StarForth

3.1.0

Generated on Tue Nov 10 2026 17:48:28 for StarForth by Doxygen 1.9.8

Tue Nov 10 2026 17:48:28

1 include/ 1

2 Topic Index 3

2.1 Topics . . . 3

3 Data Structure Index 5

3.1 Data Structures . . . 5

4 File Index 11

4.1 File List . . . 11

5 Topic Documentation 19

5.1 Architecture Detection . . . 19

5.2 Unified API Wrappers . . . 19

5.3 Runtime Architecture Detection . . . 19

5.4 Build Information . . . 20

5.5 VM Input Operations . . . 20

5.5.1 Detailed Description . . . 20

5.5.2 Function Documentation . . . 20

5.5.2.1 vm_addr_from_ptr() . . . 20

5.5.2.2 vm_input_ensure() . . . 21

5.5.2.3 vm_input_in() . . . 22

5.5.2.4 vm_input_load_line() . . . 23

5.5.2.5 vm_input_source() . . . 23

5.5.2.6 vm_input_span() . . . 24

5.5.2.7 vm_input_tib() . . . 25

5.6 Stack Operations . . . 25

5.6.1 Detailed Description . . . 26

5.6.2 Function Documentation . . . 27

5.6.2.1 vm_add_check_overflow() . . . 27

5.6.2.2 vm_cpuid() . . . 27

5.6.2.3 vm_divmod() . . . 28

5.6.2.4 vm_has_sse42() . . . 28

5.6.2.5 vm_max_asm() . . . 29

5.6.2.6 vm_memcpy_asm() . . . 29

5.6.2.7 vm_memzero_asm() . . . 31

5.6.2.8 vm_min_asm() . . . 31

5.6.2.9 vm_mul_double() . . . 31

5.6.2.10 vm_pop_asm() . . . 32

5.6.2.11 vm_push_asm() . . . 33

5.6.2.12 vm_rpop_asm() . . . 35

5.6.2.13 vm_rpush_asm() . . . 36

5.6.2.14 vm_strcmp_asm() . . . 37

5.6.2.15 vm_sub_check_overflow() . . . 37

5.7 Fast Primitive Word Implementations . . . 37
5.7.1 Detailed Description . . . 38
5.7.2 Macro Definition Documentation . . . 38
5.7.2.1 PRIM_2DROP_ARM64 . . . 38
5.7.2.2 PRIM_2DUP_ARM64 . . . 39
5.7.2.3 PRIM_AND_ARM64 . . . 39
5.7.2.4 PRIM_BRANCH_ARM64 . . . 39
5.7.2.5 PRIM_C_FETCH_ARM64 . . . 39
5.7.2.6 PRIM_C_STORE_ARM64 . . . 39
5.7.2.7 PRIM_DROP_ARM64 . . . 40
5.7.2.8 PRIM_DUP_ARM64 . . . 40
5.7.2.9 PRIM_EQUALS_ARM64 . . . 40
5.7.2.10 PRIM_EXECUTE_ARM64 . . . 40
5.7.2.11 PRIM_FETCH_ARM64 . . . 40
5.7.2.12 PRIM_GREATER_ARM64 . . . 41
5.7.2.13 PRIM_INVERT_ARM64 . . . 41
5.7.2.14 PRIM_LESS_ARM64 . . . 41
5.7.2.15 PRIM_MINUS_ARM64 . . . 41
5.7.2.16 PRIM_MOD_ARM64 . . . 41
5.7.2.17 PRIM_NEGATE_ARM64 . . . 42
5.7.2.18 PRIM_OR_ARM64 . . . 42
5.7.2.19 PRIM_OVER_ARM64 . . . 42
5.7.2.20 PRIM_PLUS_ARM64 . . . 42
5.7.2.21 PRIM_R_FETCH_ARM64 . . . 42
5.7.2.22 PRIM_R_FROM_ARM64 . . . 43
5.7.2.23 PRIM_ROT_ARM64 . . . 43
5.7.2.24 PRIM_SLASH_ARM64 . . . 43
5.7.2.25 PRIM_STAR_ARM64 . . . 43
5.7.2.26 PRIM_STORE_ARM64 . . . 43
5.7.2.27 PRIM_SWAP_ARM64 . . . 44
5.7.2.28 PRIM_TO_R_ARM64 . . . 44
5.7.2.29 PRIM_XOR_ARM64 . . . 44
5.7.2.30 PRIM_ZBRANCH_ARM64 . . . 44
5.7.2.31 PRIM_ZERO_EQUALS_ARM64 . . . 44
5.7.2.32 PRIM_ZERO_LESS_ARM64 . . . 45
5.8 Control Flow Stack . . . 45
5.8.1 Detailed Description . . . 45
5.8.2 Function Documentation . . . 45
5.8.2.1 cf_epoch_sync() . . . 45
5.8.2.2 cf_peek_item() . . . 46
5.8.2.3 cf_pop_item() . . . 47
5.8.2.4 cf_push_item() . . . 47

5.9 Runtime Branch Helpers . . . 48
5.9.1 Detailed Description . . . 49
5.9.2 Function Documentation . . . 49
5.9.2.1 control_forth_0branch() . . . 49
5.9.2.2 control_forth_again() . . . 50
5.9.2.3 control_forth_begin() . . . 51
5.9.2.4 control_forth_branch() . . . 51
5.9.2.5 control_forth_case() . . . 52
5.9.2.6 control_forth_do() . . . 53
5.9.2.7 control_forth_else() . . . 53
5.9.2.8 control_forth_endcase() . . . 54
5.9.2.9 control_forth_endof() . . . 55
5.9.2.10 control_forth_EXIT() . . . 56
5.9.2.11 control_forth_I() . . . 56
5.9.2.12 control_forth_if() . . . 57
5.9.2.13 control_forth_J() . . . 57
5.9.2.14 control_forth_leave() . . . 58
5.9.2.15 control_forth_loop() . . . 59
5.9.2.16 control_forth_of() . . . 60
5.9.2.17 control_forth_plus_loop() . . . 60
5.9.2.18 control_forth_qdo() . . . 61
5.9.2.19 control_forth_repeat() . . . 62
5.9.2.20 control_forth_runtime_do() . . . 63
5.9.2.21 control_forth_runtime_leave() . . . 63
5.9.2.22 control_forth_runtime_loop() . . . 64
5.9.2.23 control_forth_runtime_plus_loop() . . . 64
5.9.2.24 control_forth_runtime_qdo() . . . 65
5.9.2.25 control_forth_then() . . . 65
5.9.2.26 control_forth_UNLOOP() . . . 66
5.9.2.27 control_forth_until() . . . 66
5.9.2.28 control_forth_while() . . . 67
5.9.2.29 register_control_words() . . . 67
5.10 FORTH-79 Control Flow Words . . . 70
5.11 FORTH-79 Defining Words . . . 70
5.12 FORTH-79 Dictionary Manipulation Words . . . 70
5.13 FORTH-79 Dictionary & Compilation Words . . . 72
5.14 FORTH-79 Line Editor Words . . . 72
5.15 Framebuffer Words . . . 72
5.16 Keyboard Words . . . 73
5.17 Log Words . . . 74
5.18 FORTH-79 Logical Operations . . . 75
5.19 FORTH-79 Memory Operations . . . 75

5.20 FORTH-79 Return Stack Operations . . . 76
5.21 REPL Scrollback Words . . . 76
5.22 FORTH-79 System & Environment Words . . . 76
5.23 TrueType Text Words . . . 77
5.24 FORTH-79 Vocabulary System Words . . . 77

6 Data Structure Documentation **79**

6.1 __attribute__ Struct Reference . . . 79
6.1.1 Detailed Description . . . 80
6.1.2 Field Documentation . . . 80
6.1.2.1 access_size . . . 80
6.1.2.2 address . . . 80
6.1.2.3 address_space_id . . . 80
6.1.2.4 base_address . . . 80
6.1.2.5 bus . . . 80
6.1.2.6 checksum . . . 80
6.1.2.7 creator_id . . . 80
6.1.2.8 creator_revision . . . 80
6.1.2.9 end_bus . . . 81
6.1.2.10 extended_checksum . . . 81
6.1.2.11 flags [1/2] . . . 81
6.1.2.12 flags [2/2] . . . 81
6.1.2.13 gsi . . . 81
6.1.2.14 gsi_base . . . 81
6.1.2.15 hdr . . . 81
6.1.2.16 io_apic_address . . . 81
6.1.2.17 io_apic_id . . . 81
6.1.2.18 length [1/2] . . . 81
6.1.2.19 length [2/2] . . . 82
6.1.2.20 local_apic_address . . . 82
6.1.2.21 oem_id . . . 82
6.1.2.22 oem_revision . . . 82
6.1.2.23 oem_table_id . . . 82
6.1.2.24 register_bit_offset . . . 82
6.1.2.25 register_bit_width . . . 82
6.1.2.26 reserved [1/3] . . . 82
6.1.2.27 reserved [2/3] . . . 82
6.1.2.28 reserved [3/3] . . . 82
6.1.2.29 revision . . . 83
6.1.2.30 rsdt_address . . . 83
6.1.2.31 segment_group . . . 83
6.1.2.32 signature . . . 83

6.1.2.33 source . . . 83
6.1.2.34 start_bus . . . 83
6.1.2.35 type . . . 83
6.1.2.36 xsdt_address . . . 83
6.2 _EFI_BLOCK_IO_PROTOCOL Struct Reference . . . 84
6.2.1 Field Documentation . . . 84
6.2.1.1 FlushBlocks . . . 84
6.2.1.2 Media . . . 84
6.2.1.3 ReadBlocks . . . 84
6.2.1.4 Reset . . . 84
6.2.1.5 Revision . . . 85
6.2.1.6 WriteBlocks . . . 85
6.3 _EFI_BOOT_SERVICES Struct Reference . . . 85
6.3.1 Field Documentation . . . 86
6.3.1.1 AllocatePages . . . 86
6.3.1.2 AllocatePool . . . 86
6.3.1.3 CheckEvent . . . 86
6.3.1.4 CloseEvent . . . 86
6.3.1.5 CloseProtocol . . . 86
6.3.1.6 ConnectController . . . 87
6.3.1.7 CreateEvent . . . 87
6.3.1.8 DisconnectController . . . 87
6.3.1.9 Exit . . . 87
6.3.1.10 ExitBootServices . . . 87
6.3.1.11 FreePages . . . 87
6.3.1.12 FreePool . . . 87
6.3.1.13 GetMemoryMap . . . 87
6.3.1.14 GetNextMonotonicCount . . . 87
6.3.1.15 HandleProtocol . . . 88
6.3.1.16 Hdr . . . 88
6.3.1.17 InstallConfigurationTable . . . 88
6.3.1.18 InstallProtocolInterface . . . 88
6.3.1.19 LoadImage . . . 88
6.3.1.20 LocateDevicePath . . . 88
6.3.1.21 LocateHandle . . . 88
6.3.1.22 LocateHandleBuffer . . . 88
6.3.1.23 LocateProtocol . . . 88
6.3.1.24 OpenProtocol . . . 89
6.3.1.25 OpenProtocolInformation . . . 89
6.3.1.26 ProtocolsPerHandle . . . 89
6.3.1.27 RaiseTPL . . . 89
6.3.1.28 RegisterProtocolNotify . . . 89

6.3.1.29 ReinstallProtocolInterface . . . 89
6.3.1.30 Reserved . . . 89
6.3.1.31 RestoreTPL . . . 89
6.3.1.32 SetTimer . . . 89
6.3.1.33 SetWatchdogTimer . . . 89
6.3.1.34 SignalEvent . . . 90
6.3.1.35 Stall . . . 90
6.3.1.36 StartImage . . . 90
6.3.1.37 UninstallProtocolInterface . . . 90
6.3.1.38 UnloadImage . . . 90
6.3.1.39 WaitForEvent . . . 90
6.4 _EFI_FILE_PROTOCOL Struct Reference . . . 90
6.4.1 Field Documentation . . . 91
6.4.1.1 Close . . . 91
6.4.1.2 Delete . . . 91
6.4.1.3 Flush . . . 91
6.4.1.4 GetInfo . . . 91
6.4.1.5 GetPosition . . . 91
6.4.1.6 Open . . . 91
6.4.1.7 Read . . . 92
6.4.1.8 Revision . . . 92
6.4.1.9 SetInfo . . . 92
6.4.1.10 SetPosition . . . 92
6.4.1.11 Write . . . 92
6.5 _EFI_GRAPHICS_OUTPUT_PROTOCOL Struct Reference . . . 92
6.5.1 Field Documentation . . . 93
6.5.1.1 Blt . . . 93
6.5.1.2 Mode . . . 93
6.5.1.3 QueryMode . . . 94
6.5.1.4 SetMode . . . 94
6.6 _EFI_RUNTIME_SERVICES Struct Reference . . . 94
6.6.1 Field Documentation . . . 95
6.6.1.1 ConvertPointer . . . 95
6.6.1.2 GetNextHighMonotonicCount . . . 95
6.6.1.3 GetNextVariableName . . . 95
6.6.1.4 GetTime . . . 95
6.6.1.5 GetVariable . . . 95
6.6.1.6 GetWakeupTime . . . 95
6.6.1.7 Hdr . . . 95
6.6.1.8 ResetSystem . . . 95
6.6.1.9 SetTime . . . 95
6.6.1.10 SetVariable . . . 96

6.6.1.11 SetVirtualAddressMap . . . 96
6.6.1.12 SetWakeupTime . . . 96
6.7 _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL Struct Reference . . . 96
6.7.1 Field Documentation . . . 96
6.7.1.1 OpenVolume . . . 96
6.7.1.2 Revision . . . 97
6.8 _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL Struct Reference . . . 97
6.8.1 Field Documentation . . . 97
6.8.1.1 ClearScreen . . . 97
6.8.1.2 EnableCursor . . . 97
6.8.1.3 Mode . . . 98
6.8.1.4 OutputString . . . 98
6.8.1.5 QueryMode . . . 98
6.8.1.6 Reset . . . 98
6.8.1.7 SetAttribute . . . 98
6.8.1.8 SetCursorPosition . . . 98
6.8.1.9 SetMode . . . 98
6.8.1.10 TestString . . . 98
6.9 _EFI_SYSTEM_TABLE Struct Reference . . . 99
6.9.1 Field Documentation . . . 99
6.9.1.1 BootServices . . . 99
6.9.1.2 ConfigurationTable . . . 99
6.9.1.3 ConIn . . . 99
6.9.1.4 ConOut . . . 100
6.9.1.5 ConsoleInHandle . . . 100
6.9.1.6 ConsoleOutHandle . . . 100
6.9.1.7 FirmwareRevision . . . 100
6.9.1.8 FirmwareVendor . . . 100
6.9.1.9 Hdr . . . 100
6.9.1.10 NumberOfTableEntries . . . 100
6.9.1.11 RuntimeServices . . . 100
6.9.1.12 StandardErrorHandle . . . 100
6.9.1.13 StdErr . . . 101
6.10 artemis_sig_t Struct Reference . . . 101
6.10.1 Field Documentation . . . 101
6.10.1.1 _pad . . . 101
6.10.1.2 disk_uuid . . . 101
6.10.1.3 genesis_time_ns . . . 101
6.10.1.4 hdr_crc . . . 101
6.10.1.5 log_region_devblocks . . . 102
6.10.1.6 log_region_offset . . . 102
6.10.1.7 magic . . . 102

6.10.1.8 metadata_devblocks . . . 102
6.11 BayesianLatencyPosterior Struct Reference . . . 102
6.11.1 Detailed Description . . . 103
6.11.2 Field Documentation . . . 103
6.11.2.1 credible_lower_95 . . . 103
6.11.2.2 credible_lower_99 . . . 103
6.11.2.3 credible_upper_95 . . . 103
6.11.2.4 credible_upper_99 . . . 103
6.11.2.5 mean_ns_q48 . . . 103
6.11.2.6 median_ns_q48 . . . 103
6.11.2.7 sample_count . . . 104
6.11.2.8 stddev_ns_q48 . . . 104
6.12 blk_bam_entry_t Struct Reference . . . 104
6.12.1 Field Documentation . . . 104
6.12.1.1 allocated . . . 104
6.12.1.2 dirty . . . 104
6.13 blk_dev_slot Struct Reference . . . 105
6.13.1 Field Documentation . . . 105
6.13.1.1 bam . . . 105
6.13.1.2 bam_dirty . . . 105
6.13.1.3 cache . . . 106
6.13.1.4 dev . . . 106
6.13.1.5 devblock_base_4k . . . 106
6.13.1.6 format_state . . . 106
6.13.1.7 has_ever_claimed . . . 106
6.13.1.8 migration_scan_lbn . . . 106
6.13.1.9 next . . . 106
6.13.1.10 raw_base . . . 107
6.13.1.11 start_lbn . . . 107
6.13.1.12 total_blkio_blocks_1k . . . 107
6.13.1.13 user_blocks . . . 107
6.13.1.14 vol_meta . . . 107
6.13.1.15 vol_meta_dirty . . . 107
6.14 blk_meta_t Struct Reference . . . 108
6.14.1 Field Documentation . . . 108
6.14.1.1 acl_allow . . . 108
6.14.1.2 acl_reserved . . . 108
6.14.1.3 acl_ttl . . . 108
6.14.1.4 app_data . . . 109
6.14.1.5 chain_length . . . 109
6.14.1.6 checksum . . . 109
6.14.1.7 content_length . . . 109

6.14.1.8 content_type . . . 109
6.14.1.9 created_time . . . 109
6.14.1.10 encoding . . . 109
6.14.1.11 entropy . . . 109
6.14.1.12 flags . . . 109
6.14.1.13 hash . . . 110
6.14.1.14 magic . . . 110
6.14.1.15 modified_time . . . 110
6.14.1.16 next_block . . . 110
6.14.1.17 owner_fp . . . 110
6.14.1.18 padding . . . 110
6.14.1.19 parent_block . . . 110
6.14.1.20 prev_block . . . 110
6.14.1.21 reserved1 . . . 110
6.14.1.22 reserved_future . . . 111
6.14.1.23 write_count . . . 111
6.15 blk_reloc_entry_t Struct Reference . . . 111
6.15.1 Field Documentation . . . 111
6.15.1.1 actual_lbn . . . 111
6.15.1.2 home_lbn . . . 111
6.16 blk_volume_meta_t Struct Reference . . . 111
6.16.1 Field Documentation . . . 112
6.16.1.1 _pad . . . 112
6.16.1.2 bam_devblocks . . . 112
6.16.1.3 bam_start . . . 112
6.16.1.4 created_time . . . 113
6.16.1.5 devblock_base . . . 113
6.16.1.6 first_free . . . 113
6.16.1.7 flags . . . 113
6.16.1.8 free_blocks . . . 113
6.16.1.9 hdr_crc . . . 113
6.16.1.10 label . . . 113
6.16.1.11 last_allocated . . . 113
6.16.1.12 magic . . . 114
6.16.1.13 meta_fence_blocks . . . 114
6.16.1.14 mounted_time . . . 114
6.16.1.15 reloc_devblocks . . . 114
6.16.1.16 reloc_start . . . 114
6.16.1.17 reserved_disk_lo . . . 114
6.16.1.18 reserved_ram_lo . . . 114
6.16.1.19 total_blocks . . . 115
6.16.1.20 total_devblocks . . . 115

6.16.1.21 total_volumes . . . 115
6.16.1.22 tracked_blocks . . . 115
6.16.1.23 version . . . 115
6.17 blkcfg_t Struct Reference . . . 115
6.17.1 Field Documentation . . . 116
6.17.1.1 backend . . . 116
6.17.1.2 blocks . . . 116
6.17.1.3 fbs . . . 116
6.17.1.4 file_path . . . 116
6.17.1.5 read_only . . . 116
6.17.1.6 size_bytes . . . 116
6.18 blkio_dev Struct Reference . . . 116
6.18.1 Field Documentation . . . 117
6.18.1.1 forth_block_size . . . 117
6.18.1.2 is_open . . . 117
6.18.1.3 state . . . 117
6.18.1.4 total_blocks . . . 117
6.18.1.5 vt . . . 117
6.19 blkio_file_state_t Struct Reference . . . 118
6.19.1 Field Documentation . . . 118
6.19.1.1 fbs . . . 118
6.19.1.2 fd . . . 118
6.19.1.3 path . . . 118
6.19.1.4 read_only . . . 118
6.19.1.5 total_blocks . . . 118
6.20 blkio_info_t Struct Reference . . . 119
6.20.1 Field Documentation . . . 119
6.20.1.1 forth_block_size . . . 119
6.20.1.2 phys_sector_size . . . 119
6.20.1.3 phys_size_bytes . . . 119
6.20.1.4 read_only . . . 119
6.20.1.5 total_blocks . . . 119
6.21 blkio_params_t Struct Reference . . . 120
6.21.1 Field Documentation . . . 120
6.21.1.1 forth_block_size . . . 120
6.21.1.2 opaque . . . 120
6.21.1.3 total_blocks . . . 120
6.22 blkio_ram_state_t Struct Reference . . . 120
6.22.1 Field Documentation . . . 121
6.22.1.1 base . . . 121
6.22.1.2 fbs . . . 121
6.22.1.3 read_only . . . 121

6.22.1.4 total_blocks . . . 121
6.23 blkio_vtable_t Struct Reference . . . 121
6.23.1 Field Documentation . . . 122
6.23.1.1 close . . . 122
6.23.1.2 flush . . . 122
6.23.1.3 info . . . 122
6.23.1.4 open . . . 122
6.23.1.5 read . . . 122
6.23.1.6 write . . . 122
6.24 BlkioUsbState Struct Reference . . . 123
6.24.1 Field Documentation . . . 123
6.24.1.1 scsi_block_size . . . 123
6.24.1.2 scsi_blocks_per_fblock . . . 123
6.24.1.3 slot_id . . . 123
6.24.1.4 total_forth_blocks . . . 124
6.24.1.5 xdev . . . 124
6.25 BlockBuffer Struct Reference . . . 124
6.25.1 Detailed Description . . . 124
6.25.2 Field Documentation . . . 124
6.25.2.1 accessed . . . 124
6.25.2.2 block_number . . . 125
6.25.2.3 data . . . 125
6.25.2.4 dirty . . . 125
6.25.2.5 hw_address . . . 125
6.25.2.6 type . . . 125
6.26 BootDisk Struct Reference . . . 125
6.26.1 Field Documentation . . . 126
6.26.1.1 disk_guid . . . 126
6.26.1.2 gpt . . . 126
6.26.1.3 is_stick . . . 126
6.27 BootInfo Struct Reference . . . 126
6.27.1 Field Documentation . . . 127
6.27.1.1 acpi_table . . . 127
6.27.1.2 args . . . 127
6.27.1.3 boot_disk . . . 127
6.27.1.4 dtb . . . 127
6.27.1.5 framebuffer . . . 127
6.27.1.6 kernel_stack_base . . . 127
6.27.1.7 kernel_stack_size . . . 127
6.27.1.8 memory_map . . . 128
6.27.1.9 memory_map_descriptor_size . . . 128
6.27.1.10 memory_map_size . . . 128

6.27.1.11 runtime_services . . . 128
6.27.1.12 uefi_boot_services_exited . . . 128
6.28 cache_slot_t Struct Reference . . . 128
6.28.1 Field Documentation . . . 129
6.28.1.1 data . . . 129
6.28.1.2 devblock . . . 129
6.28.1.3 dirty . . . 129
6.28.1.4 loaded . . . 129
6.28.1.5 meta . . . 129
6.28.1.6 meta_dirty . . . 129
6.28.1.7 valid . . . 130
6.29 CapsuleDirHeader Struct Reference . . . 130
6.29.1 Field Documentation . . . 130
6.29.1.1 arena_base . . . 130
6.29.1.2 arena_size . . . 130
6.29.1.3 desc_capacity . . . 130
6.29.1.4 desc_count . . . 131
6.29.1.5 dir_hash . . . 131
6.29.1.6 magic . . . 131
6.29.1.7 name_count . . . 131
6.29.1.8 reserved . . . 131
6.30 CapsuleNameEntry Struct Reference . . . 131
6.30.1 Field Documentation . . . 132
6.30.1.1 name . . . 132
6.31 CapsuleRunRecord Struct Reference . . . 132
6.31.1 Field Documentation . . . 132
6.31.1.1 capsule_hash . . . 132
6.31.1.2 capsule_id . . . 132
6.31.1.3 ended_ns . . . 133
6.31.1.4 flags . . . 133
6.31.1.5 post_dict_hash . . . 133
6.31.1.6 pre_dict_hash . . . 133
6.31.1.7 reserved . . . 133
6.31.1.8 result_code . . . 133
6.31.1.9 run_id . . . 133
6.31.1.10 started_ns . . . 133
6.31.1.11 vm_id . . . 134
6.32 CapsuleSigEntry Struct Reference . . . 134
6.32.1 Field Documentation . . . 134
6.32.1.1 _pad . . . 134
6.32.1.2 has_sig . . . 134
6.32.1.3 sig . . . 134

6.33 CDConfigEntry Struct Reference . . . 134
6.33.1 Field Documentation . . . 135
6.33.1.1 _pad . . . 135
6.33.1.2 benefit_score . . . 135
6.33.1.3 config_bits . . . 135
6.33.1.4 has_prev . . . 135
6.33.1.5 prev_locality_q16 . . . 135
6.33.1.6 prev_window . . . 135
6.33.1.7 tick_count . . . 136
6.34 CDConfigTable Struct Reference . . . 136
6.34.1 Field Documentation . . . 136
6.34.1.1 current_config . . . 136
6.34.1.2 current_regime . . . 137
6.34.1.3 entries . . . 137
6.34.1.4 num_configs . . . 137
6.34.1.5 num_regimes . . . 137
6.34.1.6 regime_scores . . . 137
6.34.1.7 regime_ticks . . . 137
6.34.1.8 total_regime_ticks . . . 137
6.35 CDRegimeMetrics Struct Reference . . . 138
6.35.1 Field Documentation . . . 138
6.35.1.1 diversity_q16 . . . 138
6.35.1.2 locality_q16 . . . 138
6.35.1.3 volatility_q16 . . . 138
6.36 CDTuning Struct Reference . . . 138
6.36.1 Field Documentation . . . 139
6.36.1.1 anova_weight_q16 . . . 139
6.36.1.2 diversity_high_threshold_q16 . . . 139
6.36.1.3 id_window_depth . . . 139
6.36.1.4 joint_weight_q16 . . . 139
6.36.1.5 locality_high_threshold_q16 . . . 139
6.36.1.6 locality_lookback . . . 140
6.36.1.7 num_configs . . . 140
6.36.1.8 num_regimes . . . 140
6.36.1.9 reward_gain . . . 140
6.36.1.10 score_decay_factor_q16 . . . 140
6.36.1.11 score_max . . . 140
6.36.1.12 score_min . . . 140
6.36.1.13 ucb_k . . . 140
6.36.1.14 volatility_high_threshold_q16 . . . 141
6.37 cf_item_t Struct Reference . . . 141
6.37.1 Field Documentation . . . 141

6.37.1.1 addr . . . 141
6.37.1.2 tag . . . 141
6.38 CLIConfig Struct Reference . . . 141
6.38.1 Detailed Description . . . 142
6.38.2 Field Documentation . . . 142
6.38.2.1 benchmark . . . 142
6.38.2.2 benchmark_iterations . . . 142
6.38.2.3 break_me . . . 142
6.38.2.4 disk_img_path . . . 142
6.38.2.5 doe_experiment . . . 143
6.38.2.6 fail_fast . . . 143
6.38.2.7 heartbeat_log_mode . . . 143
6.38.2.8 log_level . . . 143
6.38.2.9 log_level_explicitly_set . . . 143
6.38.2.10 ram_disk_mb . . . 143
6.38.2.11 run_tests . . . 143
6.38.2.12 run_tests_flag_observed . . . 143
6.38.2.13 script_mode . . . 144
6.39 ControlFlowState Struct Reference . . . 144
6.39.1 Field Documentation . . . 144
6.39.1.1 cf_last_mode . . . 144
6.39.1.2 cf_sp . . . 145
6.39.1.3 cf_stack . . . 145
6.39.1.4 endof_addrs . . . 145
6.39.1.5 endof_mark_sp . . . 145
6.39.1.6 endof_mark_stack . . . 145
6.39.1.7 endof_sp . . . 145
6.39.1.8 leave_addrs . . . 145
6.39.1.9 leave_mark_sp . . . 146
6.39.1.10 leave_mark_stack . . . 146
6.39.1.11 leave_sp . . . 146
6.40 der_span_t Struct Reference . . . 146
6.40.1 Field Documentation . . . 146
6.40.1.1 len . . . 146
6.40.1.2 p . . . 146
6.41 DiagEvent Struct Reference . . . 147
6.41.1 Field Documentation . . . 147
6.41.1.1 code . . . 147
6.41.1.2 value . . . 147
6.42 DictEntry Struct Reference . . . 147
6.42.1 Field Documentation . . . 148
6.42.1.1 acl_allow . . . 148

6.42.1.2 acl_default . . . 148
6.42.1.3 acl_mode . . . 148
6.42.1.4 acl_pinned . . . 148
6.42.1.5 acl_ttl . . . 149
6.42.1.6 execution_heat . . . 149
6.42.1.7 flags . . . 149
6.42.1.8 func . . . 149
6.42.1.9 link . . . 149
6.42.1.10 name . . . 149
6.42.1.11 name_len . . . 150
6.42.1.12 physics . . . 150
6.42.1.13 transition_metrics . . . 150
6.42.1.14 word_id . . . 150
6.43 DictPhysics Struct Reference . . . 150
6.43.1 Field Documentation . . . 151
6.43.1.1 acl_hint . . . 151
6.43.1.2 avg_latency_ns . . . 151
6.43.1.3 last_active_ns . . . 151
6.43.1.4 last_decay_ns . . . 151
6.43.1.5 last_decay_tick . . . 151
6.43.1.6 mass_bytes . . . 151
6.43.1.7 pubsub_mask . . . 151
6.43.1.8 state_flags . . . 151
6.43.1.9 temperature_q8 . . . 152
6.44 DoeMetrics Struct Reference . . . 152
6.44.1 Detailed Description . . . 153
6.44.2 Field Documentation . . . 153
6.44.2.1 actual_window_size . . . 153
6.44.2.2 adaptive_shrink_rate . . . 153
6.44.2.3 avg_word_heat . . . 154
6.44.2.4 bucket_hit_percent . . . 154
6.44.2.5 bucket_hits . . . 154
6.44.2.6 bucket_search_latency_ns . . . 154
6.44.2.7 bucket_search_stddev_ns . . . 154
6.44.2.8 cache_demotions . . . 154
6.44.2.9 cache_hit_latency_ns . . . 154
6.44.2.10 cache_hit_percent . . . 154
6.44.2.11 cache_hit_stddev_ns . . . 155
6.44.2.12 cache_hits . . . 155
6.44.2.13 cache_promotions . . . 155
6.44.2.14 ci_lower_95 . . . 155
6.44.2.15 ci_upper_95 . . . 155

6.44.2.16 context_accuracy_percent . . . 155
6.44.2.17 context_correct . . . 155
6.44.2.18 context_predictions_total . . . 155
6.44.2.19 cpu_freq_delta_mhz_q48 . . . 156
6.44.2.20 cpu_temp_delta_c_q48 . . . 156
6.44.2.21 decay_min_interval_ns . . . 156
6.44.2.22 decay_rate_q16 . . . 156
6.44.2.23 decay_slope . . . 156
6.44.2.24 dictionary_lookups . . . 156
6.44.2.25 early_exit_count . . . 156
6.44.2.26 enable_hotwords_cache . . . 156
6.44.2.27 enable_loop_1_heat_tracking . . . 157
6.44.2.28 enable_loop_2_rolling_window . . . 157
6.44.2.29 enable_loop_3_linear_decay . . . 157
6.44.2.30 enable_loop_4_pipelining . . . 157
6.44.2.31 enable_loop_5_window_inference . . . 157
6.44.2.32 enable_loop_6_decay_inference . . . 157
6.44.2.33 enable_loop_7_adaptive_heartrate . . . 157
6.44.2.34 enable_pipelining . . . 157
6.44.2.35 final_effective_window_size . . . 158
6.44.2.36 heat_cache_demotion_threshold . . . 158
6.44.2.37 hot_word_count . . . 158
6.44.2.38 inference_run_count . . . 158
6.44.2.39 memory_allocated_bytes . . . 158
6.44.2.40 prefetch_accuracy_percent . . . 158
6.44.2.41 prefetch_attempts . . . 158
6.44.2.42 prefetch_hits . . . 158
6.44.2.43 rolling_window_size . . . 159
6.44.2.44 rolling_window_width . . . 159
6.44.2.45 speedup_vs_baseline . . . 159
6.44.2.46 stale_word_count . . . 159
6.44.2.47 stale_word_ratio . . . 159
6.44.2.48 tick_count . . . 159
6.44.2.49 tick_target_ns . . . 159
6.44.2.50 total_executions . . . 159
6.44.2.51 total_heat . . . 160
6.44.2.52 total_lookups . . . 160
6.44.2.53 total_runtime_ms . . . 160
6.44.2.54 vm_workload_duration_ns_q48 . . . 160
6.44.2.55 window_diversity_percent . . . 160
6.44.2.56 window_final_size_bytes . . . 160
6.44.2.57 window_tuning_checks . . . 160

6.44.2.58 window_variance_q48 . . . 160
6.44.2.59 words_executed . . . 161
6.45 ed_point_t Struct Reference . . . 161
6.45.1 Field Documentation . . . 161
6.45.1.1 x . . . 161
6.45.1.2 y . . . 162
6.45.1.3 z . . . 162
6.46 EFI_BLOCK_IO_MEDIA Struct Reference . . . 162
6.46.1 Field Documentation . . . 162
6.46.1.1 BlockSize . . . 162
6.46.1.2 IoAlign . . . 162
6.46.1.3 LastBlock . . . 163
6.46.1.4 LogicalPartition . . . 163
6.46.1.5 MediaId . . . 163
6.46.1.6 MediaPresent . . . 163
6.46.1.7 ReadOnly . . . 163
6.46.1.8 RemovableMedia . . . 163
6.46.1.9 WriteCaching . . . 163
6.47 EFI_CONFIGURATION_TABLE Struct Reference . . . 163
6.47.1 Field Documentation . . . 164
6.47.1.1 VendorGuid . . . 164
6.47.1.2 VendorTable . . . 164
6.48 EFI_DEVICE_PATH_PROTOCOL Struct Reference . . . 164
6.48.1 Field Documentation . . . 164
6.48.1.1 Length . . . 164
6.48.1.2 SubType . . . 164
6.48.1.3 Type . . . 165
6.49 EFI_FILE_INFO Struct Reference . . . 165
6.49.1 Field Documentation . . . 165
6.49.1.1 Attribute . . . 165
6.49.1.2 CreateTime . . . 166
6.49.1.3 FileName . . . 166
6.49.1.4 FileSize . . . 166
6.49.1.5 LastAccessTime . . . 166
6.49.1.6 ModificationTime . . . 166
6.49.1.7 PhysicalSize . . . 166
6.49.1.8 Size . . . 166
6.50 EFI_GRAPHICS_OUTPUT_MODE_INFORMATION Struct Reference . . . 167
6.50.1 Field Documentation . . . 167
6.50.1.1 HorizontalResolution . . . 167
6.50.1.2 PixelFormat . . . 167
6.50.1.3 PixelInformation . . . 167

6.50.1.4 PixelsPerScanLine . . . 168
6.50.1.5 Version . . . 168
6.50.1.6 VerticalResolution . . . 168
6.51 EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE Struct Reference . . . 168
6.51.1 Field Documentation . . . 169
6.51.1.1 FrameBufferBase . . . 169
6.51.1.2 FrameBufferSize . . . 169
6.51.1.3 Info . . . 169
6.51.1.4 MaxMode . . . 169
6.51.1.5 Mode . . . 169
6.51.1.6 SizeOfInfo . . . 169
6.52 EFI_GUID Struct Reference . . . 170
6.52.1 Field Documentation . . . 170
6.52.1.1 Data1 . . . 170
6.52.1.2 Data2 . . . 170
6.52.1.3 Data3 . . . 170
6.52.1.4 Data4 . . . 170
6.53 EFI_LOADED_IMAGE_PROTOCOL Struct Reference . . . 170
6.53.1 Field Documentation . . . 171
6.53.1.1 DeviceHandle . . . 171
6.53.1.2 FilePath . . . 171
6.53.1.3 ImageBase . . . 171
6.53.1.4 ImageCodeType . . . 171
6.53.1.5 ImageDataType . . . 171
6.53.1.6 ImageSize . . . 171
6.53.1.7 LoadOptions . . . 172
6.53.1.8 LoadOptionsSize . . . 172
6.53.1.9 ParentHandle . . . 172
6.53.1.10 Reserved . . . 172
6.53.1.11 Revision . . . 172
6.53.1.12 SystemTable . . . 172
6.53.1.13 Unload . . . 172
6.54 EFI_MEMORY_DESCRIPTOR Struct Reference . . . 172
6.54.1 Field Documentation . . . 173
6.54.1.1 Attribute . . . 173
6.54.1.2 NumberOfPages . . . 173
6.54.1.3 PhysicalStart . . . 173
6.54.1.4 Type . . . 173
6.54.1.5 VirtualStart . . . 173
6.55 EFI_PIXEL_BITMASK Struct Reference . . . 173
6.55.1 Field Documentation . . . 174
6.55.1.1 BlueMask . . . 174

6.55.1.2 GreenMask . . . 174
6.55.1.3 RedMask . . . 174
6.55.1.4 ReservedMask . . . 174
6.56 EFI_TABLE_HEADER Struct Reference . . . 174
6.56.1 Field Documentation . . . 175
6.56.1.1 CRC32 . . . 175
6.56.1.2 HeaderSize . . . 175
6.56.1.3 Reserved . . . 175
6.56.1.4 Revision . . . 175
6.56.1.5 Signature . . . 175
6.57 EFI_TIME Struct Reference . . . 175
6.57.1 Field Documentation . . . 176
6.57.1.1 Day . . . 176
6.57.1.2 Daylight . . . 176
6.57.1.3 Hour . . . 176
6.57.1.4 Minute . . . 176
6.57.1.5 Month . . . 176
6.57.1.6 Nanosecond . . . 176
6.57.1.7 Pad1 . . . 176
6.57.1.8 Pad2 . . . 176
6.57.1.9 Second . . . 176
6.57.1.10 TimeZone . . . 176
6.57.1.11 Year . . . 177
6.58 Elf64_Ehdr Struct Reference . . . 177
6.58.1 Field Documentation . . . 177
6.58.1.1 e_ehsize . . . 177
6.58.1.2 e_entry . . . 177
6.58.1.3 e_flags . . . 177
6.58.1.4 e_ident . . . 178
6.58.1.5 e_machine . . . 178
6.58.1.6 e_phentsize . . . 178
6.58.1.7 e_phnum . . . 178
6.58.1.8 e_phoff . . . 178
6.58.1.9 e_shentsize . . . 178
6.58.1.10 e_shnum . . . 178
6.58.1.11 e_shoff . . . 178
6.58.1.12 e_shstrndx . . . 179
6.58.1.13 e_type . . . 179
6.58.1.14 e_version . . . 179
6.59 Elf64_Phdr Struct Reference . . . 179
6.59.1 Field Documentation . . . 179
6.59.1.1 p_align . . . 179

6.59.1.2 p_filesz . . . 179
6.59.1.3 p_flags . . . 180
6.59.1.4 p_memsz . . . 180
6.59.1.5 p_offset . . . 180
6.59.1.6 p_paddr . . . 180
6.59.1.7 p_type . . . 180
6.59.1.8 p_vaddr . . . 180
6.60 Elf64_Rel Struct Reference . . . 180
6.60.1 Field Documentation . . . 180
6.60.1.1 r_info . . . 180
6.60.1.2 r_offset . . . 181
6.61 Elf64_Rela Struct Reference . . . 181
6.61.1 Field Documentation . . . 181
6.61.1.1 r_addend . . . 181
6.61.1.2 r_info . . . 181
6.61.1.3 r_offset . . . 181
6.62 Elf64_Shdr Struct Reference . . . 181
6.62.1 Field Documentation . . . 182
6.62.1.1 sh_addr . . . 182
6.62.1.2 sh_addralign . . . 182
6.62.1.3 sh_entsize . . . 182
6.62.1.4 sh_flags . . . 182
6.62.1.5 sh_info . . . 182
6.62.1.6 sh_link . . . 182
6.62.1.7 sh_name . . . 182
6.62.1.8 sh_offset . . . 183
6.62.1.9 sh_size . . . 183
6.62.1.10 sh_type . . . 183
6.63 Elf64_Sym Struct Reference . . . 183
6.63.1 Field Documentation . . . 183
6.63.1.1 st_info . . . 183
6.63.1.2 st_name . . . 183
6.63.1.3 st_other . . . 184
6.63.1.4 st_shndx . . . 184
6.63.1.5 st_size . . . 184
6.63.1.6 st_value . . . 184
6.64 ExecResult Struct Reference . . . 184
6.64.1 Field Documentation . . . 184
6.64.1.1 dsp . . . 184
6.64.1.2 error . . . 185
6.64.1.3 rsp . . . 185
6.64.1.4 stack . . . 185

6.65 FbState Struct Reference . . . 185
6.65.1 Field Documentation . . . 185
6.65.1.1 base . . . 185
6.65.1.2 fmt . . . 185
6.65.1.3 height . . . 186
6.65.1.4 ready . . . 186
6.65.1.5 scale . . . 186
6.65.1.6 stride . . . 186
6.65.1.7 width . . . 186
6.66 fdt_header_t Struct Reference . . . 186
6.66.1 Field Documentation . . . 187
6.66.1.1 boot_cpuid_phys . . . 187
6.66.1.2 last_comp_version . . . 187
6.66.1.3 magic . . . 187
6.66.1.4 off_dt_strings . . . 187
6.66.1.5 off_dt_struct . . . 187
6.66.1.6 off_mem_rsvmap . . . 187
6.66.1.7 size_dt_strings . . . 187
6.66.1.8 size_dt_struct . . . 187
6.66.1.9 totalsize . . . 188
6.66.1.10 version . . . 188
6.67 fdt_uart_t Struct Reference . . . 188
6.67.1 Field Documentation . . . 188
6.67.1.1 base . . . 188
6.67.1.2 io_width . . . 188
6.67.1.3 kind . . . 189
6.67.1.4 reg_shift . . . 189
6.68 fe25519 Struct Reference . . . 189
6.68.1 Field Documentation . . . 189
6.68.1.1 v . . . 189
6.69 FILE Struct Reference . . . 189
6.69.1 Field Documentation . . . 190
6.69.1.1 _dummy . . . 190
6.70 FramebufferInfo Struct Reference . . . 190
6.70.1 Field Documentation . . . 190
6.70.1.1 base . . . 190
6.70.1.2 height . . . 190
6.70.1.3 pixel_format . . . 190
6.70.1.4 pixels_per_scanline . . . 191
6.70.1.5 size . . . 191
6.70.1.6 width . . . 191
6.71 gdt_entry Struct Reference . . . 191

6.71.1 Detailed Description . . . 191
6.71.2 Field Documentation . . . 191
6.71.2.1 access . . . 191
6.71.2.2 base_high . . . 192
6.71.2.3 base_low . . . 192
6.71.2.4 base_mid . . . 192
6.71.2.5 granularity . . . 192
6.71.2.6 limit_low . . . 192
6.72 gdtr Struct Reference . . . 192
6.72.1 Field Documentation . . . 192
6.72.1.1 base . . . 192
6.72.1.2 limit . . . 193
6.73 gpt_part_dev_t Struct Reference . . . 193
6.73.1 Field Documentation . . . 193
6.73.1.1 disk . . . 193
6.73.1.2 fblocks . . . 194
6.73.1.3 first_fblock . . . 194
6.74 gpt_part_t Struct Reference . . . 194
6.74.1 Field Documentation . . . 194
6.74.1.1 first_lba . . . 194
6.74.1.2 last_lba . . . 194
6.74.1.3 type . . . 194
6.75 heap_block Struct Reference . . . 195
6.75.1 Field Documentation . . . 195
6.75.1.1 free . . . 195
6.75.1.2 next . . . 195
6.75.1.3 prev . . . 195
6.75.1.4 requested . . . 195
6.75.1.5 size . . . 196
6.76 HeartbeatSnapshot Struct Reference . . . 196
6.76.1 Field Documentation . . . 196
6.76.1.1 actual_window_size . . . 196
6.76.1.2 decay_slope_q48 . . . 196
6.76.1.3 hot_word_count . . . 196
6.76.1.4 published_ns . . . 197
6.76.1.5 published_tick . . . 197
6.76.1.6 stale_word_count . . . 197
6.76.1.7 total_heat . . . 197
6.76.1.8 window_width . . . 197
6.77 HeartbeatState Struct Reference . . . 197
6.77.1 Field Documentation . . . 198
6.77.1.1 bucket_collapse_flag . . . 198

6.77.1.2 bucket_mode_transitions . . . 198
6.77.1.3 bucket_sum_heat_variance . . . 198
6.77.1.4 bucket_sum_K . . . 199
6.77.1.5 bucket_sum_K_squared . . . 199
6.77.1.6 bucket_sum_window_variance . . . 199
6.77.1.7 bucket_tick_count . . . 199
6.77.1.8 check_counter . . . 199
6.77.1.9 dictionary_lookups . . . 199
6.77.1.10 early_exit_count . . . 199
6.77.1.11 heartbeat_enabled . . . 199
6.77.1.12 inference_run_count . . . 200
6.77.1.13 last_inference_tick . . . 200
6.77.1.14 m5_time_trust . . . 200
6.77.1.15 m5_variance . . . 200
6.77.1.16 run_start_ns . . . 200
6.77.1.17 snapshot_index . . . 200
6.77.1.18 snapshots . . . 200
6.77.1.19 tick_buffer . . . 200
6.77.1.20 tick_buffer_size . . . 201
6.77.1.21 tick_buffer_write_index . . . 201
6.77.1.22 tick_count . . . 201
6.77.1.23 tick_count_total . . . 201
6.77.1.24 tick_number_offset . . . 201
6.77.1.25 tick_target_ns . . . 201
6.77.1.26 words_executed . . . 201
6.77.1.27 worker . . . 202
6.78 HeartbeatTickSnapshot Struct Reference . . . 202
6.78.1 Field Documentation . . . 202
6.78.1.1 actual_window_size . . . 202
6.78.1.2 avg_word_heat . . . 202
6.78.1.3 bucket_hits_delta . . . 203
6.78.1.4 cache_hits_delta . . . 203
6.78.1.5 elapsed_ns . . . 203
6.78.1.6 estimated_jitter_ns . . . 203
6.78.1.7 hot_word_count . . . 203
6.78.1.8 l8_mode . . . 203
6.78.1.9 predicted_label_hits . . . 203
6.78.1.10 tick_interval_ns . . . 203
6.78.1.11 tick_number . . . 204
6.78.1.12 window_width . . . 204
6.78.1.13 word_executions_delta . . . 204
6.79 HeartbeatWorker Struct Reference . . . 204

6.79.1 Field Documentation . . . 204
6.79.1.1 running . . . 204
6.79.1.2 stop_requested . . . 204
6.79.1.3 tick_ns . . . 205
6.80 homeblocks_sig_t Struct Reference . . . 205
6.80.1 Field Documentation . . . 205
6.80.1.1 _pad . . . 205
6.80.1.2 cert_devblocks . . . 205
6.80.1.3 cert_offset . . . 205
6.80.1.4 drive_uuid . . . 206
6.80.1.5 hdr_crc . . . 206
6.80.1.6 identity_src_devblocks . . . 206
6.80.1.7 identity_src_offset . . . 206
6.80.1.8 magic . . . 206
6.80.1.9 metadata_devblocks . . . 206
6.80.1.10 minted_time_ns . . . 206
6.81 HotwordsCache_s Struct Reference . . . 207
6.81.1 Field Documentation . . . 207
6.81.1.1 cache . . . 207
6.81.1.2 cache_count . . . 207
6.81.1.3 enabled . . . 208
6.81.1.4 lru_index . . . 208
6.81.1.5 stats . . . 208
6.82 HotwordsStats Struct Reference . . . 208
6.82.1 Field Documentation . . . 209
6.82.1.1 bucket_hits . . . 209
6.82.1.2 bucket_reorders . . . 210
6.82.1.3 bucket_search_samples . . . 210
6.82.1.4 bucket_search_total_ns_q48 . . . 210
6.82.1.5 bucket_search_variance_sum_q48 . . . 210
6.82.1.6 cache_hit_samples . . . 210
6.82.1.7 cache_hit_total_ns_q48 . . . 211
6.82.1.8 cache_hit_variance_sum_q48 . . . 211
6.82.1.9 cache_hits . . . 211
6.82.1.10 evictions . . . 211
6.82.1.11 max_bucket_search_ns . . . 211
6.82.1.12 max_cache_hit_ns . . . 212
6.82.1.13 min_bucket_search_ns . . . 212
6.82.1.14 min_cache_hit_ns . . . 212
6.82.1.15 misses . . . 212
6.82.1.16 promotions . . . 212
6.82.1.17 total_lookups . . . 213

6.83 idt_entry Struct Reference . . . 213
6.83.1 Field Documentation . . . 213
6.83.1.1 ist . . . 213
6.83.1.2 offset_high . . . 213
6.83.1.3 offset_low . . . 213
6.83.1.4 offset_mid . . . 214
6.83.1.5 selector . . . 214
6.83.1.6 type_attr . . . 214
6.83.1.7 zero . . . 214
6.84 idtr Struct Reference . . . 214
6.84.1 Field Documentation . . . 214
6.84.1.1 base . . . 214
6.84.1.2 limit . . . 215
6.85 InferenceInputs Struct Reference . . . 215
6.85.1 Field Documentation . . . 215
6.85.1.1 hot_word_count . . . 215
6.85.1.2 last_stale_count . . . 216
6.85.1.3 last_total_heat . . . 216
6.85.1.4 prefetch_attempts . . . 216
6.85.1.5 prefetch_hits . . . 216
6.85.1.6 stale_word_count . . . 216
6.85.1.7 total_heat . . . 216
6.85.1.8 trajectory_length . . . 216
6.85.1.9 vm . . . 216
6.85.1.10 window . . . 217
6.85.1.11 word_count . . . 217
6.86 InferenceOutputs Struct Reference . . . 217
6.86.1 Field Documentation . . . 217
6.86.1.1 adaptive_decay_slope . . . 217
6.86.1.2 adaptive_window_width . . . 217
6.86.1.3 early_exited . . . 218
6.86.1.4 last_check_tick . . . 218
6.86.1.5 slope_fit_quality_q48 . . . 218
6.86.1.6 window_variance_q48 . . . 218
6.87 KernelArgs Struct Reference . . . 218
6.87.1 Field Documentation . . . 219
6.87.1.1 heap_size . . . 219
6.87.1.2 log_level . . . 219
6.87.1.3 raw . . . 219
6.87.1.4 run_doe . . . 219
6.87.1.5 screen_boot . . . 219
6.87.1.6 stack_size . . . 219

6.87.1.7 system . . . 219
6.88 kmalloc_prefix_t Struct Reference . . . 220
6.88.1 Field Documentation . . . 220
6.88.1.1 block . . . 220
6.88.1.2 magic . . . 220
6.89 kmalloc_stats_t Struct Reference . . . 220
6.89.1 Detailed Description . . . 221
6.89.2 Field Documentation . . . 221
6.89.2.1 free_bytes . . . 221
6.89.2.2 peak_bytes . . . 221
6.89.2.3 total_bytes . . . 221
6.89.2.4 used_bytes . . . 221
6.90 log_region_ctrl_t Struct Reference . . . 221
6.90.1 Field Documentation . . . 222
6.90.1.1 _pad . . . 222
6.90.1.2 devblocks . . . 222
6.90.1.3 hdr_crc . . . 222
6.90.1.4 head_slot . . . 222
6.90.1.5 magic . . . 222
6.90.1.6 record_count . . . 222
6.90.1.7 tail_slot . . . 223
6.91 log_slot_t Struct Reference . . . 223
6.91.1 Field Documentation . . . 223
6.91.1.1 level . . . 223
6.91.1.2 msg . . . 223
6.91.1.3 msg_len . . . 223
6.91.1.4 source . . . 223
6.91.1.5 timestamp . . . 224
6.92 MemoryProfile Struct Reference . . . 224
6.92.1 Field Documentation . . . 224
6.92.1.1 cache_hits . . . 224
6.92.1.2 cache_misses . . . 224
6.92.1.3 reads . . . 224
6.92.1.4 total_bytes_read . . . 224
6.92.1.5 total_bytes_written . . . 224
6.92.1.6 writes . . . 225
6.93 ParityPacket Struct Reference . . . 225
6.93.1 Detailed Description . . . 225
6.93.2 Field Documentation . . . 225
6.93.2.1 bootstrap_result . . . 225
6.93.2.2 header_hash64 . . . 225
6.93.2.3 here_offset . . . 226

6.93.2.4 latest_word_id . . . 226
6.93.2.5 tests_errors . . . 226
6.93.2.6 tests_failed . . . 226
6.93.2.7 tests_passed . . . 226
6.93.2.8 tests_skipped . . . 226
6.93.2.9 tests_total . . . 226
6.93.2.10 window_hash64 . . . 226
6.93.2.11 word_count . . . 227
6.94 PciDevice Struct Reference . . . 227
6.94.1 Field Documentation . . . 227
6.94.1.1 bus . . . 227
6.94.1.2 device . . . 227
6.94.1.3 device_id . . . 227
6.94.1.4 function . . . 228
6.94.1.5 vendor_id . . . 228
6.95 physics_analytics_event_header Struct Reference . . . 228
6.95.1 Detailed Description . . . 228
6.95.2 Field Documentation . . . 228
6.95.2.1 channel . . . 228
6.95.2.2 payload_bytes . . . 229
6.95.2.3 reserved . . . 229
6.95.2.4 timestamp_ns . . . 229
6.96 physics_analytics_header Struct Reference . . . 229
6.96.1 Detailed Description . . . 230
6.96.2 Field Documentation . . . 230
6.96.2.1 consume_seq . . . 230
6.96.2.2 dropped_events . . . 231
6.96.2.3 flags . . . 231
6.96.2.4 heap_bytes . . . 231
6.96.2.5 magic . . . 231
6.96.2.6 produce_seq . . . 231
6.96.2.7 read_offset . . . 231
6.96.2.8 ring_bytes . . . 232
6.96.2.9 ring_offset . . . 232
6.96.2.10 scratch_bytes . . . 232
6.96.2.11 scratch_offset . . . 232
6.96.2.12 summary_bytes . . . 232
6.96.2.13 summary_offset . . . 232
6.96.2.14 version_major . . . 233
6.96.2.15 version_minor . . . 233
6.96.2.16 write_offset . . . 233
6.97 physics_analytics_heap Struct Reference . . . 233

6.97.1 Detailed Description . . . 234
6.97.2 Field Documentation . . . 234
6.97.2.1 base . . . 234
6.97.2.2 bytes . . . 234
6.97.2.3 header . . . 234
6.97.2.4 ring . . . 235
6.97.2.5 ring_bytes . . . 235
6.97.2.6 scratch . . . 235
6.97.2.7 scratch_bytes . . . 235
6.97.2.8 summary . . . 235
6.97.2.9 summary_bytes . . . 235
6.98 physics_host_snapshot Struct Reference . . . 236
6.98.1 Detailed Description . . . 237
6.98.2 Field Documentation . . . 237
6.98.2.1 backend . . . 237
6.98.2.2 backend_seq . . . 237
6.98.2.3 cgroup_cpu_usage_us . . . 237
6.98.2.4 cgroup_memory_current_bytes . . . 238
6.98.2.5 cpu_count . . . 238
6.98.2.6 cpu_idle_jiffies . . . 238
6.98.2.7 cpu_total_jiffies . . . 238
6.98.2.8 flags . . . 238
6.98.2.9 load_avg_milli . . . 238
6.98.2.10 monotonic_time_ns . . . 239
6.98.2.11 psi_cpu_full_avg10_milli . . . 239
6.98.2.12 psi_cpu_full_avg300_milli . . . 239
6.98.2.13 psi_cpu_full_avg60_milli . . . 239
6.98.2.14 psi_cpu_some_avg10_milli . . . 239
6.98.2.15 psi_cpu_some_avg300_milli . . . 239
6.98.2.16 psi_cpu_some_avg60_milli . . . 240
6.98.2.17 psi_io_full_avg10_milli . . . 240
6.98.2.18 psi_io_full_avg300_milli . . . 240
6.98.2.19 psi_io_full_avg60_milli . . . 240
6.98.2.20 psi_io_some_avg10_milli . . . 240
6.98.2.21 psi_io_some_avg300_milli . . . 240
6.98.2.22 psi_io_some_avg60_milli . . . 241
6.98.2.23 psi_mem_some_avg10_milli . . . 241
6.98.2.24 psi_mem_some_avg300_milli . . . 241
6.98.2.25 psi_mem_some_avg60_milli . . . 241
6.98.2.26 realtime_ns . . . 241
6.98.2.27 runnable_threads . . . 241
6.98.2.28 scheduler_policy . . . 242

6.98.2.29 scheduler_priority . . . 242
6.98.2.30 scheduler_quantum_ns . . . 242
6.99 physics_runtime_state Struct Reference . . . 242
6.99.1 Field Documentation . . . 243
6.99.1.1 backend_seq . . . 243
6.99.1.2 heap . . . 243
6.99.1.3 heap_ready . . . 243
6.100 physics_seed_definition Struct Reference . . . 243
6.100.1 Field Documentation . . . 243
6.100.1.1 acl_hint . . . 243
6.100.1.2 avg_latency_ns . . . 244
6.100.1.3 name . . . 244
6.100.1.4 pubsub_mask . . . 244
6.100.1.5 temperature_q8 . . . 244
6.101 PhysicsSnapshot Struct Reference . . . 244
6.101.1 Field Documentation . . . 244
6.101.1.1 decay_slope_q48 . . . 244
6.101.1.2 eff_window_size . . . 245
6.101.1.3 heat_25th . . . 245
6.101.1.4 heat_50th . . . 245
6.101.1.5 heat_75th . . . 245
6.101.1.6 prefetch_attempts . . . 245
6.101.1.7 tick_target_ns . . . 245
6.101.1.8 total_executions . . . 245
6.102 PipelineGlobalMetrics Struct Reference . . . 246
6.102.1 Field Documentation . . . 246
6.102.1.1 last_checked_accuracy . . . 246
6.102.1.2 last_checked_window_size . . . 246
6.102.1.3 prefetch_attempts . . . 246
6.102.1.4 prefetch_hits . . . 246
6.102.1.5 suggested_next_size . . . 246
6.102.1.6 window_tuning_checks . . . 247
6.103 pmm_stats_t Struct Reference . . . 247
6.103.1 Field Documentation . . . 247
6.103.1.1 free_bytes . . . 247
6.103.1.2 free_pages . . . 247
6.103.1.3 total_bytes . . . 247
6.103.1.4 total_pages . . . 248
6.103.1.5 used_bytes . . . 248
6.103.1.6 used_pages . . . 248
6.104 Profiler Struct Reference . . . 248
6.104.1 Field Documentation . . . 249

6.104.1.1 branch_count . . . 249
6.104.1.2 category_timers . . . 249
6.104.1.3 compile_operations . . . 249
6.104.1.4 [struct] . . . 249
6.104.1.5 dict_lookups . . . 249
6.104.1.6 instruction_count . . . 249
6.104.1.7 level . . . 250
6.104.1.8 loop_iterations . . . 250
6.104.1.9 memory_allocations . . . 250
6.104.1.10 memory_stats . . . 250
6.104.1.11 profiling_overhead . . . 250
6.104.1.12 stack_operations . . . 250
6.104.1.13 total_execution_time . . . 250
6.104.1.14 vm_cycles . . . 250
6.104.1.15 word_profile_capacity . . . 250
6.104.1.16 word_profile_count . . . 250
6.104.1.17 word_profiles . . . 251
6.105 ProfileScope Struct Reference . . . 251
6.105.1 Field Documentation . . . 251
6.105.1.1 category . . . 251
6.105.1.2 word_name . . . 251
6.106 ProfileTimer Struct Reference . . . 251
6.106.1 Detailed Description . . . 252
6.106.2 Field Documentation . . . 252
6.106.2.1 call_count . . . 252
6.106.2.2 max_time . . . 252
6.106.2.3 min_time . . . 252
6.106.2.4 name . . . 252
6.106.2.5 start_time . . . 253
6.106.2.6 total_time . . . 253
6.107 reloc_block Struct Reference . . . 253
6.107.1 Detailed Description . . . 253
6.107.2 Field Documentation . . . 254
6.107.2.1 block_size . . . 254
6.107.2.2 entry . . . 254
6.107.2.3 page_rva . . . 254
6.108 rolling_window_view_t Struct Reference . . . 254
6.108.1 Detailed Description . . . 254
6.108.2 Field Documentation . . . 255
6.108.2.1 effective_window_size . . . 255
6.108.2.2 history . . . 255
6.108.2.3 is_warm . . . 255

6.108.2.4 total_executions . . . 255
6.108.2.5 window_pos . . . 255
6.109 RollingWindowOfTruth Struct Reference . . . 255
6.109.1 Field Documentation . . . 256
6.109.1.1 adaptive_check_accumulator . . . 256
6.109.1.2 adaptive_pending . . . 256
6.109.1.3 effective_window_size . . . 256
6.109.1.4 execution_history . . . 256
6.109.1.5 is_warm . . . 257
6.109.1.6 last_pattern_diversity . . . 257
6.109.1.7 pattern_diversity_check_count . . . 257
6.109.1.8 snapshot_buffers . . . 257
6.109.1.9 snapshot_effective_window_size . . . 257
6.109.1.10 snapshot_index . . . 257
6.109.1.11 snapshot_is_warm . . . 257
6.109.1.12 snapshot_pending . . . 258
6.109.1.13 snapshot_total_executions . . . 258
6.109.1.14 snapshot_window_pos . . . 258
6.109.1.15 total_executions . . . 258
6.109.1.16 window_pos . . . 258
6.110 rpi5_range_t Struct Reference . . . 258
6.110.1 Field Documentation . . . 259
6.110.1.1 addr . . . 259
6.110.1.2 size . . . 259
6.111 Rpi5FramebufferInfo Struct Reference . . . 259
6.111.1 Detailed Description . . . 259
6.111.2 Field Documentation . . . 259
6.111.2.1 base . . . 259
6.111.2.2 height . . . 260
6.111.2.3 pixel_format . . . 260
6.111.2.4 pixels_per_scanline . . . 260
6.111.2.5 size . . . 260
6.111.2.6 width . . . 260
6.112 sbiret_t Struct Reference . . . 260
6.112.1 Field Documentation . . . 260
6.112.1.1 error . . . 260
6.112.1.2 value . . . 261
6.113 sf_alloc_stats Struct Reference . . . 261
6.113.1 Field Documentation . . . 261
6.113.1.1 alloc_count . . . 261
6.113.1.2 free_count . . . 261
6.113.1.3 peak_bytes . . . 261

6.113.1.4 total_bytes . . . 261
6.113.1.5 used_bytes . . . 262
6.114 sf_mutex Struct Reference . . . 262
6.114.1 Field Documentation . . . 262
6.114.1.1 handle . . . 262
6.115 sf_time_backend_t Struct Reference . . . 262
6.115.1 Field Documentation . . . 262
6.115.1.1 format_timestamp . . . 262
6.115.1.2 get_monotonic_ns . . . 263
6.115.1.3 get_realtime_ns . . . 263
6.115.1.4 has_rtc . . . 263
6.115.1.5 set_realtime_ns . . . 263
6.116 sha512_ctx_t Struct Reference . . . 264
6.116.1 Field Documentation . . . 264
6.116.1.1 bitlen . . . 264
6.116.1.2 buf . . . 264
6.116.1.3 buf_len . . . 264
6.116.1.4 state . . . 265
6.117 sigaction Struct Reference . . . 265
6.117.1 Field Documentation . . . 265
6.117.1.1 sa_flags . . . 265
6.117.1.2 sa_handler . . . 265
6.118 sk_blk_attach_pending_t Struct Reference . . . 266
6.118.1 Field Documentation . . . 266
6.118.1.1 is_artemis . . . 266
6.118.1.2 pending . . . 266
6.118.1.3 sig . . . 266
6.118.1.4 sig_rc . . . 267
6.119 sk_usb_ctrl_t Struct Reference . . . 267
6.119.1 Field Documentation . . . 267
6.119.1.1 count . . . 267
6.119.1.2 pending . . . 268
6.119.1.3 slots . . . 268
6.120 sk_v4_drive_t Struct Reference . . . 268
6.120.1 Field Documentation . . . 269
6.120.1.1 art . . . 269
6.120.1.2 art_state . . . 269
6.120.1.3 art_used . . . 269
6.120.1.4 part . . . 269
6.120.1.5 part_state . . . 269
6.120.1.6 raw . . . 269
6.120.1.7 used . . . 270

6.121 sk_v4_marker_t Struct Reference . . . 270
6.121.1 Field Documentation . . . 270
6.121.1.1 has_real . . . 270
6.121.1.2 present . . . 270
6.121.1.3 proxy . . . 270
6.121.1.4 real . . . 271
6.122 SpeedupEstimate Struct Reference . . . 271
6.122.1 Detailed Description . . . 271
6.122.2 Field Documentation . . . 271
6.122.2.1 credible_lower_95_q48 . . . 271
6.122.2.2 credible_upper_95_q48 . . . 271
6.122.2.3 probability_gt_10pct_q48 . . . 272
6.122.2.4 probability_gt_double_q48 . . . 272
6.122.2.5 speedup_factor_q48 . . . 272
6.123 ssm_config_t Struct Reference . . . 272
6.123.1 Field Documentation . . . 272
6.123.1.1 L1_heat_tracking . . . 272
6.123.1.2 L2_rolling_window . . . 273
6.123.1.3 L3_linear_decay . . . 273
6.123.1.4 L4_pipelining . . . 273
6.123.1.5 L5_window_inference . . . 273
6.123.1.6 L6_decay_inference . . . 273
6.123.1.7 L7_adaptive_heartrate . . . 273
6.124 ssm_l8_metrics_t Struct Reference . . . 274
6.124.1 Field Documentation . . . 274
6.124.1.1 cv . . . 274
6.124.1.2 entropy . . . 274
6.124.1.3 inference_early_exited . . . 274
6.124.1.4 inference_ran_this_tick . . . 274
6.124.1.5 stability_score . . . 274
6.124.1.6 temporal_decay . . . 275
6.125 ssm_l8_state_t Struct Reference . . . 275
6.125.1 Field Documentation . . . 275
6.125.1.1 current_mode . . . 275
6.125.1.2 hysteresis_counter . . . 276
6.125.1.3 pending_mode . . . 276
6.125.1.4 table . . . 276
6.126 TestCase Struct Reference . . . 276
6.126.1 Detailed Description . . . 277
6.126.2 Field Documentation . . . 277
6.126.2.1 contract . . . 277
6.126.2.2 expected . . . 277

6.126.2.3 implemented . . . 277
6.126.2.4 input . . . 278
6.126.2.5 name . . . 278
6.126.2.6 should_error . . . 278
6.126.2.7 type . . . 278
6.127 TestModule Struct Reference . . . 278
6.127.1 Detailed Description . . . 279
6.127.2 Field Documentation . . . 279
6.127.2.1 module_name . . . 279
6.127.2.2 reserved . . . 279
6.127.2.3 reserved_int . . . 279
6.127.2.4 run_module_tests . . . 279
6.128 TestStats Struct Reference . . . 280
6.128.1 Detailed Description . . . 280
6.128.2 Field Documentation . . . 280
6.128.2.1 total_error . . . 280
6.128.2.2 total_fail . . . 280
6.128.2.3 total_pass . . . 280
6.128.2.4 total_skip . . . 281
6.128.2.5 total_tests . . . 281
6.129 time_trust_state Struct Reference . . . 281
6.129.1 Field Documentation . . . 282
6.129.1.1 expected_delta . . . 282
6.129.1.2 last_tsc . . . 282
6.129.1.3 ticks . . . 282
6.129.1.4 total_samples . . . 282
6.129.1.5 trust . . . 282
6.129.1.6 variance . . . 282
6.129.1.7 window . . . 282
6.130 time_window Struct Reference . . . 283
6.130.1 Field Documentation . . . 283
6.130.1.1 count . . . 283
6.130.1.2 deltas . . . 283
6.130.1.3 pos . . . 283
6.131 timer_calibration_record Struct Reference . . . 283
6.131.1 Field Documentation . . . 284
6.131.1.1 converged . . . 284
6.131.1.2 cv_hpet_ppm . . . 284
6.131.1.3 cv_pit_ppm . . . 284
6.131.1.4 diff_ppm . . . 284
6.131.1.5 hpet_hz . . . 284
6.131.1.6 pit_hz_mean . . . 284

6.131.1.7 reserved . . . 284
6.131.1.8 trust . . . 284
6.131.1.9 tsc_hz_mean . . . 285
6.131.1.10 vm_mode . . . 285
6.131.1.11 windows_used . . . 285
6.132 timespec Struct Reference . . . 285
6.132.1 Field Documentation . . . 285
6.132.1.1 tv_nsec . . . 285
6.132.1.2 tv_sec . . . 286
6.133 timeval Struct Reference . . . 286
6.133.1 Field Documentation . . . 286
6.133.1.1 tv_sec . . . 286
6.133.1.2 tv_usec . . . 286
6.134 timezone Struct Reference . . . 286
6.134.1 Field Documentation . . . 287
6.134.1.1 tz_dsttime . . . 287
6.134.1.2 tz_minuteswest . . . 287
6.135 ttf_bitmap_t Struct Reference . . . 287
6.135.1 Detailed Description . . . 287
6.135.2 Field Documentation . . . 287
6.135.2.1 height . . . 287
6.135.2.2 pixels . . . 288
6.135.2.3 width . . . 288
6.136 ttf_flat_point_t Struct Reference . . . 288
6.136.1 Field Documentation . . . 288
6.136.1.1 x . . . 288
6.136.1.2 y . . . 288
6.137 ttf_font_t Struct Reference . . . 288
6.137.1 Detailed Description . . . 289
6.137.2 Field Documentation . . . 289
6.137.2.1 cmap_subtable_off . . . 289
6.137.2.2 data . . . 289
6.137.2.3 glyf_len . . . 289
6.137.2.4 glyf_off . . . 289
6.137.2.5 has_cmap . . . 290
6.137.2.6 head_off . . . 290
6.137.2.7 hmtx_off . . . 290
6.137.2.8 index_to_loc_format . . . 290
6.137.2.9 loca_len . . . 290
6.137.2.10 loca_off . . . 290
6.137.2.11 maxp_off . . . 290
6.137.2.12 num_glyphs . . . 290

6.137.2.13 num_h_metrics . . . 291
6.137.2.14 size . . . 291
6.137.2.15 units_per_em . . . 291
6.138 ttf_glyph_header_t Struct Reference . . . 291
6.138.1 Detailed Description . . . 291
6.138.2 Field Documentation . . . 292
6.138.2.1 glyf_length . . . 292
6.138.2.2 glyf_offset . . . 292
6.138.2.3 num_contours . . . 292
6.138.2.4 x_max . . . 292
6.138.2.5 x_min . . . 292
6.138.2.6 y_max . . . 292
6.138.2.7 y_min . . . 292
6.139 ttf_outline_t Struct Reference . . . 293
6.139.1 Detailed Description . . . 293
6.139.2 Field Documentation . . . 293
6.139.2.1 contour_count . . . 293
6.139.2.2 contour_ends . . . 294
6.139.2.3 max_contours . . . 294
6.139.2.4 max_points . . . 294
6.139.2.5 point_count . . . 294
6.139.2.6 points . . . 294
6.140 ttf_point_t Struct Reference . . . 294
6.140.1 Detailed Description . . . 295
6.140.2 Field Documentation . . . 295
6.140.2.1 on_curve . . . 295
6.140.2.2 x . . . 295
6.140.2.3 y . . . 295
6.141 ttf_raster_cache_slot_t Struct Reference . . . 295
6.141.1 Field Documentation . . . 296
6.141.1.1 codepoint . . . 296
6.141.1.2 font . . . 296
6.141.1.3 height . . . 296
6.141.1.4 hits . . . 296
6.141.1.5 pixels . . . 296
6.141.1.6 size_px . . . 296
6.141.1.7 valid . . . 296
6.141.1.8 width . . . 297
6.142 ttf_raster_cache_t Struct Reference . . . 297
6.142.1 Field Documentation . . . 297
6.142.1.1 evict_next . . . 297
6.142.1.2 slot_count . . . 298

6.142.1.3 slots . . . 298
6.143 usb_bot_cbw_t Struct Reference . . . 298
6.143.1 Field Documentation . . . 298
6.143.1.1 bCBWCBLength . . . 298
6.143.1.2 bCBWLUN . . . 298
6.143.1.3 bmCBWFlags . . . 299
6.143.1.4 CBWCB . . . 299
6.143.1.5 dCBWDataTransferLength . . . 299
6.143.1.6 dCBWSignature . . . 299
6.143.1.7 dCBWTag . . . 299
6.144 usb_bot_csw_t Struct Reference . . . 299
6.144.1 Field Documentation . . . 300
6.144.1.1 bCSWStatus . . . 300
6.144.1.2 dCSWDataResidue . . . 300
6.144.1.3 dCSWSignature . . . 300
6.144.1.4 dCSWTag . . . 300
6.145 usb_setup_packet_t Struct Reference . . . 300
6.145.1 Field Documentation . . . 301
6.145.1.1 bmRequestType . . . 301
6.145.1.2 bRequest . . . 301
6.145.1.3 wIndex . . . 301
6.145.1.4 wLength . . . 301
6.145.1.5 wValue . . . 301
6.146 user_identity_seed_t Struct Reference . . . 301
6.146.1 Field Documentation . . . 302
6.146.1.1 _pad . . . 302
6.146.1.2 crc . . . 302
6.146.1.3 email . . . 302
6.146.1.4 full_name . . . 302
6.146.1.5 magic . . . 302
6.146.1.6 phone . . . 302
6.146.1.7 pubkey . . . 302
6.146.1.8 seed . . . 303
6.146.1.9 username . . . 303
6.146.1.10 version . . . 303
6.147 VirtBlkReq Struct Reference . . . 303
6.147.1 Field Documentation . . . 303
6.147.1.1 reserved . . . 303
6.147.1.2 sector . . . 303
6.147.1.3 type . . . 304
6.148 VirtBlkState Struct Reference . . . 304
6.148.1 Field Documentation . . . 304

6.148.1.1 avail . . . 304
6.148.1.2 avail_idx . . . 305
6.148.1.3 capacity_sectors . . . 305
6.148.1.4 common . . . 305
6.148.1.5 data_buf . . . 305
6.148.1.6 desc . . . 305
6.148.1.7 last_used_idx . . . 305
6.148.1.8 notify . . . 305
6.148.1.9 notify_off_mult . . . 305
6.148.1.10 queue_notify_off . . . 306
6.148.1.11 req_buf . . . 306
6.148.1.12 status_buf . . . 306
6.148.1.13 total_forth_blocks . . . 306
6.148.1.14 used . . . 306
6.149 VirtInputState Struct Reference . . . 306
6.149.1 Field Documentation . . . 307
6.149.1.1 avail . . . 307
6.149.1.2 avail_idx . . . 307
6.149.1.3 bufs . . . 307
6.149.1.4 common . . . 307
6.149.1.5 desc . . . 307
6.149.1.6 isr_status . . . 308
6.149.1.7 last_used_idx . . . 308
6.149.1.8 notify . . . 308
6.149.1.9 notify_off_mult . . . 308
6.149.1.10 queue_notify_off . . . 308
6.149.1.11 used . . . 308
6.150 VirtioCommonCfg Struct Reference . . . 309
6.150.1 Field Documentation . . . 309
6.150.1.1 config_generation . . . 309
6.150.1.2 config_msix_vector . . . 309
6.150.1.3 device_feature . . . 309
6.150.1.4 device_feature_select . . . 309
6.150.1.5 device_status . . . 309
6.150.1.6 driver_feature . . . 310
6.150.1.7 driver_feature_select . . . 310
6.150.1.8 num_queues . . . 310
6.150.1.9 queue_desc . . . 310
6.150.1.10 queue_device . . . 310
6.150.1.11 queue_driver . . . 310
6.150.1.12 queue_enable . . . 310
6.150.1.13 queue_msix_vector . . . 310

6.150.1.14 queue_notify_data . . . 311
6.150.1.15 queue_notify_off . . . 311
6.150.1.16 queue_reset . . . 311
6.150.1.17 queue_select . . . 311
6.150.1.18 queue_size . . . 311
6.151 VirtioInputEvent Struct Reference . . . 311
6.151.1 Field Documentation . . . 312
6.151.1.1 code . . . 312
6.151.1.2 type . . . 312
6.151.1.3 value . . . 312
6.152 VirtqAvail Struct Reference . . . 312
6.152.1 Field Documentation . . . 312
6.152.1.1 flags . . . 312
6.152.1.2 idx . . . 312
6.152.1.3 ring . . . 313
6.152.1.4 used_event . . . 313
6.153 VirtqDesc Struct Reference . . . 313
6.153.1 Field Documentation . . . 313
6.153.1.1 addr . . . 313
6.153.1.2 flags . . . 313
6.153.1.3 len . . . 313
6.153.1.4 next . . . 314
6.154 VirtqUsed Struct Reference . . . 314
6.154.1 Field Documentation . . . 314
6.154.1.1 avail_event . . . 314
6.154.1.2 flags . . . 314
6.154.1.3 idx . . . 315
6.154.1.4 ring . . . 315
6.155 VirtqUsedElem Struct Reference . . . 315
6.155.1 Field Documentation . . . 315
6.155.1.1 id . . . 315
6.155.1.2 len . . . 315
6.156 VirtRngState Struct Reference . . . 316
6.156.1 Field Documentation . . . 316
6.156.1.1 avail . . . 316
6.156.1.2 avail_idx . . . 316
6.156.1.3 common . . . 317
6.156.1.4 data_buf . . . 317
6.156.1.5 desc . . . 317
6.156.1.6 last_used_idx . . . 317
6.156.1.7 notify . . . 317
6.156.1.8 notify_off_mult . . . 317

6.156.1.9 queue_notify_off . . . 317
6.156.1.10 used . . . 318
6.157 VM Struct Reference . . . 318
6.157.1 Detailed Description . . . 321
6.157.2 Field Documentation . . . 321
6.157.2.1 abort_requested . . . 321
6.157.2.2 acl_skip . . . 321
6.157.2.3 base . . . 321
6.157.2.4 base_addr . . . 322
6.157.2.5 blk_vm_cbuf . . . 322
6.157.2.6 blk_vm_dirty . . . 322
6.157.2.7 blk_vm_epoch . . . 322
6.157.2.8 blk_vm_lbn . . . 322
6.157.2.9 blk_vm_next . . . 322
6.157.2.10 cf . . . 322
6.157.2.11 compile_buffer . . . 323
6.157.2.12 compile_pos . . . 323
6.157.2.13 compile_size . . . 323
6.157.2.14 compiling_word . . . 323
6.157.2.15 current_executing_entry . . . 323
6.157.2.16 current_word_name . . . 323
6.157.2.17 data_stack . . . 323
6.157.2.18 decay_slope_direction . . . 324
6.157.2.19 decay_slope_q48 . . . 324
6.157.2.20 dict_fence_here . . . 324
6.157.2.21 dict_fence_latest . . . 324
6.157.2.22 dict_lock . . . 324
6.157.2.23 doe_row_printing . . . 324
6.157.2.24 dsp . . . 325
6.157.2.25 ecw_nesting . . . 326
6.157.2.26 emergency_console . . . 326
6.157.2.27 error . . . 326
6.157.2.28 exit_colon . . . 327
6.157.2.29 halted . . . 327
6.157.2.30 heartbeat . . . 328
6.157.2.31 heartbeat_decay_cursor_id . . . 328
6.157.2.32 heat_threshold_25th . . . 328
6.157.2.33 heat_threshold_50th . . . 328
6.157.2.34 heat_threshold_75th . . . 328
6.157.2.35 here . . . 329
6.157.2.36 hold_addr . . . 329
6.157.2.37 hold_pos . . . 329

6.157.2.38 host . . . 329
6.157.2.39 hot_word_count_at_check . . . 329
6.157.2.40 hotwords_cache . . . 330
6.157.2.41 in_var . . . 330
6.157.2.42 input_buffer . . . 330
6.157.2.43 input_length . . . 330
6.157.2.44 input_pos . . . 330
6.157.2.45 interpreter_enabled . . . 330
6.157.2.46 ip . . . 331
6.157.2.47 last_bucket_reorg_ns . . . 331
6.157.2.48 last_decay_check_ns . . . 331
6.157.2.49 last_inference_outputs . . . 331
6.157.2.50 latest . . . 331
6.157.2.51 lookup_strategy . . . 332
6.157.2.52 memory . . . 332
6.157.2.53 mode . . . 332
6.157.2.54 next_word_id . . . 332
6.157.2.55 pipeline_metrics . . . 333
6.157.2.56 recycled_word_id_count . . . 333
6.157.2.57 recycled_word_ids . . . 333
6.157.2.58 return_stack . . . 333
6.157.2.59 rolling_window . . . 333
6.157.2.60 rsp . . . 334
6.157.2.61 scr_addr . . . 334
6.157.2.62 span_var . . . 334
6.157.2.63 ssm_config . . . 334
6.157.2.64 ssm_l8_state . . . 334
6.157.2.65 stale_word_count_at_check . . . 335
6.157.2.66 state_addr . . . 335
6.157.2.67 state_var . . . 335
6.157.2.68 tib_buf . . . 335
6.157.2.69 tib_cap . . . 335
6.157.2.70 total_heat_at_last_check . . . 335
6.157.2.71 tuning_lock . . . 336
6.157.2.72 vocab . . . 336
6.157.2.73 word_count_at_check . . . 336
6.157.2.74 word_id_map . . . 336
6.157.2.75 zuse_cert_installed . . . 336
6.157.2.76 zuse_cert_pubkey . . . 336
6.157.2.77 zuse_cert_seed . . . 337
6.157.2.78 zuse_root_pubkey . . . 337
6.157.2.79 zuse_root_pubkey_known . . . 337

6.157.2.80 zuse_session . . . 337
6.158 vm_node Struct Reference . . . 338
6.158.1 Field Documentation . . . 338
6.158.1.1 entry . . . 338
6.158.1.2 next . . . 338
6.159 vm_physics_node Struct Reference . . . 339
6.159.1 Field Documentation . . . 339
6.159.1.1 next . . . 339
6.159.1.2 physics . . . 339
6.159.1.3 vm_id . . . 339
6.160 VMFleetTouchSample Struct Reference . . . 340
6.160.1 Field Documentation . . . 340
6.160.1.1 amount . . . 340
6.160.1.2 clamped . . . 340
6.160.1.3 elapsed_ticks . . . 340
6.161 VMFleetWindow Struct Reference . . . 340
6.161.1 Field Documentation . . . 341
6.161.1.1 count . . . 341
6.161.1.2 head . . . 341
6.161.1.3 is_warm . . . 341
6.161.1.4 touch_samples . . . 341
6.162 VMHostServices Struct Reference . . . 341
6.162.1 Detailed Description . . . 342
6.162.2 Field Documentation . . . 342
6.162.2.1 alloc . . . 342
6.162.2.2 free . . . 342
6.162.2.3 is_executable_ptr . . . 342
6.162.2.4 monotonic_ns . . . 342
6.162.2.5 mutex_destroy . . . 343
6.162.2.6 mutex_init . . . 343
6.162.2.7 mutex_lock . . . 343
6.162.2.8 mutex_unlock . . . 343
6.162.2.9 owns_xt_entry . . . 343
6.162.2.10 panic . . . 343
6.162.2.11 parity_mode . . . 343
6.162.2.12 putc . . . 343
6.162.2.13 puts . . . 343
6.162.2.14 verbose . . . 344
6.163 vmm_page_info_t Struct Reference . . . 344
6.163.1 Field Documentation . . . 344
6.163.1.1 executable . . . 344
6.163.1.2 present . . . 344

6.163.1.3 writable . . . 344
6.164 VMPhysics Struct Reference . . . 345
6.164.1 Field Documentation . . . 345
6.164.1.1 execution_heat_q48 . . . 345
6.164.1.2 is_live . . . 345
6.164.1.3 last_active_tick . . . 345
6.164.1.4 touched . . . 345
6.165 VMRegistryEntry Struct Reference . . . 345
6.165.1 Field Documentation . . . 346
6.165.1.1 birth_capsule_id . . . 346
6.165.1.2 birth_dict_hash . . . 346
6.165.1.3 birth_timestamp_ns . . . 346
6.165.1.4 flags . . . 346
6.165.1.5 name . . . 346
6.165.1.6 parent_vm_id . . . 347
6.165.1.7 pending_reap . . . 347
6.165.1.8 stadium_patron_cell . . . 347
6.165.1.9 state . . . 347
6.165.1.10 vm_id . . . 347
6.165.1.11 vm_ptr . . . 347
6.166 VocabularyState Struct Reference . . . 348
6.166.1 Field Documentation . . . 348
6.166.1.1 context_var_addr . . . 348
6.166.1.2 context_vocab . . . 348
6.166.1.3 ctx_cached_head . . . 349
6.166.1.4 ctx_fc . . . 349
6.166.1.5 ctx_n . . . 349
6.166.1.6 current_var_addr . . . 349
6.166.1.7 current_vocab . . . 349
6.166.1.8 forth_cached_head . . . 349
6.166.1.9 forth_fc . . . 349
6.166.1.10 forth_n . . . 349
6.166.1.11 forth_vocab . . . 350
6.166.1.12 initialized . . . 350
6.167 VT100State Struct Reference . . . 350
6.167.1 Field Documentation . . . 350
6.167.1.1 bg . . . 350
6.167.1.2 bold . . . 351
6.167.1.3 cols . . . 351
6.167.1.4 cx . . . 351
6.167.1.5 cy . . . 351
6.167.1.6 def_bg . . . 351

6.167.1.7 def_fg . . . 351
6.167.1.8 fg . . . 351
6.167.1.9 initialized . . . 352
6.167.1.10 param_count . . . 352
6.167.1.11 param_has_value . . . 352
6.167.1.12 params . . . 352
6.167.1.13 reverse . . . 352
6.167.1.14 rows . . . 352
6.167.1.15 saved_cx . . . 352
6.167.1.16 saved_cy . . . 353
6.167.1.17 state . . . 353
6.167.1.18 underline . . . 353
6.168 wirebind_live_entry_t Struct Reference . . . 353
6.168.1 Field Documentation . . . 354
6.168.1.1 dev . . . 354
6.168.1.2 in_use . . . 354
6.168.1.3 switch_slot . . . 354
6.168.1.4 user_vm_id . . . 354
6.168.1.5 username . . . 354
6.169 WordContract Struct Reference . . . 354
6.169.1 Field Documentation . . . 355
6.169.1.1 flags . . . 355
6.169.1.2 fuzz_rounds . . . 355
6.170 WordProfile Struct Reference . . . 355
6.170.1 Detailed Description . . . 356
6.170.2 Field Documentation . . . 356
6.170.2.1 avg_time . . . 356
6.170.2.2 call_count . . . 356
6.170.2.3 max_time . . . 356
6.170.2.4 min_time . . . 356
6.170.2.5 percentage . . . 356
6.170.2.6 total_time . . . 356
6.170.2.7 word_name . . . 357
6.171 WordStats Struct Reference . . . 357
6.171.1 Detailed Description . . . 358
6.171.2 Field Documentation . . . 358
6.171.2.1 call_count . . . 358
6.171.2.2 entry . . . 358
6.171.2.3 max_time_ns . . . 358
6.171.2.4 min_time_ns . . . 358
6.171.2.5 total_time_ns . . . 358
6.172 WordTestSuite Struct Reference . . . 359

6.172.1 Detailed Description . . . 359
6.172.2 Field Documentation . . . 359
6.172.2.1 suite_contract . . . 359
6.172.2.2 test_count . . . 360
6.172.2.3 tests . . . 360
6.172.2.4 word_name . . . 360
6.173 WordTransitionMetrics Struct Reference . . . 360
6.173.1 Detailed Description . . . 361
6.173.2 Field Documentation . . . 362
6.173.2.1 actual_window_size . . . 362
6.173.2.2 context_transitions . . . 362
6.173.2.3 context_window . . . 362
6.173.2.4 context_window_pos . . . 363
6.173.2.5 max_transition_probability_q48 . . . 363
6.173.2.6 misprediction_cost_q48 . . . 363
6.173.2.7 most_likely_next_word_id . . . 363
6.173.2.8 prefetch_attempts . . . 363
6.173.2.9 prefetch_hits . . . 364
6.173.2.10 prefetch_latency_saved_q48 . . . 364
6.173.2.11 prefetch_misses . . . 364
6.173.2.12 total_context_transitions . . . 364
6.173.2.13 total_transitions . . . 364
6.173.2.14 transition_heat . . . 365
6.174 xhci_cap_regs_t Struct Reference . . . 365
6.174.1 Field Documentation . . . 365
6.174.1.1 cap_length . . . 365
6.174.1.2 db_off . . . 365
6.174.1.3 hcc_params1 . . . 366
6.174.1.4 hcc_params2 . . . 366
6.174.1.5 hci_version . . . 366
6.174.1.6 hcs_params1 . . . 366
6.174.1.7 hcs_params2 . . . 366
6.174.1.8 hcs_params3 . . . 366
6.174.1.9 reserved0 . . . 366
6.174.1.10 rts_off . . . 366
6.175 xhci_dev_t Struct Reference . . . 367
6.175.1 Member Enumeration Documentation . . . 369
6.175.1.1 anonymous enum . . . 369
6.175.1.2 anonymous enum . . . 369
6.175.1.3 anonymous enum . . . 369
6.175.1.4 anonymous enum . . . 369
6.175.2 Field Documentation . . . 370

6.175.2.1 bar0_phys . . . 370
6.175.2.2 bot_cap_block_size . . . 370
6.175.2.3 bot_cap_last_lba . . . 370
6.175.2.4 bot_cbw . . . 370
6.175.2.5 [] . . . 370
6.175.2.6 bot_csw . . . 370
6.175.2.7 bot_data_buf . . . 371
6.175.2.8 bot_expected_data_len . . . 371
6.175.2.9 [] . . . 371
6.175.2.10 bot_last_tag . . . 371
6.175.2.11 bot_next_tag . . . 371
6.175.2.12 bot_read10_block_size . . . 371
6.175.2.13 bot_read10_lba . . . 371
6.175.2.14 bot_read10_num_blocks . . . 372
6.175.2.15 bot_reset_clear_remaining . . . 372
6.175.2.16 bot_stall_recoveries . . . 372
6.175.2.17 [] . . . 372
6.175.2.18 bot_tur_retries . . . 372
6.175.2.19 bot_write10_block_size . . . 372
6.175.2.20 bot_write10_lba . . . 372
6.175.2.21 bot_write10_num_blocks . . . 373
6.175.2.22 cap . . . 373
6.175.2.23 cmd_ring . . . 373
6.175.2.24 cmd_ring_cycle . . . 373
6.175.2.25 cmd_ring_enq . . . 373
6.175.2.26 [] . . . 373
6.175.2.27 dcbaa . . . 373
6.175.2.28 doorbell . . . 374
6.175.2.29 evt_ring . . . 374
6.175.2.30 evt_ring_cycle . . . 374
6.175.2.31 evt_ring_deq . . . 374
6.175.2.32 evt_ring_seg_table . . . 374
6.175.2.33 intr0 . . . 374
6.175.2.34 is_connect . . . 374
6.175.2.35 max_intrs . . . 375
6.175.2.36 max_ports . . . 375
6.175.2.37 max_scratchpad_bufs . . . 375
6.175.2.38 max_slots . . . 375
6.175.2.39 msc_slots . . . 375
6.175.2.40 op . . . 375
6.175.2.41 pci . . . 375
6.175.2.42 pending_connect_port_id . . . 376

6.175.2.43 pending_connect_slot_id . . . 376
6.175.2.44 pending_connect_speed . . . 376
6.175.2.45 pending_disable_slot_id . . . 376
6.175.2.46 [struct] . . . 376
6.175.2.47 port_id . . . 376
6.175.2.48 port_slot_id . . . 376
6.175.2.49 portsc . . . 376
6.175.2.50 runtime . . . 377
6.175.2.51 scratchpad_arr . . . 377
6.175.2.52 slot_id . . . 377
6.175.2.53 stall_dci . . . 377
6.175.2.54 stall_ep_addr . . . 377
6.175.2.55 stall_retry_action . . . 377
6.175.2.56 valid . . . 377
6.176 xhci_ep_ctx32_t Struct Reference . . . 378
6.176.1 Field Documentation . . . 378
6.176.1.1 dword0 . . . 378
6.176.1.2 dword1 . . . 378
6.176.1.3 dword4 . . . 378
6.176.1.4 reserved . . . 378
6.176.1.5 tr_dequeue_ptr . . . 378
6.177 xhci_input_ctrl_ctx32_t Struct Reference . . . 379
6.177.1 Field Documentation . . . 379
6.177.1.1 add_flags . . . 379
6.177.1.2 config_word . . . 379
6.177.1.3 drop_flags . . . 379
6.177.1.4 reserved . . . 379
6.178 xhci_intr_regs_t Struct Reference . . . 379
6.178.1 Field Documentation . . . 380
6.178.1.1 erdp . . . 380
6.178.1.2 erstba . . . 380
6.178.1.3 erstsz . . . 380
6.178.1.4 iman . . . 380
6.178.1.5 imod . . . 380
6.178.1.6 reserved0 . . . 380
6.179 xhci_msc_slot_t Struct Reference . . . 381
6.179.1 Field Documentation . . . 382
6.179.1.1 bot_msc_attach_pending . . . 382
6.179.1.2 bot_msc_attached . . . 382
6.179.1.3 bot_msc_detach_pending . . . 382
6.179.1.4 bulk_in_ep_addr . . . 382
6.179.1.5 bulk_in_max_packet . . . 382

6.179.1.6 bulk_in_ring . . . 383
6.179.1.7 bulk_in_ring_cycle . . . 383
6.179.1.8 bulk_in_ring_enq . . . 383
6.179.1.9 bulk_out_ep_addr . . . 383
6.179.1.10 bulk_out_max_packet . . . 383
6.179.1.11 bulk_out_ring . . . 383
6.179.1.12 bulk_out_ring_cycle . . . 383
6.179.1.13 bulk_out_ring_enq . . . 384
6.179.1.14 config_descriptor . . . 384
6.179.1.15 config_total_length . . . 384
6.179.1.16 device_ctx . . . 384
6.179.1.17 device_descriptor . . . 384
6.179.1.18 ep0_ring . . . 384
6.179.1.19 ep0_ring_cycle . . . 384
6.179.1.20 ep0_ring_enq . . . 385
6.179.1.21 hid_buf . . . 385
6.179.1.22 hid_ep_addr . . . 385
6.179.1.23 hid_iface . . . 385
6.179.1.24 hid_interval . . . 385
6.179.1.25 hid_kbd . . . 385
6.179.1.26 hid_leds_sent . . . 385
6.179.1.27 hid_max_packet . . . 386
6.179.1.28 hid_prev . . . 386
6.179.1.29 hid_ring . . . 386
6.179.1.30 hid_ring_cycle . . . 386
6.179.1.31 hid_ring_enq . . . 386
6.179.1.32 hid_running . . . 386
6.179.1.33 in_use . . . 386
6.179.1.34 input_ctx . . . 386
6.179.1.35 next_action . . . 387
6.179.1.36 next_action_config_value . . . 387
6.179.1.37 next_action_length . . . 387
6.179.1.38 transfer_purpose . . . 387
6.180 xhci_op_regs_t Struct Reference . . . 387
6.180.1 Field Documentation . . . 388
6.180.1.1 config . . . 388
6.180.1.2 crcr . . . 388
6.180.1.3 dcbaap . . . 388
6.180.1.4 dn_ctrl . . . 388
6.180.1.5 page_size . . . 388
6.180.1.6 reserved0 . . . 388
6.180.1.7 reserved1 . . . 388

6.180.1.8 usb_cmd . . . 388
6.180.1.9 usb_sts . . . 389
6.181 xhci_port_regs_t Struct Reference . . . 389
6.181.1 Field Documentation . . . 389
6.181.1.1 porthlpmc . . . 389
6.181.1.2 portli . . . 389
6.181.1.3 portpmsc . . . 389
6.181.1.4 portsc . . . 389
6.182 xhci_runtime_regs_t Struct Reference . . . 390
6.182.1 Field Documentation . . . 390
6.182.1.1 mf_index . . . 390
6.182.1.2 reserved0 . . . 390
6.183 xhci_slot_ctx32_t Struct Reference . . . 390
6.183.1 Field Documentation . . . 390
6.183.1.1 dword0 . . . 390
6.183.1.2 dword1 . . . 391
6.183.1.3 dword2 . . . 391
6.183.1.4 dword3 . . . 391
6.183.1.5 reserved . . . 391
6.184 xhci_trb_t Struct Reference . . . 391
6.184.1 Field Documentation . . . 391
6.184.1.1 control . . . 391
6.184.1.2 parameter . . . 392
6.184.1.3 status . . . 392
6.185 XXHash64State Struct Reference . . . 392
6.185.1 Detailed Description . . . 392
6.185.2 Field Documentation . . . 392
6.185.2.1 buffer . . . 392
6.185.2.2 buffer_size . . . 393
6.185.2.3 seed . . . 393
6.185.2.4 total_len . . . 393
6.185.2.5 v1 . . . 393
6.185.2.6 v2 . . . 393
6.185.2.7 v3 . . . 393
6.185.2.8 v4 . . . 393
6.186 zuse_cert_devblock_t Struct Reference . . . 394
6.186.1 Field Documentation . . . 394
6.186.1.1 _pad . . . 394
6.186.1.2 crc . . . 394
6.186.1.3 magic . . . 394
6.186.1.4 pubkey . . . 394
6.186.1.5 seed . . . 394

6.186.1.6 version . . . 394
6.187 zuse_genesis_marker_t Struct Reference . . . 395
6.187.1 Field Documentation . . . 395
6.187.1.1 _pad . . . 395
6.187.1.2 crc . . . 395
6.187.1.3 magic . . . 395
6.187.1.4 version . . . 395
6.187.1.5 zuse_pubkey . . . 395
6.188 zuse_genesis_marker_v2_t Struct Reference . . . 396
6.188.1 Field Documentation . . . 396
6.188.1.1 _pad . . . 396
6.188.1.2 _reserved . . . 396
6.188.1.3 crc . . . 396
6.188.1.4 has_real . . . 396
6.188.1.5 magic . . . 396
6.188.1.6 proxy_pubkey . . . 396
6.188.1.7 real_pubkey . . . 396
6.188.1.8 version . . . 396

7 File Documentation **397**

7.1 kernel/include/starkernel/apic.h File Reference . . . 397
7.1.1 Macro Definition Documentation . . . 398
7.1.1.1 APIC_SPURIOUS_VECTOR . . . 398
7.1.1.2 APIC_TIMER_VECTOR . . . 398
7.1.2 Function Documentation . . . 399
7.1.2.1 apic_eoi() . . . 399
7.1.2.2 apic_id() . . . 400
7.1.2.3 apic_init() . . . 401
7.1.2.4 apic_timer_init() . . . 403
7.1.2.5 apic_timer_period_tsc() . . . 404
7.1.2.6 apic_timer_rearm() . . . 405
7.1.2.7 apic_timer_start() . . . 406
7.1.2.8 apic_timer_stop() . . . 408
7.2 kernel/include/starkernel/arch.h File Reference . . . 409
7.2.1 Function Documentation . . . 410
7.2.1.1 arch_cold_reset() . . . 410
7.2.1.2 arch_disable_interrupts() . . . 410
7.2.1.3 arch_early_init() . . . 411
7.2.1.4 arch_enable_interrupts() . . . 412
7.2.1.5 arch_halt() . . . 413
7.2.1.6 arch_interrupts_init() . . . 414
7.2.1.7 arch_mmu_init() . . . 415

7.2.1.8 arch_read_timestamp() . . . 416
7.2.1.9 arch_relax() . . . 417
7.3 kernel/include/starkernel/artemis_sig.h File Reference . . . 417
7.3.1 Macro Definition Documentation . . . 419
7.3.1.1 ARTEMIS_SIG_DEVBLOCK_FROM_TOP . . . 419
7.3.1.2 ARTEMIS_SIG_GET_MAGIC . . . 419
7.3.1.3 ARTEMIS_SIG_GET_VERSION . . . 419
7.3.1.4 ARTEMIS_SIG_MAGIC . . . 419
7.3.1.5 ARTEMIS_SIG_PACK . . . 420
7.3.1.6 ARTEMIS_SIG_VERSION_0 . . . 420
7.3.2 Typedef Documentation . . . 420
7.3.2.1 artemis_sig_size_check . . . 420
7.3.3 Enumeration Type Documentation . . . 420
7.3.3.1 artemis_sig_result_t . . . 420
7.3.4 Function Documentation . . . 420
7.3.4.1 artemis_sig_check() . . . 420
7.3.4.2 artemis_sig_compute_crc() . . . 421
7.3.4.3 artemis_sig_genesis_stamp() . . . 422
7.4 kernel/include/starkernel/blkio_usb.h File Reference . . . 423
7.4.1 Function Documentation . . . 423
7.4.1.1 blkio_usb_open_msc() . . . 423
7.5 kernel/include/starkernel/boot_info_offsets.h File Reference . . . 425
7.5.1 Macro Definition Documentation . . . 425
7.5.1.1 BOOT_INFO_KERNEL_STACK_BASE_OFFSET . . . 425
7.5.1.2 BOOT_INFO_KERNEL_STACK_SIZE_OFFSET . . . 425
7.6 kernel/include/starkernel/capsule.h File Reference . . . 425
7.6.1 Macro Definition Documentation . . . 428
7.6.1.1 CAPSULE_BIRTH_ELIGIBLE . . . 428
7.6.1.2 CAPSULE_DESC_MAGIC . . . 428
7.6.1.3 CAPSULE_DIR_MAGIC . . . 428
7.6.1.4 CAPSULE_DOE_ELIGIBLE . . . 428
7.6.1.5 CAPSULE_FLAG_ACTIVE . . . 428
7.6.1.6 CAPSULE_FLAG_CONTRIB . . . 429
7.6.1.7 CAPSULE_FLAG_DEPRECATED . . . 429
7.6.1.8 CAPSULE_FLAG_EXPERIMENT . . . 429
7.6.1.9 CAPSULE_FLAG_MAMA_INIT . . . 429
7.6.1.10 CAPSULE_FLAG_PINNED . . . 429
7.6.1.11 CAPSULE_FLAG_PRODUCTION . . . 429
7.6.1.12 CAPSULE_FLAG_REVOKED . . . 429
7.6.1.13 CAPSULE_IS_MAMA_INIT . . . 429
7.6.1.14 CAPSULE_MAGIC_GET_HASHALG . . . 430
7.6.1.15 CAPSULE_MAGIC_GET_SIG . . . 430

7.6.1.16 CAPSULE_MAGIC_GET_VERSION . . . 430
7.6.1.17 CAPSULE_MAGIC_PACK . . . 430
7.6.1.18 CAPSULE_MAX_COUNT . . . 430
7.6.1.19 CAPSULE_MODE_VALID . . . 430
7.6.1.20 CAPSULE_NAME_MAX . . . 430
7.6.1.21 CAPSULE_VERSION_0 . . . 430
7.6.2 Enumeration Type Documentation . . . 430
7.6.2.1 CapsuleHashAlg . . . 430
7.6.2.2 CapsuleValidateResult . . . 431
7.6.3 Function Documentation . . . 431
7.6.3.1 __attribute__() . . . 431
7.6.3.2 capsule_find_by_id() . . . 431
7.6.3.3 capsule_find_by_name() . . . 432
7.6.3.4 capsule_find_mama_init() . . . 433
7.6.3.5 capsule_get_payload() . . . 433
7.6.3.6 capsule_validate() . . . 434
7.6.3.7 capsule_validate_result_str() . . . 435
7.6.4 Variable Documentation . . . 435
7.6.4.1 CapsuleDesc . . . 435
7.7 kernel/include/starkernel/capsule_birth.h File Reference . . . 435
7.7.1 Typedef Documentation . . . 437
7.7.1.1 CapsuleDictHashFn . . . 437
7.7.1.2 CapsuleExecFn . . . 437
7.7.1.3 CapsuleVMAllocFn . . . 438
7.7.2 Function Documentation . . . 438
7.7.2.1 capsule_birth_baby() . . . 438
7.7.2.2 capsule_birth_mama() . . . 440
7.7.2.3 capsule_birth_set_hooks() . . . 441
7.7.2.4 capsule_run_experiment() . . . 442
7.7.2.5 capsule_vm_find_by_name() . . . 443
7.7.2.6 capsule_vm_find_by_name_nocase() . . . 444
7.7.2.7 capsule_vm_force_reap() . . . 445
7.7.2.8 capsule_vm_hooks_register() . . . 446
7.7.2.9 capsule_vm_kill() . . . 447
7.7.2.10 capsule_vm_kill_all_nonmama() . . . 448
7.7.2.11 capsule_vm_registry_count() . . . 449
7.7.2.12 capsule_vm_registry_get() . . . 449
7.7.2.13 capsule_vm_registry_get_by_index() . . . 450
7.7.2.14 capsule_vm_registry_init() . . . 451
7.7.2.15 capsule_vm_registry_set_name() . . . 451
7.7.2.16 capsule_vm_set_pending_reap() . . . 452
7.7.2.17 capsule_vm_set_state() . . . 453

7.8 kernel/include/starkernel/capsule_blocks.h File Reference . . . 453
7.8.1 Function Documentation . . . 454
7.8.1.1 capsule_block_header() . . . 454
7.9 kernel/include/starkernel/capsule_console.h File Reference . . . 455
7.10 kernel/include/starkernel/capsule_generated.h File Reference . . . 455
7.10.1 Function Documentation . . . 456
7.10.1.1 __attribute__() . . . 456
7.11 kernel/include/starkernel/capsule_loader.h File Reference . . . 456
7.11.1 Function Documentation . . . 458
7.11.1.1 capsule_clear_blocks() . . . 458
7.11.1.2 capsule_exec_init() . . . 459
7.11.1.3 capsule_exec_payload() . . . 460
7.11.1.4 capsule_load_blocks() . . . 461
7.12 kernel/include/starkernel/capsule_mint.h File Reference . . . 462
7.13 kernel/include/starkernel/capsule_run.h File Reference . . . 463
7.13.1 Macro Definition Documentation . . . 464
7.13.1.1 CAPSULE_MAX_RUN_RECORDS . . . 464
7.13.1.2 VM_NAME_MAX . . . 464
7.13.2 Enumeration Type Documentation . . . 464
7.13.2.1 CapsuleRunResult . . . 464
7.13.2.2 VMState . . . 465
7.13.3 Function Documentation . . . 465
7.13.3.1 capsule_parity_log_birth() . . . 465
7.13.3.2 capsule_parity_log_birth_failed() . . . 466
7.13.3.3 capsule_parity_log_kill() . . . 466
7.13.3.4 capsule_parity_log_mama_init() . . . 467
7.13.3.5 capsule_parity_log_run() . . . 468
7.13.3.6 capsule_parity_set_output() . . . 468
7.13.3.7 capsule_run_log_count() . . . 469
7.13.3.8 capsule_run_log_get() . . . 469
7.13.3.9 capsule_run_log_init() . . . 469
7.13.3.10 capsule_run_log_record() . . . 470
7.14 kernel/include/starkernel/capsule_runcap.h File Reference . . . 471
7.15 kernel/include/starkernel/capsule_sig.h File Reference . . . 471
7.15.1 Macro Definition Documentation . . . 472
7.15.1.1 CAPSULE_SIGNING_CERT_NAME . . . 472
7.15.2 Enumeration Type Documentation . . . 472
7.15.2.1 CapsuleSigResult . . . 472
7.15.3 Function Documentation . . . 473
7.15.3.1 capsule_sig_result_str() . . . 473
7.15.3.2 capsule_verify_signature() . . . 473
7.16 kernel/include/starkernel/capsule_vm_physics.h File Reference . . . 474

7.16.1 Function Documentation . . . 476
7.16.1.1 vm_physics_conserved() . . . 476
7.16.1.2 vm_physics_fleet_heat_sum() . . . 477
7.16.1.3 vm_physics_heartbeat_tick() . . . 477
7.16.1.4 vm_physics_heat_of() . . . 478
7.16.1.5 vm_physics_init() . . . 479
7.16.1.6 vm_physics_retire() . . . 480
7.16.1.7 vm_physics_status() . . . 481
7.16.1.8 vm_physics_tick() . . . 481
7.16.1.9 vm_physics_touch() . . . 482
7.17 kernel/include/starkernel/capsule_vm_switch_signal.h File Reference . . . 483
7.18 kernel/include/starkernel/capsule_wirebind.h File Reference . . . 483
7.19 kernel/include/starkernel/capsule_zuse_boot.h File Reference . . . 484
7.20 kernel/include/starkernel/cmdline.h File Reference . . . 484
7.20.1 Function Documentation . . . 485
7.20.1.1 cmdline_parse_ascii() . . . 485
7.20.1.2 cmdline_parse_size() . . . 486
7.21 kernel/include/starkernel/console.h File Reference . . . 487
7.21.1 Typedef Documentation . . . 490
7.21.1.1 console_user_prefix_fn . . . 490
7.21.2 Function Documentation . . . 490
7.21.2.1 console_attach_serial() . . . 490
7.21.2.2 console_cancel_deferred_line_start() . . . 491
7.21.2.3 console_describe_serial() . . . 492
7.21.2.4 console_ensure_line_start() . . . 492
7.21.2.5 console_fb_clear() . . . 493
7.21.2.6 console_fb_draw_cursor() . . . 494
7.21.2.7 console_fb_enable_ttf() . . . 494
7.21.2.8 console_fb_erase_cursor() . . . 495
7.21.2.9 console_fb_init() . . . 496
7.21.2.10 console_fb_mirror() . . . 497
7.21.2.11 console_fb_scroll_back() . . . 497
7.21.2.12 console_fb_scroll_fwd() . . . 498
7.21.2.13 console_fb_toggle_graphics() . . . 498
7.21.2.14 console_get_vm_name() . . . 499
7.21.2.15 console_getc() . . . 499
7.21.2.16 console_init() . . . 500
7.21.2.17 console_poll() . . . 501
7.21.2.18 console_println() . . . 501
7.21.2.19 console_putc() . . . 502
7.21.2.20 console_puts() . . . 503
7.21.2.21 console_save_vm_name() . . . 505

7.21.2.22 console_set_user_prefix_provider() . . . 506
7.21.2.23 console_set_vm_name() . . . 506
7.21.2.24 console_tx_count() . . . 507
7.21.3 Variable Documentation . . . 507
7.21.3.1 g_sk_fault_word . . . 507
7.22 kernel/include/starkernel/doe_log.h File Reference . . . 507
7.22.1 Function Documentation . . . 508
7.22.1.1 doe_log_tick_row() . . . 508
7.22.1.2 register_doe_log_words() . . . 509
7.22.2 Variable Documentation . . . 510
7.22.2.1 g_doe_log_enabled . . . 510
7.23 kernel/include/starkernel/ed25519.h File Reference . . . 512
7.23.1 Function Documentation . . . 513
7.23.1.1 ed25519_keygen() . . . 513
7.23.1.2 ed25519_sign() . . . 514
7.23.1.3 ed25519_verify() . . . 515
7.24 kernel/include/starkernel/elf64.h File Reference . . . 516
7.24.1 Macro Definition Documentation . . . 518
7.24.1.1 EI_ABIVERSION . . . 518
7.24.1.2 EI_CLASS . . . 518
7.24.1.3 EI_DATA . . . 518
7.24.1.4 EI_MAG0 . . . 519
7.24.1.5 EI_MAG1 . . . 519
7.24.1.6 EI_MAG2 . . . 519
7.24.1.7 EI_MAG3 . . . 519
7.24.1.8 EI_NIDENT . . . 519
7.24.1.9 EI_OSABI . . . 519
7.24.1.10 EI_PAD . . . 519
7.24.1.11 EI_VERSION . . . 519
7.24.1.12 ELF64_R_INFO . . . 519
7.24.1.13 ELF64_R_SYM . . . 520
7.24.1.14 ELF64_R_TYPE . . . 520
7.24.1.15 ELFCLASS32 . . . 520
7.24.1.16 ELFCLASS64 . . . 520
7.24.1.17 ELFDATA2LSB . . . 520
7.24.1.18 ELFDATA2MSB . . . 520
7.24.1.19 ELFMAG0 . . . 520
7.24.1.20 ELFMAG1 . . . 520
7.24.1.21 ELFMAG2 . . . 520
7.24.1.22 ELFMAG3 . . . 520
7.24.1.23 ELFOSABI_NONE . . . 521
7.24.1.24 EM_AARCH64 . . . 521

7.24.1.25 EM_RISCV . . . 521
7.24.1.26 EM_X86_64 . . . 521
7.24.1.27 ET_CORE . . . 521
7.24.1.28 ET_DYN . . . 521
7.24.1.29 ET_EXEC . . . 521
7.24.1.30 ET_NONE . . . 521
7.24.1.31 ET_REL . . . 521
7.24.1.32 EV_CURRENT . . . 521
7.24.1.33 PF_R . . . 522
7.24.1.34 PF_W . . . 522
7.24.1.35 PF_X . . . 522
7.24.1.36 PT_DYNAMIC . . . 522
7.24.1.37 PT_INTERP . . . 522
7.24.1.38 PT_LOAD . . . 522
7.24.1.39 PT_NOTE . . . 522
7.24.1.40 PT_NULL . . . 522
7.24.1.41 PT_PHDR . . . 522
7.24.1.42 PT_SHLIB . . . 522
7.24.1.43 PT_TLS . . . 523
7.24.1.44 R_AARCH64_ABS64 . . . 523
7.24.1.45 R_AARCH64_NONE . . . 523
7.24.1.46 R_AARCH64_RELATIVE . . . 523
7.24.1.47 R_RISCV_64 . . . 523
7.24.1.48 R_RISCV_NONE . . . 523
7.24.1.49 R_RISCV_RELATIVE . . . 523
7.24.1.50 R_X86_64_32 . . . 523
7.24.1.51 R_X86_64_32S . . . 523
7.24.1.52 R_X86_64_64 . . . 523
7.24.1.53 R_X86_64_NONE . . . 524
7.24.1.54 R_X86_64_PC32 . . . 524
7.24.1.55 R_X86_64_PLT32 . . . 524
7.24.1.56 R_X86_64_RELATIVE . . . 524
7.24.1.57 SHT_DYNAMIC . . . 524
7.24.1.58 SHT_DYNSYM . . . 524
7.24.1.59 SHT_HASH . . . 524
7.24.1.60 SHT_NOBITS . . . 524
7.24.1.61 SHT_NOTE . . . 524
7.24.1.62 SHT_NULL . . . 524
7.24.1.63 SHT_PROGBITS . . . 525
7.24.1.64 SHT_REL . . . 525
7.24.1.65 SHT_RELA . . . 525
7.24.1.66 SHT_SHLIB . . . 525

7.24.1.67 SHT_STRTAB . . . 525
7.24.1.68 SHT_SYMTAB . . . 525
7.24.2 Typedef Documentation . . . 525
7.24.2.1 Elf64_Addr . . . 525
7.24.2.2 Elf64_Half . . . 525
7.24.2.3 Elf64_Off . . . 525
7.24.2.4 Elf64_Sword . . . 525
7.24.2.5 Elf64_Sxword . . . 526
7.24.2.6 Elf64_Word . . . 526
7.24.2.7 Elf64_Xword . . . 526
7.25 kernel/include/starkernel/elf_loader.h File Reference . . . 526
7.25.1 Function Documentation . . . 527
7.25.1.1 elf_load_kernel() . . . 527
7.26 kernel/include/starkernel/fdt.h File Reference . . . 528
7.26.1 Function Documentation . . . 529
7.26.1.1 fdt_complete() . . . 529
7.26.1.2 fdt_find_child() . . . 530
7.26.1.3 fdt_find_node_by_compatible() . . . 531
7.26.1.4 fdt_find_node_by_device_type() . . . 532
7.26.1.5 fdt_find_node_by_name() . . . 533
7.26.1.6 fdt_find_prop() . . . 534
7.26.1.7 fdt_find_prop_in_node() . . . 535
7.26.1.8 fdt_next_child_node() . . . 537
7.26.1.9 fdt_prop_u32() . . . 538
7.26.1.10 fdt_reg_address() . . . 539
7.26.1.11 fdt_root_node() . . . 539
7.26.1.12 fdt_valid() . . . 540
7.27 kernel/include/starkernel/fdt_uart.h File Reference . . . 542
7.27.1 Enumeration Type Documentation . . . 543
7.27.1.1 anonymous enum . . . 543
7.27.2 Function Documentation . . . 543
7.27.2.1 fdt_console_uart() . . . 543
7.28 kernel/include/starkernel/fe25519.h File Reference . . . 544
7.28.1 Function Documentation . . . 545
7.28.1.1 fe25519_0() . . . 545
7.28.1.2 fe25519_1() . . . 546
7.28.1.3 fe25519_add() . . . 547
7.28.1.4 fe25519_copy() . . . 547
7.28.1.5 fe25519_eq() . . . 547
7.28.1.6 fe25519_invert() . . . 548
7.28.1.7 fe25519_mul() . . . 549
7.28.1.8 fe25519_mul_small() . . . 549

7.28.1.9 fe25519_neg() . . . 550
7.28.1.10 fe25519_pack() . . . 550
7.28.1.11 fe25519_parity() . . . 551
7.28.1.12 fe25519_sq() . . . 551
7.28.1.13 fe25519_sub() . . . 552
7.28.1.14 fe25519_unpack() . . . 553
7.29 kernel/include/starkernel/framebuffer.h File Reference . . . 553
7.29.1 Macro Definition Documentation . . . 554
7.29.1.1 FB_B . . . 554
7.29.1.2 FB_G . . . 554
7.29.1.3 FB_R . . . 555
7.29.1.4 FB_RGB . . . 555
7.29.2 Enumeration Type Documentation . . . 555
7.29.2.1 FbPixelFormat . . . 555
7.29.3 Function Documentation . . . 555
7.29.3.1 __attribute__() . . . 555
7.29.3.2 fb_cell_h() . . . 555
7.29.3.3 fb_cell_w() . . . 556
7.29.3.4 fb_draw_glyph() . . . 556
7.29.3.5 fb_draw_orientation_test() . . . 557
7.29.3.6 fb_fill_rect() . . . 558
7.29.3.7 fb_height() . . . 559
7.29.3.8 fb_init() . . . 559
7.29.3.9 fb_is_available() . . . 560
7.29.3.10 fb_put_pixel() . . . 560
7.29.3.11 fb_scroll_rect() . . . 561
7.29.3.12 fb_scroll_rows() . . . 562
7.29.3.13 fb_width() . . . 563
7.30 kernel/include/starkernel/freestanding/assert.h File Reference . . . 564
7.30.1 Macro Definition Documentation . . . 564
7.30.1.1 assert . . . 564
7.30.1.2 static_assert . . . 565
7.31 kernel/include/starkernel/freestanding/ctype.h File Reference . . . 565
7.31.1 Function Documentation . . . 565
7.31.1.1 isalnum() . . . 565
7.31.1.2 isalpha() . . . 566
7.31.1.3 iscntrl() . . . 566
7.31.1.4 isdigit() . . . 567
7.31.1.5 islower() . . . 567
7.31.1.6 isprint() . . . 567
7.31.1.7 ispunct() . . . 568
7.31.1.8 isspace() . . . 568

7.31.1.9 isupper() . . . 568
7.31.1.10 isxdigit() . . . 569
7.31.1.11 tolower() . . . 569
7.31.1.12 toupper() . . . 569
7.32 kernel/include/starkernel/freestanding/errno.h File Reference . . . 570
7.32.1 Macro Definition Documentation . . . 570
7.32.1.1 EINVAL . . . 570
7.32.1.2 ENOENT . . . 570
7.32.1.3 ENOMEM . . . 570
7.32.1.4 EPERM . . . 570
7.32.1.5 errno . . . 570
7.32.2 Variable Documentation . . . 571
7.32.2.1 _freestanding_errno . . . 571
7.33 kernel/include/starkernel/freestanding/inttypes.h File Reference . . . 571
7.33.1 Macro Definition Documentation . . . 572
7.33.1.1 PRId16 . . . 572
7.33.1.2 PRId32 . . . 572
7.33.1.3 PRId64 . . . 572
7.33.1.4 PRId8 . . . 572
7.33.1.5 PRIi16 . . . 572
7.33.1.6 PRIi32 . . . 572
7.33.1.7 PRIi64 . . . 572
7.33.1.8 PRIi8 . . . 573
7.33.1.9 PRIu16 . . . 573
7.33.1.10 PRIu32 . . . 573
7.33.1.11 PRIu64 . . . 573
7.33.1.12 PRIu8 . . . 573
7.33.1.13 PRIx16 . . . 573
7.33.1.14 PRIX16 . . . 573
7.33.1.15 PRIx32 . . . 573
7.33.1.16 PRIX32 . . . 573
7.33.1.17 PRIx64 . . . 573
7.33.1.18 PRIX64 . . . 574
7.33.1.19 PRIx8 . . . 574
7.33.1.20 PRIX8 . . . 574
7.34 kernel/include/starkernel/freestanding/math.h File Reference . . . 574
7.34.1 Macro Definition Documentation . . . 575
7.34.1.1 HUGE_VAL . . . 575
7.34.1.2 INFINITY . . . 575
7.34.1.3 NAN . . . 575
7.34.2 Function Documentation . . . 575
7.34.2.1 atan2() . . . 575

7.34.2.2 ceil() . . . 575
7.34.2.3 cos() . . . 575
7.34.2.4 exp() . . . 575
7.34.2.5 fabs() . . . 576
7.34.2.6 floor() . . . 576
7.34.2.7 log() . . . 576
7.34.2.8 pow() . . . 576
7.34.2.9 sin() . . . 576
7.34.2.10 sqrt() . . . 576
7.35 kernel/include/starkernel/freestanding/sched.h File Reference . . . 576
7.35.1 Function Documentation . . . 577
7.35.1.1 sched_yield() . . . 577
7.36 kernel/include/starkernel/freestanding/signal.h File Reference . . . 577
7.36.1 Macro Definition Documentation . . . 578
7.36.1.1 SA_RESETHAND . . . 578
7.36.1.2 SIG_DFL . . . 578
7.36.1.3 SIG_IGN . . . 578
7.36.1.4 SIGABRT . . . 578
7.36.1.5 SIGINT . . . 578
7.36.1.6 SIGSEGV . . . 578
7.36.1.7 SIGTERM . . . 578
7.36.2 Typedef Documentation . . . 578
7.36.2.1 sighandler_t . . . 578
7.36.3 Function Documentation . . . 579
7.36.3.1 sigaction() . . . 579
7.36.3.2 signal() . . . 579
7.37 kernel/include/starkernel/freestanding/stdio.h File Reference . . . 579
7.37.1 Macro Definition Documentation . . . 580
7.37.1.1 SEEK_CUR . . . 580
7.37.1.2 SEEK_END . . . 580
7.37.1.3 SEEK_SET . . . 581
7.37.1.4 stderr . . . 581
7.37.1.5 stdin . . . 581
7.37.1.6 stdout . . . 581
7.37.2 Function Documentation . . . 581
7.37.2.1 fclose() . . . 581
7.37.2.2 fflush() . . . 582
7.37.2.3 fgetc() . . . 582
7.37.2.4 fgets() . . . 582
7.37.2.5 fopen() . . . 583
7.37.2.6 fprintf() . . . 583
7.37.2.7 fputc() . . . 584

7.37.2.8 fputs() . . . 585
7.37.2.9 fread() . . . 585
7.37.2.10 fscanf() . . . 586
7.37.2.11 fseek() . . . 586
7.37.2.12 ftell() . . . 586
7.37.2.13 fwrite() . . . 587
7.37.2.14 getchar() . . . 587
7.37.2.15 printf() . . . 587
7.37.2.16 putchar() . . . 588
7.37.2.17 puts() . . . 589
7.37.2.18 rewind() . . . 590
7.37.2.19 snprintf() . . . 590
7.37.2.20 sscanf() . . . 590
7.37.2.21 vfprintf() . . . 591
7.37.2.22 vsnprintf() . . . 592
7.38 kernel/include/starkernel/freestanding/stdlib.h File Reference . . . 592
7.38.1 Function Documentation . . . 593
7.38.1.1 abort() . . . 593
7.38.1.2 atoi() . . . 593
7.38.1.3 atol() . . . 593
7.38.1.4 calloc() . . . 594
7.38.1.5 free() . . . 594
7.38.1.6 malloc() . . . 595
7.38.1.7 qsort() . . . 596
7.38.1.8 realloc() . . . 596
7.38.1.9 strtol() . . . 597
7.38.1.10 strtoll() . . . 598
7.38.1.11 strtoul() . . . 598
7.39 kernel/include/starkernel/freestanding/string.h File Reference . . . 598
7.39.1 Function Documentation . . . 599
7.39.1.1 memchr() . . . 599
7.39.1.2 memcmp() . . . 599
7.39.1.3 memcpy() . . . 600
7.39.1.4 memmove() . . . 601
7.39.1.5 memset() . . . 601
7.39.1.6 strchr() . . . 602
7.39.1.7 strcmp() . . . 602
7.39.1.8 strcpy() . . . 602
7.39.1.9 strerror() . . . 602
7.39.1.10 strlen() . . . 603
7.39.1.11 strncmp() . . . 603
7.39.1.12 strncpy() . . . 604

7.39.1.13 strrchr() . . . 605
7.40 kernel/include/starkernel/freestanding/sys/time.h File Reference . . . 605
7.40.1 Typedef Documentation . . . 606
7.40.1.1 suseconds_t . . . 606
7.40.1.2 time_t . . . 606
7.40.2 Function Documentation . . . 606
7.40.2.1 gettimeofday() . . . 606
7.41 kernel/include/starkernel/freestanding/time.h File Reference . . . 606
7.41.1 Function Documentation . . . 607
7.41.1.1 nanosleep() . . . 607
7.42 kernel/include/starkernel/freestanding/sys/types.h File Reference . . . 607
7.42.1 Typedef Documentation . . . 608
7.42.1.1 gid_t . . . 608
7.42.1.2 mode_t . . . 608
7.42.1.3 off_t . . . 608
7.42.1.4 pid_t . . . 608
7.42.1.5 ssize_t . . . 608
7.42.1.6 uid_t . . . 609
7.43 kernel/include/starkernel/gpt.h File Reference . . . 609
7.43.1 Macro Definition Documentation . . . 610
7.43.1.1 GPT_BAD . . . 610
7.43.1.2 GPT_EIO . . . 610
7.43.1.3 GPT_NONE . . . 610
7.43.1.4 GPT_OK . . . 610
7.43.2 Function Documentation . . . 611
7.43.2.1 gpt_alternate_lba() . . . 611
7.43.2.2 gpt_crc32() . . . 611
7.43.2.3 gpt_disk_guid() . . . 611
7.43.2.4 gpt_find() . . . 612
7.43.2.5 gpt_grow_last() . . . 613
7.43.2.6 gpt_part_open() . . . 613
7.43.2.7 gpt_read() . . . 614
7.43.2.8 gpt_write_one() . . . 615
7.43.3 Variable Documentation . . . 615
7.43.3.1 gpt_type_starship_artemis . . . 615
7.43.3.2 gpt_type_starship_identity . . . 616
7.44 kernel/include/starkernel/hal/hal.h File Reference . . . 616
7.45 kernel/include/starkernel/hal_memory.h File Reference . . . 616
7.45.1 Function Documentation . . . 617
7.45.1.1 hal_mem_alloc() . . . 617
7.45.1.2 hal_mem_alloc_aligned() . . . 617
7.45.1.3 hal_mem_free() . . . 618

7.46 kernel/include/starkernel/homeblocks_sig.h File Reference . . . 618
7.46.1 Macro Definition Documentation . . . 619
7.46.1.1 HOMEBLOCKS_SIG_GET_MAGIC . . . 619
7.46.1.2 HOMEBLOCKS_SIG_GET_VERSION . . . 619
7.46.1.3 HOMEBLOCKS_SIG_MAGIC . . . 620
7.46.1.4 HOMEBLOCKS_SIG_PACK . . . 620
7.46.1.5 HOMEBLOCKS_SIG_START_FBLOCK . . . 620
7.46.1.6 HOMEBLOCKS_SIG_VERSION_0 . . . 620
7.46.2 Typedef Documentation . . . 620
7.46.2.1 homeblocks_sig_size_check . . . 620
7.46.3 Enumeration Type Documentation . . . 620
7.46.3.1 homeblocks_sig_result_t . . . 620
7.46.4 Function Documentation . . . 621
7.46.4.1 homeblocks_sig_check() . . . 621
7.46.4.2 homeblocks_sig_compute_crc() . . . 622
7.47 kernel/include/starkernel/i8042.h File Reference . . . 622
7.47.1 Macro Definition Documentation . . . 623
7.47.1.1 I8042_KEYBOARD_VECTOR . . . 623
7.47.2 Function Documentation . . . 624
7.47.2.1 i8042_drain_stale() . . . 624
7.47.2.2 i8042_handle_irq() . . . 624
7.47.2.3 i8042_init() . . . 625
7.47.2.4 i8042_pop_scancode() . . . 626
7.47.3 Variable Documentation . . . 627
7.47.3.1 g_i8042_isr_count . . . 627
7.48 kernel/include/starkernel/ioapic.h File Reference . . . 627
7.48.1 Function Documentation . . . 628
7.48.1.1 ioapic_init() . . . 628
7.48.1.2 ioapic_route_legacy_irq() . . . 629
7.48.1.3 ioapic_unmask_legacy_irq() . . . 630
7.49 kernel/include/starkernel/kbd_input.h File Reference . . . 631
7.49.1 Function Documentation . . . 631
7.49.1.1 sk_input_drop() . . . 631
7.49.1.2 sk_input_getc() . . . 632
7.50 kernel/include/starkernel/kernel_args.h File Reference . . . 634
7.50.1 Macro Definition Documentation . . . 635
7.50.1.1 KARGS_DEFAULT_HEAP_SIZE . . . 635
7.50.1.2 KARGS_DEFAULT_LOG_LEVEL . . . 635
7.50.1.3 KARGS_DEFAULT_RUN_DOE . . . 635
7.50.1.4 KARGS_DEFAULT_STACK_SIZE . . . 635
7.50.1.5 KARGS_LOG_DEBUG . . . 635
7.50.1.6 KARGS_LOG_ERROR . . . 635

7.50.1.7 KARGS_LOG_INFO . . . 635
7.50.1.8 KARGS_LOG_WARN . . . 635
7.50.1.9 KARGS_SYSTEM_DEFAULT . . . 636
7.50.1.10 KARGS_SYSTEM_V3 . . . 636
7.50.1.11 KARGS_SYSTEM_V4 . . . 636
7.50.1.12 KERNEL_ARGS_CMDLINE_MAX . . . 636
7.50.1.13 REBOOT_MAX_TRIES . . . 636
7.51 kernel/include/starkernel/kmalloc.h File Reference . . . 636
7.51.1 Function Documentation . . . 637
7.51.1.1 kfree() . . . 637
7.51.1.2 kmalloc() . . . 638
7.51.1.3 kmalloc_aligned() . . . 639
7.51.1.4 kmalloc_debug_census() . . . 640
7.51.1.5 kmalloc_debug_census_bytes() . . . 640
7.51.1.6 kmalloc_get_stats() . . . 640
7.51.1.7 kmalloc_heap_base_addr() . . . 640
7.51.1.8 kmalloc_heap_end_addr() . . . 641
7.51.1.9 kmalloc_init() . . . 641
7.51.1.10 kmalloc_is_initialized() . . . 642
7.52 kernel/include/starkernel/log_attrib.h File Reference . . . 642
7.52.1 Function Documentation . . . 642
7.52.1.1 vm_log_attributed_vm() . . . 642
7.53 kernel/include/starkernel/log_region.h File Reference . . . 642
7.53.1 Macro Definition Documentation . . . 644
7.53.1.1 LOG_REGION_DEVBLOCK_FROM_TOP_BASE . . . 644
7.53.1.2 LOG_REGION_GET_MAGIC . . . 644
7.53.1.3 LOG_REGION_GET_VERSION . . . 644
7.53.1.4 LOG_REGION_GROWTH_INCREMENT . . . 644
7.53.1.5 LOG_REGION_INITIAL_DEVBLOCKS . . . 645
7.53.1.6 LOG_REGION_MAGIC . . . 645
7.53.1.7 LOG_REGION_MAX_DEVBLOCKS . . . 645
7.53.1.8 LOG_REGION_PACK . . . 645
7.53.1.9 LOG_REGION_PROTECTED_MAX_LEVEL . . . 645
7.53.1.10 LOG_REGION_VERSION_0 . . . 645
7.53.1.11 LOG_SLOT_MSG_MAX . . . 645
7.53.1.12 LOG_SLOT_SIZE . . . 645
7.53.1.13 LOG_SLOT_SOURCE_MAX . . . 645
7.53.1.14 LOG_SLOTS_PER_DEVBLOCK . . . 645
7.53.2 Typedef Documentation . . . 646
7.53.2.1 log_region_ctrl_size_check . . . 646
7.53.2.2 log_slot_size_check . . . 646
7.53.3 Function Documentation . . . 646

7.53.3.1 log_region_append() . . . 646
7.53.3.2 log_region_read_recent() . . . 646
7.54 kernel/include/starkernel/pci.h File Reference . . . 647
7.54.1 Macro Definition Documentation . . . 648
7.54.1.1 PCI_BAR_IO . . . 648
7.54.1.2 PCI_BAR_MEM_64 . . . 649
7.54.1.3 PCI_BAR_TYPE_MASK . . . 649
7.54.1.4 PCI_CFG_BAR0 . . . 649
7.54.1.5 PCI_CFG_BAR1 . . . 649
7.54.1.6 PCI_CFG_BAR2 . . . 649
7.54.1.7 PCI_CFG_BAR3 . . . 649
7.54.1.8 PCI_CFG_BAR4 . . . 649
7.54.1.9 PCI_CFG_BAR5 . . . 649
7.54.1.10 PCI_CFG_CAP_PTR . . . 649
7.54.1.11 PCI_CFG_CLASS_REV . . . 649
7.54.1.12 PCI_CFG_COMMAND . . . 650
7.54.1.13 PCI_CFG_DEVICE_ID . . . 650
7.54.1.14 PCI_CFG_HEADER_TYPE . . . 650
7.54.1.15 PCI_CFG_INT_LINE . . . 650
7.54.1.16 PCI_CFG_INT_PIN . . . 650
7.54.1.17 PCI_CFG_STATUS . . . 650
7.54.1.18 PCI_CFG_VENDOR_ID . . . 650
7.54.1.19 PCI_CMD_BUS_MASTER . . . 650
7.54.1.20 PCI_CMD_INTX_DISABLE . . . 650
7.54.1.21 PCI_CMD_IO_SPACE . . . 651
7.54.1.22 PCI_CMD_MEM_SPACE . . . 651
7.54.2 Function Documentation . . . 651
7.54.2.1 pci_bar() . . . 651
7.54.2.2 pci_bar_size() . . . 652
7.54.2.3 pci_enable() . . . 652
7.54.2.4 pci_find_class() . . . 653
7.54.2.5 pci_find_first() . . . 654
7.54.2.6 pci_init() . . . 654
7.54.2.7 pci_map_bar() . . . 655
7.54.2.8 pci_read16() . . . 656
7.54.2.9 pci_read32() . . . 656
7.54.2.10 pci_read8() . . . 657
7.54.2.11 pci_write16() . . . 657
7.54.2.12 pci_write32() . . . 658
7.54.2.13 pci_write8() . . . 658
7.55 kernel/include/starkernel/pie_hidden.h File Reference . . . 659
7.56 kernel/include/starkernel/plic.h File Reference . . . 659

7.56.1 Function Documentation . . . 660
7.56.1.1 plic_claim() . . . 660
7.56.1.2 plic_complete() . . . 660
7.56.1.3 plic_disable() . . . 661
7.56.1.4 plic_enable() . . . 661
7.56.1.5 plic_init() . . . 662
7.56.1.6 plic_set_priority() . . . 663
7.57 kernel/include/starkernel/pmm.h File Reference . . . 664
7.57.1 Macro Definition Documentation . . . 665
7.57.1.1 PMM_PAGE_SIZE . . . 665
7.57.2 Function Documentation . . . 665
7.57.2.1 pmm_alloc_contiguous() . . . 665
7.57.2.2 pmm_alloc_page() . . . 666
7.57.2.3 pmm_free_contiguous() . . . 666
7.57.2.4 pmm_free_page() . . . 667
7.57.2.5 pmm_get_stats() . . . 667
7.57.2.6 pmm_init() . . . 668
7.57.2.7 pmm_is_initialized() . . . 669
7.58 kernel/include/starkernel/rng.h File Reference . . . 669
7.58.1 Macro Definition Documentation . . . 670
7.58.1.1 RNG_ERR_NO_BACKEND . . . 670
7.58.2 Function Documentation . . . 670
7.58.2.1 rng_get_bytes() . . . 670
7.58.2.2 rng_init() . . . 671
7.58.2.3 rng_ready() . . . 671
7.59 kernel/include/starkernel/rpi5_dtb.h File Reference . . . 672
7.59.1 Function Documentation . . . 672
7.59.1.1 rpi5_mailbox_base() . . . 672
7.59.1.2 rpi5_uart_base() . . . 673
7.60 kernel/include/starkernel/rpi5_mailbox.h File Reference . . . 674
7.60.1 Function Documentation . . . 675
7.60.1.1 rpi5_mailbox_get_framebuffer() . . . 675
7.61 kernel/include/starkernel/rpi5_mmu.h File Reference . . . 676
7.61.1 Function Documentation . . . 677
7.61.1.1 rpi5_mmu_map_ram() . . . 677
7.61.1.2 rpi5_mmu_uncached() . . . 678
7.62 kernel/include/starkernel/rpi5_native_boot.h File Reference . . . 679
7.62.1 Function Documentation . . . 680
7.62.1.1 rpi5_native_boot() . . . 680
7.63 kernel/include/starkernel/rpi5_native_entry.h File Reference . . . 681
7.63.1 Variable Documentation . . . 682
7.63.1.1 g_rpi5_dtb_ptr . . . 682

7.64 kernel/include/starkernel/scalar25519.h File Reference . . . 682
7.64.1 Function Documentation . . . 683
7.64.1.1 scalar_lt_L() . . . 683
7.64.1.2 scalar_muladd() . . . 684
7.64.1.3 scalar_reduce512() . . . 685
7.65 kernel/include/starkernel/session.h File Reference . . . 686
7.66 kernel/include/starkernel/sha512.h File Reference . . . 686
7.66.1 Function Documentation . . . 687
7.66.1.1 sha512() . . . 687
7.66.1.2 sha512_final() . . . 688
7.66.1.3 sha512_init() . . . 688
7.66.1.4 sha512_update() . . . 689
7.67 kernel/include/starkernel/timer.h File Reference . . . 689
7.67.1 Macro Definition Documentation . . . 691
7.67.1.1 TIME_TRUST_HIGH . . . 691
7.67.1.2 TIME_TRUST_LOW . . . 691
7.67.1.3 TIME_WINDOW_SIZE . . . 692
7.67.2 Typedef Documentation . . . 692
7.67.2.1 time_trust_t . . . 692
7.67.2.2 timer_calibration_record_t . . . 692
7.67.2.3 timer_trust_level_t . . . 692
7.67.2.4 TimeTrustState . . . 692
7.67.2.5 TimeWindow . . . 692
7.67.3 Enumeration Type Documentation . . . 692
7.67.3.1 timer_trust_level . . . 692
7.67.4 Function Documentation . . . 693
7.67.4.1 heartbeat_init() . . . 693
7.67.4.2 heartbeat_next_period_ns() . . . 693
7.67.4.3 heartbeat_read_counter() . . . 694
7.67.4.4 heartbeat_service() . . . 696
7.67.4.5 heartbeat_set_adaptive_period_ns() . . . 696
7.67.4.6 heartbeat_state() . . . 697
7.67.4.7 heartbeat_tick() . . . 698
7.67.4.8 heartbeat_ticks() . . . 699
7.67.4.9 heartbeat_trust() . . . 699
7.67.4.10 timer_calibration_record() . . . 700
7.67.4.11 timer_check_drift_now() . . . 701
7.67.4.12 timer_init() . . . 702
7.67.4.13 timer_now_ns() . . . 704
7.67.4.14 timer_tsc_hz() . . . 705
7.68 kernel/include/starkernel/ttf.h File Reference . . . 706
7.68.1 Macro Definition Documentation . . . 709

7.68.1.1 TTF_CACHE_BITMAP_BYTES . . . 709
7.68.1.2 TTF_CACHE_BITMAP_DIM . . . 709
7.68.1.3 TTF_CACHE_MARGIN . . . 709
7.68.1.4 TTF_CACHE_MAX_SIZE_PX . . . 709
7.68.1.5 TTF_ERR_BAD_GLYPH_INDEX . . . 709
7.68.1.6 TTF_ERR_BAD_SFNT . . . 709
7.68.1.7 TTF_ERR_BAD_TABLE . . . 709
7.68.1.8 TTF_ERR_OUT_OF_BOUNDS . . . 709
7.68.1.9 TTF_ERR_TABLE_MISSING . . . 709
7.68.1.10 TTF_ERR_TOO_DEEP . . . 709
7.68.1.11 TTF_ERR_TOO_MANY_CONTOURS . . . 710
7.68.1.12 TTF_ERR_TOO_MANY_POINTS . . . 710
7.68.1.13 TTF_ERR_UNSUPPORTED . . . 710
7.68.1.14 TTF_GLYPH_MISSING . . . 710
7.68.1.15 TTF_MAX_COMPOSITE_DEPTH . . . 710
7.68.1.16 TTF_OK . . . 710
7.68.2 Function Documentation . . . 711
7.68.2.1 ttf_codepoint_to_glyph() . . . 711
7.68.2.2 ttf_glyph_advance_width() . . . 711
7.68.2.3 ttf_glyph_header() . . . 712
7.68.2.4 ttf_glyph_outline() . . . 713
7.68.2.5 ttf_parse() . . . 714
7.68.2.6 ttf_raster_cache_get() . . . 715
7.68.2.7 ttf_raster_cache_init() . . . 716
7.68.2.8 ttf_rasterize_glyph() . . . 716
7.69 kernel/include/starkernel/uefi.h File Reference . . . 717
7.69.1 Macro Definition Documentation . . . 721
7.69.1.1 EFI_ABORTED . . . 721
7.69.1.2 EFI_BAD_BUFFER_SIZE . . . 721
7.69.1.3 EFI_BUFFER_TOO_SMALL . . . 721
7.69.1.4 EFI_DEVICE_ERROR . . . 721
7.69.1.5 EFI_DEVICE_PATH_SUBTYPE_END_ENTIRE . . . 721
7.69.1.6 EFI_DEVICE_PATH_SUBTYPE_HARDDRIVE . . . 721
7.69.1.7 EFI_DEVICE_PATH_TYPE_END . . . 722
7.69.1.8 EFI_DEVICE_PATH_TYPE_MEDIA . . . 722
7.69.1.9 EFI_FILE_MODE_READ . . . 722
7.69.1.10 EFI_INVALID_PARAMETER . . . 722
7.69.1.11 EFI_LOAD_ERROR . . . 722
7.69.1.12 EFI_MEMORY_RP . . . 722
7.69.1.13 EFI_MEMORY_RUNTIME . . . 722
7.69.1.14 EFI_MEMORY_UC . . . 722
7.69.1.15 EFI_MEMORY_UCE . . . 722

7.69.1.16 EFI_MEMORY_WB . . . 722
7.69.1.17 EFI_MEMORY_WC . . . 723
7.69.1.18 EFI_MEMORY_WP . . . 723
7.69.1.19 EFI_MEMORY_WT . . . 723
7.69.1.20 EFI_MEMORY_XP . . . 723
7.69.1.21 EFI_NOT_FOUND . . . 723
7.69.1.22 EFI_NOT_READY . . . 723
7.69.1.23 EFI_OUT_OF_RESOURCES . . . 723
7.69.1.24 EFI_SUCCESS . . . 723
7.69.1.25 EFI_UNSUPPORTED . . . 723
7.69.1.26 EFI_VARIABLE_BOOTSERVICE_ACCESS . . . 723
7.69.1.27 EFI_VARIABLE_NON_VOLATILE . . . 724
7.69.1.28 EFI_VARIABLE_RUNTIME_ACCESS . . . 724
7.69.1.29 EFI_WRITE_PROTECTED . . . 724
7.69.1.30 EFIAPI . . . 724
7.69.1.31 FALSE . . . 724
7.69.1.32 SF_VAR_BOOT_ARGS . . . 724
7.69.1.33 SF_VAR_REBOOT_TRIES . . . 724
7.69.1.34 SF_VAR_ZUSE_CERT . . . 724
7.69.1.35 STARFORTH_VENDOR_GUID . . . 724
7.69.1.36 TPL_APPLICATION . . . 725
7.69.1.37 TPL_CALLBACK . . . 725
7.69.1.38 TPL_HIGH_LEVEL . . . 725
7.69.1.39 TPL_NOTIFY . . . 725
7.69.1.40 TRUE . . . 725
7.69.2 Typedef Documentation . . . 725
7.69.2.1 BOOLEAN . . . 725
7.69.2.2 CHAR16 . . . 725
7.69.2.3 EFI_ALLOCATE_PAGES . . . 725
7.69.2.4 EFI_ALLOCATE_POOL . . . 725
7.69.2.5 EFI_BLOCK_IO_PROTOCOL . . . 725
7.69.2.6 EFI_BLOCK_READ . . . 726
7.69.2.7 EFI_BOOT_SERVICES . . . 726
7.69.2.8 EFI_EXIT_BOOT_SERVICES . . . 726
7.69.2.9 EFI_FILE_CLOSE . . . 726
7.69.2.10 EFI_FILE_GET_INFO . . . 726
7.69.2.11 EFI_FILE_OPEN . . . 726
7.69.2.12 EFI_FILE_PROTOCOL . . . 726
7.69.2.13 EFI_FILE_READ . . . 726
7.69.2.14 EFI_FREE_PAGES . . . 726
7.69.2.15 EFI_FREE_POOL . . . 726
7.69.2.16 EFI_GET_MEMORY_MAP . . . 727

7.69.2.17 EFI_GET_VARIABLE . . . 727
7.69.2.18 EFI_GRAPHICS_OUTPUT_PROTOCOL . . . 727
7.69.2.19 EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE . . . 727
7.69.2.20 EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE . . . 727
7.69.2.21 EFI_HANDLE . . . 727
7.69.2.22 EFI_HANDLE_PROTOCOL . . . 727
7.69.2.23 EFI_LOCATE_DEVICE_PATH . . . 727
7.69.2.24 EFI_LOCATE_HANDLE . . . 727
7.69.2.25 EFI_LOCATE_PROTOCOL . . . 728
7.69.2.26 EFI_PHYSICAL_ADDRESS . . . 728
7.69.2.27 EFI_RAISE_TPL . . . 728
7.69.2.28 EFI_RESET_SYSTEM . . . 728
7.69.2.29 EFI_RESTORE_TPL . . . 728
7.69.2.30 EFI_RUNTIME_SERVICES . . . 728
7.69.2.31 EFI_SET_VARIABLE . . . 728
7.69.2.32 EFI_SIMPLE_FILE_SYSTEM_PROTOCOL . . . 728
7.69.2.33 EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME . . . 728
7.69.2.34 EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL . . . 728
7.69.2.35 EFI_STALL . . . 729
7.69.2.36 EFI_STATUS . . . 729
7.69.2.37 EFI_SYSTEM_TABLE . . . 729
7.69.2.38 EFI_TEXT_RESET . . . 729
7.69.2.39 EFI_TEXT_STRING . . . 729
7.69.2.40 EFI_TPL . . . 729
7.69.2.41 INT16 . . . 729
7.69.2.42 INT32 . . . 729
7.69.2.43 INT64 . . . 729
7.69.2.44 INT8 . . . 729
7.69.2.45 INTN . . . 730
7.69.2.46 UINT16 . . . 730
7.69.2.47 UINT32 . . . 730
7.69.2.48 UINT64 . . . 730
7.69.2.49 UINT8 . . . 730
7.69.2.50 UINTN . . . 730
7.69.2.51 VOID . . . 730
7.69.3 Enumeration Type Documentation . . . 730
7.69.3.1 EFI_ALLOCATE_TYPE . . . 730
7.69.3.2 EFI_GRAPHICS_PIXEL_FORMAT . . . 731
7.69.3.3 EFI_LOCATE_SEARCH_TYPE . . . 731
7.69.3.4 EFI_MEMORY_TYPE . . . 731
7.69.3.5 EFI_RESET_TYPE . . . 732
7.69.4 Variable Documentation . . . 732

7.69.4.1 EFI_ACPI_20_TABLE_GUID . . . 732
7.69.4.2 EFI_ACPI_TABLE_GUID . . . 732
7.69.4.3 EFI_BLOCK_IO_PROTOCOL_GUID . . . 732
7.69.4.4 EFI_DEVICE_PATH_PROTOCOL_GUID . . . 732
7.69.4.5 EFI_DTB_TABLE_GUID . . . 732
7.69.4.6 EFI_FILE_INFO_GUID . . . 733
7.69.4.7 EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID . . . 733
7.69.4.8 EFI_LOADED_IMAGE_PROTOCOL_GUID . . . 733
7.69.4.9 EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID . . . 733
7.70 kernel/include/starkernel/usb_kbd.h File Reference . . . 734
7.70.1 Macro Definition Documentation . . . 735
7.70.1.1 USB_KBD_LED_CAPS . . . 735
7.70.1.2 USB_KBD_LED_NUM . . . 735
7.70.1.3 USB_KBD_LED_SCROLL . . . 735
7.70.2 Function Documentation . . . 735
7.70.2.1 usb_kbd_forget() . . . 735
7.70.2.2 usb_kbd_leds_wanted() . . . 736
7.70.2.3 usb_kbd_pop_event() . . . 736
7.70.2.4 usb_kbd_report() . . . 736
7.70.2.5 usb_kbd_set_leds() . . . 737
7.71 kernel/include/starkernel/user_identity_seed.h File Reference . . . 738
7.71.1 Macro Definition Documentation . . . 739
7.71.1.1 USER_IDENTITY_EMAIL_MAX . . . 739
7.71.1.2 USER_IDENTITY_FULL_NAME_MAX . . . 739
7.71.1.3 USER_IDENTITY_PHONE_MAX . . . 739
7.71.1.4 USER_IDENTITY_SEED_MAGIC . . . 739
7.71.1.5 USER_IDENTITY_SEED_VERSION . . . 739
7.71.1.6 USER_IDENTITY_USERNAME_MAX . . . 739
7.71.2 Typedef Documentation . . . 739
7.71.2.1 user_identity_seed_size_check . . . 739
7.72 kernel/include/starkernel/v4/sk_v4.h File Reference . . . 740
7.72.1 Function Documentation . . . 741
7.72.1.1 sk_v4_run() . . . 741
7.73 kernel/include/starkernel/v4/sk_v4_grow.h File Reference . . . 742
7.73.1 Function Documentation . . . 743
7.73.1.1 sk_v4_grow_artemis() . . . 743
7.73.1.2 sk_v4_grow_cap_sectors() . . . 743
7.74 kernel/include/starkernel/v4/sk_v4_identity.h File Reference . . . 744
7.74.1 Function Documentation . . . 745
7.74.1.1 sk_v4_identity_beat() . . . 745
7.74.1.2 sk_v4_identity_boot() . . . 746
7.74.1.3 sk_v4_identity_check_now() . . . 747

7.74.1.4 sk_v4_identity_marker() . . . 748
7.74.1.5 sk_v4_identity_offer_wait() . . . 749
7.74.1.6 sk_v4_identity_offered_present() . . . 749
7.74.1.7 sk_v4_identity_present() . . . 750
7.74.1.8 sk_v4_identity_recheck_offered() . . . 750
7.74.1.9 sk_v4_identity_release_offered() . . . 751
7.74.1.10 sk_v4_identity_set_marker() . . . 752
7.74.1.11 sk_v4_identity_stick_in() . . . 752
7.74.1.12 sk_v4_identity_wait() . . . 753
7.75 kernel/include/starkernel/v4/sk_v4_marker.h File Reference . . . 754
7.75.1 Function Documentation . . . 755
7.75.1.1 sk_v4_marker_read() . . . 755
7.75.1.2 sk_v4_marker_write() . . . 755
7.76 kernel/include/starkernel/v4/sk_v4_mint.h File Reference . . . 756
7.76.1 Function Documentation . . . 757
7.76.1.1 sk_v4_mint_zuse() . . . 757
7.77 kernel/include/starkernel/virtio_blk.h File Reference . . . 758
7.77.1 Macro Definition Documentation . . . 759
7.77.1.1 VBLK_ARTEMIS_SECTORS . . . 759
7.77.1.2 VBLK_SECTOR_SIZE . . . 759
7.77.1.3 VBLK_SECTORS_PER_BLOK . . . 759
7.77.1.4 VIRTIO_BLK_DEVICE_LEGACY . . . 759
7.77.1.5 VIRTIO_BLK_DEVICE_MODERN . . . 759
7.77.1.6 VIRTIO_BLK_S_IOERR . . . 759
7.77.1.7 VIRTIO_BLK_S_OK . . . 760
7.77.1.8 VIRTIO_BLK_S_UNSUPP . . . 760
7.77.1.9 VIRTIO_BLK_T_IN . . . 760
7.77.1.10 VIRTIO_BLK_T_OUT . . . 760
7.77.1.11 VIRTIO_PCI_VENDOR_ID . . . 760
7.77.2 Function Documentation . . . 760
7.77.2.1 virtio_blk_find_artemis() . . . 760
7.78 kernel/include/starkernel/virtio_input.h File Reference . . . 761
7.78.1 Macro Definition Documentation . . . 762
7.78.1.1 VIRTIO_INPUT_EV_KEY . . . 762
7.78.1.2 VIRTIO_INPUT_PCI_DEVICE_ID . . . 762
7.78.1.3 VIRTIO_INPUT_PCI_VENDOR_ID . . . 762
7.78.2 Function Documentation . . . 762
7.78.2.1 virtio_input_find_keyboard() . . . 762
7.78.2.2 virtio_input_isr() . . . 763
7.78.2.3 virtio_input_pop_event() . . . 763
7.79 kernel/include/starkernel/virtio_rng.h File Reference . . . 764
7.79.1 Macro Definition Documentation . . . 765

7.79.1.1 VIRTIO_RNG_DEVICE_LEGACY . . . 765
7.79.1.2 VIRTIO_RNG_DEVICE_MODERN . . . 765
7.79.1.3 VIRTIO_RNG_PCI_VENDOR_ID . . . 765
7.79.2 Function Documentation . . . 765
7.79.2.1 virtio_rng_get_bytes() . . . 765
7.79.2.2 virtio_rng_init() . . . 766
7.79.2.3 virtio_rng_ready() . . . 766
7.80 kernel/include/starkernel/vm/arena.h File Reference . . . 767
7.81 kernel/include/starkernel/vm/bootstrap/sk_vm_bootstrap.h File Reference . . . 767
7.82 kernel/include/starkernel/vm/kernel_hermes.h File Reference . . . 768
7.83 kernel/include/starkernel/vm/parity.h File Reference . . . 768
7.83.1 Macro Definition Documentation . . . 769
7.83.1.1 FNV1A_64_OFFSET_BASIS . . . 769
7.83.1.2 FNV1A_64_PRIME . . . 770
7.83.1.3 SK_BOOTSTRAP_ARENA_FAIL . . . 770
7.83.1.4 SK_BOOTSTRAP_DICT_FAIL . . . 770
7.83.1.5 SK_BOOTSTRAP_INIT_FAIL . . . 770
7.83.1.6 SK_BOOTSTRAP_OK . . . 770
7.83.2 Function Documentation . . . 770
7.83.2.1 fnv1a_64() . . . 770
7.83.2.2 sk_dict_canonical_hash() . . . 771
7.83.2.3 sk_dict_word_count() . . . 772
7.83.2.4 sk_parity_collect() . . . 773
7.83.2.5 sk_parity_print() . . . 774
7.84 kernel/include/starkernel/vm/stadium.h File Reference . . . 775
7.85 kernel/include/starkernel/vm/stadium_blocks.h File Reference . . . 775
7.86 kernel/include/starkernel/vm/stadium_words.h File Reference . . . 776
7.87 kernel/include/starkernel/vm/switch.h File Reference . . . 776
7.88 kernel/include/starkernel/vm_identity.h File Reference . . . 777
7.89 kernel/include/starkernel/vm_log_buffer.h File Reference . . . 777
7.90 kernel/include/starkernel/vm_uuid.h File Reference . . . 777
7.91 kernel/include/starkernel/vmm.h File Reference . . . 777
7.91.1 Macro Definition Documentation . . . 779
7.91.1.1 VMM_FLAG_CACHE_DISABLE . . . 779
7.91.1.2 VMM_FLAG_NX . . . 779
7.91.1.3 VMM_FLAG_PRESENT . . . 779
7.91.1.4 VMM_FLAG_USER . . . 779
7.91.1.5 VMM_FLAG_WRITABLE . . . 779
7.91.1.6 VMM_PAGE_SIZE . . . 779
7.91.2 Function Documentation . . . 780
7.91.2.1 vmm_get_paddr() . . . 780
7.91.2.2 vmm_init() . . . 780

7.91.2.3 vmm_map_page() . . . 781
7.91.2.4 vmm_map_range() . . . 782
7.91.2.5 vmm_query_page() . . . 783
7.91.2.6 vmm_unmap_page() . . . 783
7.92 kernel/include/starkernel/vt100.h File Reference . . . 784
7.92.1 Macro Definition Documentation . . . 786
7.92.1.1 VT100_DEFAULT_BG . . . 786
7.92.1.2 VT100_DEFAULT_FG . . . 786
7.92.2 Function Documentation . . . 786
7.92.2.1 vt100_cols() . . . 786
7.92.2.2 vt100_draw_cursor() . . . 786
7.92.2.3 vt100_enable_ttf() . . . 787
7.92.2.4 vt100_erase_cursor() . . . 788
7.92.2.5 vt100_init() . . . 789
7.92.2.6 vt100_putc() . . . 790
7.92.2.7 vt100_puts() . . . 792
7.92.2.8 vt100_reset() . . . 793
7.92.2.9 vt100_rows() . . . 794
7.92.2.10 vt100_scroll_back() . . . 794
7.92.2.11 vt100_scroll_fwd() . . . 795
7.92.2.12 vt100_toggle_graphics() . . . 796
7.93 kernel/include/starkernel/x509_ed25519.h File Reference . . . 796
7.93.1 Function Documentation . . . 797
7.93.1.1 x509_build_user_cert() . . . 797
7.93.1.2 x509_extract_ed25519_pubkey() . . . 798
7.93.1.3 x509_extract_serial() . . . 799
7.93.1.4 x509_verify_signature() . . . 800
7.94 kernel/include/starkernel/xhci.h File Reference . . . 801
7.94.1 Macro Definition Documentation . . . 805
7.94.1.1 SCSI_CDB_LEN_READ10 . . . 805
7.94.1.2 SCSI_CDB_LEN_READ_CAPACITY10 . . . 805
7.94.1.3 SCSI_CDB_LEN_TEST_UNIT_READY . . . 805
7.94.1.4 SCSI_CDB_LEN_WRITE10 . . . 805
7.94.1.5 SCSI_CMD_READ10 . . . 805
7.94.1.6 SCSI_CMD_READ_CAPACITY10 . . . 805
7.94.1.7 SCSI_CMD_TEST_UNIT_READY . . . 805
7.94.1.8 SCSI_CMD_WRITE10 . . . 805
7.94.1.9 SCSI_READ_CAPACITY10_DATA_LEN . . . 805
7.94.1.10 USB_BOT_CBW_FLAG_DATA_IN . . . 806
7.94.1.11 USB_BOT_CBW_LENGTH . . . 806
7.94.1.12 USB_BOT_CBW_LUN_DEFAULT . . . 806
7.94.1.13 USB_BOT_CBW_SIGNATURE . . . 806

7.94.1.14 USB_BOT_CSW_LENGTH . . . 806
7.94.1.15 USB_BOT_CSW_SIGNATURE . . . 806
7.94.1.16 USB_BOT_CSW_STATUS_FAILED . . . 806
7.94.1.17 USB_BOT_CSW_STATUS_PASS . . . 806
7.94.1.18 USB_BOT_CSW_STATUS_PHASE_ERROR . . . 806
7.94.1.19 USB_BOT_MASS_STORAGE_RESET . . . 806
7.94.1.20 USB_CLASS_HID . . . 807
7.94.1.21 USB_CLASS_MASS_STORAGE . . . 807
7.94.1.22 USB_CONFIG_OFF_CONFIG_VALUE . . . 807
7.94.1.23 USB_CONFIG_OFF_TOTAL_LENGTH . . . 807
7.94.1.24 USB_DESC_OFF_LENGTH . . . 807
7.94.1.25 USB_DESC_OFF_TYPE . . . 807
7.94.1.26 USB_DESC_TYPE_CONFIG . . . 807
7.94.1.27 USB_DESC_TYPE_DEVICE . . . 807
7.94.1.28 USB_DESC_TYPE_ENDPOINT . . . 807
7.94.1.29 USB_DESC_TYPE_INTERFACE . . . 807
7.94.1.30 USB_DIR_DEVICE_TO_HOST . . . 808
7.94.1.31 USB_DIR_HOST_TO_DEVICE . . . 808
7.94.1.32 USB_EP_ADDR_DIR_MASK . . . 808
7.94.1.33 USB_EP_ADDR_NUM_MASK . . . 808
7.94.1.34 USB_EP_ATTR_TYPE_MASK . . . 808
7.94.1.35 USB_EP_OFF_ADDRESS . . . 808
7.94.1.36 USB_EP_OFF_ATTRIBUTES . . . 808
7.94.1.37 USB_EP_OFF_INTERVAL . . . 808
7.94.1.38 USB_EP_OFF_MAX_PACKET_SIZE . . . 808
7.94.1.39 USB_EP_TYPE_BULK . . . 808
7.94.1.40 USB_EP_TYPE_INTERRUPT . . . 809
7.94.1.41 USB_FEATURE_ENDPOINT_HALT . . . 809
7.94.1.42 USB_HID_PROTOCOL_BOOT . . . 809
7.94.1.43 USB_HID_REQ_SET_PROTOCOL . . . 809
7.94.1.44 USB_IFACE_OFF_CLASS . . . 809
7.94.1.45 USB_IFACE_OFF_NUMBER . . . 809
7.94.1.46 USB_IFACE_OFF_PROTOCOL . . . 809
7.94.1.47 USB_IFACE_OFF_SUBCLASS . . . 809
7.94.1.48 USB_PROTOCOL_BOT . . . 809
7.94.1.49 USB_PROTOCOL_HID_KEYBOARD . . . 809
7.94.1.50 USB_RECIP_DEVICE . . . 810
7.94.1.51 USB_RECIP_ENDPOINT . . . 810
7.94.1.52 USB_RECIP_INTERFACE . . . 810
7.94.1.53 USB_REQ_CLEAR_FEATURE . . . 810
7.94.1.54 USB_REQ_GET_DESCRIPTOR . . . 810
7.94.1.55 USB_REQ_SET_CONFIGURATION . . . 810

7.94.1.56 USB_REQ_TYPE_CLASS . . . 810
7.94.1.57 USB_REQ_TYPE_STANDARD . . . 810
7.94.1.58 USB_SUBCLASS_HID_BOOT . . . 810
7.94.1.59 USB_SUBCLASS_SCSI . . . 810
7.94.1.60 XHCI_BOT_STALL_MAX_RECOVERIES . . . 811
7.94.1.61 XHCI_BOT_TUR_MAX_RETRIES . . . 811
7.94.1.62 XHCI_COMPLETION_CODE_SHORT_PACKET . . . 811
7.94.1.63 XHCI_COMPLETION_CODE_STALL_ERROR . . . 811
7.94.1.64 XHCI_COMPLETION_CODE_SUCCESS . . . 811
7.94.1.65 XHCI_CONFIG_MAX_SLOTS_EN . . . 811
7.94.1.66 XHCI_CRCR_CA . . . 811
7.94.1.67 XHCI_CRCR_CRR . . . 811
7.94.1.68 XHCI_CRCR_CS . . . 811
7.94.1.69 XHCI_CRCR_PTR_MASK . . . 811
7.94.1.70 XHCI_CRCR_RCS . . . 812
7.94.1.71 XHCI_DB_STREAM_ID . . . 812
7.94.1.72 XHCI_DB_TARGET . . . 812
7.94.1.73 XHCI_EP_ADDR_TO_DCI . . . 812
7.94.1.74 XHCI_EP_CTX_CERR_SHIFT . . . 812
7.94.1.75 XHCI_EP_CTX_INTERVAL_SHIFT . . . 812
7.94.1.76 XHCI_EP_CTX_MAX_PACKET_SHIFT . . . 812
7.94.1.77 XHCI_EP_CTX_TYPE_BULK_IN . . . 812
7.94.1.78 XHCI_EP_CTX_TYPE_BULK_OUT . . . 812
7.94.1.79 XHCI_EP_CTX_TYPE_CONTROL_BIDI . . . 813
7.94.1.80 XHCI_EP_CTX_TYPE_INTERRUPT_IN . . . 813
7.94.1.81 XHCI_EP_CTX_TYPE_SHIFT . . . 813
7.94.1.82 XHCI_ERDP_EHB . . . 813
7.94.1.83 XHCI_ERDP_PTR_MASK . . . 813
7.94.1.84 XHCI_EVT_COMPLETION_CODE . . . 813
7.94.1.85 XHCI_EVT_EP_ID . . . 813
7.94.1.86 XHCI_EVT_RING_MAX_DRAIN . . . 813
7.94.1.87 XHCI_EVT_SLOT_ID . . . 813
7.94.1.88 XHCI_HCCPARAMS1_CSZ . . . 813
7.94.1.89 XHCI_HCSPARAMS1_MAX_INTRS . . . 814
7.94.1.90 XHCI_HCSPARAMS1_MAX_PORTS . . . 814
7.94.1.91 XHCI_HCSPARAMS1_MAX_SLOTS . . . 814
7.94.1.92 XHCI_HCSPARAMS2_MAX_SCRATCHPAD_BUFS . . . 814
7.94.1.93 XHCI_IMAN_IE . . . 814
7.94.1.94 XHCI_IMAN_IP . . . 814
7.94.1.95 XHCI_INPUT_CTRL_ADD_EP0 . . . 814
7.94.1.96 XHCI_INPUT_CTRL_ADD_SLOT . . . 814
7.94.1.97 XHCI_MAX_TRACKED_PORTS . . . 814

7.94.1.98 XHCI_PORT_REGS_OFFSET . . . 814
7.94.1.99 XHCI_PORTSC_CCS . . . 815
7.94.1.100 XHCI_PORTSC_CSC . . . 815
7.94.1.101 XHCI_PORTSC_PEC . . . 815
7.94.1.102 XHCI_PORTSC_PED . . . 815
7.94.1.103 XHCI_PORTSC_PLS_MASK . . . 815
7.94.1.104 XHCI_PORTSC_PP . . . 815
7.94.1.105 XHCI_PORTSC_PR . . . 815
7.94.1.106 XHCI_PORTSC_PRC . . . 815
7.94.1.107 XHCI_PORTSC_SPEED . . . 815
7.94.1.108 XHCI_PORTSC_SPEED_MASK . . . 815
7.94.1.109 XHCI_PSC_EVT_PORT_ID . . . 816
7.94.1.110 XHCI_RING_BYTES . . . 816
7.94.1.111 XHCI_RING_TRB_COUNT . . . 816
7.94.1.112 XHCI_SETUP_TRT_IN_DATA . . . 816
7.94.1.113 XHCI_SETUP_TRT_NO_DATA . . . 816
7.94.1.114 XHCI_SETUP_TRT_OUT_DATA . . . 816
7.94.1.115 XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT . . . 816
7.94.1.116 XHCI_SLOT_CTX_INTR_TARGET_SHIFT . . . 816
7.94.1.117 XHCI_SLOT_CTX_ROOT_PORT_SHIFT . . . 816
7.94.1.118 XHCI_SLOT_CTX_SPEED_SHIFT . . . 816
7.94.1.119 XHCI_TRB_CONTROL_CYCLE . . . 817
7.94.1.120 XHCI_TRB_CONTROL_DIR_IN . . . 817
7.94.1.121 XHCI_TRB_CONTROL_IDT . . . 817
7.94.1.122 XHCI_TRB_CONTROL_IOC . . . 817
7.94.1.123 XHCI_TRB_CONTROL_TC . . . 817
7.94.1.124 XHCI_TRB_CONTROL_TRT_SHIFT . . . 817
7.94.1.125 XHCI_TRB_CONTROL_TYPE_MASK . . . 817
7.94.1.126 XHCI_TRB_CONTROL_TYPE_SHIFT . . . 817
7.94.1.127 XHCI_TRB_TYPE . . . 817
7.94.1.128 XHCI_TRB_TYPE_ADDRESS_DEVICE_CMD . . . 817
7.94.1.129 XHCI_TRB_TYPE_COMMAND_COMPLETION_EVT . . . 818
7.94.1.130 XHCI_TRB_TYPE_CONFIGURE_ENDPOINT_CMD . . . 818
7.94.1.131 XHCI_TRB_TYPE_DATA_STAGE . . . 818
7.94.1.132 XHCI_TRB_TYPE_DISABLE_SLOT_CMD . . . 818
7.94.1.133 XHCI_TRB_TYPE_ENABLE_SLOT_CMD . . . 818
7.94.1.134 XHCI_TRB_TYPE_LINK . . . 818
7.94.1.135 XHCI_TRB_TYPE_NORMAL . . . 818
7.94.1.136 XHCI_TRB_TYPE_PORT_STATUS_CHANGE_EVT . . . 818
7.94.1.137 XHCI_TRB_TYPE_RESET_ENDPOINT_CMD . . . 818
7.94.1.138 XHCI_TRB_TYPE_SET_TR_DEQUEUE_POINTER_CMD . . . 818
7.94.1.139 XHCI_TRB_TYPE_SETUP_STAGE . . . 819

7.94.1.140 XHCI_TRB_TYPE_STATUS_STAGE . . . 819
7.94.1.141 XHCI_TRB_TYPE_TRANSFER_EVENT . . . 819
7.94.1.142 XHCI_USBCMD_HCRST . . . 819
7.94.1.143 XHCI_USBCMD_HSEE . . . 819
7.94.1.144 XHCI_USBCMD_INTE . . . 819
7.94.1.145 XHCI_USBCMD_RUN . . . 819
7.94.1.146 XHCI_USBSTS_CNR . . . 819
7.94.1.147 XHCI_USBSTS_EINT . . . 819
7.94.1.148 XHCI_USBSTS_HCE . . . 819
7.94.1.149 XHCI_USBSTS_HCH . . . 820
7.94.1.150 XHCI_USBSTS_HSE . . . 820
7.94.1.151 XHCI_USBSTS_PCD . . . 820
7.94.2 Typedef Documentation . . . 820
7.94.2.1 xhci_doorbell_t . . . 820
7.95 kernel/include/starkernel/xhci_driver.h File Reference . . . 820
7.95.1 Macro Definition Documentation . . . 822
7.95.1.1 XHCI_MAX_CONTROLLERS . . . 822
7.95.2 Enumeration Type Documentation . . . 822
7.95.2.1 xhci_next_action_t . . . 822
7.95.2.2 xhci_transfer_purpose_t . . . 823
7.95.3 Function Documentation . . . 823
7.95.3.1 xhci_bot_get_capacity() . . . 823
7.95.3.2 xhci_bot_read_block() . . . 824
7.95.3.3 xhci_bot_read_data_in() . . . 825
7.95.3.4 xhci_bot_receive_csw() . . . 826
7.95.3.5 xhci_bot_send_read10() . . . 826
7.95.3.6 xhci_bot_send_read_capacity10() . . . 827
7.95.3.7 xhci_bot_send_test_unit_ready() . . . 828
7.95.3.8 xhci_bot_send_write10() . . . 829
7.95.3.9 xhci_bot_wait_for_idle() . . . 829
7.95.3.10 xhci_bot_write_block() . . . 830
7.95.3.11 xhci_bot_write_data_out() . . . 831
7.95.3.12 xhci_bringup() . . . 832
7.95.3.13 xhci_cmd_address_device() . . . 833
7.95.3.14 xhci_cmd_configure_endpoint() . . . 834
7.95.3.15 xhci_cmd_configure_hid_endpoint() . . . 835
7.95.3.16 xhci_cmd_disable_slot() . . . 836
7.95.3.17 xhci_cmd_enable_slot() . . . 837
7.95.3.18 xhci_cmd_reset_endpoint() . . . 837
7.95.3.19 xhci_cmd_set_tr_dequeue_pointer() . . . 838
7.95.3.20 xhci_dev_at() . . . 839
7.95.3.21 xhci_dev_count() . . . 839

7.95.3.22 xhci_ep0_bot_mass_storage_reset() . . . 840
7.95.3.23 xhci_ep0_clear_endpoint_halt() . . . 840
7.95.3.24 xhci_ep0_get_config_descriptor() . . . 841
7.95.3.25 xhci_ep0_get_device_descriptor() . . . 842
7.95.3.26 xhci_ep0_hid_set_protocol() . . . 843
7.95.3.27 xhci_ep0_set_configuration() . . . 843
7.95.3.28 xhci_find_and_map() . . . 844
7.95.3.29 xhci_keyboard_running() . . . 845
7.95.3.30 xhci_msc_slot_for() . . . 846
7.95.3.31 xhci_poll_events() . . . 846
7.96 kernel/include/starkernel/xxhash64.h File Reference . . . 848
7.96.1 Macro Definition Documentation . . . 849
7.96.1.1 XXHASH64_CAPSULE_SEED . . . 849
7.96.2 Function Documentation . . . 849
7.96.2.1 xxhash64() . . . 849
7.96.2.2 xxhash64_capsule() . . . 850
7.96.2.3 xxhash64_digest() . . . 851
7.96.2.4 xxhash64_reset() . . . 852
7.96.2.5 xxhash64_update() . . . 853
7.97 kernel/include/starkernel/zuse_cert_devblock.h File Reference . . . 853
7.97.1 Macro Definition Documentation . . . 854
7.97.1.1 ZUSE_CERT_DEVBLOCK_MAGIC . . . 854
7.97.1.2 ZUSE_CERT_DEVBLOCK_VERSION . . . 854
7.98 kernel/include/starkernel/zuse_genesis_marker.h File Reference . . . 855
7.98.1 Macro Definition Documentation . . . 856
7.98.1.1 ZUSE_GENESIS_MARKER_MAGIC . . . 856
7.98.1.2 ZUSE_GENESIS_MARKER_VERSION . . . 856
7.98.1.3 ZUSE_GENESIS_MARKER_VERSION_2 . . . 856
7.99 kernel/src/arch/aarch64/rpi5_dtb.c File Reference . . . 856
7.99.1 Macro Definition Documentation . . . 857
7.99.1.1 BCM2712_SOC_RANGES_OFFSET . . . 857
7.99.2 Function Documentation . . . 857
7.99.2.1 reg_first_cell_be32() . . . 857
7.99.2.2 rpi5_mailbox_base() . . . 858
7.99.2.3 rpi5_peripheral_base() . . . 858
7.99.2.4 rpi5_uart_base() . . . 859
7.100 kernel/src/arch/aarch64/rpi5_mailbox.c File Reference . . . 860
7.100.1 Macro Definition Documentation . . . 861
7.100.1.1 MBOX_CH_PROPERTY . . . 861
7.100.1.2 MBOX_CODE_REQUEST . . . 861
7.100.1.3 MBOX_CODE_RESPONSE_OK . . . 861
7.100.1.4 MBOX_EMPTY . . . 861

7.100.1.5 MBOX_FULL . . . 861
7.100.1.6 MBOX_OFF_READ . . . 862
7.100.1.7 MBOX_OFF_STATUS0 . . . 862
7.100.1.8 MBOX_OFF_STATUS1 . . . 862
7.100.1.9 MBOX_OFF_WRITE . . . 862
7.100.1.10 PIXEL_ORDER_RGB . . . 862
7.100.1.11 TAG_ALLOCATE_BUFFER . . . 863
7.100.1.12 TAG_END . . . 863
7.100.1.13 TAG_GET_PITCH . . . 863
7.100.1.14 TAG_SET_DEPTH . . . 863
7.100.1.15 TAG_SET_PHYS_SIZE . . . 863
7.100.1.16 TAG_SET_PIXEL_ORDER . . . 863
7.100.1.17 TAG_SET_VIRT_SIZE . . . 863
7.100.1.18 TAG_SET_VIRTUAL_OFFS . . . 863
7.100.2 Function Documentation . . . 863
7.100.2.1 __attribute__() . . . 863
7.100.2.2 append_tag() . . . 864
7.100.2.3 buffer_to_ram() . . . 864
7.100.2.4 bus_to_phys() . . . 864
7.100.2.5 counter_freq() . . . 865
7.100.2.6 counter_now() . . . 865
7.100.2.7 mb_be32() . . . 866
7.100.2.8 mb_cell_prop() . . . 866
7.100.2.9 mb_cells() . . . 867
7.100.2.10 mb_compatible() . . . 867
7.100.2.11 mb_find_path() . . . 868
7.100.2.12 mmio_read32() . . . 869
7.100.2.13 mmio_write32() . . . 869
7.100.2.14 rpi5_mailbox_get_framebuffer() . . . 869
7.101 kernel/src/arch/aarch64/rpi5_mmu.c File Reference . . . 871
7.101.1 Macro Definition Documentation . . . 872
7.101.1.1 BLOCK_NORMAL_NC . . . 872
7.101.1.2 BLOCK_NORMAL_WB . . . 872
7.101.1.3 GIB . . . 872
7.101.1.4 MIB2 . . . 872
7.101.2 Function Documentation . . . 872
7.101.2.1 be32_at() . . . 872
7.101.2.2 cells_at() . . . 873
7.101.2.3 current_el() . . . 873
7.101.2.4 l2_gib() . . . 874
7.101.2.5 replace_entry() . . . 874
7.101.2.6 rpi5_mmu_map_ram() . . . 875

7.101.2.7 rpi5_mmu_uncached() . . . 875
7.101.2.8 tlb_flush_all() . . . 876
7.101.3 Variable Documentation . . . 877
7.101.3.1 __kernel_end . . . 877
7.101.3.2 __kernel_start . . . 877
7.101.3.3 g_rpi5_l1 . . . 877
7.101.3.4 g_rpi5_l2 . . . 877
7.102 kernel/src/arch/aarch64/rpi5_native_boot.c File Reference . . . 878
7.102.1 Macro Definition Documentation . . . 879
7.102.1.1 RPI5_FB_DEFAULT_BPP . . . 879
7.102.1.2 RPI5_FB_DEFAULT_HEIGHT . . . 879
7.102.1.3 RPI5_FB_DEFAULT_WIDTH . . . 879
7.102.1.4 RPI5_MAX_MEMMAP_ENTRIES . . . 880
7.102.1.5 RPI5_MAX_RAM_REGIONS . . . 880
7.102.1.6 RPI5_MAX_RESERVED_RANGES . . . 880
7.102.2 Function Documentation . . . 880
7.102.2.1 add_memmap_entry() . . . 880
7.102.2.2 be32_at() . . . 881
7.102.2.3 build_memory_map() . . . 881
7.102.2.4 collect_ram_regions() . . . 882
7.102.2.5 collect_reserved_ranges() . . . 883
7.102.2.6 emit_region_with_carveouts() . . . 884
7.102.2.7 kernel_main() . . . 884
7.102.2.8 node_cells_or_fallback() . . . 887
7.102.2.9 read_cells() . . . 888
7.102.2.10 rpi5_native_boot() . . . 888
7.102.2.11 rpi5_native_halt() . . . 889
7.102.2.12 sort_ranges_by_addr() . . . 890
7.102.3 Variable Documentation . . . 890
7.102.3.1 g_rpi5_boot_info . . . 890
7.102.3.2 g_rpi5_memmap . . . 890
7.103 kernel/src/arch/amd64/i8042.c File Reference . . . 891
7.103.1 Macro Definition Documentation . . . 892
7.103.1.1 I8042_CMD_PORT . . . 892
7.103.1.2 I8042_CMD_READ_CONFIG . . . 892
7.103.1.3 I8042_CMD_WRITE_CONFIG . . . 892
7.103.1.4 I8042_CONFIG_IRQ1_ENABLE . . . 892
7.103.1.5 I8042_DATA_PORT . . . 892
7.103.1.6 I8042_DEV_ACK . . . 892
7.103.1.7 I8042_DEV_CMD_ENABLE_SCANNING . . . 893
7.103.1.8 I8042_INIT_WAIT_ITERS . . . 893
7.103.1.9 I8042_STATUS_INPUT_FULL . . . 893

7.103.1.10 I8042_STATUS_OUTPUT_FULL . . . 893
7.103.1.11 I8042_STATUS_PORT . . . 893
7.103.1.12 RING_SIZE . . . 893
7.103.2 Function Documentation . . . 893
7.103.2.1 i8042_drain() . . . 893
7.103.2.2 i8042_drain_stale() . . . 894
7.103.2.3 i8042_handle_irq() . . . 894
7.103.2.4 i8042_init() . . . 895
7.103.2.5 i8042_pop_scancode() . . . 896
7.103.2.6 inb() . . . 897
7.103.2.7 outb() . . . 897
7.103.3 Variable Documentation . . . 898
7.103.3.1 g_i8042_isr_count . . . 898
7.103.3.2 g_i8042_present . . . 898
7.103.3.3 ring . . . 898
7.103.3.4 ring_head . . . 899
7.103.3.5 ring_tail . . . 899
7.104 kernel/src/arch/amd64/ioapic.c File Reference . . . 899
7.104.1 Macro Definition Documentation . . . 900
7.104.1.1 ISO_POLARITY_HIGH . . . 900
7.104.1.2 ISO_POLARITY_LOW . . . 900
7.104.1.3 ISO_POLARITY_MASK . . . 900
7.104.1.4 ISO_TRIGGER_EDGE . . . 901
7.104.1.5 ISO_TRIGGER_LEVEL . . . 901
7.104.1.6 ISO_TRIGGER_MASK . . . 901
7.104.1.7 MADT_TYPE_INT_SRC . . . 901
7.104.1.8 MADT_TYPE_IOAPIC . . . 901
7.104.1.9 MAX_OVERRIDES . . . 901
7.104.1.10 MAX_REDIR_ENTRIES . . . 901
7.104.2 Function Documentation . . . 901
7.104.2.1 ioapic_init() . . . 901
7.104.2.2 ioapic_route_legacy_irq() . . . 902
7.104.2.3 ioapic_unmask_legacy_irq() . . . 903
7.104.2.4 ioapic_write() . . . 904
7.104.2.5 parse_madt() . . . 904
7.104.2.6 sig4_eq() . . . 905
7.104.2.7 sig8_eq() . . . 905
7.104.3 Variable Documentation . . . 906
7.104.3.1 g_ioapic_base . . . 906
7.104.3.2 g_ioapic_gsi_base . . . 906
7.104.3.3 g_ioapic_ready . . . 906
7.104.3.4 g_override_count . . . 906

7.104.3.5 g_overrides . . . 906
7.104.3.6 g_redir_low . . . 907
7.104.3.7 g_redir_valid . . . 907
7.105 kernel/src/arch/aarch64/apic.c File Reference . . . 907
7.105.1 Macro Definition Documentation . . . 909
7.105.1.1 GICC_BASE_PA_QEMU_DEFAULT . . . 909
7.105.1.2 GICC_CTLR . . . 909
7.105.1.3 GICC_CTLR_ENABLE_GRP0 . . . 909
7.105.1.4 GICC_EOIR . . . 909
7.105.1.5 GICC_IAR . . . 909
7.105.1.6 GICC_PMR . . . 909
7.105.1.7 GICD_BASE_PA_QEMU_DEFAULT . . . 909
7.105.1.8 GICD_CTLR . . . 909
7.105.1.9 GICD_CTLR_ENABLE_GRP0 . . . 909
7.105.1.10 GICD_ICFGR . . . 909
7.105.1.11 GICD_IPRIORITYR . . . 910
7.105.1.12 GICD_ISENABLER0 . . . 910
7.105.1.13 GICD_ISPENDR . . . 910
7.105.1.14 GICD_ITARGETSR . . . 910
7.105.1.15 PMR_ALLOW_ALL . . . 910
7.105.1.16 TIMER_PPI_EL1 . . . 910
7.105.1.17 TIMER_PPI_EL2 . . . 910
7.105.1.18 TIMER_PRIORITY . . . 910
7.105.2 Function Documentation . . . 911
7.105.2.1 aarch64_current_el() . . . 911
7.105.2.2 aarch64_psci_conduit_init() . . . 912
7.105.2.3 apic_eoi() . . . 913
7.105.2.4 apic_eoi_intid() . . . 913
7.105.2.5 apic_init() . . . 914
7.105.2.6 apic_read_iar() . . . 915
7.105.2.7 apic_spi_enable() . . . 916
7.105.2.8 apic_timer_init() . . . 916
7.105.2.9 apic_timer_period_tsc() . . . 917
7.105.2.10 apic_timer_ppi() . . . 917
7.105.2.11 apic_timer_rearm() . . . 918
7.105.2.12 apic_timer_start() . . . 918
7.105.2.13 apic_timer_stop() . . . 919
7.105.2.14 gic_bases_from_dtb() . . . 919
7.105.2.15 mmio_read32() . . . 920
7.105.2.16 mmio_write32() . . . 921
7.105.3 Variable Documentation . . . 921
7.105.3.1 s_counter_hz_apic . . . 921

7.105.3.2 s_gicc_base . . . 921
7.105.3.3 s_gicd_base . . . 921
7.105.3.4 s_timer_period_tsc . . . 921
7.105.3.5 s_timer_ppi . . . 922
7.106 kernel/src/arch/amd64/apic.c File Reference . . . 922
7.106.1 Macro Definition Documentation . . . 924
7.106.1.1 APIC_REG_EOI . . . 924
7.106.1.2 APIC_REG_ID . . . 924
7.106.1.3 APIC_REG_ISR_BASE . . . 924
7.106.1.4 APIC_REG_LVT_TIMER . . . 924
7.106.1.5 APIC_REG_SIVR . . . 924
7.106.1.6 APIC_REG_TIMER_CCR . . . 924
7.106.1.7 APIC_REG_TIMER_DCR . . . 924
7.106.1.8 APIC_REG_TIMER_ICR . . . 924
7.106.1.9 APIC_REG_TPR . . . 924
7.106.1.10 IA32_APIC_BASE_GLOBAL_EN . . . 924
7.106.1.11 IA32_APIC_BASE_MSR . . . 925
7.106.1.12 IA32_APIC_BASE_X2APIC_EN . . . 925
7.106.1.13 LAPIC_DEFAULT_PHYS . . . 925
7.106.1.14 LAPIC_VIRT_BASE . . . 925
7.106.1.15 LVT_MASKED . . . 925
7.106.1.16 LVT_MODE_ONESHOT . . . 925
7.106.1.17 LVT_MODE_PERIODIC . . . 925
7.106.1.18 TIMER_DIV_1 . . . 925
7.106.1.19 TIMER_DIV_128 . . . 925
7.106.1.20 TIMER_DIV_16 . . . 926
7.106.1.21 TIMER_DIV_2 . . . 926
7.106.1.22 TIMER_DIV_32 . . . 926
7.106.1.23 TIMER_DIV_4 . . . 926
7.106.1.24 TIMER_DIV_64 . . . 926
7.106.1.25 TIMER_DIV_8 . . . 926
7.106.2 Function Documentation . . . 926
7.106.2.1 apic_eoi() . . . 926
7.106.2.2 apic_id() . . . 927
7.106.2.3 apic_init() . . . 927
7.106.2.4 apic_timer_init() . . . 928
7.106.2.5 apic_timer_period_tsc() . . . 930
7.106.2.6 apic_timer_rearm() . . . 930
7.106.2.7 apic_timer_start() . . . 931
7.106.2.8 apic_timer_stop() . . . 932
7.106.2.9 calibrate_apic_timer() . . . 933
7.106.2.10 lapic_read() . . . 934

7.106.2.11 lapic_write() . . . 935
7.106.2.12 rdmsr64() . . . 936
7.106.2.13 wrmsr64() . . . 937
7.106.3 Variable Documentation . . . 938
7.106.3.1 lapic_base . . . 938
7.106.3.2 lapic_phys_base . . . 938
7.106.3.3 s_apic_hz . . . 938
7.106.3.4 timer_initial_count . . . 938
7.106.3.5 timer_period_tsc_ticks . . . 938
7.106.3.6 timer_tick_hz . . . 938
7.107 kernel/src/arch/riscv64/apic.c File Reference . . . 939
7.107.1 Macro Definition Documentation . . . 940
7.107.1.1 SBI_BASE_FID_PROBE_EXT . . . 940
7.107.1.2 SBI_EXT_BASE . . . 940
7.107.1.3 SBI_EXT_TIME . . . 940
7.107.1.4 SBI_SUCCESS . . . 940
7.107.1.5 SBI_TIME_FID_SET_TIMER . . . 940
7.107.1.6 SIE_STIE . . . 941
7.107.2 Function Documentation . . . 941
7.107.2.1 apic_eoi() . . . 941
7.107.2.2 apic_init() . . . 941
7.107.2.3 apic_timer_init() . . . 943
7.107.2.4 apic_timer_period_tsc() . . . 945
7.107.2.5 apic_timer_start() . . . 945
7.107.2.6 apic_timer_stop() . . . 946
7.107.2.7 rdtime() . . . 946
7.107.2.8 riscv64_timer_rearm() . . . 947
7.107.2.9 sbi_call1() . . . 948
7.107.2.10 sbi_set_timer() . . . 948
7.107.3 Variable Documentation . . . 949
7.107.3.1 s_next_deadline . . . 949
7.107.3.2 s_sbi_time_ok . . . 949
7.107.3.3 s_time_hz . . . 949
7.107.3.4 s_timer_period_tsc . . . 949
7.108 kernel/src/arch/aarch64/arch.c File Reference . . . 949
7.108.1 Function Documentation . . . 950
7.108.1.1 aarch64_current_el() . . . 950
7.108.1.2 aarch64_install_vectors() . . . 951
7.108.1.3 aarch64_psci_conduit_init() . . . 951
7.108.1.4 arch_cold_reset() . . . 952
7.108.1.5 arch_disable_interrupts() . . . 952
7.108.1.6 arch_early_init() . . . 953

7.108.1.7 arch_enable_interrupts() . . . 953
7.108.1.8 arch_halt() . . . 953
7.108.1.9 arch_mmu_init() . . . 954
7.108.1.10 arch_read_timestamp() . . . 954
7.108.2 Variable Documentation . . . 954
7.108.2.1 s_current_el . . . 954
7.108.2.2 s_psci_use_smc . . . 954
7.109 kernel/src/arch/amd64/arch.c File Reference . . . 955
7.109.1 Function Documentation . . . 956
7.109.1.1 __attribute__() [1/2] . . . 956
7.109.1.2 __attribute__() [2/2] . . . 956
7.109.1.3 arch_cold_reset() . . . 956
7.109.1.4 arch_disable_interrupts() . . . 956
7.109.1.5 arch_early_init() . . . 957
7.109.1.6 arch_enable_interrupts() . . . 957
7.109.1.7 arch_halt() . . . 957
7.109.1.8 arch_mmu_init() . . . 958
7.109.1.9 arch_read_timestamp() . . . 958
7.109.1.10 gdt_init() . . . 958
7.109.1.11 gdt_set_entry() . . . 959
7.109.2 Variable Documentation . . . 960
7.109.2.1 access . . . 960
7.109.2.2 base . . . 960
7.109.2.3 base_high . . . 960
7.109.2.4 base_low . . . 960
7.109.2.5 base_mid . . . 960
7.109.2.6 granularity . . . 960
7.109.2.7 limit . . . 961
7.109.2.8 limit_low . . . 961
7.110 kernel/src/arch/riscv64/arch.c File Reference . . . 961
7.110.1 Function Documentation . . . 962
7.110.1.1 arch_cold_reset() . . . 962
7.110.1.2 arch_disable_interrupts() . . . 963
7.110.1.3 arch_early_init() . . . 963
7.110.1.4 arch_enable_interrupts() . . . 964
7.110.1.5 arch_halt() . . . 964
7.110.1.6 arch_mmu_init() . . . 965
7.110.1.7 arch_read_timestamp() . . . 965
7.110.1.8 print_hex64() . . . 966
7.110.1.9 riscv64_install_vectors() . . . 966
7.110.2 Variable Documentation . . . 966
7.110.2.1 __kernel_start . . . 966

7.111 kernel/src/arch/aarch64/interrupts.c File Reference . . . 967
7.111.1 Macro Definition Documentation . . . 968
7.111.1.1 GIC_INTID_SPURIOUS . . . 968
7.111.2 Function Documentation . . . 968
7.111.2.1 aarch64_current_el() . . . 968
7.111.2.2 aarch64_exception_handler() . . . 969
7.111.2.3 aarch64_install_vectors() . . . 970
7.111.2.4 aarch64_irq_handler() . . . 970
7.111.2.5 apic_eoi_intid() . . . 971
7.111.2.6 apic_read_iar() . . . 972
7.111.2.7 apic_timer_ppi() . . . 973
7.111.2.8 apic_timer_rearm() . . . 973
7.111.2.9 arch_interrupts_init() . . . 974
7.111.2.10 print_hex() . . . 975
7.111.2.11 virtio_input_isr() . . . 976
7.111.3 Variable Documentation . . . 977
7.111.3.1 g_sk_fault_word . . . 977
7.111.3.2 g_virtio_input_gic_intid . . . 977
7.112 kernel/src/arch/amd64/interrupts.c File Reference . . . 977
7.112.1 Macro Definition Documentation . . . 978
7.112.1.1 IDT_ENTRIES . . . 978
7.112.1.2 INTERRUPT_GATE . . . 979
7.112.2 Function Documentation . . . 979
7.112.2.1 __attribute__() [1/2] . . . 979
7.112.2.2 __attribute__() [2/2] . . . 979
7.112.2.3 decode_pf_error() . . . 979
7.112.2.4 isr_common_handler() . . . 980
7.112.2.5 isr_stub0() . . . 981
7.112.2.6 lidt() . . . 982
7.112.2.7 outb() . . . 982
7.112.2.8 pic_disable() . . . 984
7.112.2.9 print_dec_u64() . . . 984
7.112.2.10 print_hex64() . . . 985
7.112.2.11 set_idt_entry() . . . 986
7.112.3 Variable Documentation . . . 987
7.112.3.1 base . . . 987
7.112.3.2 g_sk_fault_word . . . 987
7.112.3.3 g_spurious_count . . . 987
7.112.3.4 idt . . . 987
7.112.3.5 ist . . . 987
7.112.3.6 limit . . . 988
7.112.3.7 offset_high . . . 988

7.112.3.8 offset_low . . . 988
7.112.3.9 offset_mid . . . 988
7.112.3.10 selector . . . 988
7.112.3.11 type_attr . . . 988
7.112.3.12 zero . . . 988
7.113 kernel/src/arch/riscv64/interrupts.c File Reference . . . 989
7.113.1 Macro Definition Documentation . . . 990
7.113.1.1 SCAUSE_INTERRUPT_BIT . . . 990
7.113.1.2 SCAUSE_S_EXTERNAL . . . 990
7.113.1.3 SCAUSE_S_TIMER . . . 990
7.113.2 Function Documentation . . . 990
7.113.2.1 arch_interrupts_init() . . . 990
7.113.2.2 print_hex() . . . 991
7.113.2.3 riscv64_exception_handler() . . . 992
7.113.2.4 riscv64_install_vectors() . . . 993
7.113.2.5 riscv64_interrupt_handler() . . . 993
7.113.2.6 riscv64_timer_rearm() . . . 994
7.113.2.7 virtio_input_isr() . . . 995
7.113.3 Variable Documentation . . . 995
7.113.3.1 g_plic_claim_count . . . 995
7.113.3.2 g_sk_fault_word . . . 995
7.113.3.3 g_virtio_input_plic_source . . . 995
7.114 kernel/src/arch/riscv64/plic.c File Reference . . . 996
7.114.1 Macro Definition Documentation . . . 997
7.114.1.1 PLIC_BASE_QEMU_DEFAULT . . . 997
7.114.1.2 PLIC_CLAIM . . . 997
7.114.1.3 PLIC_CONTEXT_S . . . 998
7.114.1.4 PLIC_ENABLE . . . 998
7.114.1.5 PLIC_PRIORITY . . . 998
7.114.1.6 PLIC_THRESHOLD . . . 998
7.114.2 Function Documentation . . . 998
7.114.2.1 be32_cell() . . . 998
7.114.2.2 be64_addr_cell_pair() . . . 999
7.114.2.3 plic_base_from_dtb() . . . 999
7.114.2.4 plic_claim() . . . 1000
7.114.2.5 plic_complete() . . . 1001
7.114.2.6 plic_disable() . . . 1002
7.114.2.7 plic_enable() . . . 1002
7.114.2.8 plic_init() . . . 1002
7.114.2.9 plic_set_priority() . . . 1004
7.114.2.10 reg32() . . . 1005
7.114.3 Variable Documentation . . . 1005

7.114.3.1 s_plic_base . . . 1005
7.115 kernel/src/arch/aarch64/timer.c File Reference . . . 1006
7.115.1 Function Documentation . . . 1007
7.115.1.1 cntfrq_read() . . . 1007
7.115.1.2 cntpct_read() . . . 1008
7.115.1.3 heartbeat_read_counter() . . . 1008
7.115.1.4 running_under_hypervisor() . . . 1009
7.115.1.5 timer_calibration_record() . . . 1009
7.115.1.6 timer_check_drift_now() . . . 1010
7.115.1.7 timer_init() . . . 1010
7.115.1.8 timer_now_ns() . . . 1010
7.115.1.9 timer_tsc_hz() . . . 1010
7.115.2 Variable Documentation . . . 1011
7.115.2.1 s_base_count . . . 1011
7.115.2.2 s_base_ns . . . 1011
7.115.2.3 s_cal . . . 1011
7.115.2.4 s_counter_hz . . . 1011
7.115.2.5 s_ns_per_tick . . . 1011
7.116 kernel/src/arch/amd64/timer.c File Reference . . . 1012
7.116.1 Macro Definition Documentation . . . 1015
7.116.1.1 CV_EPSILON_PPM . . . 1015
7.116.1.2 CV_STABLE_COUNT . . . 1015
7.116.1.3 DRIFT_EPSILON_PPM . . . 1015
7.116.1.4 HPET_GEN_CAP_ID . . . 1015
7.116.1.5 HPET_GEN_CONFIG . . . 1015
7.116.1.6 HPET_MAIN_COUNTER . . . 1015
7.116.1.7 HPET_PHYS_BASE . . . 1016
7.116.1.8 HPET_VIRT_BASE . . . 1016
7.116.1.9 PIT_CHANNEL0_DATA . . . 1016
7.116.1.10 PIT_COMMAND . . . 1016
7.116.1.11 PIT_FREQ_HZ . . . 1016
7.116.1.12 PMTIMER_FREQ_HZ . . . 1016
7.116.1.13 PMTIMER_IO_PORT_DEFAULT . . . 1016
7.116.1.14 PMTIMER_MASK . . . 1016
7.116.1.15 PMTIMER_WIDTH_BITS . . . 1016
7.116.1.16 SAMPLE_WINDOWS . . . 1016
7.116.1.17 TIMER_REQUIRE_PIT . . . 1017
7.116.1.18 TIMER_VM_STRICT_INVARIANT_TSC . . . 1017
7.116.1.19 WARMUP_WINDOWS . . . 1017
7.116.1.20 WINDOW_TICKS_HPET . . . 1017
7.116.1.21 WINDOW_TICKS_PIT . . . 1017
7.116.2 Function Documentation . . . 1017

7.116.2.1 calibrate_tsc_with_pmtimer() . . . 1017
7.116.2.2 calibrate_window_hpet() . . . 1018
7.116.2.3 calibrate_window_pit() . . . 1019
7.116.2.4 converge_tsc_bare_metal() . . . 1020
7.116.2.5 cpu_has_invariant_tsc() . . . 1021
7.116.2.6 cpu_has_rdtscp() . . . 1022
7.116.2.7 cpuid() . . . 1023
7.116.2.8 derive_tsc_hz_from_cpuid() . . . 1024
7.116.2.9 enable_hpet() . . . 1025
7.116.2.10 fadt_find_pm_tmr_port() . . . 1026
7.116.2.11 heartbeat_read_counter() . . . 1027
7.116.2.12 hpet_read() . . . 1027
7.116.2.13 hpet_write() . . . 1028
7.116.2.14 inb() . . . 1028
7.116.2.15 init_timer_vm_mode() . . . 1030
7.116.2.16 inl() . . . 1031
7.116.2.17 muldiv64() . . . 1032
7.116.2.18 outb() . . . 1033
7.116.2.19 pit_probe_running() . . . 1034
7.116.2.20 pit_program_reload() . . . 1034
7.116.2.21 pit_read_counter16() . . . 1035
7.116.2.22 pmtimer_delta() . . . 1036
7.116.2.23 pmtimer_read() . . . 1037
7.116.2.24 print_dec() . . . 1038
7.116.2.25 rdtsc() . . . 1039
7.116.2.26 rdtscp_aux() . . . 1039
7.116.2.27 running_under_hypervisor() . . . 1040
7.116.2.28 sanity_check_hpet_running_quick() . . . 1041
7.116.2.29 timer_calibration_record() . . . 1041
7.116.2.30 timer_check_drift_now() . . . 1042
7.116.2.31 timer_drift_check_bare_metal() . . . 1042
7.116.2.32 timer_fatal() . . . 1043
7.116.2.33 timer_init() . . . 1044
7.116.2.34 timer_now_ns() . . . 1046
7.116.2.35 timer_now_ns_vm_relative() . . . 1047
7.116.2.36 timer_sig4_eq() . . . 1048
7.116.2.37 timer_sig8_eq() . . . 1048
7.116.2.38 timer_tsc_hz() . . . 1048
7.116.2.39 update_stats() . . . 1049
7.116.2.40 vm_timebase_init_from_pmtimer() . . . 1049
7.116.3 Variable Documentation . . . 1050
7.116.3.1 calib_record . . . 1050

7.116.3.2 hpet_freq_hz . . . 1050
7.116.3.3 hpet_regs . . . 1050
7.116.3.4 pit_available . . . 1051
7.116.3.5 pit_tsc_hz_mean . . . 1051
7.116.3.6 pmtimer_port . . . 1051
7.116.3.7 tsc_hz_locked . . . 1051
7.116.3.8 vm_ns_base . . . 1051
7.116.3.9 vm_pm_start . . . 1051
7.116.3.10 windows_used . . . 1051
7.117 kernel/src/arch/riscv64/timer.c File Reference . . . 1052
7.117.1 Macro Definition Documentation . . . 1053
7.117.1.1 RISCV_TIMEBASE_HZ_FALLBACK . . . 1053
7.117.2 Function Documentation . . . 1053
7.117.2.1 bytes_contain() . . . 1053
7.117.2.2 heartbeat_read_counter() . . . 1054
7.117.2.3 print_dec() . . . 1055
7.117.2.4 rdtime() . . . 1056
7.117.2.5 running_under_hypervisor() . . . 1057
7.117.2.6 timer_calibration_record() . . . 1057
7.117.2.7 timer_check_drift_now() . . . 1057
7.117.2.8 timer_init() . . . 1058
7.117.2.9 timer_now_ns() . . . 1059
7.117.2.10 timer_tsc_hz() . . . 1060
7.117.3 Variable Documentation . . . 1060
7.117.3.1 s_base_count . . . 1060
7.117.3.2 s_base_ns . . . 1061
7.117.3.3 s_cal . . . 1061
7.117.3.4 s_counter_hz . . . 1061
7.117.3.5 s_ns_per_tick . . . 1061
7.118 kernel/src/artemis_sig.c File Reference . . . 1061
7.118.1 Function Documentation . . . 1062
7.118.1.1 artemis_sig_check() . . . 1062
7.118.1.2 artemis_sig_compute_crc() . . . 1063
7.118.1.3 artemis_sig_genesis_stamp() . . . 1063
7.118.1.4 artemis_sig_start_fblock() . . . 1064
7.119 kernel/src/boot/cmdline.c File Reference . . . 1065
7.119.1 Function Documentation . . . 1066
7.119.1.1 cmdline_parse_ascii() . . . 1066
7.119.1.2 cmdline_parse_size() . . . 1067
7.119.1.3 parse_log_level() . . . 1068
7.119.1.4 sf_isspace() . . . 1069
7.119.1.5 sf_strlcpy() . . . 1070

7.119.1.6 sf_strncmp() . . . 1071
7.120 kernel/src/boot/elf_loader.c File Reference . . . 1071
7.120.1 Function Documentation . . . 1072
7.120.1.1 elf_apply_relocations() . . . 1072
7.120.1.2 elf_load_kernel() . . . 1074
7.120.1.3 elf_load_segments() . . . 1075
7.120.1.4 elf_memcpy() . . . 1076
7.120.1.5 elf_memset() . . . 1077
7.120.1.6 elf_validate_header() . . . 1078
7.121 kernel/src/boot/reloc_stub.c File Reference . . . 1078
7.121.1 Function Documentation . . . 1079
7.121.1.1 __attribute__() [1/2] . . . 1079
7.121.1.2 __attribute__() [2/2] . . . 1080
7.121.2 Variable Documentation . . . 1080
7.121.2.1 block_size . . . 1080
7.121.2.2 entry . . . 1080
7.121.2.3 page_rva . . . 1081
7.122 kernel/src/boot/uefi_loader.c File Reference . . . 1081
7.122.1 Macro Definition Documentation . . . 1082
7.122.1.1 DEBUG_CHECKPOINT_STALL_US . . . 1082
7.122.1.2 KERNEL_MAX_SIZE . . . 1082
7.122.1.3 RAW_LOG . . . 1082
7.122.1.4 UEFI_DEBUG_CHECKPOINTS . . . 1082
7.122.2 Typedef Documentation . . . 1082
7.122.2.1 KernelEntry . . . 1082
7.122.3 Function Documentation . . . 1083
7.122.3.1 debug_checkpoint() . . . 1083
7.122.3.2 efi_main() . . . 1083
7.122.3.3 guid_equals() . . . 1085
7.122.3.4 load_cfg_from_esp() . . . 1086
7.122.3.5 load_kernel_from_esp() . . . 1087
7.122.4 Variable Documentation . . . 1088
7.122.4.1 g_boot_info . . . 1088
7.122.4.2 kernel_elf_buffer . . . 1089
7.122.4.3 kernel_elf_size . . . 1089
7.123 kernel/src/capsule/capsule_birth.c File Reference . . . 1089
7.123.1 Typedef Documentation . . . 1091
7.123.1.1 vm_node_t . . . 1091
7.123.2 Function Documentation . . . 1091
7.123.2.1 capsule_birth_baby() . . . 1091
7.123.2.2 capsule_birth_mama() . . . 1093
7.123.2.3 capsule_birth_set_hooks() . . . 1094

7.123.2.4 capsule_run_experiment() . . . 1095
7.123.2.5 capsule_vm_find_by_name() . . . 1096
7.123.2.6 capsule_vm_find_by_name_nocase() . . . 1097
7.123.2.7 capsule_vm_force_reap() . . . 1098
7.123.2.8 capsule_vm_kill() . . . 1099
7.123.2.9 capsule_vm_kill_all_nonmama() . . . 1100
7.123.2.10 capsule_vm_registry_count() . . . 1100
7.123.2.11 capsule_vm_registry_get() . . . 1100
7.123.2.12 capsule_vm_registry_get_by_index() . . . 1101
7.123.2.13 capsule_vm_registry_init() . . . 1102
7.123.2.14 capsule_vm_registry_set_name() . . . 1103
7.123.2.15 capsule_vm_set_pending_reap() . . . 1103
7.123.2.16 capsule_vm_set_state() . . . 1104
7.123.2.17 contrib_capsule_refused() . . . 1105
7.123.2.18 dictionary_snapshot_acl_from_parent() . . . 1105
7.123.2.19 dispatch_init_forth() . . . 1106
7.123.2.20 vm_find_entry_ptr() . . . 1106
7.123.2.21 vm_name_copy() . . . 1106
7.123.2.22 vm_name_eq() . . . 1107
7.123.2.23 vm_name_eq_nocase() . . . 1107
7.123.2.24 vm_name_prefix_eq_nocase() . . . 1108
7.123.2.25 vm_registry_alloc() . . . 1108
7.123.2.26 vm_registry_live_count() . . . 1109
7.123.2.27 vm_to_lower() . . . 1109
7.123.3 Variable Documentation . . . 1109
7.123.3.1 vm_alloc_fn . . . 1109
7.123.3.2 vm_dict_hash_fn . . . 1109
7.123.3.3 vm_exec_fn . . . 1110
7.123.3.4 vm_registry_count . . . 1110
7.123.3.5 vm_registry_head . . . 1110
7.124 kernel/src/capsule/capsule_blocks.c File Reference . . . 1111
7.124.1 Function Documentation . . . 1111
7.124.1.1 capsule_block_header() . . . 1111
7.125 kernel/src/capsule/capsule_console.c File Reference . . . 1111
7.125.1 Function Documentation . . . 1112
7.125.1.1 capsule_console_birth() . . . 1112
7.125.2 Variable Documentation . . . 1113
7.125.2.1 CONSOLE_IDENTITY_SRC . . . 1113
7.126 kernel/src/capsule/capsule_find.c File Reference . . . 1114
7.126.1 Function Documentation . . . 1114
7.126.1.1 capsule_find_by_id() . . . 1114
7.126.1.2 capsule_find_by_name() . . . 1115

7.126.1.3 capsule_find_mama_init() . . . 1116
7.126.1.4 capsule_get_payload() . . . 1116
7.126.1.5 capsule_streq() . . . 1117
7.127 kernel/src/capsule/capsule_loader.c File Reference . . . 1117
7.127.1 Macro Definition Documentation . . . 1118
7.127.1.1 CAPSULE_BLOCK_LINES . . . 1118
7.127.1.2 CAPSULE_EXEC_LINE_MAX . . . 1119
7.127.1.3 CAPSULE_LINE_COLS . . . 1119
7.127.1.4 CAPSULE_MAX_DEFERRED . . . 1119
7.127.1.5 CAPSULE_RAM_OFFSET . . . 1119
7.127.1.6 LOADER_BLOCK_SIZE . . . 1119
7.127.2 Function Documentation . . . 1119
7.127.2.1 capsule_clear_blocks() . . . 1119
7.127.2.2 capsule_exec_init() . . . 1120
7.127.2.3 capsule_exec_payload() . . . 1122
7.127.2.4 capsule_load_blocks() . . . 1124
7.127.2.5 exec_block_with_retry() . . . 1125
7.127.2.6 is_block_header() . . . 1126
7.127.2.7 popcount16() . . . 1126
7.127.2.8 u32_to_dec() . . . 1126
7.127.2.9 write_ramdrive_block() . . . 1127
7.128 kernel/src/capsule/capsule_mint.c File Reference . . . 1127
7.128.1 Macro Definition Documentation . . . 1128
7.128.1.1 MINT_CERT_DEVBLOCK . . . 1128
7.128.1.2 MINT_CERT_DEVBLOCKS . . . 1129
7.128.1.3 MINT_IDENTITY_SRC_DEVBLOCKS . . . 1129
7.128.1.4 MINT_IDENTITY_SRC_OFFSET . . . 1129
7.128.1.5 MINT_METADATA_DEVBLOCKS . . . 1129
7.128.2 Function Documentation . . . 1129
7.128.2.1 blkio_ram_init_state() . . . 1129
7.128.2.2 blkio_ram_state_size() . . . 1130
7.128.2.3 blkio_ram_vtable() . . . 1131
7.128.2.4 capsule_mint_identity() . . . 1131
7.128.2.5 capsule_mint_identity_scratch() . . . 1132
7.128.2.6 read_devblock() . . . 1133
7.128.2.7 verify_mint() . . . 1134
7.128.2.8 write_devblock() . . . 1135
7.128.3 Variable Documentation . . . 1135
7.128.3.1 MINT_DEFAULT_PERSONALITY . . . 1135
7.128.3.2 MINT_RESTRICTED_PERSONALITY . . . 1136
7.129 kernel/src/capsule/capsule_run.c File Reference . . . 1136
7.129.1 Macro Definition Documentation . . . 1137

7.129.1.1 PARITY_EMIT . . . 1137
7.129.1.2 PARITY_HAVE_SINK . . . 1137
7.129.2 Function Documentation . . . 1138
7.129.2.1 capsule_parity_log_birth() . . . 1138
7.129.2.2 capsule_parity_log_birth_failed() . . . 1138
7.129.2.3 capsule_parity_log_kill() . . . 1139
7.129.2.4 capsule_parity_log_mama_init() . . . 1140
7.129.2.5 capsule_parity_log_run() . . . 1140
7.129.2.6 capsule_parity_set_output() . . . 1141
7.129.2.7 capsule_run_log_count() . . . 1142
7.129.2.8 capsule_run_log_get() . . . 1142
7.129.2.9 capsule_run_log_init() . . . 1142
7.129.2.10 capsule_run_log_record() . . . 1143
7.129.2.11 parity_put_hex64() . . . 1143
7.129.2.12 parity_put_u32() . . . 1144
7.129.2.13 parity_put_u64() . . . 1144
7.129.2.14 parity_put_uuid() . . . 1144
7.129.3 Variable Documentation . . . 1145
7.129.3.1 parity_putc . . . 1145
7.129.3.2 parity_puts . . . 1145
7.129.3.3 run_id_counter . . . 1145
7.129.3.4 run_log . . . 1145
7.129.3.5 run_log_count . . . 1145
7.129.3.6 run_log_head . . . 1145
7.130 kernel/src/capsule/capsule_runcap.c File Reference . . . 1146
7.130.1 Function Documentation . . . 1146
7.130.1.1 capsule_runcap_birth() . . . 1146
7.131 kernel/src/capsule/capsule_sig.c File Reference . . . 1147
7.131.1 Function Documentation . . . 1148
7.131.1.1 capsule_sig_result_str() . . . 1148
7.131.1.2 capsule_verify_signature() . . . 1149
7.131.1.3 find_and_cache_pubkey() . . . 1150
7.131.2 Variable Documentation . . . 1150
7.131.2.1 cached_pubkey . . . 1150
7.131.2.2 pubkey_state . . . 1150
7.132 kernel/src/capsule/capsule_validate.c File Reference . . . 1151
7.132.1 Function Documentation . . . 1151
7.132.1.1 capsule_validate() . . . 1151
7.132.1.2 capsule_validate_result_str() . . . 1152
7.132.2 Variable Documentation . . . 1153
7.132.2.1 validate_result_strings . . . 1153
7.133 kernel/src/capsule/capsule_vm_hooks.c File Reference . . . 1153

7.133.1 Function Documentation . . . 1154
7.133.1.1 capsule_dict_hash_hook() . . . 1154
7.133.1.2 capsule_exec_hook() . . . 1154
7.133.1.3 capsule_vm_alloc_hook() . . . 1155
7.133.1.4 capsule_vm_hooks_register() . . . 1157
7.134 kernel/src/capsule/capsule_vm_physics.c File Reference . . . 1158
7.134.1 Macro Definition Documentation . . . 1159
7.134.1.1 VM_FLEET_WINDOW_DEPTH . . . 1159
7.134.1.2 VM_PHYSICS_EPSILON_Q48 . . . 1160
7.134.1.3 VM_PHYSICS_MIN_INFORMATIVE_SAMPLES . . . 1160
7.134.2 Typedef Documentation . . . 1160
7.134.2.1 vm_physics_node_t . . . 1160
7.134.3 Function Documentation . . . 1160
7.134.3.1 console_put_u64() . . . 1160
7.134.3.2 l8_regime_modulation_q16() . . . 1161
7.134.3.3 vm_physics_conserved() . . . 1161
7.134.3.4 vm_physics_find() . . . 1162
7.134.3.5 vm_physics_find_root_id() . . . 1163
7.134.3.6 vm_physics_fleet_heat_sum() . . . 1164
7.134.3.7 vm_physics_heartbeat_tick() . . . 1164
7.134.3.8 vm_physics_heat_of() . . . 1165
7.134.3.9 vm_physics_init() . . . 1166
7.134.3.10 vm_physics_retire() . . . 1167
7.134.3.11 vm_physics_status() . . . 1168
7.134.3.12 vm_physics_tick() . . . 1168
7.134.3.13 vm_physics_touch() . . . 1169
7.134.3.14 vm_physics_transfer() . . . 1170
7.134.4 Variable Documentation . . . 1170
7.134.4.1 fleet_heartbeat_tick_count . . . 1170
7.134.4.2 fleet_last_capacity_tick . . . 1171
7.134.4.3 fleet_transfer_slope_q48 . . . 1171
7.134.4.4 fleet_window . . . 1171
7.134.4.5 slope_fit_quality_q48 . . . 1171
7.134.4.6 vm_physics_head . . . 1171
7.135 kernel/src/capsule/capsule_vm_switch_signal.c File Reference . . . 1171
7.136 kernel/src/capsule/capsule_wirebind.c File Reference . . . 1171
7.136.1 Macro Definition Documentation . . . 1173
7.136.1.1 WIREBIND_CERT_MAX_DEVBLOCKS . . . 1173
7.136.1.2 WIREBIND_MAX_LIVE . . . 1173
7.136.1.3 WIREBIND_OVERFLOW_CLAIM_DEVBLOCKS . . . 1173
7.136.1.4 WIREBIND_OVERFLOW_FREE_THRESHOLD_PCT . . . 1173
7.136.2 Function Documentation . . . 1173

7.136.2.1 capsule_wirebind_attached_username() . . . 1173
7.136.2.2 capsule_wirebind_eject() . . . 1174
7.136.2.3 capsule_wirebind_overflow_idle_check() . . . 1174
7.136.2.4 capsule_wirebind_reap_idle_check() . . . 1175
7.136.2.5 capsule_wirebind_try_attach() . . . 1176
7.136.2.6 capsule_wirebind_unclean_detach() . . . 1177
7.136.2.7 capsule_wirebind_verify_cert() . . . 1178
7.136.2.8 read_devblock() . . . 1179
7.136.2.9 wirebind_live_alloc() . . . 1180
7.136.2.10 wirebind_live_find_by_dev() . . . 1180
7.136.2.11 wirebind_live_remove() . . . 1181
7.136.2.12 wirebind_resolve_attached() . . . 1181
7.136.2.13 wirebind_teardown_console() . . . 1182
7.136.3 Variable Documentation . . . 1182
7.136.3.1 g_wirebind_attached_dev . . . 1182
7.136.3.2 g_wirebind_attached_username . . . 1182
7.136.3.3 g_wirebind_attached_valid . . . 1183
7.136.3.4 g_wirebind_attached_vm_id . . . 1183
7.136.3.5 g_wirebind_live . . . 1183
7.137 kernel/src/capsule/capsule_zuse_boot.c File Reference . . . 1183
7.137.1 Function Documentation . . . 1184
7.137.1.1 capsule_zuse_boot_attached_dev() . . . 1184
7.137.1.2 capsule_zuse_boot_check() . . . 1184
7.137.1.3 capsule_zuse_boot_check_key() . . . 1185
7.137.1.4 capsule_zuse_boot_genesis_mint() . . . 1186
7.137.1.5 capsule_zuse_boot_load_root_pubkey() . . . 1187
7.137.1.6 capsule_zuse_boot_logout() . . . 1187
7.137.1.7 capsule_zuse_boot_root_pubkey() . . . 1188
7.137.1.8 capsule_zuse_boot_try_attach() . . . 1188
7.137.1.9 genesis_marker_read() . . . 1189
7.137.1.10 install_and_activate() . . . 1190
7.137.1.11 read_devblock() . . . 1191
7.137.2 Variable Documentation . . . 1192
7.137.2.1 g_zuse_attached_dev . . . 1192
7.137.2.2 g_zuse_attached_this_device . . . 1192
7.138 kernel/src/capsule/mama_forth_words.c File Reference . . . 1192
7.138.1 Detailed Description . . . 1192
7.139 kernel/src/capsule/vm_identity.c File Reference . . . 1193
7.139.1 Function Documentation . . . 1193
7.139.1.1 vm_identity_from_cert() . . . 1193
7.139.1.2 vm_identity_has_cap() . . . 1194
7.140 kernel/src/capsule/vm_uuid.c File Reference . . . 1195

7.141 kernel/src/crypto/ed25519.c File Reference . . . 1195
7.141.1 Function Documentation . . . 1196
7.141.1.1 base_point() . . . 1196
7.141.1.2 clamp_scalar() . . . 1197
7.141.1.3 ed25519_keygen() . . . 1197
7.141.1.4 ed25519_sign() . . . 1198
7.141.1.5 ed25519_verify() . . . 1199
7.141.1.6 expand_seed() . . . 1200
7.141.1.7 fe25519_pow() . . . 1200
7.141.1.8 point_add() . . . 1201
7.141.1.9 point_compress() . . . 1202
7.141.1.10 point_decompress() . . . 1202
7.141.1.11 point_identity() . . . 1203
7.141.1.12 scalar_mult() . . . 1204
7.141.2 Variable Documentation . . . 1205
7.141.2.1 BX_BYTES . . . 1205
7.141.2.2 BY_BYTES . . . 1205
7.141.2.3 D_BYTES . . . 1205
7.141.2.4 POW_2252M2_EXP . . . 1206
7.141.2.5 SQRTM1_BYTES . . . 1206
7.142 kernel/src/crypto/fe25519.c File Reference . . . 1206
7.142.1 Macro Definition Documentation . . . 1207
7.142.1.1 MASK51 . . . 1207
7.142.1.2 WBITS . . . 1207
7.142.2 Typedef Documentation . . . 1207
7.142.2.1 u128 . . . 1207
7.142.3 Function Documentation . . . 1208
7.142.3.1 fe25519_0() . . . 1208
7.142.3.2 fe25519_1() . . . 1208
7.142.3.3 fe25519_add() . . . 1209
7.142.3.4 fe25519_carry() . . . 1209
7.142.3.5 fe25519_copy() . . . 1210
7.142.3.6 fe25519_eq() . . . 1210
7.142.3.7 fe25519_invert() . . . 1210
7.142.3.8 fe25519_mul() . . . 1211
7.142.3.9 fe25519_mul_small() . . . 1211
7.142.3.10 fe25519_neg() . . . 1212
7.142.3.11 fe25519_pack() . . . 1212
7.142.3.12 fe25519_parity() . . . 1213
7.142.3.13 fe25519_sq() . . . 1214
7.142.3.14 fe25519_sub() . . . 1214
7.142.3.15 fe25519_unpack() . . . 1215

7.143 kernel/src/crypto/scalar25519.c File Reference . . . 1215
7.143.1 Function Documentation . . . 1216
7.143.1.1 cmp32() . . . 1216
7.143.1.2 getbit() . . . 1216
7.143.1.3 mul256() . . . 1217
7.143.1.4 scalar_lt_L() . . . 1217
7.143.1.5 scalar_muladd() . . . 1218
7.143.1.6 scalar_reduce512() . . . 1218
7.143.1.7 sub_L() . . . 1219
7.143.2 Variable Documentation . . . 1220
7.143.2.1 L_BYTES . . . 1220
7.144 kernel/src/crypto/sha512.c File Reference . . . 1220
7.144.1 Function Documentation . . . 1221
7.144.1.1 rotr64() . . . 1221
7.144.1.2 sha512() . . . 1221
7.144.1.3 sha512_block() . . . 1222
7.144.1.4 sha512_final() . . . 1222
7.144.1.5 sha512_init() . . . 1223
7.144.1.6 sha512_update() . . . 1223
7.144.2 Variable Documentation . . . 1224
7.144.2.1 K . . . 1224
7.145 kernel/src/crypto/x509_ed25519.c File Reference . . . 1224
7.145.1 Function Documentation . . . 1225
7.145.1.1 der_next() . . . 1225
7.145.1.2 der_put_ed25519_algid() . . . 1225
7.145.1.3 der_put_tlv() . . . 1226
7.145.1.4 der_put_uint_integer() . . . 1226
7.145.1.5 x509_build_user_cert() . . . 1227
7.145.1.6 x509_extract_ed25519_pubkey() . . . 1228
7.145.1.7 x509_extract_serial() . . . 1229
7.145.1.8 x509_verify_signature() . . . 1230
7.146 kernel/src/doe_log.c File Reference . . . 1231
7.146.1 Macro Definition Documentation . . . 1232
7.146.1.1 DOE_BUF_SIZE . . . 1232
7.146.1.2 DOE_PREFIX . . . 1232
7.146.2 Function Documentation . . . 1232
7.146.2.1 doe_log_heat_by_name() . . . 1232
7.146.2.2 doe_log_tick_row() . . . 1233
7.146.2.3 doe_word_hb_off() . . . 1234
7.146.2.4 doe_word_hb_on() . . . 1234
7.146.2.5 register_doe_log_words() . . . 1235
7.146.3 Variable Documentation . . . 1235

7.146.3.1 doe_header . . . 1235
7.146.3.2 g_doe_log_enabled . . . 1236
7.147 kernel/src/gpt.c File Reference . . . 1237
7.147.1 Macro Definition Documentation . . . 1238
7.147.1.1 GPT_ENT_SECTORS . . . 1238
7.147.1.2 GPT_ENTRIES_MAX . . . 1239
7.147.1.3 GPT_ESIZE . . . 1239
7.147.1.4 GPT_NENT . . . 1239
7.147.1.5 GPT_SECTOR . . . 1239
7.147.2 Function Documentation . . . 1239
7.147.2.1 gpt_alternate_lba() . . . 1239
7.147.2.2 gpt_crc32() . . . 1239
7.147.2.3 gpt_disk_guid() . . . 1240
7.147.2.4 gpt_find() . . . 1240
7.147.2.5 gpt_grow_last() . . . 1241
7.147.2.6 gpt_new_guid() . . . 1242
7.147.2.7 gpt_part_open() . . . 1242
7.147.2.8 gpt_read() . . . 1243
7.147.2.9 gpt_write_one() . . . 1244
7.147.2.10 le32() . . . 1244
7.147.2.11 le64() . . . 1245
7.147.2.12 part_close() . . . 1245
7.147.2.13 part_flush() . . . 1245
7.147.2.14 part_info() . . . 1245
7.147.2.15 part_open() . . . 1246
7.147.2.16 part_read() . . . 1246
7.147.2.17 part_write() . . . 1246
7.147.2.18 put_le32() . . . 1247
7.147.2.19 put_le64() . . . 1247
7.147.2.20 put_sectors() . . . 1248
7.147.3 Variable Documentation . . . 1248
7.147.3.1 g_gpt_alternate_lba . . . 1248
7.147.3.2 g_gpt_disk_guid . . . 1248
7.147.3.3 g_gpt_entries . . . 1249
7.147.3.4 g_gpt_hdr_block . . . 1249
7.147.3.5 g_gpt_new_entries . . . 1249
7.147.3.6 g_gpt_rmw . . . 1249
7.147.3.7 g_part_vt . . . 1249
7.147.3.8 gpt_type_starship_artemis . . . 1249
7.147.3.9 gpt_type_starship_identity . . . 1249
7.148 kernel/src/hal/console.c File Reference . . . 1250
7.148.1 Macro Definition Documentation . . . 1252

7.148.1.1 CONSOLE_VM_NAME_BUF . . . 1252
7.148.1.2 PL011_FR . . . 1252
7.148.1.3 PL011_FR_RXFE . . . 1252
7.148.1.4 PL011_FR_TXFF . . . 1252
7.148.1.5 SERIAL_TX_MAX_SPIN . . . 1252
7.148.1.6 UART16550_LSR . . . 1252
7.148.1.7 UART16550_LSR_DR . . . 1252
7.148.1.8 UART16550_LSR_THRE . . . 1253
7.148.2 Enumeration Type Documentation . . . 1253
7.148.2.1 anonymous enum . . . 1253
7.148.3 Function Documentation . . . 1253
7.148.3.1 console_attach_serial() . . . 1253
7.148.3.2 console_cancel_deferred_line_start() . . . 1254
7.148.3.3 console_describe_serial() . . . 1255
7.148.3.4 console_ensure_line_start() . . . 1255
7.148.3.5 console_fb_clear() . . . 1256
7.148.3.6 console_fb_draw_cursor() . . . 1257
7.148.3.7 console_fb_enable_ttf() . . . 1257
7.148.3.8 console_fb_erase_cursor() . . . 1258
7.148.3.9 console_fb_init() . . . 1259
7.148.3.10 console_fb_mirror() . . . 1260
7.148.3.11 console_fb_scroll_back() . . . 1260
7.148.3.12 console_fb_scroll_fwd() . . . 1261
7.148.3.13 console_fb_toggle_graphics() . . . 1261
7.148.3.14 console_get_vm_name() . . . 1262
7.148.3.15 console_getc() . . . 1262
7.148.3.16 console_init() . . . 1263
7.148.3.17 console_poll() . . . 1264
7.148.3.18 console_println() . . . 1264
7.148.3.19 console_putc() . . . 1265
7.148.3.20 console_putc_inner() . . . 1266
7.148.3.21 console_puts() . . . 1268
7.148.3.22 console_save_vm_name() . . . 1270
7.148.3.23 console_set_user_prefix_provider() . . . 1271
7.148.3.24 console_set_vm_name() . . . 1271
7.148.3.25 console_tx_count() . . . 1272
7.148.3.26 emit_prefix() . . . 1272
7.148.3.27 fb_mirrored() . . . 1273
7.148.3.28 raw_putc() . . . 1274
7.148.3.29 serial_putc() . . . 1276
7.148.3.30 serial_reg_read() . . . 1278
7.148.3.31 serial_reg_write() . . . 1279

7.148.3.32 serial_rx_ready() . . . 1280
7.148.3.33 serial_tx_ready() . . . 1281
7.148.4 Variable Documentation . . . 1282
7.148.4.1 g_active_vm_name . . . 1282
7.148.4.2 g_active_vm_name_buf . . . 1282
7.148.4.3 g_console_tx_count . . . 1283
7.148.4.4 g_fb_mirror . . . 1283
7.148.4.5 g_line_start . . . 1283
7.148.4.6 g_pending_line_close . . . 1283
7.148.4.7 g_serial_base . . . 1283
7.148.4.8 g_serial_from_dt . . . 1283
7.148.4.9 g_serial_kind . . . 1283
7.148.4.10 g_serial_shift . . . 1284
7.148.4.11 g_serial_width . . . 1284
7.148.4.12 g_user_prefix_fn . . . 1284
7.149 kernel/src/hal/fdt.c File Reference . . . 1284
7.149.1 Macro Definition Documentation . . . 1286
7.149.1.1 FDT_BEGIN_NODE . . . 1286
7.149.1.2 FDT_END . . . 1286
7.149.1.3 FDT_END_NODE . . . 1286
7.149.1.4 FDT_MAGIC . . . 1286
7.149.1.5 FDT_NOP . . . 1286
7.149.1.6 FDT_PATH_MAX . . . 1286
7.149.1.7 FDT_PROP . . . 1286
7.149.2 Function Documentation . . . 1287
7.149.2.1 be32() . . . 1287
7.149.2.2 compat_list_contains() . . . 1287
7.149.2.3 fdt_complete() . . . 1288
7.149.2.4 fdt_find_child() . . . 1289
7.149.2.5 fdt_find_node_by_compatible() . . . 1290
7.149.2.6 fdt_find_node_by_device_type() . . . 1291
7.149.2.7 fdt_find_node_by_name() . . . 1292
7.149.2.8 fdt_find_prop() . . . 1293
7.149.2.9 fdt_find_prop_in_node() . . . 1294
7.149.2.10 fdt_next_child_node() . . . 1295
7.149.2.11 fdt_prop_u32() . . . 1297
7.149.2.12 fdt_reg_address() . . . 1298
7.149.2.13 fdt_root_node() . . . 1299
7.149.2.14 fdt_valid() . . . 1300
7.149.2.15 node_cells() . . . 1302
7.149.2.16 node_enabled() . . . 1302
7.149.2.17 node_name_eq() . . . 1303

7.149.2.18 path_to() . . . 1303
7.149.2.19 read_cells_at() . . . 1304
7.149.2.20 skip_node_body() . . . 1304
7.149.2.21 str_eq() . . . 1305
7.150 kernel/src/hal/fdt_uart.c File Reference . . . 1306
7.150.1 Macro Definition Documentation . . . 1307
7.150.1.1 PATH_DEPTH . . . 1307
7.150.2 Function Documentation . . . 1307
7.150.2.1 be32() . . . 1307
7.150.2.2 cell_prop() . . . 1308
7.150.2.3 cells() . . . 1308
7.150.2.4 compatible_has() . . . 1309
7.150.2.5 fdt_console_uart() . . . 1309
7.151 kernel/src/hal/font_8x16.c File Reference . . . 1310
7.151.1 Function Documentation . . . 1311
7.151.1.1 font_8x16_glyph() . . . 1311
7.151.2 Variable Documentation . . . 1311
7.151.2.1 font_8x16_data . . . 1311
7.152 kernel/src/hal/framebuffer.c File Reference . . . 1312
7.152.1 Function Documentation . . . 1313
7.152.1.1 __attribute__() . . . 1313
7.152.1.2 fb_cell_h() . . . 1313
7.152.1.3 fb_cell_w() . . . 1314
7.152.1.4 fb_draw_glyph() . . . 1314
7.152.1.5 fb_draw_orientation_test() . . . 1315
7.152.1.6 fb_fill_rect() . . . 1316
7.152.1.7 fb_height() . . . 1316
7.152.1.8 fb_init() . . . 1317
7.152.1.9 fb_is_available() . . . 1318
7.152.1.10 fb_put_pixel() . . . 1318
7.152.1.11 fb_scroll_rect() . . . 1319
7.152.1.12 fb_scroll_rows() . . . 1320
7.152.1.13 fb_width() . . . 1321
7.152.1.14 font_8x16_glyph() . . . 1322
7.152.1.15 pack_pixel() . . . 1323
7.152.2 Variable Documentation . . . 1323
7.152.2.1 g_fb . . . 1323
7.153 kernel/src/hal/hal.c File Reference . . . 1323
7.154 kernel/src/hal/host_services.c File Reference . . . 1323
7.155 kernel/src/hal/kbd_input.c File Reference . . . 1323
7.155.1 Macro Definition Documentation . . . 1324
7.155.1.1 SK_KBD_TABLE_SIZE . . . 1324

7.155.1.2 SK_KEY_102ND . . . 1325
7.155.1.3 SK_KEY_BACKSPACE . . . 1325
7.155.1.4 SK_KEY_CAPSLOCK . . . 1325
7.155.1.5 SK_KEY_ENTER . . . 1325
7.155.1.6 SK_KEY_KPENTER . . . 1325
7.155.1.7 SK_KEY_KPSLASH . . . 1325
7.155.1.8 SK_KEY_LEFTALT . . . 1325
7.155.1.9 SK_KEY_LEFTSHIFT . . . 1325
7.155.1.10 SK_KEY_NUMLOCK . . . 1325
7.155.1.11 SK_KEY_RIGHTALT . . . 1325
7.155.1.12 SK_KEY_RIGHTSHIFT . . . 1326
7.155.1.13 SK_KEY_TAB . . . 1326
7.155.2 Function Documentation . . . 1326
7.155.2.1 sk_input_drop() . . . 1326
7.155.2.2 sk_input_getc() . . . 1327
7.155.2.3 sk_kbd_getc() . . . 1329
7.155.2.4 sk_kbd_keypad() . . . 1330
7.155.2.5 sk_kbd_show_locks() . . . 1330
7.155.3 Variable Documentation . . . 1331
7.155.3.1 g_kbd_alt_down . . . 1331
7.155.3.2 g_kbd_caps_lock . . . 1331
7.155.3.3 g_kbd_num_lock . . . 1331
7.155.3.4 g_kbd_poll_tick . . . 1331
7.155.3.5 g_kbd_shift_down . . . 1331
7.155.3.6 sk_kbd_shifted . . . 1331
7.155.3.7 sk_kbd_unshifted . . . 1332
7.156 kernel/src/hal/memory.c File Reference . . . 1332
7.156.1 Function Documentation . . . 1333
7.156.1.1 hal_mem_alloc() . . . 1333
7.156.1.2 hal_mem_alloc_aligned() . . . 1333
7.156.1.3 hal_mem_free() . . . 1334
7.157 kernel/src/hal/ttf.c File Reference . . . 1334
7.157.1 Macro Definition Documentation . . . 1336
7.157.1.1 TAG . . . 1336
7.157.1.2 TTF_COMP_ARGS_ARE_WORDS . . . 1336
7.157.1.3 TTF_COMP_ARGS_ARE_XY_VALUES . . . 1337
7.157.1.4 TTF_COMP_MORE_COMPONENTS . . . 1337
7.157.1.5 TTF_COMP_WE_HAVE_2X2 . . . 1337
7.157.1.6 TTF_COMP_WE_HAVE_A_SCALE . . . 1337
7.157.1.7 TTF_COMP_WE_HAVE_XY_SCALE . . . 1337
7.157.1.8 TTF_F2DOT14_ONE . . . 1337
7.157.1.9 TTF_FLAG_ON_CURVE . . . 1337

7.157.1.10 TTF_FLAG_REPEAT . . . 1337
7.157.1.11 TTF_FLAG_X_SAME_OR_POS . . . 1337
7.157.1.12 TTF_FLAG_X_SHORT . . . 1337
7.157.1.13 TTF_FLAG_Y_SAME_OR_POS . . . 1338
7.157.1.14 TTF_FLAG_Y_SHORT . . . 1338
7.157.1.15 TTF_MAX_SIMPLE_POINTS . . . 1338
7.157.1.16 TTF_QUAD_SEGMENTS . . . 1338
7.157.1.17 TTF_RASTER_FLAT_MAX_CONTOURS . . . 1338
7.157.1.18 TTF_RASTER_FLAT_MAX_POINTS . . . 1338
7.157.1.19 TTF_RASTER_RAW_MAX_CONTOURS . . . 1338
7.157.1.20 TTF_RASTER_RAW_MAX_POINTS . . . 1338
7.157.2 Function Documentation . . . 1339
7.157.2.1 decode_composite_glyph() . . . 1339
7.157.2.2 decode_glyph_r() . . . 1340
7.157.2.3 decode_simple_glyph() . . . 1341
7.157.2.4 find_table() . . . 1341
7.157.2.5 fits() . . . 1342
7.157.2.6 flat_push() . . . 1343
7.157.2.7 flatten_contour() . . . 1343
7.157.2.8 flatten_quad() . . . 1344
7.157.2.9 in_bounds() . . . 1345
7.157.2.10 q48_from_i32() . . . 1346
7.157.2.11 q48_mid() . . . 1346
7.157.2.12 q48_smul() . . . 1346
7.157.2.13 raw_midpoint() . . . 1347
7.157.2.14 rd_i16() . . . 1347
7.157.2.15 rd_i8() . . . 1348
7.157.2.16 rd_u16() . . . 1348
7.157.2.17 rd_u32() . . . 1349
7.157.2.18 rd_u8() . . . 1349
7.157.2.19 to_pixel_space() . . . 1349
7.157.2.20 ttf_codepoint_to_glyph() . . . 1350
7.157.2.21 ttf_glyph_advance_width() . . . 1350
7.157.2.22 ttf_glyph_header() . . . 1351
7.157.2.23 ttf_glyph_outline() . . . 1352
7.157.2.24 ttf_parse() . . . 1353
7.157.2.25 ttf_raster_cache_get() . . . 1354
7.157.2.26 ttf_raster_cache_init() . . . 1355
7.157.2.27 ttf_rasterize_glyph() . . . 1355
7.158 kernel/src/hal/vt100.c File Reference . . . 1356
7.158.1 Macro Definition Documentation . . . 1358
7.158.1.1 UNDERLINE_ROW . . . 1358

7.158.1.2 VT100_BOOT_SCROLLBACK_MAX_LINES . . . 1358
7.158.1.3 VT100_MAX_PARAMS . . . 1359
7.158.1.4 VT100_SCROLLBACK_MAX_LINES . . . 1359
7.158.1.5 VT100_TTF_CACHE_SLOTS . . . 1359
7.158.1.6 VT100_TTF_CELL_H_PX . . . 1359
7.158.1.7 VT100_TTF_CELL_W_PX . . . 1359
7.158.1.8 VT100_TTF_FONT_CAPSULE . . . 1359
7.158.1.9 VT100_TTF_SIZE_PX . . . 1359
7.158.2 Enumeration Type Documentation . . . 1359
7.158.2.1 ParseState . . . 1359
7.158.2.2 vt_glyph_mode_t . . . 1360
7.158.3 Function Documentation . . . 1360
7.158.3.1 advance_cursor() . . . 1360
7.158.3.2 ansi256_to_rgb() . . . 1361
7.158.3.3 apply_sgr() . . . 1362
7.158.3.4 cell_h() . . . 1364
7.158.3.5 cell_w() . . . 1364
7.158.3.6 dispatch_csi() . . . 1365
7.158.3.7 draw_cursor_glyph() . . . 1366
7.158.3.8 eff_bg() . . . 1368
7.158.3.9 eff_fg() . . . 1369
7.158.3.10 erase_display() . . . 1370
7.158.3.11 erase_line() . . . 1371
7.158.3.12 erase_line_range() . . . 1372
7.158.3.13 param() . . . 1373
7.158.3.14 put_char() . . . 1374
7.158.3.15 px_of() . . . 1375
7.158.3.16 py_of() . . . 1376
7.158.3.17 reset_attrs() . . . 1377
7.158.3.18 reset_parser() . . . 1378
7.158.3.19 scroll_up() . . . 1378
7.158.3.20 scrollback_line_at() . . . 1379
7.158.3.21 scrollback_redraw() . . . 1380
7.158.3.22 set_cursor() . . . 1381
7.158.3.23 ttf_draw_glyph_cell() . . . 1381
7.158.3.24 vt100_cols() . . . 1382
7.158.3.25 vt100_draw_cursor() . . . 1383
7.158.3.26 vt100_enable_ttf() . . . 1383
7.158.3.27 vt100_erase_cursor() . . . 1384
7.158.3.28 vt100_init() . . . 1385
7.158.3.29 vt100_putc() . . . 1386
7.158.3.30 vt100_puts() . . . 1388

7.158.3.31 vt100_reset() . . . 1389
7.158.3.32 vt100_rows() . . . 1390
7.158.3.33 vt100_scroll_back() . . . 1390
7.158.3.34 vt100_scroll_fwd() . . . 1391
7.158.3.35 vt100_toggle_graphics() . . . 1392
7.158.4 Variable Documentation . . . 1392
7.158.4.1 g_boot_line_stride . . . 1392
7.158.4.2 g_boot_ring . . . 1393
7.158.4.3 g_boot_ring_cap . . . 1393
7.158.4.4 g_boot_ring_count . . . 1393
7.158.4.5 g_boot_ring_head . . . 1393
7.158.4.6 g_boot_rows . . . 1393
7.158.4.7 g_boot_shadow . . . 1393
7.158.4.8 g_glyph_mode . . . 1393
7.158.4.9 g_line_stride . . . 1394
7.158.4.10 g_ring . . . 1394
7.158.4.11 g_ring_cap . . . 1394
7.158.4.12 g_ring_count . . . 1394
7.158.4.13 g_ring_head . . . 1394
7.158.4.14 g_scroll_offset . . . 1394
7.158.4.15 g_shadow . . . 1394
7.158.4.16 g_terminal_visible . . . 1395
7.158.4.17 g_ttf_cache . . . 1395
7.158.4.18 g_ttf_font . . . 1395
7.158.4.19 g_ttf_ready . . . 1395
7.158.4.20 g_vt . . . 1395
7.159 kernel/src/hash/xxhash64.c File Reference . . . 1395
7.159.1 Macro Definition Documentation . . . 1396
7.159.1.1 XXHASH64_PRIME1 . . . 1396
7.159.1.2 XXHASH64_PRIME2 . . . 1396
7.159.1.3 XXHASH64_PRIME3 . . . 1397
7.159.1.4 XXHASH64_PRIME4 . . . 1397
7.159.1.5 XXHASH64_PRIME5 . . . 1397
7.159.2 Function Documentation . . . 1397
7.159.2.1 xxh64_avalanche() . . . 1397
7.159.2.2 xxh64_merge_round() . . . 1397
7.159.2.3 xxh64_read32() . . . 1398
7.159.2.4 xxh64_read64() . . . 1398
7.159.2.5 xxh64_rotl() . . . 1399
7.159.2.6 xxh64_round() . . . 1399
7.159.2.7 xxhash64() . . . 1399
7.159.2.8 xxhash64_digest() . . . 1400

7.159.2.9 xxhash64_reset() . . . 1401
7.159.2.10 xxhash64_update() . . . 1402
7.160 kernel/src/heartbeat.c File Reference . . . 1403
7.160.1 Macro Definition Documentation . . . 1404
7.160.1.1 HEARTBEAT_BASE_PERIOD_NS . . . 1404
7.160.1.2 HEARTBEAT_HAS_VARIANCE . . . 1404
7.160.2 Function Documentation . . . 1405
7.160.2.1 heartbeat_init() . . . 1405
7.160.2.2 heartbeat_next_period_ns() . . . 1405
7.160.2.3 heartbeat_service() . . . 1406
7.160.2.4 heartbeat_set_adaptive_period_ns() . . . 1406
7.160.2.5 heartbeat_state() . . . 1407
7.160.2.6 heartbeat_tick() . . . 1408
7.160.2.7 heartbeat_ticks() . . . 1409
7.160.2.8 heartbeat_trust() . . . 1409
7.160.3 Variable Documentation . . . 1410
7.160.3.1 g_adaptive_period_ns . . . 1410
7.160.3.2 g_heartbeat . . . 1410
7.160.3.3 g_pending_counter . . . 1410
7.160.3.4 g_pending_valid . . . 1410
7.161 kernel/src/homeblocks_sig.c File Reference . . . 1410
7.161.1 Function Documentation . . . 1411
7.161.1.1 homeblocks_sig_check() . . . 1411
7.161.1.2 homeblocks_sig_compute_crc() . . . 1412
7.162 kernel/src/kernel_main.c File Reference . . . 1412
7.162.1 Function Documentation . . . 1413
7.162.1.1 is_ram_type() . . . 1413
7.162.1.2 itoa_simple() . . . 1414
7.162.1.3 kernel_main() . . . 1415
7.162.1.4 kernel_main_deep() . . . 1418
7.162.1.5 print_banner() . . . 1421
7.162.1.6 print_boot_info() . . . 1422
7.162.1.7 print_heap_stats() . . . 1423
7.162.1.8 print_pmm_stats() . . . 1424
7.162.1.9 print_uint() . . . 1425
7.162.2 Variable Documentation . . . 1426
7.162.2.1 g_sk_runtime_services . . . 1426
7.163 kernel/src/log_region.c File Reference . . . 1427
7.163.1 Function Documentation . . . 1427
7.163.1.1 log_region_append() . . . 1427
7.163.1.2 log_region_ctrl_compute_crc() . . . 1428
7.163.1.3 log_region_ctrl_init_fresh() . . . 1429

7.163.1.4 log_region_ctrl_read() . . . 1430
7.163.1.5 log_region_ctrl_write() . . . 1430
7.163.1.6 log_region_read_recent() . . . 1431
7.163.1.7 log_region_read_slot() . . . 1431
7.163.1.8 log_region_slot_devblock_from_top() . . . 1432
7.163.1.9 log_region_write_slot() . . . 1433
7.164 kernel/src/math/q48_16.c File Reference . . . 1433
7.164.1 Function Documentation . . . 1434
7.164.1.1 q48_cos_approx() . . . 1434
7.164.1.2 q48_div() . . . 1436
7.164.1.3 q48_exp_approx() . . . 1436
7.164.1.4 q48_log_approx() . . . 1437
7.164.1.5 q48_mul() . . . 1439
7.164.1.6 q48_reduce_angle() . . . 1439
7.164.1.7 q48_sin_approx() . . . 1439
7.164.1.8 q48_sqrt_approx() . . . 1441
7.165 kernel/src/memory/kmalloc.c File Reference . . . 1441
7.165.1 Macro Definition Documentation . . . 1443
7.165.1.1 KMALLOC_DEFAULT_HEAP_SIZE . . . 1443
7.165.1.2 KMALLOC_MAGIC . . . 1443
7.165.1.3 KMALLOC_MIN_ALIGN . . . 1443
7.165.1.4 KMALLOC_MIN_PAGES . . . 1444
7.165.2 Typedef Documentation . . . 1444
7.165.2.1 heap_block_t . . . 1444
7.165.3 Function Documentation . . . 1444
7.165.3.1 account_allocation() . . . 1444
7.165.3.2 account_free() . . . 1445
7.165.3.3 align_up_size() . . . 1445
7.165.3.4 allocate_from_block() . . . 1446
7.165.3.5 can_split() . . . 1447
7.165.3.6 coalesce_neighbors() . . . 1448
7.165.3.7 kfree() . . . 1449
7.165.3.8 kmalloc() . . . 1449
7.165.3.9 kmalloc_aligned() . . . 1450
7.165.3.10 kmalloc_debug_census() . . . 1451
7.165.3.11 kmalloc_debug_census_bytes() . . . 1451
7.165.3.12 kmalloc_get_stats() . . . 1452
7.165.3.13 kmalloc_heap_base_addr() . . . 1452
7.165.3.14 kmalloc_heap_end_addr() . . . 1452
7.165.3.15 kmalloc_init() . . . 1453
7.165.3.16 kmalloc_is_initialized() . . . 1453
7.165.3.17 min_splittable_payload() . . . 1454

7.165.3.18 normalize_alignment() . . . 1454
7.165.3.19 pointer_within_heap() . . . 1455
7.165.3.20 update_free_bytes() . . . 1456
7.165.4 Variable Documentation . . . 1456
7.165.4.1 heap_base . . . 1456
7.165.4.2 heap_head . . . 1456
7.165.4.3 heap_initialized . . . 1457
7.165.4.4 heap_size . . . 1457
7.165.4.5 heap_stats . . . 1457
7.166 kernel/src/memory/pmm.c File Reference . . . 1457
7.166.1 Macro Definition Documentation . . . 1458
7.166.1.1 PMM_BITMAP_BITS . . . 1458
7.166.1.2 PMM_MAX_BITMAP_BYTES . . . 1458
7.166.2 Function Documentation . . . 1459
7.166.2.1 bitmap_clear() . . . 1459
7.166.2.2 bitmap_mark_range() . . . 1459
7.166.2.3 bitmap_set() . . . 1460
7.166.2.4 bitmap_test() . . . 1460
7.166.2.5 find_contiguous_free() . . . 1460
7.166.2.6 is_ram_type() . . . 1461
7.166.2.7 is_usable_memory() . . . 1462
7.166.2.8 pmm_alloc_contiguous() . . . 1462
7.166.2.9 pmm_alloc_page() . . . 1463
7.166.2.10 pmm_bytes_to_pages() . . . 1463
7.166.2.11 pmm_free_contiguous() . . . 1464
7.166.2.12 pmm_free_page() . . . 1464
7.166.2.13 pmm_get_stats() . . . 1465
7.166.2.14 pmm_init() . . . 1465
7.166.2.15 pmm_is_initialized() . . . 1466
7.166.2.16 pmm_memset() . . . 1466
7.166.2.17 recompute_free_pages() . . . 1467
7.166.3 Variable Documentation . . . 1467
7.166.3.1 __kernel_end . . . 1467
7.166.3.2 __kernel_start . . . 1467
7.166.3.3 pmm_alloc_floor_page . . . 1467
7.166.3.4 pmm_bitmap . . . 1468
7.166.3.5 pmm_free_pages . . . 1468
7.166.3.6 pmm_initialized . . . 1468
7.166.3.7 pmm_total_pages . . . 1468
7.166.3.8 pmm_tracked_pages . . . 1468
7.167 kernel/src/memory/vmm.c File Reference . . . 1468
7.167.1 Macro Definition Documentation . . . 1470

7.167.1.1 ADDR_MASK . . . 1470
7.167.1.2 PTE_ACCESSED . . . 1470
7.167.1.3 PTE_DIRTY . . . 1470
7.167.1.4 PTE_GLOBAL . . . 1470
7.167.1.5 PTE_NX . . . 1470
7.167.1.6 PTE_PCD . . . 1470
7.167.1.7 PTE_PRESENT . . . 1471
7.167.1.8 PTE_PS . . . 1471
7.167.1.9 PTE_PWT . . . 1471
7.167.1.10 PTE_USER . . . 1471
7.167.1.11 PTE_WRITABLE . . . 1471
7.167.2 Function Documentation . . . 1471
7.167.2.1 alloc_page_zeroed() . . . 1471
7.167.2.2 get_or_alloc_table() . . . 1472
7.167.2.3 get_table() . . . 1472
7.167.2.4 invlpg() . . . 1473
7.167.2.5 is_ram_type() . . . 1473
7.167.2.6 load_cr3() . . . 1474
7.167.2.7 make_pte() . . . 1474
7.167.2.8 paddr_to_virt() . . . 1474
7.167.2.9 pd_index() . . . 1475
7.167.2.10 pdpt_index() . . . 1475
7.167.2.11 pml4_index() . . . 1475
7.167.2.12 pt_index() . . . 1476
7.167.2.13 resolve_pt() . . . 1476
7.167.2.14 vmm_get_paddr() . . . 1477
7.167.2.15 vmm_init() . . . 1477
7.167.2.16 vmm_map_page() . . . 1478
7.167.2.17 vmm_map_range() . . . 1479
7.167.2.18 vmm_query_page() . . . 1480
7.167.2.19 vmm_unmap_page() . . . 1480
7.167.3 Variable Documentation . . . 1481
7.167.3.1 hpet_phys_base . . . 1481
7.167.3.2 lapic_phys_base . . . 1481
7.167.3.3 vmm_root_pml4_phys . . . 1481
7.168 kernel/src/pci/pci.c File Reference . . . 1482
7.168.1 Function Documentation . . . 1483
7.168.1.1 cfg_read16() . . . 1483
7.168.1.2 cfg_read32() . . . 1484
7.168.1.3 cfg_read8() . . . 1484
7.168.1.4 cfg_write16() . . . 1485
7.168.1.5 cfg_write32() . . . 1485

7.168.1.6 cfg_write8() . . . 1486
7.168.1.7 ecam_cfg_ptr() . . . 1487
7.168.1.8 ecam_read16() . . . 1487
7.168.1.9 ecam_read32() . . . 1488
7.168.1.10 ecam_read8() . . . 1488
7.168.1.11 ecam_write16() . . . 1489
7.168.1.12 ecam_write32() . . . 1489
7.168.1.13 ecam_write8() . . . 1490
7.168.1.14 parse_mcfg() . . . 1491
7.168.1.15 pci_bar() . . . 1491
7.168.1.16 pci_bar_size() . . . 1492
7.168.1.17 pci_enable() . . . 1492
7.168.1.18 pci_find_class() . . . 1493
7.168.1.19 pci_find_first() . . . 1494
7.168.1.20 pci_init() . . . 1494
7.168.1.21 pci_map_bar() . . . 1495
7.168.1.22 pci_read16() . . . 1496
7.168.1.23 pci_read32() . . . 1496
7.168.1.24 pci_read8() . . . 1497
7.168.1.25 pci_write16() . . . 1497
7.168.1.26 pci_write32() . . . 1498
7.168.1.27 pci_write8() . . . 1498
7.168.1.28 sig4_eq() . . . 1499
7.168.1.29 sig8_eq() . . . 1499
7.168.2 Variable Documentation . . . 1499
7.168.2.1 g_bus_first . . . 1499
7.168.2.2 g_bus_last . . . 1499
7.168.2.3 g_ecam_base . . . 1500
7.169 kernel/src/rng/rng.c File Reference . . . 1500
7.169.1 Enumeration Type Documentation . . . 1500
7.169.1.1 anonymous enum . . . 1500
7.169.2 Function Documentation . . . 1501
7.169.2.1 rng_get_bytes() . . . 1501
7.169.2.2 rng_init() . . . 1501
7.169.2.3 rng_ready() . . . 1502
7.169.3 Variable Documentation . . . 1502
7.169.3.1 g_rng_backend . . . 1502
7.170 kernel/src/usb/blkio_usb.c File Reference . . . 1503
7.170.1 Function Documentation . . . 1504
7.170.1.1 blkio_usb_open_msc() . . . 1504
7.170.1.2 usb_blk_close() . . . 1505
7.170.1.3 usb_blk_flush() . . . 1505

7.170.1.4 usb_blk_info() . . . 1505
7.170.1.5 usb_blk_open() . . . 1505
7.170.1.6 usb_blk_read() . . . 1505
7.170.1.7 usb_blk_state_for() . . . 1506
7.170.1.8 usb_blk_write() . . . 1507
7.170.2 Variable Documentation . . . 1508
7.170.2.1 count . . . 1508
7.170.2.2 [struct] . . . 1508
7.170.2.3 g_usb_blk_vtable . . . 1508
7.170.2.4 slots . . . 1508
7.170.2.5 xdev . . . 1508
7.171 kernel/src/usb/usb_kbd.c File Reference . . . 1509
7.171.1 Macro Definition Documentation . . . 1510
7.171.1.1 KBD_EVENTS . . . 1510
7.171.2 Function Documentation . . . 1510
7.171.2.1 held() . . . 1510
7.171.2.2 key_for() . . . 1510
7.171.2.3 push() . . . 1511
7.171.2.4 usb_kbd_forget() . . . 1511
7.171.2.5 usb_kbd_leds_wanted() . . . 1512
7.171.2.6 usb_kbd_pop_event() . . . 1512
7.171.2.7 usb_kbd_report() . . . 1512
7.171.2.8 usb_kbd_set_leds() . . . 1513
7.171.3 Variable Documentation . . . 1514
7.171.3.1 g_count . . . 1514
7.171.3.2 [struct] . . . 1514
7.171.3.3 g_head . . . 1514
7.171.3.4 g_leds_want . . . 1514
7.171.3.5 keycode . . . 1514
7.171.3.6 modifier_key . . . 1514
7.171.3.7 pressed . . . 1514
7.171.3.8 usage_to_key . . . 1515
7.172 kernel/src/usb/xhci.c File Reference . . . 1515
7.172.1 Macro Definition Documentation . . . 1517
7.172.1.1 USB_HID_REPORT_OUTPUT . . . 1517
7.172.1.2 USB_HID_REQ_SET_REPORT . . . 1518
7.172.1.3 XHCI_BAR0_MIN_MAP_SIZE . . . 1518
7.172.1.4 XHCI_LEGSUP_BIOS_OWNED . . . 1518
7.172.1.5 XHCI_LEGSUP_OS_OWNED . . . 1518
7.172.1.6 XHCI_PCI_CLASS . . . 1518
7.172.1.7 XHCI_PCI_PROG_IF . . . 1518
7.172.1.8 XHCI_PCI_SUBCLASS . . . 1518

7.172.1.9 XHCI_XCAP_LEGACY . . . 1518
7.172.2 Function Documentation . . . 1519
7.172.2.1 xhci_bot_get_capacity() . . . 1519
7.172.2.2 xhci_bot_read_block() . . . 1519
7.172.2.3 xhci_bot_read_data_in() . . . 1520
7.172.2.4 xhci_bot_receive_csw() . . . 1521
7.172.2.5 xhci_bot_send_read10() . . . 1522
7.172.2.6 xhci_bot_send_read_capacity10() . . . 1522
7.172.2.7 xhci_bot_send_test_unit_ready() . . . 1523
7.172.2.8 xhci_bot_send_write10() . . . 1524
7.172.2.9 xhci_bot_wait_for_idle() . . . 1525
7.172.2.10 xhci_bot_write_block() . . . 1526
7.172.2.11 xhci_bot_write_data_out() . . . 1527
7.172.2.12 xhci_bringup() . . . 1528
7.172.2.13 xhci_bulk_in_enqueue_and_ring() . . . 1529
7.172.2.14 xhci_bulk_out_enqueue_and_ring() . . . 1530
7.172.2.15 xhci_bulk_purpose_stalled() . . . 1531
7.172.2.16 xhci_cmd_address_device() . . . 1531
7.172.2.17 xhci_cmd_configure_endpoint() . . . 1532
7.172.2.18 xhci_cmd_configure_hid_endpoint() . . . 1533
7.172.2.19 xhci_cmd_disable_slot() . . . 1534
7.172.2.20 xhci_cmd_enable_slot() . . . 1535
7.172.2.21 xhci_cmd_reset_endpoint() . . . 1535
7.172.2.22 xhci_cmd_set_tr_dequeue_pointer() . . . 1536
7.172.2.23 xhci_ctx_bytes() . . . 1537
7.172.2.24 xhci_default_ep0_max_packet() . . . 1537
7.172.2.25 xhci_dev_at() . . . 1538
7.172.2.26 xhci_dev_count() . . . 1538
7.172.2.27 xhci_drain_pending_event() . . . 1538
7.172.2.28 xhci_ep0_bot_mass_storage_reset() . . . 1539
7.172.2.29 xhci_ep0_clear_endpoint_halt() . . . 1540
7.172.2.30 xhci_ep0_control_read() . . . 1541
7.172.2.31 xhci_ep0_control_write_nodata() . . . 1541
7.172.2.32 xhci_ep0_enqueue_trb() . . . 1542
7.172.2.33 xhci_ep0_get_config_descriptor() . . . 1543
7.172.2.34 xhci_ep0_get_device_descriptor() . . . 1544
7.172.2.35 xhci_ep0_hid_set_leds() . . . 1544
7.172.2.36 xhci_ep0_hid_set_protocol() . . . 1545
7.172.2.37 xhci_ep0_set_configuration() . . . 1546
7.172.2.38 xhci_find_and_map() . . . 1547
7.172.2.39 xhci_handle_bulk_stall() . . . 1547
7.172.2.40 xhci_handle_port_connected() . . . 1548

7.172.2.41 xhci_hid_queue_report() . . . 1549
7.172.2.42 xhci_init_bulk_ring() . . . 1549
7.172.2.43 xhci_keyboard_running() . . . 1550
7.172.2.44 xhci_log_hex32() . . . 1550
7.172.2.45 xhci_msc_slot_for() . . . 1551
7.172.2.46 xhci_poll_events() . . . 1551
7.172.2.47 xhci_poll_one() . . . 1553
7.172.2.48 xhci_port_regs() . . . 1556
7.172.2.49 xhci_queue_pending_event() . . . 1556
7.172.2.50 xhci_scan_ports_for_already_connected() . . . 1557
7.172.2.51 xhci_stall_fail() . . . 1558
7.172.2.52 xhci_submit_command() . . . 1558
7.172.2.53 xhci_take_from_bios() . . . 1559
7.172.2.54 xhci_wait_bit() . . . 1559
7.172.3 Variable Documentation . . . 1560
7.172.3.1 g_xhci_dev_count . . . 1560
7.172.3.2 g_xhci_devs . . . 1560
7.172.3.3 g_xhci_keyboards . . . 1560
7.173 kernel/src/v4/sk_v4.c File Reference . . . 1560
7.173.1 Macro Definition Documentation . . . 1561
7.173.1.1 SK_V4_CURSOR_BLINK . . . 1561
7.173.1.2 SK_V4_LINE . . . 1561
7.173.1.3 SK_V4_PROMPT . . . 1561
7.173.2 Function Documentation . . . 1562
7.173.2.1 sk_v4_key() . . . 1562
7.173.2.2 sk_v4_out() . . . 1563
7.173.2.3 sk_v4_readline() . . . 1564
7.173.2.4 sk_v4_run() . . . 1565
7.173.2.5 sk_v4_serve() . . . 1566
7.173.3 Variable Documentation . . . 1567
7.173.3.1 sk_v4_es . . . 1567
7.173.3.2 sk_v4_heat . . . 1567
7.173.3.3 sk_v4_image . . . 1567
7.173.3.4 sk_v4_node . . . 1567
7.173.3.5 sk_v4_post_blk_ram . . . 1567
7.173.3.6 sk_v4_text . . . 1567
7.173.3.7 sk_v4_words . . . 1568
7.174 kernel/src/v4/sk_v4_grow.c File Reference . . . 1568
7.174.1 Function Documentation . . . 1569
7.174.1.1 bam_cover_devblocks() . . . 1569
7.174.1.2 dev_rw4k() . . . 1569
7.174.1.3 read_header() . . . 1570

7.174.1.4 sk_v4_grow_artemis() . . . 1570
7.174.1.5 sk_v4_grow_cap_sectors() . . . 1571
7.174.2 Variable Documentation . . . 1571
7.174.2.1 bytes . . . 1571
7.174.2.2 [union] . . . 1572
7.174.2.3 g_move . . . 1572
7.174.2.4 meta . . . 1572
7.175 kernel/src/v4/sk_v4_identity.c File Reference . . . 1572
7.175.1 Macro Definition Documentation . . . 1574
7.175.1.1 SK_V4_CAP_SECTORS . . . 1574
7.175.1.2 SK_V4_GPT_PARTS . . . 1574
7.175.1.3 SK_V4_GROW_MIN_BYTES . . . 1574
7.175.1.4 SK_V4_IDLE_BEAT . . . 1574
7.175.2 Function Documentation . . . 1574
7.175.2.1 sk_v4_discard_input() . . . 1574
7.175.2.2 sk_v4_drive_arrived() . . . 1575
7.175.2.3 sk_v4_drive_left() . . . 1577
7.175.2.4 sk_v4_fixed_screen() . . . 1578
7.175.2.5 sk_v4_genesis() . . . 1578
7.175.2.6 sk_v4_grow_stick() . . . 1579
7.175.2.7 sk_v4_identity_beat() . . . 1580
7.175.2.8 sk_v4_identity_boot() . . . 1581
7.175.2.9 sk_v4_identity_check_now() . . . 1582
7.175.2.10 sk_v4_identity_marker() . . . 1583
7.175.2.11 sk_v4_identity_offer_wait() . . . 1584
7.175.2.12 sk_v4_identity_offered_present() . . . 1584
7.175.2.13 sk_v4_identity_present() . . . 1585
7.175.2.14 sk_v4_identity_recheck_offered() . . . 1585
7.175.2.15 sk_v4_identity_release_offered() . . . 1586
7.175.2.16 sk_v4_identity_set_marker() . . . 1587
7.175.2.17 sk_v4_identity_stick_in() . . . 1587
7.175.2.18 sk_v4_identity_wait() . . . 1588
7.175.2.19 sk_v4_idle() . . . 1589
7.175.2.20 sk_v4_is_this_stick() . . . 1590
7.175.2.21 sk_v4_login() . . . 1590
7.175.2.22 sk_v4_real_zuse_arrived() . . . 1591
7.175.2.23 sk_v4_session_end() . . . 1592
7.175.2.24 sk_v4_user_prefix() . . . 1593
7.175.3 Variable Documentation . . . 1593
7.175.3.1 count . . . 1593
7.175.3.2 g_boot_disk . . . 1594
7.175.3.3 g_gpt_parts . . . 1594

7.175.3.4 g_last_beat . . . 1594
7.175.3.5 g_marker . . . 1594
7.175.3.6 g_mint_waiting . . . 1594
7.175.3.7 g_offered . . . 1594
7.175.3.8 g_pending_real . . . 1594
7.175.3.9 g_proxy_fallback . . . 1595
7.175.3.10 g_real_drive . . . 1595
7.175.3.11 g_spare . . . 1595
7.175.3.12 g_stick_in . . . 1595
7.175.3.13 [struct] . . . 1595
7.175.3.14 g_zuse_cert_installed . . . 1595
7.175.3.15 g_zuse_dev . . . 1595
7.175.3.16 g_zuse_pubkey . . . 1596
7.175.3.17 g_zuse_root_known . . . 1596
7.175.3.18 g_zuse_root_pubkey . . . 1596
7.175.3.19 g_zuse_seed . . . 1596
7.175.3.20 g_zuse_session . . . 1596
7.175.3.21 slots . . . 1596
7.176 kernel/src/v4/sk_v4_marker.c File Reference . . . 1597
7.176.1 Function Documentation . . . 1597
7.176.1.1 sk_v4_marker_read() . . . 1597
7.176.1.2 sk_v4_marker_write() . . . 1598
7.176.2 Variable Documentation . . . 1599
7.176.2.1 bytes . . . 1599
7.176.2.2 [union] . . . 1599
7.176.2.3 v1 . . . 1599
7.176.2.4 v2 . . . 1599
7.177 kernel/src/v4/sk_v4_mint.c File Reference . . . 1600
7.177.1 Macro Definition Documentation . . . 1600
7.177.1.1 MINT_IDENT_SECTORS . . . 1600
7.177.1.2 MINT_MIN_BYTES . . . 1600
7.177.1.3 MINT_ZERO_FBLOCKS . . . 1601
7.177.2 Function Documentation . . . 1601
7.177.2.1 failed() . . . 1601
7.177.2.2 read_answer() . . . 1601
7.177.2.3 say_size() . . . 1602
7.177.2.4 sk_v4_mint_zuse() . . . 1603
7.178 kernel/src/virtio/virtio_blk.c File Reference . . . 1604
7.178.1 Macro Definition Documentation . . . 1606
7.178.1.1 VBLK_IO_TIMEOUT_NS . . . 1606
7.178.1.2 VCAP_OFF_BAR . . . 1606
7.178.1.3 VCAP_OFF_CAP_LEN . . . 1606

7.178.1.4 VCAP_OFF_CAP_NEXT . . . 1606
7.178.1.5 VCAP_OFF_CAP_VNDR . . . 1607
7.178.1.6 VCAP_OFF_CFG_TYPE . . . 1607
7.178.1.7 VCAP_OFF_LENGTH . . . 1607
7.178.1.8 VCAP_OFF_NOTIFY_MULT . . . 1607
7.178.1.9 VCAP_OFF_OFFSET . . . 1607
7.178.1.10 VIRTIO_F_VERSION_1 . . . 1607
7.178.1.11 VIRTIO_PCI_CAP_COMMON_CFG . . . 1607
7.178.1.12 VIRTIO_PCI_CAP_DEVICE_CFG . . . 1607
7.178.1.13 VIRTIO_PCI_CAP_ISR_CFG . . . 1607
7.178.1.14 VIRTIO_PCI_CAP_NOTIFY_CFG . . . 1607
7.178.1.15 VIRTIO_PCI_CAP_VENDOR_ID . . . 1608
7.178.1.16 VIRTIO_STATUS_ACKNOWLEDGE . . . 1608
7.178.1.17 VIRTIO_STATUS_DRIVER . . . 1608
7.178.1.18 VIRTIO_STATUS_DRIVER_OK . . . 1608
7.178.1.19 VIRTIO_STATUS_FAILED . . . 1608
7.178.1.20 VIRTIO_STATUS_FEATURES_OK . . . 1608
7.178.1.21 VQUEUE_SIZE . . . 1608
7.178.1.22 VRING_DESC_F_NEXT . . . 1608
7.178.1.23 VRING_DESC_F_WRITE . . . 1608
7.178.2 Function Documentation . . . 1609
7.178.2.1 rmb() . . . 1609
7.178.2.2 vblk_close() . . . 1609
7.178.2.3 vblk_flush() . . . 1609
7.178.2.4 vblk_info() . . . 1609
7.178.2.5 vblk_init_device() . . . 1610
7.178.2.6 vblk_io() . . . 1610
7.178.2.7 vblk_open() . . . 1611
7.178.2.8 vblk_read() . . . 1612
7.178.2.9 vblk_write() . . . 1612
7.178.2.10 virtio_blk_find_artemis() . . . 1613
7.178.2.11 walk_virtio_caps() . . . 1614
7.178.2.12 wmb() . . . 1614
7.178.3 Variable Documentation . . . 1615
7.178.3.1 g_vblk . . . 1615
7.178.3.2 g_vblk_dead . . . 1615
7.178.3.3 g_vblk_ready . . . 1615
7.178.3.4 g_vblk_vtable . . . 1615
7.179 kernel/src/virtio/virtio_input.c File Reference . . . 1615
7.179.1 Macro Definition Documentation . . . 1617
7.179.1.1 DIAG_RING_SIZE . . . 1617
7.179.1.2 EVENTQ_SIZE . . . 1617

7.179.1.3 VCAP_OFF_BAR . . . 1617
7.179.1.4 VCAP_OFF_CAP_NEXT . . . 1617
7.179.1.5 VCAP_OFF_CAP_VNDR . . . 1617
7.179.1.6 VCAP_OFF_CFG_TYPE . . . 1618
7.179.1.7 VCAP_OFF_LENGTH . . . 1618
7.179.1.8 VCAP_OFF_NOTIFY_MULT . . . 1618
7.179.1.9 VCAP_OFF_OFFSET . . . 1618
7.179.1.10 VIRTIO_F_VERSION_1 . . . 1618
7.179.1.11 VIRTIO_PCI_CAP_COMMON_CFG . . . 1618
7.179.1.12 VIRTIO_PCI_CAP_ISR_CFG . . . 1618
7.179.1.13 VIRTIO_PCI_CAP_NOTIFY_CFG . . . 1618
7.179.1.14 VIRTIO_PCI_CAP_VENDOR_ID . . . 1618
7.179.1.15 VIRTIO_STATUS_ACKNOWLEDGE . . . 1618
7.179.1.16 VIRTIO_STATUS_DRIVER . . . 1619
7.179.1.17 VIRTIO_STATUS_DRIVER_OK . . . 1619
7.179.1.18 VIRTIO_STATUS_FAILED . . . 1619
7.179.1.19 VIRTIO_STATUS_FEATURES_OK . . . 1619
7.179.1.20 VRING_DESC_F_WRITE . . . 1619
7.179.2 Function Documentation . . . 1619
7.179.2.1 diag_push() . . . 1619
7.179.2.2 enable_interrupt_route() . . . 1620
7.179.2.3 vinput_init_device() . . . 1620
7.179.2.4 virtio_input_find_keyboard() . . . 1621
7.179.2.5 virtio_input_isr() . . . 1622
7.179.2.6 virtio_input_pop_event() . . . 1622
7.179.2.7 walk_virtio_caps() . . . 1623
7.179.2.8 wmb() . . . 1624
7.179.3 Variable Documentation . . . 1624
7.179.3.1 g_diag_head . . . 1624
7.179.3.2 g_diag_ring . . . 1624
7.179.3.3 g_diag_tail . . . 1624
7.179.3.4 g_vinput . . . 1624
7.179.3.5 g_vinput_ready . . . 1624
7.179.3.6 g_virtio_input_gic_intid . . . 1625
7.179.3.7 g_virtio_input_isr_count . . . 1625
7.179.3.8 g_virtio_input_plic_source . . . 1625
7.180 kernel/src/virtio/virtio_rng.c File Reference . . . 1625
7.180.1 Macro Definition Documentation . . . 1626
7.180.1.1 VCAP_OFF_BAR . . . 1626
7.180.1.2 VCAP_OFF_CAP_NEXT . . . 1627
7.180.1.3 VCAP_OFF_CAP_VNDR . . . 1627
7.180.1.4 VCAP_OFF_CFG_TYPE . . . 1627

7.180.1.5 VCAP_OFF_LENGTH . . . 1627
7.180.1.6 VCAP_OFF_NOTIFY_MULT . . . 1627
7.180.1.7 VCAP_OFF_OFFSET . . . 1627
7.180.1.8 VIRTIO_F_VERSION_1 . . . 1627
7.180.1.9 VIRTIO_PCI_CAP_COMMON_CFG . . . 1627
7.180.1.10 VIRTIO_PCI_CAP_NOTIFY_CFG . . . 1627
7.180.1.11 VIRTIO_PCI_CAP_VENDOR_ID . . . 1627
7.180.1.12 VIRTIO_STATUS_ACKNOWLEDGE . . . 1628
7.180.1.13 VIRTIO_STATUS_DRIVER . . . 1628
7.180.1.14 VIRTIO_STATUS_DRIVER_OK . . . 1628
7.180.1.15 VIRTIO_STATUS_FAILED . . . 1628
7.180.1.16 VIRTIO_STATUS_FEATURES_OK . . . 1628
7.180.1.17 VQUEUE_SIZE . . . 1628
7.180.1.18 VRING_DESC_F_WRITE . . . 1628
7.180.1.19 VRNG_BUF_SIZE . . . 1628
7.180.2 Function Documentation . . . 1628
7.180.2.1 rmb() . . . 1628
7.180.2.2 virtio_rng_get_bytes() . . . 1629
7.180.2.3 virtio_rng_init() . . . 1629
7.180.2.4 virtio_rng_ready() . . . 1630
7.180.2.5 vrng_init_device() . . . 1630
7.180.2.6 vrng_request() . . . 1631
7.180.2.7 walk_virtio_caps() . . . 1632
7.180.2.8 wmb() . . . 1632
7.180.3 Variable Documentation . . . 1633
7.180.3.1 g_vrng . . . 1633
7.180.3.2 g_vrng_ready . . . 1633
7.181 kernel/src/vm/arena.c File Reference . . . 1633
7.182 kernel/src/vm/bootstrap/sk_vm_bootstrap.c File Reference . . . 1633
7.183 kernel/src/vm/host/shim.c File Reference . . . 1633
7.184 kernel/src/vm/kernel_hermes.c File Reference . . . 1633
7.185 kernel/src/vm/parity.c File Reference . . . 1633
7.185.1 Macro Definition Documentation . . . 1634
7.185.1.1 MAX_DICT_ENTRIES . . . 1634
7.185.1.2 SK_PARITY_DEBUG . . . 1634
7.185.1.3 sk_parity_debug_check_ptr . . . 1635
7.185.1.4 sk_parity_debug_log_msg . . . 1635
7.185.2 Function Documentation . . . 1635
7.185.2.1 fnv1a_64() . . . 1635
7.185.2.2 fnv1a_64_u32() . . . 1636
7.185.2.3 fnv1a_64_u8() . . . 1637
7.185.2.4 hash_colon_body() . . . 1637

7.185.2.5 print_hex64() . . . 1638
7.185.2.6 print_nl() . . . 1639
7.185.2.7 print_str() . . . 1640
7.185.2.8 print_u32() . . . 1641
7.185.2.9 sk_dict_canonical_hash() . . . 1642
7.185.2.10 sk_dict_word_count() . . . 1643
7.185.2.11 sk_parity_collect() . . . 1644
7.185.2.12 sk_parity_print() . . . 1645
7.186 kernel/src/vm/q48_stubs.c File Reference . . . 1646
7.186.1 Function Documentation . . . 1646
7.186.1.1 q48_from_double() . . . 1646
7.186.1.2 q48_to_double() . . . 1647
7.187 kernel/src/vm/session.c File Reference . . . 1648
7.188 kernel/src/vm/stadium.c File Reference . . . 1648
7.189 kernel/src/vm/stadium_blocks.c File Reference . . . 1648
7.190 kernel/src/vm/stadium_words.c File Reference . . . 1648
7.191 kernel/src/vm/switch.c File Reference . . . 1649
7.192 kernel/src/vm/vm_core.c File Reference . . . 1649
7.192.1 Macro Definition Documentation . . . 1651
7.192.1.1 vm_check_arena . . . 1651
7.192.2 Function Documentation . . . 1651
7.192.2.1 acl_recheck() . . . 1651
7.192.2.2 execute_colon_word() . . . 1652
7.192.2.3 hosted_alloc() . . . 1654
7.192.2.4 hosted_free() . . . 1655
7.192.2.5 hosted_mutex_destroy() . . . 1655
7.192.2.6 hosted_mutex_init() . . . 1656
7.192.2.7 hosted_mutex_lock() . . . 1656
7.192.2.8 hosted_mutex_unlock() . . . 1656
7.192.2.9 hosted_panic() . . . 1657
7.192.2.10 hosted_putc() . . . 1657
7.192.2.11 hosted_puts() . . . 1657
7.192.2.12 hosted_time_ns() . . . 1658
7.192.2.13 hosted_xt_entry_owned() . . . 1658
7.192.2.14 hosted_xt_is_executable() . . . 1658
7.192.2.15 sk_vm_at_outermost_interpret() . . . 1659
7.192.2.16 vm_addr_ok() . . . 1659
7.192.2.17 vm_assert_interpreter_enabled() . . . 1660
7.192.2.18 vm_cleanup() . . . 1661
7.192.2.19 vm_compile_call() . . . 1662
7.192.2.20 vm_compile_exit() . . . 1663
7.192.2.21 vm_compile_literal() . . . 1664

7.192.2.22 vm_compile_word() . . . 1665
7.192.2.23 vm_default_host_services() . . . 1667
7.192.2.24 vm_enable_interpreter() . . . 1667
7.192.2.25 vm_enter_compile_mode() . . . 1668
7.192.2.26 vm_exit_compile_mode() . . . 1670
7.192.2.27 vm_get_base() . . . 1670
7.192.2.28 vm_host() . . . 1672
7.192.2.29 vm_host_alloc() . . . 1673
7.192.2.30 vm_host_calloc() . . . 1674
7.192.2.31 vm_host_free() . . . 1674
7.192.2.32 vm_interpret() . . . 1675
7.192.2.33 vm_interpret_word() . . . 1677
7.192.2.34 vm_load_cell() . . . 1679
7.192.2.35 vm_load_u8() . . . 1680
7.192.2.36 vm_log_attributed_vm() . . . 1681
7.192.2.37 vm_make_immediate() . . . 1681
7.192.2.38 vm_monotonic_ns() . . . 1682
7.192.2.39 vm_parse_number() . . . 1683
7.192.2.40 vm_parse_word() . . . 1684
7.192.2.41 vm_ptr() . . . 1686
7.192.2.42 vm_reset_hosted_fake_ns() . . . 1688
7.192.2.43 vm_set_base() . . . 1689
7.192.2.44 vm_store_cell() . . . 1690
7.192.2.45 vm_store_u8() . . . 1691
7.192.2.46 vm_zuse_cert_install() . . . 1692
7.192.3 Variable Documentation . . . 1693
7.192.3.1 g_log_attrib_vm . . . 1693
7.192.3.2 g_vm_interpret_depth . . . 1693
7.192.3.3 hosted_fake_ns . . . 1693
7.192.3.4 hosted_services . . . 1693
7.193 kernel/src/vm/vm_log_buffer.c File Reference . . . 1694
7.194 kernel/src/vm/vm_runtime.c File Reference . . . 1694
7.194.1 Function Documentation . . . 1695
7.194.1.1 heartbeat_publish_snapshot() . . . 1695
7.194.1.2 heartbeat_snapshot_index_load() . . . 1696
7.194.1.3 heartbeat_snapshot_index_store() . . . 1696
7.194.1.4 heartbeat_thread_main() . . . 1697
7.194.1.5 vm_heartbeat_run_cycle() . . . 1698
7.194.1.6 vm_heartbeat_update_l8() . . . 1700
7.194.1.7 vm_snapshot_read() . . . 1701
7.194.1.8 vm_tick() . . . 1702
7.194.1.9 vm_tick_apply_background_decay() . . . 1703

7.194.1.10 vm_tick_inference_engine() . . . 1704
7.194.1.11 vm_tick_slope_validator() . . . 1706
7.194.1.12 vm_tick_window_tuner() . . . 1707
7.195 v3/include/arch_detect.h File Reference . . . 1708
7.195.1 Detailed Description . . . 1709
7.195.2 Macro Definition Documentation . . . 1709
7.195.2.1 ARCH_ARM64 . . . 1709
7.195.2.2 ARCH_NAME . . . 1710
7.195.2.3 ARCH_RISCV64 . . . 1710
7.195.2.4 ARCH_X86_64 . . . 1710
7.195.2.5 NEXT_OPT . . . 1710
7.195.2.6 PRIM_BRANCH_OPT . . . 1710
7.195.2.7 PRIM_DROP_OPT . . . 1710
7.195.2.8 PRIM_DUP_OPT . . . 1710
7.195.2.9 PRIM_FETCH_OPT . . . 1710
7.195.2.10 PRIM_MINUS_OPT . . . 1710
7.195.2.11 PRIM_PLUS_OPT . . . 1710
7.195.2.12 PRIM_R_FETCH_OPT . . . 1711
7.195.2.13 PRIM_R_FROM_OPT . . . 1711
7.195.2.14 PRIM_STAR_OPT . . . 1711
7.195.2.15 PRIM_STORE_OPT . . . 1711
7.195.2.16 PRIM_SWAP_OPT . . . 1711
7.195.2.17 PRIM_TO_R_OPT . . . 1711
7.195.2.18 PRIM_ZBRANCH_OPT . . . 1711
7.195.2.19 vm_abs_opt . . . 1711
7.195.2.20 vm_add_overflow_opt . . . 1711
7.195.2.21 vm_clz_opt . . . 1712
7.195.2.22 vm_ctz_opt . . . 1712
7.195.2.23 vm_divmod_opt . . . 1712
7.195.2.24 vm_max_opt . . . 1712
7.195.2.25 vm_memcpy_opt . . . 1712
7.195.2.26 vm_memzero_opt . . . 1712
7.195.2.27 vm_min_opt . . . 1712
7.195.2.28 vm_mul_double_opt . . . 1713
7.195.2.29 vm_pop_opt . . . 1713
7.195.2.30 vm_popcnt_opt . . . 1713
7.195.2.31 vm_prefetch_opt . . . 1713
7.195.2.32 vm_prefetch_stream_opt . . . 1713
7.195.2.33 vm_push_opt . . . 1713
7.195.2.34 vm_rpop_opt . . . 1713
7.195.2.35 vm_rpush_opt . . . 1713
7.195.2.36 vm_save_registers_opt . . . 1714

7.195.2.37 vm_setup_registers_opt . . . 1714
7.195.2.38 vm_strcmp_opt . . . 1714
7.195.2.39 vm_sub_overflow_opt . . . 1714
7.195.3 Function Documentation . . . 1714
7.195.3.1 vm_get_arch_name() . . . 1714
7.195.3.2 vm_get_build_info() . . . 1715
7.195.3.3 vm_has_asm_opt() . . . 1715
7.195.3.4 vm_has_direct_threading() . . . 1716
7.195.3.5 vm_has_sse42_opt() . . . 1716
7.196 v3/include/blkcfg.h File Reference . . . 1716
7.196.1 Macro Definition Documentation . . . 1717
7.196.1.1 BLKCFG_DEFAULT_FBS . . . 1717
7.196.1.2 BLKCFG_PATH_MAX . . . 1717
7.196.2 Enumeration Type Documentation . . . 1717
7.196.2.1 blkcfg_backend_t . . . 1717
7.196.3 Function Documentation . . . 1718
7.196.3.1 blkcfg_format_summary() . . . 1718
7.196.3.2 blkcfg_init_from_options() . . . 1718
7.196.4 Variable Documentation . . . 1718
7.196.4.1 g_blkcfg . . . 1718
7.197 v3/include/blkio.h File Reference . . . 1719
7.197.1 Macro Definition Documentation . . . 1720
7.197.1.1 BLKIO_FORTH_BLOCK_SIZE . . . 1720
7.197.2 Typedef Documentation . . . 1720
7.197.2.1 blkio_dev_t . . . 1720
7.197.3 Enumeration Type Documentation . . . 1720
7.197.3.1 anonymous enum . . . 1720
7.197.4 Function Documentation . . . 1720
7.197.4.1 blkio_bzero() . . . 1720
7.197.4.2 blkio_close() . . . 1721
7.197.4.3 blkio_flush() . . . 1721
7.197.4.4 blkio_info() . . . 1722
7.197.4.5 blkio_open() . . . 1722
7.197.4.6 blkio_read() . . . 1723
7.197.4.7 blkio_write() . . . 1723
7.198 v3/include/blkio_factory.h File Reference . . . 1724
7.198.1 Function Documentation . . . 1725
7.198.1.1 blkio_factory_open() . . . 1725
7.199 v3/include/block_subsystem.h File Reference . . . 1727
7.199.1 Macro Definition Documentation . . . 1730
7.199.1.1 BLK_DEVICE_SECTOR . . . 1730
7.199.1.2 BLK_DISK_START . . . 1730

7.199.1.3 BLK_DISK_SYS_RESERVED . . . 1730
7.199.1.4 BLK_FLAG_CLAIMED . . . 1730
7.199.1.5 BLK_FLAG_MIGRATING . . . 1730
7.199.1.6 BLK_FLAG_STALE . . . 1730
7.199.1.7 BLK_FORTH_SIZE . . . 1730
7.199.1.8 BLK_FORTH_SYS_RESERVED . . . 1730
7.199.1.9 BLK_META_FENCE_INIT . . . 1730
7.199.1.10 BLK_META_PER_BLOCK . . . 1731
7.199.1.11 BLK_META_TOTAL . . . 1731
7.199.1.12 BLK_PACK_RATIO . . . 1731
7.199.1.13 BLK_RAM_BLOCKS . . . 1731
7.199.2 Enumeration Type Documentation . . . 1731
7.199.2.1 anonymous enum . . . 1731
7.199.2.2 anonymous enum . . . 1731
7.199.3 Function Documentation . . . 1732
7.199.3.1 blk_acl_allow_get() . . . 1732
7.199.3.2 blk_acl_allow_set() . . . 1732
7.199.3.3 blk_acl_ttl_get() . . . 1733
7.199.3.4 blk_acl_ttl_set() . . . 1734
7.199.3.5 blk_allocate() . . . 1734
7.199.3.6 blk_firsttouch_claim() . . . 1735
7.199.3.7 blk_flags_get() . . . 1735
7.199.3.8 blk_flags_set() . . . 1736
7.199.3.9 blk_flush() . . . 1736
7.199.3.10 blk_get_buffer() . . . 1737
7.199.3.11 blk_get_device_free_blocks() . . . 1738
7.199.3.12 blk_get_device_range() . . . 1739
7.199.3.13 blk_get_empty_buffer() . . . 1740
7.199.3.14 blk_get_first_disk_range() . . . 1740
7.199.3.15 blk_get_meta() . . . 1741
7.199.3.16 blk_get_total_blocks() . . . 1741
7.199.3.17 blk_get_volume_meta() . . . 1742
7.199.3.18 blk_is_allocated() . . . 1742
7.199.3.19 blk_is_valid() . . . 1743
7.199.3.20 blk_lbn_device_handle() . . . 1743
7.199.3.21 blk_mark_allocated() . . . 1744
7.199.3.22 blk_mark_free() . . . 1745
7.199.3.23 blk_meta_relocate_devblock() . . . 1745
7.199.3.24 blk_meta_zone_read() . . . 1746
7.199.3.25 blk_meta_zone_write() . . . 1747
7.199.3.26 blk_migration_idle_check() . . . 1748
7.199.3.27 blk_owner_fp_get() . . . 1748

7.199.3.28 blk_owner_fp_set() . . . 1749
7.199.3.29 blk_owner_has_claim() . . . 1749
7.199.3.30 blk_set_meta() . . . 1750
7.199.3.31 blk_set_volume_meta() . . . 1751
7.199.3.32 blk_subsys_add_raw_device() . . . 1751
7.199.3.33 blk_subsys_attach_device() . . . 1751
7.199.3.34 blk_subsys_confirm_format() . . . 1752
7.199.3.35 blk_subsys_detach_device() . . . 1753
7.199.3.36 blk_subsys_epoch() . . . 1754
7.199.3.37 blk_subsys_init() . . . 1754
7.199.3.38 blk_subsys_relocate_block() . . . 1755
7.199.3.39 blk_subsys_set_format_bam_for() . . . 1756
7.199.3.40 blk_subsys_shutdown() . . . 1756
7.199.3.41 blk_update() . . . 1757
7.199.3.42 blk_volume_compute_totals() . . . 1758
7.199.3.43 compute_crc64() . . . 1758
7.200 v3/include/cli.h File Reference . . . 1759
7.200.1 Enumeration Type Documentation . . . 1760
7.200.1.1 HeartbeatLogMode . . . 1760
7.200.2 Function Documentation . . . 1761
7.200.2.1 cli_parse() . . . 1761
7.200.2.2 cli_print_usage() . . . 1762
7.201 v3/include/compudynamics.h File Reference . . . 1763
7.201.1 Macro Definition Documentation . . . 1764
7.201.1.1 CD_MAX_CLASSIFY_DEPTH . . . 1764
7.201.2 Function Documentation . . . 1764
7.201.2.1 cd_classify_ids() . . . 1764
7.201.2.2 cd_config_table_force() . . . 1766
7.201.2.3 cd_config_table_free() . . . 1766
7.201.2.4 cd_config_table_init() . . . 1767
7.201.2.5 cd_config_table_seed() . . . 1767
7.201.2.6 cd_config_table_tick() . . . 1768
7.201.2.7 cd_tuning_vm() . . . 1769
7.201.2.8 cd_tuning_word() . . . 1770
7.202 v3/include/dictionary_heat_optimization.h File Reference . . . 1770
7.202.1 Macro Definition Documentation . . . 1771
7.202.1.1 SF_FC_BUCKETS . . . 1771
7.202.2 Function Documentation . . . 1771
7.202.2.1 dict_adaptive_optimization_pass() . . . 1771
7.202.2.2 dict_find_word_heat_aware() . . . 1773
7.202.2.3 dict_reorganize_buckets_by_heat() . . . 1776
7.202.2.4 dict_update_heat_percentiles() . . . 1778

7.202.3 Variable Documentation . . . 1779
7.202.3.1 sf_fc_cap . . . 1779
7.202.3.2 sf_fc_count . . . 1779
7.202.3.3 sf_fc_list . . . 1779
7.203 v3/include/dictionary_management.h File Reference . . . 1780
7.203.1 Function Documentation . . . 1781
7.203.1.1 dictionary_clear() . . . 1781
7.203.1.2 dictionary_init() . . . 1781
7.203.1.3 dictionary_lookup() . . . 1781
7.203.1.4 dictionary_register() . . . 1782
7.204 v3/include/doe_metrics.h File Reference . . . 1782
7.204.1 Detailed Description . . . 1783
7.204.2 Function Documentation . . . 1784
7.204.2.1 metrics_from_vm() . . . 1784
7.204.2.2 metrics_get_cpu_freq_mhz() . . . 1785
7.204.2.3 metrics_get_cpu_temp_c() . . . 1786
7.204.2.4 metrics_get_timestamp() . . . 1787
7.204.2.5 metrics_print_text() . . . 1788
7.204.2.6 metrics_write_csv_header() . . . 1788
7.204.2.7 metrics_write_csv_row() . . . 1789
7.204.2.8 metrics_write_james_law_csv_row() . . . 1790
7.205 v3/include/inference_engine.h File Reference . . . 1791
7.205.1 Function Documentation . . . 1792
7.205.1.1 compute_variance_q48() . . . 1792
7.205.1.2 find_variance_inflection() . . . 1793
7.205.1.3 infer_decay_slope_q48() . . . 1795
7.205.1.4 inference_engine_run() . . . 1796
7.205.1.5 inference_outputs_to_string() . . . 1798
7.205.1.6 inference_outputs_validate() . . . 1799
7.206 v3/include/io.h File Reference . . . 1800
7.206.1 Macro Definition Documentation . . . 1802
7.206.1.1 BLOCK_COUNT . . . 1802
7.206.1.2 MEMORY_SIZE . . . 1802
7.206.2 Function Documentation . . . 1802
7.206.2.1 io_init() . . . 1802
7.206.2.2 io_read_block() . . . 1802
7.206.2.3 io_write_block() . . . 1803
7.207 kernel/include/starkernel/log.h File Reference . . . 1804
7.207.1 Macro Definition Documentation . . . 1804
7.207.1.1 LOG_MSG_LINE_MAX . . . 1804
7.207.2 Enumeration Type Documentation . . . 1804
7.207.2.1 LogLevel . . . 1804

7.207.2.2 TestResult . . . 1805
7.207.3 Function Documentation . . . 1805
7.207.3.1 log_get_level() . . . 1805
7.207.3.2 log_message() . . . 1805
7.207.3.3 log_set_level() . . . 1805
7.207.3.4 log_set_vm() . . . 1806
7.207.3.5 log_test_result() . . . 1806
7.208 v3/include/log.h File Reference . . . 1806
7.208.1 Macro Definition Documentation . . . 1807
7.208.1.1 LOG_MSG_LINE_MAX . . . 1807
7.208.2 Enumeration Type Documentation . . . 1807
7.208.2.1 LogLevel . . . 1807
7.208.2.2 TestResult . . . 1807
7.208.3 Function Documentation . . . 1808
7.208.3.1 log_get_level() . . . 1808
7.208.3.2 log_message() . . . 1808
7.208.3.3 log_set_level() . . . 1810
7.208.3.4 log_set_vm() . . . 1811
7.208.3.5 log_test_result() . . . 1811
7.209 v3/include/math_portable.h File Reference . . . 1812
7.209.1 Macro Definition Documentation . . . 1813
7.209.1.1 Q48_SCALE . . . 1813
7.209.2 Function Documentation . . . 1814
7.209.2.1 erf_q48() . . . 1814
7.209.2.2 sqrt_q48() . . . 1815
7.210 v3/include/memory_management.h File Reference . . . 1816
7.210.1 Detailed Description . . . 1817
7.210.2 Function Documentation . . . 1817
7.210.2.1 vm_align() . . . 1817
7.210.2.2 vm_allot() . . . 1818
7.211 v3/include/physics_execution_hooks.h File Reference . . . 1820
7.211.1 Function Documentation . . . 1821
7.211.1.1 physics_on_lookup() . . . 1821
7.211.1.2 physics_post_execute() . . . 1823
7.211.1.3 physics_pre_execute() . . . 1825
7.212 v3/include/physics_hotwords_cache.h File Reference . . . 1826
7.212.1 Macro Definition Documentation . . . 1828
7.212.1.1 HOTWORDS_CACHE_SIZE . . . 1828
7.212.1.2 HOTWORDS_EXECUTION_HEAT_DELTA_THRESHOLD . . . 1828
7.212.1.3 HOTWORDS_EXECUTION_HEAT_THRESHOLD . . . 1829
7.212.2 Typedef Documentation . . . 1829
7.212.2.1 HotwordsCache . . . 1829

7.212.3 Function Documentation . . . 1829
7.212.3.1 hotwords_bayesian_report() . . . 1829
7.212.3.2 hotwords_bucket_reorder() . . . 1830
7.212.3.3 hotwords_cache_cleanup() . . . 1831
7.212.3.4 hotwords_cache_evict_entry() . . . 1832
7.212.3.5 hotwords_cache_evict_name() . . . 1832
7.212.3.6 hotwords_cache_init() . . . 1834
7.212.3.7 hotwords_cache_lookup() . . . 1835
7.212.3.8 hotwords_cache_promote() . . . 1837
7.212.3.9 hotwords_cache_set_enabled() . . . 1839
7.212.3.10 hotwords_posterior_bucket_searches() . . . 1840
7.212.3.11 hotwords_posterior_cache_hits() . . . 1841
7.212.3.12 hotwords_speedup_estimate() . . . 1842
7.212.3.13 hotwords_stats_print() . . . 1843
7.212.3.14 hotwords_stats_reset() . . . 1844
7.213 v3/include/physics_metadata.h File Reference . . . 1845
7.213.1 Function Documentation . . . 1846
7.213.1.1 physics_decay_slope_load() . . . 1846
7.213.1.2 physics_execution_heat_increment() . . . 1847
7.213.1.3 physics_execution_heat_load() . . . 1847
7.213.1.4 physics_metadata_apply_linear_decay() . . . 1847
7.213.1.5 physics_metadata_apply_seed() . . . 1849
7.213.1.6 physics_metadata_init() . . . 1850
7.213.1.7 physics_metadata_record_latency() . . . 1852
7.213.1.8 physics_metadata_refresh_state() . . . 1854
7.213.1.9 physics_metadata_set_mass() . . . 1855
7.213.1.10 physics_metadata_touch() . . . 1856
7.214 v3/include/physics_pipelining_metrics.h File Reference . . . 1857
7.214.1 Typedef Documentation . . . 1859
7.214.1.1 WordTransitionMetrics . . . 1859
7.214.2 Function Documentation . . . 1859
7.214.2.1 transition_metrics_binary_chop_suggest_window() . . . 1859
7.214.2.2 transition_metrics_cleanup() . . . 1860
7.214.2.3 transition_metrics_context_accuracy_string() . . . 1861
7.214.2.4 transition_metrics_get_probability_q48() . . . 1862
7.214.2.5 transition_metrics_init() . . . 1863
7.214.2.6 transition_metrics_record() . . . 1865
7.214.2.7 transition_metrics_record_context() . . . 1867
7.214.2.8 transition_metrics_record_prefetch_hit() . . . 1868
7.214.2.9 transition_metrics_record_prefetch_miss() . . . 1868
7.214.2.10 transition_metrics_reset() . . . 1869
7.214.2.11 transition_metrics_should_speculate() . . . 1870

7.214.2.12 transition_metrics_stats_string() . . . 1871
7.214.2.13 transition_metrics_update_cache() . . . 1872
7.214.2.14 transition_metrics_update_context_window() . . . 1873
7.215 v3/include/physics_runtime.h File Reference . . . 1874
7.215.1 Macro Definition Documentation . . . 1876
7.215.1.1 HOLA_PROTOCOL_MAJOR . . . 1876
7.215.1.2 HOLA_PROTOCOL_MINOR . . . 1876
7.215.1.3 HOLA_SHARED_MAGIC . . . 1876
7.215.1.4 PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT . . . 1876
7.215.1.5 PHYSICS_ANALYTICS_DEFAULT_HEAP_BYTES . . . 1876
7.215.1.6 PHYSICS_HOST_FLAG_CGROUP . . . 1876
7.215.1.7 PHYSICS_HOST_FLAG_CPU_STAT . . . 1877
7.215.1.8 PHYSICS_HOST_FLAG_PSI . . . 1877
7.215.2 Typedef Documentation . . . 1877
7.215.2.1 physics_analytics_event_header_t . . . 1877
7.215.2.2 physics_analytics_header_t . . . 1877
7.215.2.3 physics_analytics_heap_t . . . 1877
7.215.2.4 physics_host_backend_t . . . 1877
7.215.2.5 physics_host_snapshot_t . . . 1877
7.215.3 Enumeration Type Documentation . . . 1877
7.215.3.1 physics_host_backend . . . 1877
7.215.4 Function Documentation . . . 1878
7.215.4.1 physics_analytics_heap_info() . . . 1878
7.215.4.2 physics_analytics_publish_event() . . . 1878
7.215.4.3 physics_host_snapshot() . . . 1879
7.215.4.4 physics_runtime_init() . . . 1880
7.215.4.5 physics_runtime_shutdown() . . . 1881
7.216 v3/include/platform_alloc.h File Reference . . . 1882
7.216.1 Typedef Documentation . . . 1883
7.216.1.1 sf_alloc_stats_t . . . 1883
7.216.2 Function Documentation . . . 1883
7.216.2.1 sf_alloc_get_stats() . . . 1883
7.216.2.2 sf_alloc_init() . . . 1884
7.216.2.3 sf_calloc() . . . 1885
7.216.2.4 sf_free() . . . 1886
7.216.2.5 sf_malloc() . . . 1887
7.216.2.6 sf_realloc() . . . 1888
7.217 v3/include/platform_io.h File Reference . . . 1891
7.217.1 Function Documentation . . . 1891
7.217.1.1 sf_terminal_ready() . . . 1891
7.218 v3/include/platform_lock.h File Reference . . . 1892
7.218.1 Typedef Documentation . . . 1892

7.218.1.1 sf_mutex_t . . . 1892
7.218.2 Function Documentation . . . 1893
7.218.2.1 sf_mutex_destroy() . . . 1893
7.218.2.2 sf_mutex_init() . . . 1893
7.218.2.3 sf_mutex_lock() . . . 1894
7.218.2.4 sf_mutex_unlock() . . . 1895
7.219 v3/include/platform_time.h File Reference . . . 1897
7.219.1 Macro Definition Documentation . . . 1898
7.219.1.1 SF_TIME_STAMP_SIZE . . . 1898
7.219.2 Typedef Documentation . . . 1898
7.219.2.1 sf_time_ns_t . . . 1898
7.219.3 Function Documentation . . . 1898
7.219.3.1 sf_format_timestamp() . . . 1898
7.219.3.2 sf_has_rtc() . . . 1899
7.219.3.3 sf_monotonic_ns() . . . 1900
7.219.3.4 sf_ns_to_ms() . . . 1901
7.219.3.5 sf_ns_to_seconds() . . . 1901
7.219.3.6 sf_ns_to_us() . . . 1901
7.219.3.7 sf_realtime_ns() . . . 1901
7.219.3.8 sf_seconds_to_ns() . . . 1902
7.219.3.9 sf_set_realtime_ns() . . . 1902
7.219.3.10 sf_time_init() . . . 1903
7.219.4 Variable Documentation . . . 1903
7.219.4.1 sf_time_backend . . . 1903
7.219.4.2 sf_time_backend_l4re . . . 1903
7.219.4.3 sf_time_backend_posix . . . 1903
7.220 v3/include/profiler.h File Reference . . . 1904
7.220.1 Macro Definition Documentation . . . 1906
7.220.1.1 PROFILE_ENABLED . . . 1906
7.220.1.2 PROFILE_END . . . 1906
7.220.1.3 PROFILE_INC_CYCLES . . . 1906
7.220.1.4 PROFILE_INC_DICT_LOOKUP . . . 1906
7.220.1.5 PROFILE_INC_STACK_OP . . . 1906
7.220.1.6 PROFILE_MEMORY_READ . . . 1906
7.220.1.7 PROFILE_MEMORY_WRITE . . . 1906
7.220.1.8 PROFILE_START . . . 1906
7.220.1.9 PROFILE_WORD_ENTER . . . 1907
7.220.1.10 PROFILE_WORD_EXIT . . . 1907
7.220.2 Typedef Documentation . . . 1907
7.220.2.1 DictEntry . . . 1907
7.220.2.2 VM . . . 1907
7.220.3 Enumeration Type Documentation . . . 1907

7.220.3.1 ProfileCategory . . . 1907
7.220.3.2 ProfileLevel . . . 1908
7.220.4 Function Documentation . . . 1908
7.220.4.1 profiler_analyze_performance() . . . 1908
7.220.4.2 profiler_end_timer() . . . 1908
7.220.4.3 profiler_generate_report() . . . 1908
7.220.4.4 profiler_get_time_ns() . . . 1909
7.220.4.5 profiler_inc_branch() . . . 1910
7.220.4.6 profiler_inc_dict_lookup() . . . 1910
7.220.4.7 profiler_inc_loop_iter() . . . 1910
7.220.4.8 profiler_inc_stack_op() . . . 1910
7.220.4.9 profiler_inc_vm_cycles() . . . 1910
7.220.4.10 profiler_init() . . . 1910
7.220.4.11 profiler_memory_alloc() . . . 1911
7.220.4.12 profiler_memory_read() . . . 1911
7.220.4.13 profiler_memory_write() . . . 1911
7.220.4.14 profiler_print_hotspots() . . . 1911
7.220.4.15 profiler_print_memory_analysis() . . . 1912
7.220.4.16 profiler_print_summary() . . . 1912
7.220.4.17 profiler_print_word_analysis() . . . 1913
7.220.4.18 profiler_reset() . . . 1913
7.220.4.19 profiler_set_level() . . . 1913
7.220.4.20 profiler_shutdown() . . . 1914
7.220.4.21 profiler_start_timer() . . . 1914
7.220.4.22 profiler_suggest_optimizations() . . . 1914
7.220.4.23 profiler_word_count() . . . 1915
7.220.4.24 profiler_word_enter() . . . 1916
7.220.4.25 profiler_word_exit() . . . 1916
7.220.5 Variable Documentation . . . 1917
7.220.5.1 g_profiler . . . 1917
7.221 kernel/include/starkernel/q48_16.h File Reference . . . 1918
7.221.1 Macro Definition Documentation . . . 1919
7.221.1.1 Q48_ONE . . . 1919
7.221.2 Typedef Documentation . . . 1919
7.221.2.1 q48_16_t . . . 1919
7.221.3 Function Documentation . . . 1919
7.221.3.1 q48_abs() . . . 1919
7.221.3.2 q48_add() . . . 1920
7.221.3.3 q48_cos_approx() . . . 1920
7.221.3.4 q48_div() . . . 1921
7.221.3.5 q48_exp_approx() . . . 1921
7.221.3.6 q48_from_u64() . . . 1922

7.221.3.7 q48_log_approx() . . . 1923
7.221.3.8 q48_mul() . . . 1923
7.221.3.9 q48_sin_approx() . . . 1924
7.221.3.10 q48_sqrt_approx() . . . 1925
7.221.3.11 q48_sub() . . . 1926
7.221.3.12 q48_to_u64() . . . 1926
7.222 v3/include/q48_16.h File Reference . . . 1926
7.222.1 Typedef Documentation . . . 1928
7.222.1.1 q48_16_t . . . 1928
7.222.2 Function Documentation . . . 1928
7.222.2.1 q48_abs() . . . 1928
7.222.2.2 q48_add() . . . 1928
7.222.2.3 q48_cos_approx() . . . 1929
7.222.2.4 q48_div() . . . 1930
7.222.2.5 q48_exp_approx() . . . 1931
7.222.2.6 q48_from_double() . . . 1932
7.222.2.7 q48_from_u64() . . . 1933
7.222.2.8 q48_is_valid() . . . 1934
7.222.2.9 q48_log_approx() . . . 1934
7.222.2.10 q48_mul() . . . 1935
7.222.2.11 q48_sin_approx() . . . 1936
7.222.2.12 q48_sqrt_approx() . . . 1937
7.222.2.13 q48_sub() . . . 1938
7.222.2.14 q48_to_double() . . . 1939
7.222.2.15 q48_to_string() . . . 1939
7.222.2.16 q48_to_u64() . . . 1940
7.223 kernel/include/starkernel/freestanding/README.md File Reference . . . 1943
7.224 kernel/include/starkernel/freestanding/sys/README.md File Reference . . . 1943
7.225 kernel/include/starkernel/hal/README.md File Reference . . . 1943
7.226 kernel/include/starkernel/README.md File Reference . . . 1943
7.227 kernel/include/starkernel/vm/bootstrap/README.md File Reference . . . 1943
7.228 kernel/include/starkernel/vm/README.md File Reference . . . 1943
7.229 kernel/src/arch/aarch64/README.md File Reference . . . 1943
7.230 kernel/src/arch/amd64/README.md File Reference . . . 1943
7.231 kernel/src/arch/riscv64/README.md File Reference . . . 1943
7.232 kernel/src/boot/README.md File Reference . . . 1943
7.233 kernel/src/capsule/README.md File Reference . . . 1943
7.234 kernel/src/hal/README.md File Reference . . . 1943
7.235 kernel/src/hash/README.md File Reference . . . 1943
7.236 kernel/src/math/README.md File Reference . . . 1943
7.237 kernel/src/memory/README.md File Reference . . . 1943
7.238 kernel/src/pci/README.md File Reference . . . 1943

7.239 kernel/src/README.md File Reference . . . 1943
7.240 kernel/src/virtio/README.md File Reference . . . 1943
7.241 kernel/src/vm/bootstrap/README.md File Reference . . . 1943
7.242 kernel/src/vm/host/README.md File Reference . . . 1943
7.243 kernel/src/vm/README.md File Reference . . . 1943
7.244 README.md File Reference . . . 1943
7.245 v3/include/README.md File Reference . . . 1943
7.246 v3/src/platform/l4re/README.md File Reference . . . 1943
7.247 v3/src/platform/linux/README.md File Reference . . . 1943
7.248 v3/src/platform/README.md File Reference . . . 1943
7.249 v3/src/README.md File Reference . . . 1943
7.250 v3/src/test_runner/include/README.md File Reference . . . 1943
7.251 v3/src/test_runner/modules/README.md File Reference . . . 1943
7.252 v3/src/test_runner/README.md File Reference . . . 1943
7.253 v3/src/word_source/include/README.md File Reference . . . 1943
7.254 v3/src/word_source/README.md File Reference . . . 1943
7.255 kernel/include/starkernel/repl.h File Reference . . . 1943
7.255.1 Function Documentation . . . 1945
7.255.1.1 sk_console_getkey() . . . 1945
7.255.1.2 sk_console_key_available() . . . 1946
7.255.1.3 sk_console_readline() . . . 1947
7.255.1.4 sk_repl() . . . 1949
7.255.1.5 sk_repl_get_active_vm() . . . 1951
7.255.1.6 sk_repl_get_attached_blk_dev() . . . 1951
7.255.1.7 sk_repl_get_homeblocks_dev() . . . 1951
7.255.1.8 sk_repl_get_homeblocks_sig() . . . 1952
7.255.1.9 sk_repl_headless_wait() . . . 1952
7.255.1.10 sk_repl_register_words() . . . 1954
7.255.1.11 sk_repl_run() . . . 1955
7.255.1.12 sk_repl_set_active_vm() . . . 1956
7.255.1.13 sk_repl_step() . . . 1957
7.256 v3/include/repl.h File Reference . . . 1958
7.256.1 Function Documentation . . . 1959
7.256.1.1 vm_repl() . . . 1959
7.257 v3/include/rolling_window_knobs.h File Reference . . . 1961
7.258 v3/include/rolling_window_of_truth.h File Reference . . . 1962
7.258.1 Function Documentation . . . 1963
7.258.1.1 rolling_window_check_adaptive_shrink() . . . 1963
7.258.1.2 rolling_window_cleanup() . . . 1963
7.258.1.3 rolling_window_count_transition() . . . 1964
7.258.1.4 rolling_window_export_execution_history() . . . 1965
7.258.1.5 rolling_window_find_hottest_word() . . . 1966

7.258.1.6 rolling_window_get_recent_sequence() . . . 1967
7.258.1.7 rolling_window_init() . . . 1969
7.258.1.8 rolling_window_is_warm() . . . 1970
7.258.1.9 rolling_window_measure_diversity() . . . 1970
7.258.1.10 rolling_window_pattern_capture_rate() . . . 1972
7.258.1.11 rolling_window_record_execution() . . . 1973
7.258.1.12 rolling_window_reset() . . . 1973
7.258.1.13 rolling_window_service() . . . 1975
7.258.1.14 rolling_window_stats_string() . . . 1976
7.259 v3/include/ssm_jacquard.h File Reference . . . 1977
7.259.1 Macro Definition Documentation . . . 1978
7.259.1.1 SSM_CFG_L1 . . . 1978
7.259.1.2 SSM_CFG_L2 . . . 1979
7.259.1.3 SSM_CFG_L3 . . . 1979
7.259.1.4 SSM_CFG_L4 . . . 1979
7.259.1.5 SSM_CFG_L5 . . . 1979
7.259.1.6 SSM_CFG_L6 . . . 1979
7.259.1.7 SSM_CFG_L7 . . . 1979
7.259.2 Enumeration Type Documentation . . . 1979
7.259.2.1 ssm_l8_mode_t . . . 1979
7.259.3 Function Documentation . . . 1980
7.259.3.1 ssm_apply_mode() . . . 1980
7.259.3.2 ssm_apply_mode_from_table() . . . 1980
7.259.3.3 ssm_l8_force_config() . . . 1981
7.259.3.4 ssm_l8_free_table() . . . 1982
7.259.3.5 ssm_l8_init() . . . 1983
7.259.3.6 ssm_l8_init_table() . . . 1984
7.259.3.7 ssm_l8_mode_name() . . . 1985
7.259.3.8 ssm_l8_update() . . . 1986
7.259.3.9 ssm_l8_update_table() . . . 1987
7.260 v3/include/starforth_config.h File Reference . . . 1988
7.260.1 Macro Definition Documentation . . . 1990
7.260.1.1 ADAPTIVE_CHECK_FREQUENCY . . . 1990
7.260.1.2 ADAPTIVE_GROWTH_THRESHOLD . . . 1990
7.260.1.3 ADAPTIVE_MIN_WINDOW_SIZE . . . 1990
7.260.1.4 ADAPTIVE_SHRINK_RATE . . . 1990
7.260.1.5 DECAY_MIN_INTERVAL . . . 1990
7.260.1.6 DECAY_RATE_PER_US_Q16 . . . 1990
7.260.1.7 ENABLE_HOTWORDS_CACHE . . . 1990
7.260.1.8 ENABLE_PIPELINING . . . 1991
7.260.1.9 HEARTBEAT_CHECK_FREQUENCY . . . 1991
7.260.1.10 HEARTBEAT_INFERENCE_FREQUENCY . . . 1991

7.260.1.11 HEARTBEAT_SLOPE_VALIDATION_FREQUENCY 1991
7.260.1.12 HEARTBEAT_THREAD_ENABLED 1991
7.260.1.13 HEARTBEAT_TICK_NS 1991
7.260.1.14 HEARTBEAT_WINDOW_TUNING_FREQUENCY 1991
7.260.1.15 INITIAL_DECAY_SLOPE_Q48 1991
7.260.1.16 MIN_SAMPLES_FOR_SPECULATION 1991
7.260.1.17 MINIMUM_PREFETCH_ROI 1991
7.260.1.18 MISPREDICTION_COST_Q48 1992
7.260.1.19 ROLLING_WINDOW_SIZE 1992
7.260.1.20 SPECULATION_DEPTH 1992
7.260.1.21 SPECULATION_THRESHOLD_Q48 1992
7.260.1.22 SSM_CV_HIGH_THRESHOLD 1992
7.260.1.23 SSM_ENTROPY_HIGH_THRESHOLD 1992
7.260.1.24 SSM_HYSTERESIS_TICKS 1992
7.260.1.25 SSM_TEMPORAL_DECAY_LOW_THRESHOLD 1992
7.260.1.26 SSM_TEMPORAL_DECAY_THRESHOLD 1992
7.260.1.27 STADIUM_BLOCK_COOL_RATE_Q48 1992
7.260.1.28 STADIUM_BLOCK_HEAT_QUANTUM 1993
7.260.1.29 STADIUM_BLOCK_TRACK_CAP_MULT 1993
7.260.1.30 STADIUM_CAPACITY_TICK 1993
7.260.1.31 STADIUM_CONTAINS_DEPTH_MAX 1993
7.260.1.32 STADIUM_MEMORY_PERCENT 1993
7.260.1.33 STADIUM_VM_MEMORY_PERCENT 1993
7.260.1.34 STADIUM_WORD_COOL_RATE_Q48 1993
7.260.1.35 STADIUM_WORD_HEAT_QUANTUM 1993
7.260.1.36 STARFORTH_CONFIG_ADAPTIVE_CHECK_FREQUENCY_DEFAULT 1993
7.260.1.37 STARFORTH_CONFIG_ADAPTIVE_GROWTH_THRESHOLD_DEFAULT . . . 1993
7.260.1.38 STARFORTH_CONFIG_ADAPTIVE_MIN_WINDOW_SIZE_DEFAULT 1994
7.260.1.39 STARFORTH_CONFIG_ADAPTIVE_SHRINK_RATE_DEFAULT 1994
7.260.1.40 STARFORTH_CONFIG_DECAY_MIN_INTERVAL_DEFAULT 1994
7.260.1.41 STARFORTH_CONFIG_DECAY_RATE_PER_US_Q16_DEFAULT 1994
7.260.1.42 STARFORTH_CONFIG_ENABLE_HOTWORDS_CACHE_DEFAULT 1994
7.260.1.43 STARFORTH_CONFIG_ENABLE_PIPELINING_DEFAULT 1994
7.260.1.44 STARFORTH_CONFIG_HEARTBEAT_CHECK_FREQUENCY_DEFAULT . . . 1994
7.260.1.45 STARFORTH_CONFIG_HEARTBEAT_INFERENCE_FREQUENCY_DEFAULT 1994
7.260.1.46 STARFORTH_CONFIG_HEARTBEAT_SLOPE_VALIDATION_FREQUENCY_DEFAULT 1994
7.260.1.47 STARFORTH_CONFIG_HEARTBEAT_THREAD_ENABLED_DEFAULT 1994
7.260.1.48 STARFORTH_CONFIG_HEARTBEAT_TICK_NS_DEFAULT 1995
7.260.1.49 STARFORTH_CONFIG_HEARTBEAT_WINDOW_TUNING_FREQUENCY_DEFAULT 1995
7.260.1.50 STARFORTH_CONFIG_INITIAL_DECAY_SLOPE_Q48_DEFAULT 1995
7.260.1.51 STARFORTH_CONFIG_MIN_SAMPLES_FOR_SPECULATION_DEFAULT . . 1995
7.260.1.52 STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT 1995

7.260.1.53 STARFORTH_CONFIG_MISPREDICTION_COST_Q48_DEFAULT 1995
7.260.1.54 STARFORTH_CONFIG_ROLLING_WINDOW_SIZE_DEFAULT 1995
7.260.1.55 STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT 1995
7.260.1.56 STARFORTH_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT 1995
7.260.1.57 STARFORTH_CONFIG_SSM_CV_HIGH_THRESHOLD_DEFAULT 1995
7.260.1.58 STARFORTH_CONFIG_SSM_ENTROPY_HIGH_THRESHOLD_DEFAULT . . 1996
7.260.1.59 STARFORTH_CONFIG_SSM_HYSTERESIS_TICKS_DEFAULT 1996
7.260.1.60 STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_LOW_THRESHOLD_DEFAULT 1996
7.260.1.61 STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_THRESHOLD_DEFAULT . 1996
7.260.1.62 STARFORTH_CONFIG_STADIUM_BLOCK_COOL_RATE_Q48_DEFAULT . . 1996
7.260.1.63 STARFORTH_CONFIG_STADIUM_BLOCK_HEAT_QUANTUM_DEFAULT . . . 1996
7.260.1.64 STARFORTH_CONFIG_STADIUM_BLOCK_TRACK_CAP_MULT_DEFAULT . . 1996
7.260.1.65 STARFORTH_CONFIG_STADIUM_CAPACITY_TICK_DEFAULT 1996
7.260.1.66 STARFORTH_CONFIG_STADIUM_CONTAINS_DEPTH_MAX_DEFAULT . . . 1996
7.260.1.67 STARFORTH_CONFIG_STADIUM_MEMORY_PERCENT_DEFAULT 1996
7.260.1.68 STARFORTH_CONFIG_STADIUM_VM_MEMORY_PERCENT_DEFAULT . . . 1997
7.260.1.69 STARFORTH_CONFIG_STADIUM_WORD_COOL_RATE_Q48_DEFAULT . . . 1997
7.260.1.70 STARFORTH_CONFIG_STADIUM_WORD_HEAT_QUANTUM_DEFAULT . . . 1997
7.260.1.71 STARFORTH_CONFIG_STRICT_PTR_DEFAULT 1997
7.260.1.72 STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT 1997
7.260.1.73 STRICT_PTR 1997
7.260.1.74 TRANSITION_WINDOW_SIZE 1997
7.260.1.75 VM_STRICT_PTR 1997
7.261 v3/include/startup.h File Reference 1998
7.261.1 Function Documentation 1998
7.261.1.1 startup_blkio_cleanup() 1998
7.261.1.2 startup_blkio_init() 1998
7.261.1.3 startup_block_subsys_init() 1999
7.261.1.4 startup_vm_init() 1999
7.262 v3/include/version.h File Reference 2000
7.262.1 Macro Definition Documentation 2000
7.262.1.1 LITHOS_VERSION 2000
7.262.1.2 LITHOS_VERSION_STR 2000
7.262.1.3 STARFORTH_ARCH 2000
7.262.1.4 STARFORTH_TARGET 2000
7.262.1.5 STARFORTH_TIMESTAMP 2001
7.262.1.6 STARFORTH_VERSION 2001
7.262.1.7 STARFORTH_VERSION_FULL 2001
7.263 v3/include/vm.h File Reference 2001
7.263.1 Macro Definition Documentation 2006
7.263.1.1 ACL_MODE_STRICT 2006
7.263.1.2 ACL_MODE_TTL 2006

7.263.1.3 ACL_TTL_OPEN . . . 2006
7.263.1.4 ACL_USER_DEFAULT . . . 2006
7.263.1.5 BLK_VM_SLOTS . . . 2006
7.263.1.6 BLK_VM_WINDOW_BASE . . . 2006
7.263.1.7 BLK_VM_WINDOW_SIZE . . . 2006
7.263.1.8 BLOCK_SIZE . . . 2006
7.263.1.9 CF_STACK_MAX . . . 2006
7.263.1.10 CHARGE_NEUTRAL . . . 2006
7.263.1.11 COMPILE_BUFFER_SIZE . . . 2007
7.263.1.12 DICTIONARY_BLOCKS . . . 2007
7.263.1.13 DICTIONARY_MEMORY_SIZE . . . 2007
7.263.1.14 DICTIONARY_SIZE . . . 2007
7.263.1.15 HEARTBEAT_TICK_BUFFER_SIZE . . . 2007
7.263.1.16 HEAT_CACHE_DEMOTION_THRESHOLD . . . 2007
7.263.1.17 INPUT_BUFFER_SIZE . . . 2007
7.263.1.18 LIKELY . . . 2007
7.263.1.19 LOG_BLOCKS . . . 2007
7.263.1.20 LOG_BLOCKS_END . . . 2007
7.263.1.21 LOG_BLOCKS_START . . . 2008
7.263.1.22 LOG_LAYER1_MAX_LINES . . . 2008
7.263.1.23 LOG_LINE_MAX . . . 2008
7.263.1.24 LOG_LINES_PER_BLOCK . . . 2008
7.263.1.25 MAX_BLOCKS . . . 2008
7.263.1.26 PHYSICS_STATE_COMPILED . . . 2008
7.263.1.27 PHYSICS_STATE_HIDDEN . . . 2008
7.263.1.28 PHYSICS_STATE_IMMEDIATE . . . 2008
7.263.1.29 PHYSICS_STATE_PINNED . . . 2008
7.263.1.30 SPIN_IDLE . . . 2008
7.263.1.31 STACK_SIZE . . . 2009
7.263.1.32 UNLIKELY . . . 2009
7.263.1.33 USER_BLOCKS_START . . . 2009
7.263.1.34 VM_MEMORY_SIZE . . . 2009
7.263.1.35 VM_POP . . . 2009
7.263.1.36 VM_PUSH . . . 2009
7.263.1.37 VM_RPOP . . . 2009
7.263.1.38 VM_RPUSH . . . 2009
7.263.1.39 VOCAB_FC_BUCKETS . . . 2009
7.263.1.40 WORD_COMPILED . . . 2010
7.263.1.41 WORD_FROZEN . . . 2010
7.263.1.42 WORD_HIDDEN . . . 2010
7.263.1.43 WORD_ID_INVALID . . . 2010
7.263.1.44 WORD_IMMEDIATE . . . 2010

7.263.1.45 WORD_NAME_MAX . . . 2010
7.263.1.46 WORD_PINNED . . . 2010
7.263.1.47 WORD_SMUDGED . . . 2010
7.263.2 Typedef Documentation . . . 2010
7.263.2.1 cell_t . . . 2010
7.263.2.2 HeartbeatWorker . . . 2010
7.263.2.3 HotwordsCache . . . 2011
7.263.2.4 InferenceOutputs . . . 2011
7.263.2.5 ucell_t . . . 2011
7.263.2.6 vaddr_t . . . 2011
7.263.2.7 VM . . . 2011
7.263.2.8 VMHostServices . . . 2011
7.263.2.9 word_func_t . . . 2011
7.263.2.10 WordTransitionMetrics . . . 2011
7.263.3 Enumeration Type Documentation . . . 2011
7.263.3.1 cf_tag_t . . . 2011
7.263.3.2 vm_mode_t . . . 2012
7.263.4 Function Documentation . . . 2012
7.263.4.1 __attribute__() . . . 2012
7.263.4.2 CELL() . . . 2012
7.263.4.3 execute_colon_word() . . . 2013
7.263.4.4 heartbeat_capture_tick_snapshot() . . . 2014
7.263.4.5 heartbeat_emit_tick_row() . . . 2015
7.263.4.6 VM_ADDR() . . . 2016
7.263.4.7 vm_addr_ok() . . . 2017
7.263.4.8 vm_addr_to_block() . . . 2019
7.263.4.9 vm_align() . . . 2020
7.263.4.10 vm_allot() . . . 2020
7.263.4.11 vm_bootstrap_root_vocabulary() . . . 2021
7.263.4.12 vm_cleanup() . . . 2022
7.263.4.13 vm_compile_call() . . . 2023
7.263.4.14 vm_compile_exit() . . . 2024
7.263.4.15 vm_compile_literal() . . . 2025
7.263.4.16 vm_compile_word() . . . 2026
7.263.4.17 vm_create_word() . . . 2028
7.263.4.18 vm_dict_entry_ok() . . . 2029
7.263.4.19 vm_dict_resolve_in_bucket() . . . 2030
7.263.4.20 vm_dictionary_find_by_func() . . . 2031
7.263.4.21 vm_dictionary_find_latest_by_func() . . . 2032
7.263.4.22 vm_dictionary_get_data_field() . . . 2033
7.263.4.23 vm_dictionary_lookup_by_word_id() . . . 2033
7.263.4.24 vm_dictionary_track_entry() . . . 2034

7.263.4.25 vm_dictionary_untrack_entry() . . . 2035
7.263.4.26 vm_enable_interpreter() . . . 2036
7.263.4.27 vm_enter_compile_mode() . . . 2036
7.263.4.28 vm_exit_compile_mode() . . . 2039
7.263.4.29 vm_find_word() . . . 2039
7.263.4.30 vm_get_base() . . . 2041
7.263.4.31 vm_get_block_addr() . . . 2042
7.263.4.32 vm_heartbeat_start() . . . 2042
7.263.4.33 vm_hide_word() . . . 2042
7.263.4.34 vm_init() . . . 2043
7.263.4.35 vm_init_with_host() . . . 2046
7.263.4.36 vm_interpret() . . . 2048
7.263.4.37 vm_interpret_word() . . . 2049
7.263.4.38 vm_load_cell() . . . 2051
7.263.4.39 vm_load_u8() . . . 2052
7.263.4.40 vm_make_immediate() . . . 2053
7.263.4.41 vm_parse_number() . . . 2054
7.263.4.42 vm_parse_word() . . . 2055
7.263.4.43 vm_pin_execution_heat() . . . 2057
7.263.4.44 vm_pop() . . . 2058
7.263.4.45 vm_pop_fast() . . . 2059
7.263.4.46 vm_ptr() . . . 2059
7.263.4.47 vm_push() . . . 2061
7.263.4.48 vm_push_fast() . . . 2063
7.263.4.49 vm_repl() . . . 2063
7.263.4.50 vm_rpop() . . . 2063
7.263.4.51 vm_rpop_fast() . . . 2064
7.263.4.52 vm_rpush() . . . 2064
7.263.4.53 vm_rpush_fast() . . . 2065
7.263.4.54 vm_run_smoke_tests() . . . 2065
7.263.4.55 vm_set_base() . . . 2065
7.263.4.56 vm_smudge_word() . . . 2066
7.263.4.57 vm_snapshot_read() . . . 2067
7.263.4.58 vm_store_cell() . . . 2067
7.263.4.59 vm_store_u8() . . . 2068
7.263.4.60 vm_tick() . . . 2070
7.263.4.61 vm_tick_inference_engine() . . . 2071
7.263.4.62 vm_tick_slope_validator() . . . 2072
7.263.4.63 vm_tick_window_tuner() . . . 2073
7.263.4.64 vm_unpin_execution_heat() . . . 2074
7.263.4.65 vm_zuse_cert_install() . . . 2074
7.263.5 Variable Documentation . . . 2075

7.263.5.1 __attribute__ . . . 2075
7.263.5.2 acl_allow . . . 2075
7.263.5.3 acl_default . . . 2075
7.263.5.4 acl_mode . . . 2075
7.263.5.5 acl_pinned . . . 2075
7.263.5.6 acl_ttl . . . 2075
7.263.5.7 execution_heat . . . 2076
7.263.5.8 flags . . . 2076
7.263.5.9 func . . . 2076
7.263.5.10 link . . . 2076
7.263.5.11 name . . . 2076
7.263.5.12 name_len . . . 2076
7.263.5.13 physics . . . 2077
7.263.5.14 transition_metrics . . . 2077
7.263.5.15 word_id . . . 2077
7.264 v3/include/vm_api.h File Reference . . . 2077
7.264.1 Detailed Description . . . 2078
7.265 v3/include/vm_asm_opt.h File Reference . . . 2078
7.266 v3/include/vm_asm_opt_arm64.h File Reference . . . 2080
7.266.1 Function Documentation . . . 2081
7.266.1.1 vm_abs_asm() . . . 2081
7.266.1.2 vm_add_check_overflow() . . . 2081
7.266.1.3 vm_clz() . . . 2082
7.266.1.4 vm_ctz() . . . 2082
7.266.1.5 vm_dc_zva() . . . 2082
7.266.1.6 vm_divmod() . . . 2082
7.266.1.7 vm_has_neon() . . . 2082
7.266.1.8 vm_is_cortex_a72() . . . 2082
7.266.1.9 vm_max_asm() . . . 2083
7.266.1.10 vm_memcpy_asm() . . . 2083
7.266.1.11 vm_memzero_asm() . . . 2083
7.266.1.12 vm_min_asm() . . . 2083
7.266.1.13 vm_mul_double() . . . 2083
7.266.1.14 vm_pop_asm() . . . 2084
7.266.1.15 vm_popcnt() . . . 2084
7.266.1.16 vm_prefetch() . . . 2084
7.266.1.17 vm_prefetch_stream() . . . 2084
7.266.1.18 vm_push_asm() . . . 2084
7.266.1.19 vm_read_midr() . . . 2085
7.266.1.20 vm_rpop_asm() . . . 2085
7.266.1.21 vm_rpush_asm() . . . 2085
7.266.1.22 vm_strcmp_asm() . . . 2086

7.266.1.23 vm_strcmp_short_asm() . . . 2086
7.266.1.24 vm_sub_check_overflow() . . . 2087
7.267 v3/include/vm_asm_opt_riscv64.h File Reference . . . 2087
7.267.1 Function Documentation . . . 2088
7.267.1.1 vm_abs_asm() . . . 2088
7.267.1.2 vm_add_check_overflow() . . . 2089
7.267.1.3 vm_clz() . . . 2089
7.267.1.4 vm_ctz() . . . 2089
7.267.1.5 vm_divmod() . . . 2089
7.267.1.6 vm_has_riscv_b_ext() . . . 2089
7.267.1.7 vm_has_riscv_m_ext() . . . 2089
7.267.1.8 vm_max_asm() . . . 2089
7.267.1.9 vm_memcpy_asm() . . . 2089
7.267.1.10 vm_memzero_asm() . . . 2090
7.267.1.11 vm_min_asm() . . . 2090
7.267.1.12 vm_mul_double() . . . 2090
7.267.1.13 vm_pop_asm() . . . 2090
7.267.1.14 vm_popcnt() . . . 2090
7.267.1.15 vm_prefetch() . . . 2090
7.267.1.16 vm_prefetch_stream() . . . 2090
7.267.1.17 vm_push_asm() . . . 2091
7.267.1.18 vm_rpop_asm() . . . 2091
7.267.1.19 vm_rpush_asm() . . . 2091
7.267.1.20 vm_strcmp_asm() . . . 2091
7.267.1.21 vm_sub_check_overflow() . . . 2091
7.268 v3/include/vm_debug.h File Reference . . . 2092
7.268.1 Function Documentation . . . 2092
7.268.1.1 vm_debug_dump_state() . . . 2092
7.268.1.2 vm_debug_install_signal_handlers() . . . 2093
7.268.1.3 vm_debug_set_current_vm() . . . 2094
7.269 v3/include/vm_host.h File Reference . . . 2095
7.269.1 Macro Definition Documentation . . . 2097
7.269.1.1 SK_ALLOC . . . 2097
7.269.1.2 SK_ALLOC_ALIGNED . . . 2097
7.269.1.3 SK_FREE . . . 2097
7.269.1.4 SK_MUTEX_DESTROY . . . 2097
7.269.1.5 SK_MUTEX_INIT . . . 2097
7.269.1.6 SK_MUTEX_LOCK . . . 2097
7.269.1.7 SK_MUTEX_UNLOCK . . . 2097
7.269.1.8 SK_PUTC . . . 2098
7.269.1.9 SK_PUTS . . . 2098
7.269.1.10 SK_TIME_NS . . . 2098

7.269.2 Typedef Documentation . . . 2098
7.269.2.1 sk_alloc_fn . . . 2098
7.269.2.2 sk_free_fn . . . 2098
7.269.2.3 sk_mutex_destroy_fn . . . 2098
7.269.2.4 sk_mutex_init_fn . . . 2098
7.269.2.5 sk_mutex_lock_fn . . . 2098
7.269.2.6 sk_mutex_unlock_fn . . . 2099
7.269.2.7 sk_panic_fn . . . 2099
7.269.2.8 sk_putc_fn . . . 2099
7.269.2.9 sk_puts_fn . . . 2099
7.269.2.10 sk_time_fn . . . 2099
7.269.2.11 sk_xt_entry_owns_fn . . . 2099
7.269.2.12 sk_xt_validate_fn . . . 2099
7.269.2.13 VMHostServices . . . 2099
7.269.3 Function Documentation . . . 2100
7.269.3.1 sk_host_services() . . . 2100
7.270 v3/include/vm_inner_interp_arm64.h File Reference . . . 2100
7.270.1 Macro Definition Documentation . . . 2101
7.270.1.1 NEXT_ARM64 . . . 2101
7.270.2 Function Documentation . . . 2102
7.270.2.1 vm_docol_arm64() . . . 2102
7.270.2.2 vm_exit_arm64() . . . 2102
7.270.2.3 vm_save_registers_arm64() . . . 2102
7.270.2.4 vm_setup_registers_arm64() . . . 2102
7.271 v3/include/vm_inner_interp_asm.h File Reference . . . 2102
7.271.1 Macro Definition Documentation . . . 2104
7.271.1.1 NEXT_ASM . . . 2104
7.271.1.2 PRIM_2DROP . . . 2104
7.271.1.3 PRIM_2DUP . . . 2105
7.271.1.4 PRIM_AND . . . 2105
7.271.1.5 PRIM_BRANCH . . . 2105
7.271.1.6 PRIM_C_FETCH . . . 2105
7.271.1.7 PRIM_C_STORE . . . 2105
7.271.1.8 PRIM_DROP . . . 2106
7.271.1.9 PRIM_DUP . . . 2106
7.271.1.10 PRIM_EQUALS . . . 2106
7.271.1.11 PRIM_EXECUTE . . . 2106
7.271.1.12 PRIM_FETCH . . . 2106
7.271.1.13 PRIM_GREATER . . . 2107
7.271.1.14 PRIM_INVERT . . . 2107
7.271.1.15 PRIM_LESS . . . 2107
7.271.1.16 PRIM_MINUS . . . 2107

7.271.1.17 PRIM_MOD . . . 2107
7.271.1.18 PRIM_NEGATE . . . 2108
7.271.1.19 PRIM_OR . . . 2108
7.271.1.20 PRIM_OVER . . . 2108
7.271.1.21 PRIM_PLUS . . . 2108
7.271.1.22 PRIM_R_FETCH . . . 2108
7.271.1.23 PRIM_R_FROM . . . 2109
7.271.1.24 PRIM_ROT . . . 2109
7.271.1.25 PRIM_SLASH . . . 2109
7.271.1.26 PRIM_STAR . . . 2109
7.271.1.27 PRIM_STORE . . . 2109
7.271.1.28 PRIM_SWAP . . . 2110
7.271.1.29 PRIM_TO_R . . . 2110
7.271.1.30 PRIM_XOR . . . 2110
7.271.1.31 PRIM_ZBRANCH . . . 2110
7.271.1.32 PRIM_ZERO_EQUALS . . . 2110
7.271.1.33 PRIM_ZERO_LESS . . . 2111
7.271.2 Function Documentation . . . 2111
7.271.2.1 vm_docol_asm() . . . 2111
7.271.2.2 vm_exit_asm() . . . 2111
7.271.2.3 vm_save_registers() . . . 2111
7.271.2.4 vm_setup_registers() . . . 2111
7.272 v3/include/vm_inner_interp_riscv64.h File Reference . . . 2112
7.272.1 Macro Definition Documentation . . . 2113
7.272.1.1 NEXT_RV64 . . . 2113
7.272.1.2 PRIM_2DROP_RV64 . . . 2113
7.272.1.3 PRIM_2DUP_RV64 . . . 2113
7.272.1.4 PRIM_AND_RV64 . . . 2114
7.272.1.5 PRIM_BRANCH_RV64 . . . 2114
7.272.1.6 PRIM_C_FETCH_RV64 . . . 2114
7.272.1.7 PRIM_C_STORE_RV64 . . . 2114
7.272.1.8 PRIM_DROP_RV64 . . . 2114
7.272.1.9 PRIM_DUP_RV64 . . . 2115
7.272.1.10 PRIM_EQUALS_RV64 . . . 2115
7.272.1.11 PRIM_EXECUTE_RV64 . . . 2115
7.272.1.12 PRIM_FETCH_RV64 . . . 2115
7.272.1.13 PRIM_GREATER_RV64 . . . 2115
7.272.1.14 PRIM_INVERT_RV64 . . . 2116
7.272.1.15 PRIM_LESS_RV64 . . . 2116
7.272.1.16 PRIM_MINUS_RV64 . . . 2116
7.272.1.17 PRIM_MOD_RV64 . . . 2116
7.272.1.18 PRIM_NEGATE_RV64 . . . 2116

7.272.1.19 PRIM_OR_RV64 . . . 2117
7.272.1.20 PRIM_OVER_RV64 . . . 2117
7.272.1.21 PRIM_PLUS_RV64 . . . 2117
7.272.1.22 PRIM_R_FETCH_RV64 . . . 2117
7.272.1.23 PRIM_R_FROM_RV64 . . . 2117
7.272.1.24 PRIM_ROT_RV64 . . . 2118
7.272.1.25 PRIM_SLASH_RV64 . . . 2118
7.272.1.26 PRIM_STAR_RV64 . . . 2118
7.272.1.27 PRIM_STORE_RV64 . . . 2118
7.272.1.28 PRIM_SWAP_RV64 . . . 2118
7.272.1.29 PRIM_TO_R_RV64 . . . 2119
7.272.1.30 PRIM_XOR_RV64 . . . 2119
7.272.1.31 PRIM_ZBRANCH_RV64 . . . 2119
7.272.1.32 PRIM_ZERO_EQUALS_RV64 . . . 2119
7.272.1.33 PRIM_ZERO_LESS_RV64 . . . 2119
7.272.2 Function Documentation . . . 2120
7.272.2.1 vm_docol_rv64() . . . 2120
7.272.2.2 vm_exit_rv64() . . . 2120
7.272.2.3 vm_save_registers_rv64() . . . 2120
7.272.2.4 vm_setup_registers_rv64() . . . 2120
7.273 v3/include/word_registry.h File Reference . . . 2120
7.273.1 Function Documentation . . . 2121
7.273.1.1 register_forth79_words() . . . 2121
7.273.1.2 register_word() . . . 2123
7.274 v3/src/blkio_factory.c File Reference . . . 2124
7.274.1 Function Documentation . . . 2125
7.274.1.1 blkio_factory_open() . . . 2125
7.274.1.2 blkio_file_init_state() . . . 2127
7.274.1.3 blkio_file_state_size() . . . 2129
7.274.1.4 blkio_file_vtable() . . . 2129
7.274.1.5 blkio_ram_init_state() . . . 2130
7.274.1.6 blkio_ram_state_size() . . . 2130
7.274.1.7 blkio_ram_vtable() . . . 2131
7.274.1.8 nonempty() . . . 2131
7.275 v3/src/blkio_file.c File Reference . . . 2132
7.275.1 Macro Definition Documentation . . . 2133
7.275.1.1 PATH_MAX . . . 2133
7.275.2 Function Documentation . . . 2133
7.275.2.1 blkio_file_init_state() . . . 2133
7.275.2.2 blkio_file_state_size() . . . 2134
7.275.2.3 blkio_file_vtable() . . . 2135
7.275.2.4 file_close() . . . 2135

7.275.2.5 file_flush() . . . 2135
7.275.2.6 file_info() . . . 2136
7.275.2.7 file_open() . . . 2137
7.275.2.8 file_read() . . . 2137
7.275.2.9 file_write() . . . 2138
7.275.2.10 full_read_at() . . . 2139
7.275.2.11 full_write_at() . . . 2139
7.275.3 Variable Documentation . . . 2140
7.275.3.1 BLKIO_FILE_VT . . . 2140
7.276 v3/src/blkio_ram.c File Reference . . . 2141
7.276.1 Function Documentation . . . 2142
7.276.1.1 blkio_ram_init_state() . . . 2142
7.276.1.2 blkio_ram_state_size() . . . 2143
7.276.1.3 blkio_ram_vtable() . . . 2143
7.276.1.4 in_bounds() . . . 2143
7.276.1.5 ram_close() . . . 2144
7.276.1.6 ram_flush() . . . 2144
7.276.1.7 ram_info() . . . 2145
7.276.1.8 ram_open() . . . 2145
7.276.1.9 ram_read() . . . 2146
7.276.1.10 ram_write() . . . 2147
7.276.2 Variable Documentation . . . 2147
7.276.2.1 BLKIO_RAM_VT . . . 2147
7.277 v3/src/block_subsystem.c File Reference . . . 2148
7.277.1 Macro Definition Documentation . . . 2151
7.277.1.1 BLK_DISK_SYS_RESERVED . . . 2151
7.277.1.2 BLK_FIRSTTOUCH_MAX_CLAIM . . . 2151
7.277.1.3 BLK_FORTH_SYS_RESERVED . . . 2151
7.277.1.4 BLK_RELOC_MAX_ENTRIES . . . 2151
7.277.1.5 DISK_CACHE_SLOTS . . . 2151
7.277.1.6 META_PER_BLOCK . . . 2151
7.277.1.7 META_REGION_OFFSET . . . 2151
7.277.1.8 META_REGION_SIZE . . . 2151
7.277.1.9 MIGRATION_SCAN_BUDGET . . . 2151
7.277.1.10 MIGRATION_WEAR_THRESHOLD . . . 2151
7.277.2 Typedef Documentation . . . 2152
7.277.2.1 blk_dev_slot_t . . . 2152
7.277.3 Function Documentation . . . 2152
7.277.3.1 __attribute__() . . . 2152
7.277.3.2 bam_flush_to_disk() . . . 2152
7.277.3.3 bam_pages_used() . . . 2153
7.277.3.4 bam_sync_from_disk() . . . 2154

7.277.3.5 blk_acl_allow_get() . . . 2155
7.277.3.6 blk_acl_allow_set() . . . 2155
7.277.3.7 blk_acl_ttl_get() . . . 2156
7.277.3.8 blk_acl_ttl_set() . . . 2157
7.277.3.9 blk_allocate() . . . 2157
7.277.3.10 blk_commit_format() . . . 2158
7.277.3.11 blk_compute_fresh_geometry() . . . 2159
7.277.3.12 blk_firsttouch_claim() . . . 2160
7.277.3.13 blk_flags_get() . . . 2161
7.277.3.14 blk_flags_set() . . . 2161
7.277.3.15 blk_flush() . . . 2162
7.277.3.16 blk_format_or_load_disk() . . . 2163
7.277.3.17 blk_get_buffer() . . . 2164
7.277.3.18 blk_get_device_free_blocks() . . . 2165
7.277.3.19 blk_get_device_range() . . . 2165
7.277.3.20 blk_get_empty_buffer() . . . 2166
7.277.3.21 blk_get_first_disk_range() . . . 2166
7.277.3.22 blk_get_meta() . . . 2167
7.277.3.23 blk_get_timestamp() . . . 2168
7.277.3.24 blk_get_total_blocks() . . . 2168
7.277.3.25 blk_get_volume_meta() . . . 2169
7.277.3.26 blk_is_allocated() . . . 2169
7.277.3.27 blk_is_valid() . . . 2170
7.277.3.28 blk_lbn_device_handle() . . . 2170
7.277.3.29 blk_mark_allocated() . . . 2171
7.277.3.30 blk_mark_free() . . . 2172
7.277.3.31 blk_meta_relocate_devblock() . . . 2172
7.277.3.32 blk_meta_zone_read() . . . 2173
7.277.3.33 blk_meta_zone_write() . . . 2174
7.277.3.34 blk_migration_idle_check() . . . 2175
7.277.3.35 blk_migration_idle_check_slot() . . . 2175
7.277.3.36 blk_owner_fp_get() . . . 2176
7.277.3.37 blk_owner_fp_set() . . . 2177
7.277.3.38 blk_owner_has_claim() . . . 2177
7.277.3.39 blk_set_meta() . . . 2178
7.277.3.40 blk_set_volume_meta() . . . 2179
7.277.3.41 blk_subsys_add_raw_device() . . . 2179
7.277.3.42 blk_subsys_attach_device() . . . 2179
7.277.3.43 blk_subsys_confirm_format() . . . 2180
7.277.3.44 blk_subsys_detach_device() . . . 2181
7.277.3.45 blk_subsys_epoch() . . . 2182
7.277.3.46 blk_subsys_init() . . . 2182

7.277.3.47 blk_subsys_relocate_block() . . . 2183
7.277.3.48 blk_subsys_set_format_bam_for() . . . 2184
7.277.3.49 blk_subsys_shutdown() . . . 2184
7.277.3.50 blk_update() . . . 2185
7.277.3.51 blk_volume_compute_totals() . . . 2186
7.277.3.52 cache_get_slot() . . . 2186
7.277.3.53 cache_load_devblock() . . . 2187
7.277.3.54 cache_writeback() . . . 2189
7.277.3.55 chain_append() . . . 2190
7.277.3.56 choose_B() . . . 2190
7.277.3.57 compute_crc64() . . . 2190
7.277.3.58 compute_totals_from_B() . . . 2191
7.277.3.59 crc64_init() . . . 2192
7.277.3.60 devblock4k_to_lba1k() . . . 2192
7.277.3.61 devblock_claimed_by_lbn() . . . 2193
7.277.3.62 devblock_is_free() . . . 2194
7.277.3.63 first_disk_slot() . . . 2194
7.277.3.64 lbn_to_slot() . . . 2195
7.277.3.65 lbn_to_slot_pbn() . . . 2196
7.277.3.66 meta_from_slice() . . . 2197
7.277.3.67 meta_to_slice() . . . 2198
7.277.3.68 minzu() . . . 2200
7.277.3.69 pbam_clr() . . . 2200
7.277.3.70 pbam_set() . . . 2201
7.277.3.71 pbam_test() . . . 2201
7.277.3.72 read_devblock_4k() . . . 2201
7.277.3.73 read_header_4k() . . . 2202
7.277.3.74 reloc_flush_to_disk() . . . 2203
7.277.3.75 reloc_load_from_disk() . . . 2204
7.277.3.76 resolve_lbn() . . . 2205
7.277.3.77 slot_by_dev() . . . 2206
7.277.3.78 slot_pbn_pack_offset() . . . 2206
7.277.3.79 slot_pbn_to_devblock() . . . 2207
7.277.3.80 udiv_floor() . . . 2208
7.277.3.81 volmeta_from_buf() . . . 2209
7.277.3.82 volmeta_to_buf() . . . 2209
7.277.3.83 write_devblock_4k() . . . 2210
7.277.3.84 write_header_4k() . . . 2211
7.277.4 Variable Documentation . . . 2212
7.277.4.1 crc64_inited . . . 2212
7.277.4.2 crc64_table . . . 2212
7.277.4.3 dirty_ram . . . 2212

7.277.4.4 epoch . . . 2212
7.277.4.5 [struct] . . . 2212
7.277.4.6 g_format_bam_for . . . 2213
7.277.4.7 head . . . 2213
7.277.4.8 initialized . . . 2213
7.277.4.9 ram_base . . . 2213
7.277.4.10 ram_size . . . 2213
7.277.4.11 ram_user . . . 2213
7.277.4.12 reloc . . . 2213
7.277.4.13 reloc_count . . . 2213
7.277.4.14 total_user_lbn . . . 2213
7.278 v3/src/cli.c File Reference . . . 2214
7.278.1 Function Documentation . . . 2214
7.278.1.1 cli_parse() . . . 2214
7.278.1.2 cli_print_usage() . . . 2216
7.278.1.3 parse_u32() . . . 2216
7.279 v3/src/compudynamics.c File Reference . . . 2217
7.279.1 Function Documentation . . . 2218
7.279.1.1 cd_classify_ids() . . . 2218
7.279.1.2 cd_config_table_force() . . . 2219
7.279.1.3 cd_config_table_free() . . . 2219
7.279.1.4 cd_config_table_init() . . . 2220
7.279.1.5 cd_config_table_seed() . . . 2221
7.279.1.6 cd_config_table_tick() . . . 2222
7.279.1.7 cd_isqrt32() . . . 2223
7.279.1.8 cd_isqrt64() . . . 2224
7.279.1.9 cd_ln_approx_q16() . . . 2224
7.279.1.10 cd_tuning_vm() . . . 2225
7.279.1.11 cd_tuning_word() . . . 2225
7.280 v3/src/dictionary_heat_optimization.c File Reference . . . 2226
7.280.1 Function Documentation . . . 2227
7.280.1.1 compare_dict_entry_heat_desc() . . . 2227
7.280.1.2 compare_heat_ascending() . . . 2228
7.280.1.3 dict_adaptive_optimization_pass() . . . 2228
7.280.1.4 dict_find_word_heat_aware() . . . 2230
7.280.1.5 dict_reorganize_buckets_by_heat() . . . 2232
7.280.1.6 dict_update_heat_percentiles() . . . 2234
7.281 v3/src/dictionary_management.c File Reference . . . 2235
7.281.1 Macro Definition Documentation . . . 2237
7.281.1.1 SF_FC_BUCKETS . . . 2237
7.281.2 Function Documentation . . . 2237
7.281.2.1 dict_newer_of() . . . 2237

7.281.2.2 sf_fc_clear_all() . . . 2238
7.281.2.3 sf_fc_reserve() . . . 2239
7.281.2.4 sf_rebuild_fc_index() . . . 2240
7.281.2.5 sf_try_fast_append() . . . 2242
7.281.2.6 sf_xrealloc() . . . 2244
7.281.2.7 vm_create_word() . . . 2246
7.281.2.8 vm_dict_entry_ok() . . . 2248
7.281.2.9 vm_dict_resolve_in_bucket() . . . 2249
7.281.2.10 vm_dictionary_acquire_word_id() . . . 2250
7.281.2.11 vm_dictionary_find_by_func() . . . 2251
7.281.2.12 vm_dictionary_find_latest_by_func() . . . 2252
7.281.2.13 vm_dictionary_get_data_field() . . . 2252
7.281.2.14 vm_dictionary_index_reset() . . . 2253
7.281.2.15 vm_dictionary_lookup_by_word_id() . . . 2254
7.281.2.16 vm_dictionary_track_entry() . . . 2254
7.281.2.17 vm_dictionary_untrack_entry() . . . 2255
7.281.2.18 vm_find_word() . . . 2256
7.281.2.19 vm_hide_word() . . . 2258
7.281.2.20 vm_pin_execution_heat() . . . 2259
7.281.2.21 vm_smudge_word() . . . 2260
7.281.2.22 vm_unpin_execution_heat() . . . 2260
7.281.3 Variable Documentation . . . 2261
7.281.3.1 sf_cached_latest . . . 2261
7.281.3.2 sf_fc_cap . . . 2261
7.281.3.3 sf_fc_count . . . 2261
7.281.3.4 sf_fc_list . . . 2261
7.282 v3/src/doe_metrics.c File Reference . . . 2261
7.282.1 Detailed Description . . . 2262
7.282.2 Function Documentation . . . 2262
7.282.2.1 extract_cache_metrics() . . . 2262
7.282.2.2 isqrt() . . . 2263
7.282.2.3 metrics_from_vm() . . . 2263
7.282.2.4 metrics_get_cpu_freq_mhz() . . . 2264
7.282.2.5 metrics_get_cpu_temp_c() . . . 2265
7.282.2.6 metrics_get_timestamp() . . . 2266
7.282.2.7 metrics_print_text() . . . 2266
7.282.2.8 metrics_write_csv_header() . . . 2267
7.282.2.9 metrics_write_csv_row() . . . 2268
7.282.2.10 metrics_write_james_law_csv_row() . . . 2268
7.282.3 Variable Documentation . . . 2269
7.282.3.1 bucket_hits . . . 2269
7.282.3.2 cache_hits . . . 2269

7.282.3.3 [struct] . . . 2269
7.282.3.4 total_lookups . . . 2269
7.283 v3/src/heartbeat_export.c File Reference . . . 2270
7.283.1 Function Documentation . . . 2270
7.283.1.1 heartbeat_capture_tick_snapshot() . . . 2270
7.283.1.2 heartbeat_emit_tick_row() . . . 2271
7.284 v3/src/inference_engine.c File Reference . . . 2272
7.284.1 Function Documentation . . . 2273
7.284.1.1 compute_fit_quality() . . . 2273
7.284.1.2 compute_levene_statistic() . . . 2275
7.284.1.3 compute_mean_q48() . . . 2276
7.284.1.4 compute_median_q48() . . . 2277
7.284.1.5 compute_variance_q48() . . . 2278
7.284.1.6 extract_heat_trajectory() . . . 2279
7.284.1.7 find_variance_inflection() . . . 2280
7.284.1.8 has_variance_stabilized() . . . 2282
7.284.1.9 infer_decay_slope_q48() . . . 2283
7.284.1.10 inference_engine_run() . . . 2284
7.284.1.11 inference_outputs_to_string() . . . 2286
7.284.1.12 inference_outputs_validate() . . . 2287
7.285 v3/src/io.c File Reference . . . 2288
7.285.1 Function Documentation . . . 2289
7.285.1.1 io_init() . . . 2289
7.285.1.2 io_write_block() . . . 2290
7.286 v3/src/platform/linux/io.c File Reference . . . 2291
7.286.1 Function Documentation . . . 2291
7.286.1.1 sf_terminal_ready() . . . 2291
7.287 v3/src/log.c File Reference . . . 2292
7.287.1 Function Documentation . . . 2292
7.287.1.1 get_timestamp() . . . 2292
7.287.1.2 log_get_level() . . . 2294
7.287.1.3 log_message() . . . 2295
7.287.1.4 log_set_level() . . . 2297
7.287.1.5 log_test_result() . . . 2297
7.287.2 Variable Documentation . . . 2299
7.287.2.1 color_reset . . . 2299
7.287.2.2 current_level . . . 2300
7.287.2.3 level_colors . . . 2300
7.287.2.4 level_names . . . 2300
7.287.2.5 test_fail_color . . . 2300
7.287.2.6 test_pass_color . . . 2300
7.287.2.7 test_skip_color . . . 2300

7.288 v3/src/main.c File Reference . . . 2301
7.288.1 Macro Definition Documentation . . . 2302
7.288.1.1 STARFORTH_STATE_BYTES . . . 2302
7.288.2 Function Documentation . . . 2302
7.288.2.1 __attribute__() . . . 2302
7.288.2.2 cleanup_and_exit() . . . 2302
7.288.2.3 cleanup_blkio() . . . 2303
7.288.2.4 init_blkio() . . . 2304
7.288.2.5 init_vm_and_subsystem() . . . 2305
7.288.2.6 main() . . . 2306
7.288.2.7 print_usage() . . . 2307
7.288.2.8 run_benchmark_mode() . . . 2308
7.288.2.9 run_doe_experiment() . . . 2309
7.288.2.10 run_optional_tests() . . . 2310
7.288.2.11 signal_handler() . . . 2310
7.288.3 Variable Documentation . . . 2311
7.288.3.1 g_blkio_opened . . . 2311
7.288.3.2 g_ram_base . . . 2311
7.288.3.3 g_ram_bytes . . . 2311
7.288.3.4 g_ram_is_mmap . . . 2311
7.289 v3/src/math_portable.c File Reference . . . 2312
7.289.1 Function Documentation . . . 2312
7.289.1.1 erf_q48() . . . 2312
7.289.1.2 exp_q48() . . . 2314
7.289.1.3 sqrt_q48() . . . 2315
7.290 v3/src/memory_management.c File Reference . . . 2316
7.290.1 Function Documentation . . . 2316
7.290.1.1 vm_addr_to_block() . . . 2316
7.290.1.2 vm_align() . . . 2317
7.290.1.3 vm_allot() . . . 2318
7.290.1.4 vm_get_block_addr() . . . 2320
7.291 v3/src/physics_execution_hooks.c File Reference . . . 2321
7.291.1 Function Documentation . . . 2322
7.291.1.1 physics_on_lookup() . . . 2322
7.291.1.2 physics_post_execute() . . . 2323
7.291.1.3 physics_pre_execute() . . . 2325
7.292 v3/src/physics_hotwords_cache.c File Reference . . . 2326
7.292.1 Function Documentation . . . 2328
7.292.1.1 calculate_posterior() . . . 2328
7.292.1.2 hotwords_bayesian_report() . . . 2329
7.292.1.3 hotwords_bucket_reorder() . . . 2331
7.292.1.4 hotwords_cache_cleanup() . . . 2331

7.292.1.5 hotwords_cache_evict_entry() . . . 2332
7.292.1.6 hotwords_cache_evict_name() . . . 2333
7.292.1.7 hotwords_cache_init() . . . 2334
7.292.1.8 hotwords_cache_lookup() . . . 2336
7.292.1.9 hotwords_cache_promote() . . . 2338
7.292.1.10 hotwords_cache_remove_at() . . . 2340
7.292.1.11 hotwords_cache_set_enabled() . . . 2341
7.292.1.12 hotwords_posterior_bucket_searches() . . . 2342
7.292.1.13 hotwords_posterior_cache_hits() . . . 2343
7.292.1.14 hotwords_speedup_estimate() . . . 2344
7.292.1.15 hotwords_stats_print() . . . 2345
7.292.1.16 hotwords_stats_reset() . . . 2346
7.292.1.17 q48_to_ns_double() . . . 2347
7.292.2 Variable Documentation . . . 2348
7.292.2.1 g_hotwords_cache . . . 2348
7.293 v3/src/physics_metadata.c File Reference . . . 2348
7.293.1 Typedef Documentation . . . 2349
7.293.1.1 physics_seed_definition_t . . . 2349
7.293.2 Function Documentation . . . 2349
7.293.2.1 clamp_u32() . . . 2349
7.293.2.2 derive_state_flags() . . . 2350
7.293.2.3 physics_metadata_apply_linear_decay() . . . 2351
7.293.2.4 physics_metadata_apply_seed() . . . 2352
7.293.2.5 physics_metadata_init() . . . 2354
7.293.2.6 physics_metadata_record_latency() . . . 2356
7.293.2.7 physics_metadata_refresh_state() . . . 2358
7.293.2.8 physics_metadata_set_mass() . . . 2359
7.293.2.9 physics_metadata_touch() . . . 2360
7.293.2.10 temperature_from_execution_heat() . . . 2361
7.293.3 Variable Documentation . . . 2362
7.293.3.1 physics_seed_table . . . 2362
7.294 v3/src/physics_pipelining_metrics.c File Reference . . . 2362
7.294.1 Function Documentation . . . 2364
7.294.1.1 ns_to_q48() . . . 2364
7.294.1.2 q48_div_u64() . . . 2364
7.294.1.3 q48_to_ns() . . . 2365
7.294.1.4 transition_metrics_binary_chop_suggest_window() . . . 2366
7.294.1.5 transition_metrics_cleanup() . . . 2366
7.294.1.6 transition_metrics_context_accuracy_string() . . . 2367
7.294.1.7 transition_metrics_get_probability_q48() . . . 2368
7.294.1.8 transition_metrics_init() . . . 2369
7.294.1.9 transition_metrics_record() . . . 2371

7.294.1.10 transition_metrics_record_context() . . . 2373
7.294.1.11 transition_metrics_record_prefetch_hit() . . . 2374
7.294.1.12 transition_metrics_record_prefetch_miss() . . . 2374
7.294.1.13 transition_metrics_reset() . . . 2375
7.294.1.14 transition_metrics_should_speculate() . . . 2376
7.294.1.15 transition_metrics_stats_string() . . . 2377
7.294.1.16 transition_metrics_update_cache() . . . 2378
7.294.1.17 transition_metrics_update_context_window() . . . 2379
7.295 v3/src/physics_runtime.c File Reference . . . 2380
7.295.1 Macro Definition Documentation . . . 2381
7.295.1.1 PATH_MAX . . . 2381
7.295.2 Typedef Documentation . . . 2382
7.295.2.1 physics_runtime_state_t . . . 2382
7.295.3 Function Documentation . . . 2382
7.295.3.1 align_up() . . . 2382
7.295.3.2 analytics_heap_allocate() . . . 2382
7.295.3.3 analytics_heap_release() . . . 2384
7.295.3.4 analytics_ring_free() . . . 2384
7.295.3.5 parse_avg_milli() . . . 2385
7.295.3.6 physics_analytics_heap_info() . . . 2386
7.295.3.7 physics_analytics_publish_event() . . . 2386
7.295.3.8 physics_host_snapshot() . . . 2387
7.295.3.9 physics_runtime_init() . . . 2388
7.295.3.10 physics_runtime_shutdown() . . . 2389
7.295.3.11 read_cgroup_cpu_usage_us() . . . 2390
7.295.3.12 read_cgroup_memory_current() . . . 2391
7.295.3.13 read_proc_stat_totals() . . . 2392
7.295.3.14 read_psi_file() . . . 2393
7.295.3.15 resolve_cgroup_path() . . . 2394
7.295.3.16 snapshot_posix() . . . 2395
7.295.4 Variable Documentation . . . 2397
7.295.4.1 g_runtime . . . 2397
7.296 v3/src/platform/alloc_host.c File Reference . . . 2397
7.296.1 Function Documentation . . . 2398
7.296.1.1 sf_alloc_get_stats() . . . 2398
7.296.1.2 sf_alloc_init() . . . 2398
7.296.1.3 sf_calloc() . . . 2399
7.296.1.4 sf_free() . . . 2399
7.296.1.5 sf_malloc() . . . 2399
7.296.1.6 sf_realloc() . . . 2400
7.296.2 Variable Documentation . . . 2400
7.296.2.1 g_stats . . . 2400

7.297 kernel/src/vm/alloc_kernel.c File Reference . . . 2400
7.297.1 Macro Definition Documentation . . . 2401
7.297.1.1 SF_ALIGN . . . 2401
7.297.2 Function Documentation . . . 2402
7.297.2.1 sf_alloc_get_stats() . . . 2402
7.297.2.2 sf_alloc_init() . . . 2402
7.297.2.3 sf_calloc() . . . 2403
7.297.2.4 sf_free() . . . 2404
7.297.2.5 sf_malloc() . . . 2405
7.297.2.6 sf_realloc() . . . 2406
7.297.3 Variable Documentation . . . 2408
7.297.3.1 g_alloc_count . . . 2408
7.297.3.2 g_free_count . . . 2408
7.298 v3/src/platform/alloc_kernel.c File Reference . . . 2409
7.298.1 Macro Definition Documentation . . . 2410
7.298.1.1 SF_ALIGN . . . 2410
7.298.1.2 SF_ALIGN_UP . . . 2410
7.298.1.3 SF_ARENA_SIZE . . . 2410
7.298.2 Function Documentation . . . 2410
7.298.2.1 sf_alloc_get_stats() . . . 2410
7.298.2.2 sf_alloc_init() . . . 2410
7.298.2.3 sf_calloc() . . . 2411
7.298.2.4 sf_free() . . . 2411
7.298.2.5 sf_malloc() . . . 2412
7.298.2.6 sf_realloc() . . . 2412
7.298.3 Variable Documentation . . . 2413
7.298.3.1 __attribute__ . . . 2413
7.298.3.2 g_arena_offset . . . 2413
7.298.3.3 g_initialized . . . 2413
7.298.3.4 g_stats . . . 2413
7.299 v3/src/platform/l4re/time.c File Reference . . . 2414
7.299.1 Function Documentation . . . 2414
7.299.1.1 sf_time_init_l4re() . . . 2414
7.299.2 Variable Documentation . . . 2414
7.299.2.1 sf_time_backend_l4re . . . 2414
7.300 v3/src/platform/linux/time.c File Reference . . . 2415
7.300.1 Macro Definition Documentation . . . 2415
7.300.1.1 _POSIX_C_SOURCE . . . 2415
7.300.2 Function Documentation . . . 2416
7.300.2.1 posix_format_timestamp() . . . 2416
7.300.2.2 posix_get_monotonic_ns() . . . 2416
7.300.2.3 posix_get_realtime_ns() . . . 2417

7.300.2.4 posix_has_rtc() . . . 2417
7.300.2.5 posix_set_realtime_ns() . . . 2417
7.300.3 Variable Documentation . . . 2418
7.300.3.1 sf_time_backend_posix . . . 2418
7.301 v3/src/platform/platform_init.c File Reference . . . 2418
7.301.1 Function Documentation . . . 2419
7.301.1.1 sf_time_init() . . . 2419
7.301.2 Variable Documentation . . . 2419
7.301.2.1 sf_time_backend . . . 2419
7.302 v3/src/platform/threading.c File Reference . . . 2419
7.302.1 Function Documentation . . . 2420
7.302.1.1 sf_mutex_destroy() . . . 2420
7.302.1.2 sf_mutex_init() . . . 2421
7.302.1.3 sf_mutex_lock() . . . 2421
7.302.1.4 sf_mutex_unlock() . . . 2423
7.303 v3/src/profiler.c File Reference . . . 2424
7.303.1 Detailed Description . . . 2427
7.303.2 Macro Definition Documentation . . . 2427
7.303.2.1 _POSIX_C_SOURCE . . . 2427
7.303.3 Typedef Documentation . . . 2427
7.303.3.1 DictEntry . . . 2427
7.303.4 Function Documentation . . . 2427
7.303.4.1 compare_by_calls() . . . 2427
7.303.4.2 compare_by_time() . . . 2428
7.303.4.3 get_time_ns() . . . 2429
7.303.4.4 get_word_stats() . . . 2429
7.303.4.5 print_word_name_safe() . . . 2430
7.303.4.6 profiler_analyze_performance() . . . 2431
7.303.4.7 profiler_end_timer() . . . 2431
7.303.4.8 profiler_generate_report() . . . 2431
7.303.4.9 profiler_get_time_ns() . . . 2432
7.303.4.10 profiler_inc_branch() . . . 2433
7.303.4.11 profiler_inc_dict_lookup() . . . 2433
7.303.4.12 profiler_inc_loop_iter() . . . 2433
7.303.4.13 profiler_inc_stack_op() . . . 2433
7.303.4.14 profiler_inc_vm_cycles() . . . 2433
7.303.4.15 profiler_init() . . . 2433
7.303.4.16 profiler_memory_alloc() . . . 2434
7.303.4.17 profiler_memory_read() . . . 2434
7.303.4.18 profiler_memory_write() . . . 2434
7.303.4.19 profiler_print_hotspots() . . . 2434
7.303.4.20 profiler_print_memory_analysis() . . . 2435

7.303.4.21 profiler_print_summary() . . . 2435
7.303.4.22 profiler_print_word_analysis() . . . 2436
7.303.4.23 profiler_reset() . . . 2436
7.303.4.24 profiler_set_level() . . . 2436
7.303.4.25 profiler_shutdown() . . . 2437
7.303.4.26 profiler_start_timer() . . . 2437
7.303.4.27 profiler_suggest_optimizations() . . . 2437
7.303.4.28 profiler_word_count() . . . 2438
7.303.4.29 profiler_word_enter() . . . 2439
7.303.4.30 profiler_word_exit() . . . 2439
7.303.5 Variable Documentation . . . 2440
7.303.5.1 current_start_time . . . 2440
7.303.5.2 current_word . . . 2441
7.303.5.3 enabled . . . 2441
7.303.5.4 g_profiler . . . 2441
7.303.5.5 level . . . 2441
7.303.5.6 [struct] . . . 2441
7.303.5.7 total_dict_lookups . . . 2441
7.303.5.8 total_memory_reads . . . 2441
7.303.5.9 total_memory_writes . . . 2441
7.303.5.10 total_stack_ops . . . 2441
7.303.5.11 word_capacity . . . 2442
7.303.5.12 word_count . . . 2442
7.303.5.13 word_stats . . . 2442
7.304 kernel/src/repl.c File Reference . . . 2442
7.304.1 Macro Definition Documentation . . . 2444
7.304.1.1 SK_CURSOR_BLINK_INTERVAL . . . 2444
7.304.1.2 SK_IDLE_BEAT_INTERVAL . . . 2444
7.304.1.3 SK_KH_BLK_ATTACH_BUF_SIZE . . . 2444
7.304.1.4 SK_PROMPT_TEXT . . . 2444
7.304.2 Function Documentation . . . 2445
7.304.2.1 sk_console_getkey() . . . 2445
7.304.2.2 sk_console_identity_present() . . . 2446
7.304.2.3 sk_console_key_available() . . . 2447
7.304.2.4 sk_console_readline() . . . 2448
7.304.2.5 sk_console_user_prefix() . . . 2450
7.304.2.6 sk_cursor_show() . . . 2451
7.304.2.7 sk_print_prompt() . . . 2452
7.304.2.8 sk_repl() . . . 2452
7.304.2.9 sk_repl_dispatch_line() . . . 2454
7.304.2.10 sk_repl_get_active_vm() . . . 2455
7.304.2.11 sk_repl_get_artemis_usb_dev() . . . 2455

7.304.2.12 sk_repl_get_attached_blk_dev() . . . 2455
7.304.2.13 sk_repl_get_homeblocks_dev() . . . 2456
7.304.2.14 sk_repl_get_homeblocks_sig() . . . 2456
7.304.2.15 sk_repl_headless_wait() . . . 2456
7.304.2.16 sk_repl_idle() . . . 2458
7.304.2.17 sk_repl_line_calls_use() . . . 2460
7.304.2.18 sk_repl_register_words() . . . 2460
7.304.2.19 sk_repl_run() . . . 2462
7.304.2.20 sk_repl_set_active_vm() . . . 2463
7.304.2.21 sk_repl_step() . . . 2463
7.304.2.22 sk_word_blk_attach_ack() . . . 2464
7.304.2.23 sk_word_kh_blk_attach_send() . . . 2465
7.304.3 Variable Documentation . . . 2466
7.304.3.1 g_artemis_usb_dev . . . 2466
7.304.3.2 g_attached_blk_dev . . . 2466
7.304.3.3 g_console_pending_key . . . 2466
7.304.3.4 g_cursor_blink_tick . . . 2466
7.304.3.5 g_cursor_visible . . . 2467
7.304.3.6 g_homeblocks_dev . . . 2467
7.304.3.7 g_homeblocks_sig . . . 2467
7.304.3.8 g_homeblocks_sig_valid . . . 2467
7.304.3.9 g_idle_pump_active . . . 2467
7.304.3.10 g_kh_blk_attach_buf . . . 2467
7.304.3.11 g_kh_console_cmd_buf . . . 2467
7.304.3.12 g_last_beat_tick . . . 2467
7.304.3.13 g_mama_interpreting . . . 2468
7.304.3.14 g_repl_active_vm . . . 2468
7.304.3.15 g_usb_ctrl . . . 2468
7.304.3.16 lithos_version . . . 2468
7.305 v3/src/repl.c File Reference . . . 2468
7.305.1 Function Documentation . . . 2469
7.305.1.1 __attribute__() . . . 2469
7.305.1.2 vm_repl() . . . 2469
7.306 v3/src/rolling_window_of_truth.c File Reference . . . 2471
7.306.1 Function Documentation . . . 2473
7.306.1.1 rolling_window_check_adaptive_shrink() . . . 2473
7.306.1.2 rolling_window_cleanup() . . . 2473
7.306.1.3 rolling_window_count_transition() . . . 2474
7.306.1.4 rolling_window_export_execution_history() . . . 2475
7.306.1.5 rolling_window_find_hottest_word() . . . 2476
7.306.1.6 rolling_window_get_recent_sequence() . . . 2477
7.306.1.7 rolling_window_init() . . . 2478

7.306.1.8 rolling_window_is_warm() . . . 2480
7.306.1.9 rolling_window_measure_diversity() . . . 2480
7.306.1.10 rolling_window_measure_diversity_view() . . . 2481
7.306.1.11 rolling_window_pattern_capture_rate() . . . 2482
7.306.1.12 rolling_window_publish_snapshot() . . . 2483
7.306.1.13 rolling_window_publish_snapshot_if_needed() . . . 2484
7.306.1.14 rolling_window_record_execution() . . . 2485
7.306.1.15 rolling_window_reset() . . . 2486
7.306.1.16 rolling_window_run_adaptive_pass() . . . 2487
7.306.1.17 rolling_window_seed_hotwords_cache() . . . 2488
7.306.1.18 rolling_window_seed_pipelining_context() . . . 2489
7.306.1.19 rolling_window_service() . . . 2490
7.306.1.20 rolling_window_service_internal() . . . 2491
7.306.1.21 rolling_window_snapshot_view() . . . 2492
7.306.1.22 rolling_window_stats_string() . . . 2493
7.307 v3/src/ssm_jacquard.c File Reference . . . 2494
7.307.1 Function Documentation . . . 2496
7.307.1.1 ssm_apply_mode() . . . 2496
7.307.1.2 ssm_apply_mode_from_table() . . . 2496
7.307.1.3 ssm_l8_force_config() . . . 2497
7.307.1.4 ssm_l8_free_table() . . . 2498
7.307.1.5 ssm_l8_init() . . . 2499
7.307.1.6 ssm_l8_init_table() . . . 2500
7.307.1.7 ssm_l8_mode_name() . . . 2501
7.307.1.8 ssm_l8_seed_from_doe() . . . 2502
7.307.1.9 ssm_l8_tick_score_and_select() . . . 2503
7.307.1.10 ssm_l8_update() . . . 2504
7.307.1.11 ssm_l8_update_table() . . . 2505
7.308 v3/src/stack_management.c File Reference . . . 2506
7.308.1 Detailed Description . . . 2507
7.308.2 Function Documentation . . . 2507
7.308.2.1 vm_pop() . . . 2507
7.308.2.2 vm_push() . . . 2509
7.308.2.3 vm_rpop() . . . 2510
7.308.2.4 vm_rpush() . . . 2511
7.309 v3/src/test_runner/include/test_common.h File Reference . . . 2512
7.309.1 Detailed Description . . . 2514
7.309.2 Macro Definition Documentation . . . 2514
7.309.2.1 MAX_TEST_INPUT . . . 2514
7.309.2.2 MAX_TEST_OUTPUT . . . 2514
7.309.2.3 MAX_TESTS_PER_WORD . . . 2514
7.309.3 Enumeration Type Documentation . . . 2514

7.309.3.1 TestType . . . 2514
7.309.4 Function Documentation . . . 2515
7.309.4.1 assert_stack_depth() . . . 2515
7.309.4.2 assert_stack_top() . . . 2515
7.309.4.3 assert_vm_error() . . . 2516
7.309.4.4 print_module_summary() . . . 2516
7.309.4.5 restore_dict_state() . . . 2518
7.309.4.6 restore_vm_state() . . . 2521
7.309.4.7 run_single_test() . . . 2523
7.309.4.8 run_test_suite() . . . 2526
7.309.4.9 run_test_suite_m() . . . 2528
7.309.4.10 save_dict_state() . . . 2530
7.309.4.11 save_vm_state() . . . 2531
7.309.5 Variable Documentation . . . 2533
7.309.5.1 fail_fast . . . 2533
7.310 v3/src/test_runner/include/test_contracts.h File Reference . . . 2534
7.310.1 Macro Definition Documentation . . . 2535
7.310.1.1 CONTRACT_DEFAULT_FUZZ_ROUNDS . . . 2535
7.310.1.2 CONTRACT_HEARTBEAT_SAFE . . . 2535
7.310.1.3 CONTRACT_NONE . . . 2535
7.310.1.4 CONTRACT_PHYSICS_TRANSPARENT . . . 2536
7.310.2 Typedef Documentation . . . 2536
7.310.2.1 contract_flags_t . . . 2536
7.310.3 Function Documentation . . . 2536
7.310.3.1 assert_heartbeat_safe() . . . 2536
7.310.3.2 capture_exec_result() . . . 2537
7.310.3.3 check_physics_transparent() . . . 2538
7.310.3.4 contract_get_violation_count() . . . 2540
7.310.3.5 contract_reset_violation_count() . . . 2541
7.310.3.6 exec_results_equal() . . . 2541
7.310.3.7 perturb_physics() . . . 2542
7.310.3.8 restore_physics() . . . 2543
7.310.3.9 save_physics() . . . 2544
7.310.4 Variable Documentation . . . 2545
7.310.4.1 global_contract_violations . . . 2545
7.311 v3/src/test_runner/include/test_runner.h File Reference . . . 2545
7.311.1 Function Documentation . . . 2548
7.311.1.1 enable_benchmark_mode() . . . 2548
7.311.1.2 run_acl_words_tests() . . . 2548
7.311.1.3 run_all_tests() . . . 2550
7.311.1.4 run_arithmetic_words_tests() . . . 2551
7.311.1.5 run_block_words_tests() . . . 2552

7.311.1.6 run_break_me_tests() . . . 2553
7.311.1.7 run_control_words_tests() . . . 2554
7.311.1.8 run_defining_words_tests() . . . 2555
7.311.1.9 run_dictionary_manipulation_words_tests() . . . 2556
7.311.1.10 run_dictionary_words_tests() . . . 2557
7.311.1.11 run_double_words_tests() . . . 2558
7.311.1.12 run_format_words_tests() . . . 2559
7.311.1.13 run_inference_words_tests() . . . 2560
7.311.1.14 run_integration_tests() . . . 2561
7.311.1.15 run_io_words_tests() . . . 2562
7.311.1.16 run_logical_words_tests() . . . 2563
7.311.1.17 run_mama_forth_words_tests() . . . 2564
7.311.1.18 run_memory_words_tests() . . . 2565
7.311.1.19 run_mixed_arithmetic_words_tests() . . . 2566
7.311.1.20 run_module_tests() . . . 2567
7.311.1.21 run_physics_freeze_words_tests() . . . 2568
7.311.1.22 run_q48_words_tests() . . . 2569
7.311.1.23 run_return_stack_words_tests() . . . 2570
7.311.1.24 run_stack_words_tests() . . . 2571
7.311.1.25 run_starforth_words_tests() . . . 2572
7.311.1.26 run_stress_tests() . . . 2573
7.311.1.27 run_string_words_tests() . . . 2574
7.311.1.28 run_system_words_tests() . . . 2575
7.311.1.29 run_vocabulary_words_tests() . . . 2576
7.311.1.30 run_word_tests() . . . 2577
7.311.1.31 test_stats_accumulate() . . . 2578
7.311.2 Variable Documentation . . . 2579
7.311.2.1 global_test_stats . . . 2579
7.312 v3/src/test_runner/modules/acl_words_test.c File Reference . . . 2580
7.312.1 Macro Definition Documentation . . . 2581
7.312.1.1 ACL_ASSERT . . . 2581
7.312.2 Function Documentation . . . 2581
7.312.2.1 run_acl_words_tests() . . . 2581
7.312.2.2 test_acl_adaptive_ttl() . . . 2583
7.312.2.3 test_acl_deny_aborts() . . . 2584
7.312.2.4 test_acl_inherit() . . . 2585
7.312.2.5 test_acl_inherit_word_dispatch() . . . 2586
7.312.2.6 test_acl_init_primitives() . . . 2588
7.312.2.7 test_acl_pin_blocks_shadow() . . . 2589
7.312.2.8 test_acl_pin_immutable() . . . 2591
7.312.2.9 test_acl_read_accessors() . . . 2592
7.312.2.10 test_acl_strict_mode() . . . 2594

7.312.2.11 test_acl_ttl_hot_path() . . . 2596
7.312.2.12 test_emergency_console_bypass() . . . 2597
7.312.3 Variable Documentation . . . 2599
7.312.3.1 tests_failed . . . 2599
7.312.3.2 tests_passed . . . 2599
7.313 v3/src/test_runner/modules/arithmetic_words_test.c File Reference . . . 2599
7.313.1 Function Documentation . . . 2600
7.313.1.1 run_arithmetic_words_tests() . . . 2600
7.313.2 Variable Documentation . . . 2600
7.313.2.1 arithmetic_word_suites . . . 2600
7.314 v3/src/test_runner/modules/block_words_test.c File Reference . . . 2601
7.314.1 Function Documentation . . . 2601
7.314.1.1 run_block_words_tests() . . . 2601
7.314.2 Variable Documentation . . . 2602
7.314.2.1 block_word_suites . . . 2602
7.315 v3/src/test_runner/modules/break_me_tests.c File Reference . . . 2603
7.315.1 Function Documentation . . . 2604
7.315.1.1 finalize_report() . . . 2604
7.315.1.2 init_report() . . . 2605
7.315.1.3 run_break_me_tests() . . . 2605
7.315.2 Variable Documentation . . . 2606
7.315.2.1 report_file . . . 2606
7.315.2.2 test_end_time . . . 2606
7.315.2.3 test_start_time . . . 2606
7.315.2.4 total_tests_run . . . 2607
7.316 v3/src/test_runner/modules/control_words_test.c File Reference . . . 2607
7.316.1 Function Documentation . . . 2607
7.316.1.1 run_control_words_tests() . . . 2607
7.316.2 Variable Documentation . . . 2608
7.316.2.1 control_word_suites . . . 2608
7.317 v3/src/test_runner/modules/defining_words_tests.c File Reference . . . 2609
7.317.1 Function Documentation . . . 2609
7.317.1.1 run_defining_words_tests() . . . 2609
7.317.2 Variable Documentation . . . 2610
7.317.2.1 defining_word_suites . . . 2610
7.318 v3/src/test_runner/modules/dictionary_manipulation_words_test.c File Reference . . . 2611
7.318.1 Function Documentation . . . 2611
7.318.1.1 run_dictionary_manipulation_words_tests() . . . 2611
7.318.2 Variable Documentation . . . 2612
7.318.2.1 dict_manip_word_suites . . . 2612
7.319 v3/src/test_runner/modules/dictionary_words_test.c File Reference . . . 2613
7.319.1 Function Documentation . . . 2613

7.319.1.1 run_dictionary_words_tests() . . . 2613
7.319.2 Variable Documentation . . . 2614
7.319.2.1 dictionary_word_suites . . . 2614
7.320 v3/src/test_runner/modules/double_words_test.c File Reference . . . 2615
7.320.1 Function Documentation . . . 2615
7.320.1.1 run_double_words_tests() . . . 2615
7.320.2 Variable Documentation . . . 2616
7.320.2.1 double_word_suites . . . 2616
7.321 v3/src/test_runner/modules/format_words_test.c File Reference . . . 2616
7.321.1 Function Documentation . . . 2617
7.321.1.1 run_format_words_tests() . . . 2617
7.321.2 Variable Documentation . . . 2618
7.321.2.1 format_word_suites . . . 2618
7.322 v3/src/test_runner/modules/inference_words_test.c File Reference . . . 2618
7.322.1 Function Documentation . . . 2619
7.322.1.1 run_inference_words_tests() . . . 2619
7.322.2 Variable Documentation . . . 2620
7.322.2.1 inference_word_suites . . . 2620
7.323 v3/src/test_runner/modules/integration_tests.c File Reference . . . 2620
7.323.1 Detailed Description . . . 2621
7.323.2 Function Documentation . . . 2621
7.323.2.1 run_integration_tests() . . . 2621
7.323.3 Variable Documentation . . . 2622
7.323.3.1 integration_suites . . . 2622
7.324 v3/src/test_runner/modules/io_words_test.c File Reference . . . 2623
7.324.1 Function Documentation . . . 2623
7.324.1.1 run_io_words_tests() . . . 2623
7.324.2 Variable Documentation . . . 2624
7.324.2.1 io_word_suites . . . 2624
7.325 v3/src/test_runner/modules/logical_words_test.c File Reference . . . 2625
7.325.1 Function Documentation . . . 2625
7.325.1.1 run_logical_words_tests() . . . 2625
7.325.2 Variable Documentation . . . 2626
7.325.2.1 logical_word_suites . . . 2626
7.326 v3/src/test_runner/modules/mama_forth_words_test.c File Reference . . . 2626
7.326.1 Detailed Description . . . 2627
7.326.2 Macro Definition Documentation . . . 2628
7.326.2.1 MAMA_IMPLEMENTED . . . 2628
7.326.3 Function Documentation . . . 2628
7.326.3.1 run_mama_forth_words_tests() . . . 2628
7.326.4 Variable Documentation . . . 2629
7.326.4.1 mama_word_suites . . . 2629

7.327 v3/src/test_runner/modules/memory_words_test.c File Reference . . . 2629
7.327.1 Function Documentation . . . 2630
7.327.1.1 run_memory_words_tests() . . . 2630
7.327.2 Variable Documentation . . . 2630
7.327.2.1 memory_word_suites . . . 2630
7.328 v3/src/test_runner/modules/mixed_arithmetic_words_test.c File Reference . . . 2631
7.328.1 Function Documentation . . . 2631
7.328.1.1 run_mixed_arithmetic_words_tests() . . . 2631
7.328.2 Variable Documentation . . . 2632
7.328.2.1 mixed_arithmetic_word_suites . . . 2632
7.329 v3/src/test_runner/modules/physics_freeze_words_test.c File Reference . . . 2633
7.329.1 Function Documentation . . . 2633
7.329.1.1 run_physics_freeze_words_tests() . . . 2633
7.329.2 Variable Documentation . . . 2634
7.329.2.1 physics_freeze_word_suites . . . 2634
7.330 v3/src/test_runner/modules/q48_words_test.c File Reference . . . 2635
7.330.1 Function Documentation . . . 2635
7.330.1.1 run_q48_words_tests() . . . 2635
7.330.2 Variable Documentation . . . 2636
7.330.2.1 q48_word_suites . . . 2636
7.331 v3/src/test_runner/modules/return_stack_words_test.c File Reference . . . 2636
7.331.1 Function Documentation . . . 2637
7.331.1.1 run_return_stack_words_tests() . . . 2637
7.331.2 Variable Documentation . . . 2638
7.331.2.1 return_stack_word_suites . . . 2638
7.332 v3/src/test_runner/modules/stack_words_test.c File Reference . . . 2638
7.332.1 Detailed Description . . . 2639
7.332.2 Function Documentation . . . 2639
7.332.2.1 run_stack_words_tests() . . . 2639
7.332.3 Variable Documentation . . . 2640
7.332.3.1 stack_word_suites . . . 2640
7.333 v3/src/test_runner/modules/starforth_words_test.c File Reference . . . 2641
7.333.1 Function Documentation . . . 2641
7.333.1.1 run_starforth_words_tests() . . . 2641
7.333.2 Variable Documentation . . . 2642
7.333.2.1 starforth_word_suites . . . 2642
7.334 v3/src/test_runner/modules/stress_tests.c File Reference . . . 2643
7.334.1 Detailed Description . . . 2643
7.334.2 Function Documentation . . . 2644
7.334.2.1 run_stress_tests() . . . 2644
7.334.3 Variable Documentation . . . 2645
7.334.3.1 stress_suites . . . 2645

7.335 v3/src/test_runner/modules/string_words_test.c File Reference . . . 2645
7.335.1 Function Documentation . . . 2646
7.335.1.1 run_string_words_tests() . . . 2646
7.335.2 Variable Documentation . . . 2647
7.335.2.1 string_word_suites . . . 2647
7.336 v3/src/test_runner/modules/system_words_test.c File Reference . . . 2647
7.336.1 Function Documentation . . . 2648
7.336.1.1 run_system_words_tests() . . . 2648
7.336.2 Variable Documentation . . . 2649
7.336.2.1 system_word_suites . . . 2649
7.337 v3/src/test_runner/modules/vocabulary_words_test.c File Reference . . . 2649
7.337.1 Function Documentation . . . 2650
7.337.1.1 run_vocabulary_words_tests() . . . 2650
7.337.2 Variable Documentation . . . 2651
7.337.2.1 vocabulary_word_suites . . . 2651
7.338 v3/src/test_runner/test_common.c File Reference . . . 2651
7.338.1 Macro Definition Documentation . . . 2653
7.338.1.1 DIE . . . 2653
7.338.1.2 test_fatal_exit . . . 2653
7.338.2 Function Documentation . . . 2653
7.338.2.1 assert_stack_depth() . . . 2653
7.338.2.2 assert_stack_top() . . . 2654
7.338.2.3 assert_vm_error() . . . 2654
7.338.2.4 dump_and_die() . . . 2655
7.338.2.5 fail_and_exit() . . . 2656
7.338.2.6 print_module_summary() . . . 2657
7.338.2.7 resolve_contract() . . . 2659
7.338.2.8 restore_dict_state() . . . 2660
7.338.2.9 restore_vm_state() . . . 2661
7.338.2.10 run_single_test() . . . 2663
7.338.2.11 run_test_suite() . . . 2666
7.338.2.12 run_test_suite_m() . . . 2668
7.338.2.13 save_dict_state() . . . 2670
7.338.2.14 save_vm_state() . . . 2671
7.338.3 Variable Documentation . . . 2673
7.338.3.1 fail_fast . . . 2673
7.339 v3/src/test_runner/test_contracts.c File Reference . . . 2674
7.339.1 Function Documentation . . . 2675
7.339.1.1 assert_heartbeat_safe() . . . 2675
7.339.1.2 capture_exec_result() . . . 2676
7.339.1.3 check_physics_transparent() . . . 2677
7.339.1.4 contract_get_violation_count() . . . 2679

7.339.1.5 contract_rand() . . . 2680
7.339.1.6 contract_reset_violation_count() . . . 2680
7.339.1.7 exec_results_equal() . . . 2681
7.339.1.8 perturb_physics() . . . 2682
7.339.1.9 restore_physics() . . . 2683
7.339.1.10 rng_seed() . . . 2684
7.339.1.11 save_physics() . . . 2685
7.339.2 Variable Documentation . . . 2685
7.339.2.1 global_contract_violations . . . 2685
7.339.2.2 rng_state . . . 2686
7.340 v3/src/test_runner/test_runner.c File Reference . . . 2686
7.340.1 Function Documentation . . . 2687
7.340.1.1 enable_benchmark_mode() . . . 2687
7.340.1.2 get_time_ns() . . . 2688
7.340.1.3 run_all_tests() . . . 2688
7.340.1.4 run_module_tests() . . . 2689
7.340.1.5 run_word_tests() . . . 2690
7.340.1.6 test_stats_accumulate() . . . 2691
7.340.2 Variable Documentation . . . 2692
7.340.2.1 benchmark_iterations . . . 2692
7.340.2.2 benchmark_mode . . . 2693
7.340.2.3 global_test_stats . . . 2693
7.340.2.4 test_modules . . . 2693
7.341 v3/src/vm.c File Reference . . . 2694
7.341.1 Function Documentation . . . 2695
7.341.1.1 acl_recheck() . . . 2695
7.341.1.2 execute_colon_word() . . . 2696
7.341.1.3 vm_addr_ok() . . . 2697
7.341.1.4 vm_compile_call() . . . 2699
7.341.1.5 vm_compile_exit() . . . 2699
7.341.1.6 vm_compile_literal() . . . 2699
7.341.1.7 vm_compile_word() . . . 2700
7.341.1.8 vm_enter_compile_mode() . . . 2701
7.341.1.9 vm_exit_compile_mode() . . . 2702
7.341.1.10 vm_get_base() . . . 2702
7.341.1.11 vm_interpret() . . . 2703
7.341.1.12 vm_interpret_word() . . . 2704
7.341.1.13 vm_load_cell() . . . 2705
7.341.1.14 vm_load_u8() . . . 2706
7.341.1.15 vm_make_immediate() . . . 2706
7.341.1.16 vm_parse_number() . . . 2707
7.341.1.17 vm_parse_word() . . . 2708

7.341.1.18 vm_ptr() . . . 2709
7.341.1.19 vm_set_base() . . . 2710
7.341.1.20 vm_store_cell() . . . 2710
7.341.1.21 vm_store_u8() . . . 2711
7.341.1.22 vm_zuse_cert_install() . . . 2711
7.341.2 Variable Documentation . . . 2712
7.341.2.1 g_vm_interpret_depth . . . 2712
7.342 v3/src/vm_api.c File Reference . . . 2712
7.342.1 Function Documentation . . . 2714
7.342.1.1 vm_api_create_word() . . . 2714
7.342.1.2 vm_api_find_word() . . . 2715
7.342.1.3 vm_api_hide_word() . . . 2715
7.342.1.4 vm_api_pop() . . . 2716
7.342.1.5 vm_api_push() . . . 2716
7.342.1.6 vm_api_rpop() . . . 2717
7.342.1.7 vm_api_rpush() . . . 2718
7.342.1.8 vm_api_smudge_word() . . . 2719
7.343 kernel/src/vm/vm_bootstrap.c File Reference . . . 2719
7.343.1 Function Documentation . . . 2720
7.343.1.1 vm_bootstrap_root_vocabulary() . . . 2720
7.343.1.2 vm_bootstrap_scr() . . . 2721
7.343.1.3 vm_init() . . . 2722
7.343.1.4 vm_init_with_host() . . . 2723
7.344 v3/src/vm_bootstrap.c File Reference . . . 2726
7.344.1 Function Documentation . . . 2726
7.344.1.1 vm_bootstrap_root_vocabulary() . . . 2726
7.344.1.2 vm_bootstrap_scr() . . . 2727
7.344.1.3 vm_cleanup() . . . 2727
7.344.1.4 vm_init() . . . 2727
7.345 v3/src/vm_debug.c File Reference . . . 2728
7.345.1 Function Documentation . . . 2729
7.345.1.1 dbg_get_base() . . . 2729
7.345.1.2 dump_block_preview() . . . 2729
7.345.1.3 mode_str() . . . 2730
7.345.1.4 sig_handler() . . . 2731
7.345.1.5 vm_debug_dump_state() . . . 2732
7.345.1.6 vm_debug_install_signal_handlers() . . . 2733
7.345.1.7 vm_debug_set_current_vm() . . . 2733
7.345.2 Variable Documentation . . . 2734
7.345.2.1 g_vm . . . 2734
7.346 kernel/src/vm/vm_internal.h File Reference . . . 2734
7.346.1 Macro Definition Documentation . . . 2735

7.346.1.1 HEARTBEAT_DECAY_BATCH . . . 2735
7.346.1.2 HEARTBEAT_HAS_THREADS . . . 2735
7.346.1.3 PARITY_MODE . . . 2735
7.346.1.4 SK_PARITY_DEBUG . . . 2736
7.346.2 Typedef Documentation . . . 2736
7.346.2.1 HeartbeatWorker . . . 2736
7.346.3 Function Documentation . . . 2736
7.346.3.1 heartbeat_publish_snapshot() . . . 2736
7.346.3.2 heartbeat_snapshot_index_load() . . . 2737
7.346.3.3 heartbeat_snapshot_index_store() . . . 2738
7.346.3.4 vm_default_host_services() . . . 2738
7.346.3.5 vm_get_base() . . . 2739
7.346.3.6 vm_heartbeat_run_cycle() . . . 2740
7.346.3.7 vm_host() . . . 2741
7.346.3.8 vm_host_alloc() . . . 2742
7.346.3.9 vm_host_calloc() . . . 2743
7.346.3.10 vm_host_free() . . . 2744
7.346.3.11 vm_monotonic_ns() . . . 2744
7.346.3.12 vm_reset_hosted_fake_ns() . . . 2745
7.346.3.13 vm_set_base() . . . 2745
7.347 v3/src/vm_internal.h File Reference . . . 2746
7.347.1 Macro Definition Documentation . . . 2747
7.347.1.1 HEARTBEAT_DECAY_BATCH . . . 2747
7.347.1.2 HEARTBEAT_HAS_THREADS . . . 2748
7.347.2 Typedef Documentation . . . 2748
7.347.2.1 HeartbeatWorker . . . 2748
7.347.3 Function Documentation . . . 2748
7.347.3.1 execute_colon_word() . . . 2748
7.347.3.2 heartbeat_snapshot_index_load() . . . 2750
7.347.3.3 heartbeat_snapshot_index_store() . . . 2751
7.347.3.4 vm_cleanup() . . . 2752
7.347.3.5 vm_get_base() . . . 2752
7.347.3.6 vm_heartbeat_publish_snapshot() . . . 2753
7.347.3.7 vm_heartbeat_run_cycle() . . . 2754
7.347.3.8 vm_heartbeat_start_thread() . . . 2755
7.347.3.9 vm_heartbeat_stop_thread() . . . 2756
7.347.3.10 vm_init() . . . 2757
7.347.3.11 vm_set_base() . . . 2758
7.347.3.12 vm_snapshot_read() . . . 2760
7.347.3.13 vm_tick() . . . 2760
7.347.3.14 vm_tick_inference_engine() . . . 2762
7.347.3.15 vm_tick_slope_validator() . . . 2763

7.347.3.16 vm_tick_window_tuner() . . . 2764
7.348 v3/src/vm_time.c File Reference . . . 2765
7.348.1 Function Documentation . . . 2766
7.348.1.1 vm_heartbeat_publish_snapshot() . . . 2766
7.348.1.2 vm_heartbeat_run_cycle() . . . 2767
7.348.1.3 vm_heartbeat_start_thread() . . . 2767
7.348.1.4 vm_heartbeat_stop_thread() . . . 2768
7.348.1.5 vm_heartbeat_update_l8() . . . 2768
7.348.1.6 vm_snapshot_read() . . . 2770
7.348.1.7 vm_tick() . . . 2770
7.348.1.8 vm_tick_apply_background_decay() . . . 2771
7.348.1.9 vm_tick_inference_engine() . . . 2772
7.348.1.10 vm_tick_slope_validator() . . . 2772
7.348.1.11 vm_tick_window_tuner() . . . 2772
7.349 v3/src/word_registry.c File Reference . . . 2773
7.349.1 Function Documentation . . . 2774
7.349.1.1 register_forth79_words() . . . 2774
7.349.1.2 register_word() . . . 2776
7.350 v3/src/word_source/acl_words.c File Reference . . . 2777
7.350.1 Function Documentation . . . 2779
7.350.1.1 acl_inherit_entry() . . . 2779
7.350.1.2 forth_acl_allow_fetch() . . . 2779
7.350.1.3 forth_acl_allow_store() . . . 2780
7.350.1.4 forth_acl_heat_fetch() . . . 2781
7.350.1.5 forth_acl_inherit() . . . 2781
7.350.1.6 forth_acl_init_primitives() . . . 2782
7.350.1.7 forth_acl_mode_fetch() . . . 2783
7.350.1.8 forth_acl_mode_store() . . . 2783
7.350.1.9 forth_acl_pin() . . . 2785
7.350.1.10 forth_acl_pinned_query() . . . 2786
7.350.1.11 forth_acl_ttl_fetch() . . . 2787
7.350.1.12 forth_acl_ttl_store() . . . 2787
7.350.1.13 forth_acl_word_id() . . . 2789
7.350.1.14 pop_xt() . . . 2790
7.350.1.15 register_acl_words() . . . 2791
7.351 v3/src/word_source/arithmetic_words.c File Reference . . . 2792
7.351.1 Function Documentation . . . 2794
7.351.1.1 arithmetic_word_abs() . . . 2794
7.351.1.2 arithmetic_word_max() . . . 2794
7.351.1.3 arithmetic_word_min() . . . 2795
7.351.1.4 arithmetic_word_minus() . . . 2795
7.351.1.5 arithmetic_word_multiply() . . . 2796

7.351.1.6 arithmetic_word_negate() . . . 2797
7.351.1.7 arithmetic_word_one_minus() . . . 2797
7.351.1.8 arithmetic_word_one_plus() . . . 2798
7.351.1.9 arithmetic_word_plus() . . . 2798
7.351.1.10 arithmetic_word_star_slash() . . . 2799
7.351.1.11 arithmetic_word_star_slash_mod() . . . 2800
7.351.1.12 arithmetic_word_two_divide() . . . 2800
7.351.1.13 arithmetic_word_two_minus() . . . 2801
7.351.1.14 arithmetic_word_two_multiply() . . . 2801
7.351.1.15 arithmetic_word_two_plus() . . . 2802
7.351.1.16 mixed_math_word_mod() . . . 2802
7.351.1.17 register_arithmetic_words() . . . 2803
7.351.1.18 word_divide() . . . 2804
7.351.1.19 word_divmod() . . . 2805
7.352 v3/src/word_source/block_words.c File Reference . . . 2806
7.352.1 Function Documentation . . . 2807
7.352.1.1 blk_vm_assign() . . . 2807
7.352.1.2 blk_vm_check_epoch() . . . 2808
7.352.1.3 blk_vm_evict() . . . 2809
7.352.1.4 blk_vm_find() . . . 2809
7.352.1.5 blk_vm_flush_all() . . . 2810
7.352.1.6 blk_vm_is_cross_device_copy() . . . 2811
7.352.1.7 blk_vm_load() . . . 2812
7.352.1.8 blk_vm_slot_for_addr() . . . 2812
7.352.1.9 block_word_acl_allow_fetch() . . . 2813
7.352.1.10 block_word_acl_allow_store() . . . 2814
7.352.1.11 block_word_acl_ttl_fetch() . . . 2814
7.352.1.12 block_word_acl_ttl_store() . . . 2815
7.352.1.13 block_word_blk_attach() . . . 2815
7.352.1.14 block_word_block() . . . 2816
7.352.1.15 block_word_buffer() . . . 2817
7.352.1.16 block_word_confirm_format() . . . 2818
7.352.1.17 block_word_empty_buffers() . . . 2819
7.352.1.18 block_word_flush() . . . 2819
7.352.1.19 block_word_list() . . . 2820
7.352.1.20 block_word_load() . . . 2820
7.352.1.21 block_word_next_block() . . . 2821
7.352.1.22 block_word_owner_fetch() . . . 2822
7.352.1.23 block_word_relocate() . . . 2823
7.352.1.24 block_word_save_buffers() . . . 2824
7.352.1.25 block_word_scr() . . . 2825
7.352.1.26 block_word_thru() . . . 2825

7.352.1.27 block_word_update() . . . 2826
7.352.1.28 empty_all_buffers() . . . 2827
7.352.1.29 get_block_buffer() . . . 2828
7.352.1.30 get_empty_buffer() . . . 2829
7.352.1.31 init_block_system() . . . 2830
7.352.1.32 mark_buffer_dirty() . . . 2830
7.352.1.33 register_block_words() . . . 2831
7.352.1.34 save_all_buffers() . . . 2833
7.352.1.35 set_scr() . . . 2833
7.353 v3/src/word_source/control_words.c File Reference . . . 2834
7.353.1 Function Documentation . . . 2836
7.353.1.1 emit_cell() . . . 2836
7.354 v3/src/word_source/defer_words.c File Reference . . . 2836
7.354.1 Function Documentation . . . 2837
7.354.1.1 defer_runtime() . . . 2837
7.354.1.2 register_defer_words() . . . 2838
7.354.1.3 word_defer() . . . 2839
7.354.1.4 word_defer_fetch() . . . 2840
7.354.1.5 word_is() . . . 2840
7.355 v3/src/word_source/defining_words.c File Reference . . . 2841
7.355.1 Function Documentation . . . 2842
7.355.1.1 defining_runtime_constant() . . . 2842
7.355.1.2 defining_runtime_create() . . . 2843
7.355.1.3 defining_runtime_dodoes() . . . 2843
7.355.1.4 defining_runtime_does_rt() . . . 2844
7.355.1.5 defining_runtime_lit() . . . 2845
7.355.1.6 defining_runtime_variable() . . . 2846
7.355.1.7 defining_word_bracket_compile() . . . 2847
7.355.1.8 defining_word_colon() . . . 2847
7.355.1.9 defining_word_compile() . . . 2849
7.355.1.10 defining_word_constant() . . . 2849
7.355.1.11 defining_word_create() . . . 2850
7.355.1.12 defining_word_does() . . . 2851
7.355.1.13 defining_word_immediate() . . . 2852
7.355.1.14 defining_word_left_bracket() . . . 2853
7.355.1.15 defining_word_literal() . . . 2853
7.355.1.16 defining_word_right_bracket() . . . 2854
7.355.1.17 defining_word_semicolon() . . . 2855
7.355.1.18 defining_word_state() . . . 2855
7.355.1.19 defining_word_variable() . . . 2856
7.355.1.20 dictionary_word_fence() . . . 2856
7.355.1.21 dictionary_word_forget() . . . 2857

7.355.1.22 register_defining_words() . . . 2858
7.356 v3/src/word_source/dictionary_heat_diagnostic_words.c File Reference . . . 2860
7.356.1 Function Documentation . . . 2860
7.356.1.1 forth_COMPARE_LOOKUPS() . . . 2860
7.356.1.2 forth_HEAT_PERCENTILES() . . . 2861
7.356.1.3 forth_LOOKUP_STRATEGY_FETCH() . . . 2862
7.356.1.4 forth_LOOKUP_STRATEGY_STORE() . . . 2862
7.356.1.5 forth_REORG_BUCKETS() . . . 2863
7.356.1.6 forth_SHOW_HEAT_OPTIMIZATION() . . . 2864
7.356.1.7 register_dictionary_heat_diagnostic_words() . . . 2865
7.357 v3/src/word_source/dictionary_manipulation_words.c File Reference . . . 2866
7.357.1 Function Documentation . . . 2868
7.357.1.1 align_address() . . . 2868
7.357.1.2 dictionary_m_word_cfa() . . . 2868
7.357.1.3 dictionary_m_word_find() . . . 2869
7.357.1.4 dictionary_m_word_hidden() . . . 2869
7.357.1.5 dictionary_m_word_interpret() . . . 2870
7.357.1.6 dictionary_m_word_lfa() . . . 2870
7.357.1.7 dictionary_m_word_link_from() . . . 2871
7.357.1.8 dictionary_m_word_name_to() . . . 2871
7.357.1.9 dictionary_m_word_nfa() . . . 2872
7.357.1.10 dictionary_m_word_pfa() . . . 2872
7.357.1.11 dictionary_m_word_smudge() . . . 2873
7.357.1.12 dictionary_m_word_tick() . . . 2873
7.357.1.13 dictionary_m_word_to_body() . . . 2874
7.357.1.14 dictionary_m_word_to_link() . . . 2874
7.357.1.15 dictionary_m_word_to_name() . . . 2875
7.357.1.16 dictionary_m_word_traverse() . . . 2875
7.357.1.17 find_entry_from_name() . . . 2876
7.357.1.18 get_body_address() . . . 2876
7.357.1.19 get_name_field() . . . 2877
7.357.1.20 register_dictionary_manipulation_words() . . . 2877
7.357.1.21 traverse_name_field() . . . 2878
7.357.2 Variable Documentation . . . 2879
7.357.2.1 state_variable . . . 2879
7.358 v3/src/word_source/dictionary_words.c File Reference . . . 2879
7.358.1 Function Documentation . . . 2880
7.358.1.1 dictionary_word_2comma() . . . 2880
7.358.1.2 dictionary_word_align() . . . 2881
7.358.1.3 dictionary_word_allot() . . . 2881
7.358.1.4 dictionary_word_c_comma() . . . 2882
7.358.1.5 dictionary_word_comma() . . . 2882

7.358.1.6 dictionary_word_here() . . . 2883
7.358.1.7 dictionary_word_latest() . . . 2884
7.358.1.8 dictionary_word_pad() . . . 2885
7.358.1.9 dictionary_word_sp_fetch() . . . 2885
7.358.1.10 dictionary_word_sp_store() . . . 2886
7.358.1.11 register_dictionary_words() . . . 2887
7.359 v3/src/word_source/double_words.c File Reference . . . 2888
7.359.1 Function Documentation . . . 2889
7.359.1.1 d_compare() . . . 2889
7.359.1.2 double_word_2drop() . . . 2890
7.359.1.3 double_word_2dup() . . . 2890
7.359.1.4 double_word_2over() . . . 2891
7.359.1.5 double_word_2r_fetch() . . . 2891
7.359.1.6 double_word_2r_from() . . . 2892
7.359.1.7 double_word_2rot() . . . 2892
7.359.1.8 double_word_2swap() . . . 2893
7.359.1.9 double_word_2to_r() . . . 2893
7.359.1.10 double_word_d_equals() . . . 2894
7.359.1.11 double_word_d_less() . . . 2894
7.359.1.12 double_word_d_minus() . . . 2895
7.359.1.13 double_word_d_plus() . . . 2895
7.359.1.14 double_word_d_two_slash() . . . 2896
7.359.1.15 double_word_d_two_star() . . . 2896
7.359.1.16 double_word_d_zero_equals() . . . 2897
7.359.1.17 double_word_d_zero_less() . . . 2897
7.359.1.18 double_word_dabs() . . . 2898
7.359.1.19 double_word_dmax() . . . 2898
7.359.1.20 double_word_dmin() . . . 2899
7.359.1.21 double_word_dnegate() . . . 2899
7.359.1.22 double_word_s_to_d() . . . 2900
7.359.1.23 register_double_words() . . . 2900
7.360 v3/src/word_source/editor_words.c File Reference . . . 2902
7.360.1 Function Documentation . . . 2902
7.360.1.1 current_scr() . . . 2902
7.360.1.2 editor_word_edit() . . . 2903
7.360.1.3 editor_word_l() . . . 2904
7.360.1.4 editor_word_s() . . . 2905
7.360.1.5 editor_word_show() . . . 2906
7.360.1.6 line_ptr() . . . 2906
7.360.1.7 print_line_64() . . . 2907
7.360.1.8 register_editor_words() . . . 2908
7.361 v3/src/word_source/format_words.c File Reference . . . 2909

7.361.1 Macro Definition Documentation . . . 2911
7.361.1.1 CELL_BITS . . . 2911
7.361.1.2 CHUNK_BITS . . . 2911
7.361.1.3 CHUNK_MASK . . . 2911
7.361.1.4 PN_BUF . . . 2911
7.361.1.5 PN_CAP . . . 2911
7.361.2 Function Documentation . . . 2912
7.361.2.1 current_base() . . . 2912
7.361.2.2 digit_for() . . . 2912
7.361.2.3 div_ud_by_base() . . . 2913
7.361.2.4 format_word_base() . . . 2913
7.361.2.5 format_word_begin_conversion() . . . 2914
7.361.2.6 format_word_d_dot() . . . 2914
7.361.2.7 format_word_d_dot_r() . . . 2915
7.361.2.8 format_word_decimal() . . . 2915
7.361.2.9 format_word_dot() . . . 2916
7.361.2.10 format_word_dot_r() . . . 2916
7.361.2.11 format_word_dot_s() . . . 2917
7.361.2.12 format_word_dump() . . . 2917
7.361.2.13 format_word_end_conversion() . . . 2918
7.361.2.14 format_word_hash() . . . 2919
7.361.2.15 format_word_hash_s() . . . 2919
7.361.2.16 format_word_hex() . . . 2920
7.361.2.17 format_word_hold() . . . 2920
7.361.2.18 format_word_octal() . . . 2921
7.361.2.19 format_word_question() . . . 2922
7.361.2.20 format_word_sign() . . . 2922
7.361.2.21 format_word_u_dot() . . . 2923
7.361.2.22 format_word_u_dot_r() . . . 2923
7.361.2.23 print_number_formatted() . . . 2924
7.361.2.24 register_format_words() . . . 2925
7.362 v3/src/word_source/framebuffer_words.c File Reference . . . 2926
7.362.1 Function Documentation . . . 2927
7.362.1.1 fbw_height() . . . 2927
7.362.1.2 fbw_plot() . . . 2928
7.362.1.3 fbw_width() . . . 2928
7.362.1.4 register_framebuffer_words() . . . 2929
7.363 v3/src/word_source/include/acl_words.h File Reference . . . 2930
7.363.1 Function Documentation . . . 2930
7.363.1.1 acl_inherit_entry() . . . 2930
7.363.1.2 register_acl_words() . . . 2932
7.364 v3/src/word_source/include/arithmetic_words.h File Reference . . . 2933

7.364.1 Function Documentation . . . 2934
7.364.1.1 register_arithmetic_words() . . . 2934
7.365 v3/src/word_source/include/block_words.h File Reference . . . 2936
7.365.1 Detailed Description . . . 2938
7.365.2 Macro Definition Documentation . . . 2939
7.365.2.1 BLOCK_SIZE . . . 2939
7.365.2.2 BUFFER_COUNT . . . 2939
7.365.2.3 MAX_BLOCKS . . . 2939
7.365.3 Enumeration Type Documentation . . . 2939
7.365.3.1 buffer_type_t . . . 2939
7.365.4 Function Documentation . . . 2939
7.365.4.1 blk_vm_check_epoch() . . . 2939
7.365.4.2 blk_vm_flush_all() . . . 2940
7.365.4.3 blk_vm_is_cross_device_copy() . . . 2941
7.365.4.4 blk_vm_slot_for_addr() . . . 2942
7.365.4.5 empty_all_buffers() . . . 2943
7.365.4.6 get_block_buffer() . . . 2943
7.365.4.7 get_empty_buffer() . . . 2944
7.365.4.8 init_block_system() . . . 2945
7.365.4.9 mark_buffer_dirty() . . . 2945
7.365.4.10 register_block_words() . . . 2946
7.365.4.11 save_all_buffers() . . . 2948
7.366 v3/src/word_source/include/control_words.h File Reference . . . 2948
7.367 v3/src/word_source/include/defer_words.h File Reference . . . 2949
7.367.1 Function Documentation . . . 2950
7.367.1.1 register_defer_words() . . . 2950
7.368 v3/src/word_source/include/defining_words.h File Reference . . . 2951
7.368.1 Function Documentation . . . 2952
7.368.1.1 register_defining_words() . . . 2952
7.369 v3/src/word_source/include/dictionary_heat_diagnostic_words.h File Reference . . . 2955
7.369.1 Function Documentation . . . 2956
7.369.1.1 register_dictionary_heat_diagnostic_words() . . . 2956
7.370 v3/src/word_source/include/dictionary_manipulation_words.h File Reference . . . 2957
7.370.1 Function Documentation . . . 2958
7.370.1.1 register_dictionary_manipulation_words() . . . 2958
7.371 v3/src/word_source/include/dictionary_words.h File Reference . . . 2959
7.371.1 Function Documentation . . . 2960
7.371.1.1 register_dictionary_words() . . . 2960
7.372 v3/src/word_source/include/double_words.h File Reference . . . 2961
7.372.1 Function Documentation . . . 2962
7.372.1.1 register_double_words() . . . 2962
7.373 v3/src/word_source/include/editor_words.h File Reference . . . 2964

7.373.1 Function Documentation . . . 2964
7.373.1.1 register_editor_words() . . . 2964
7.374 v3/src/word_source/include/format_words.h File Reference . . . 2966
7.374.1 Function Documentation . . . 2966
7.374.1.1 register_format_words() . . . 2966
7.375 v3/src/word_source/include/framebuffer_words.h File Reference . . . 2968
7.375.1 Function Documentation . . . 2969
7.375.1.1 register_framebuffer_words() . . . 2969
7.376 v3/src/word_source/include/inference_words.h File Reference . . . 2970
7.376.1 Function Documentation . . . 2971
7.376.1.1 register_inference_words() . . . 2971
7.377 v3/src/word_source/include/io_words.h File Reference . . . 2972
7.377.1 Macro Definition Documentation . . . 2974
7.377.1.1 IO_WORDS_H . . . 2974
7.377.2 Function Documentation . . . 2974
7.377.2.1 register_io_words() . . . 2974
7.378 v3/src/word_source/include/keyboard_words.h File Reference . . . 2975
7.378.1 Function Documentation . . . 2976
7.378.1.1 register_keyboard_words() . . . 2976
7.378.1.2 sk_key_event_poll() . . . 2977
7.379 v3/src/word_source/include/log_words.h File Reference . . . 2978
7.379.1 Function Documentation . . . 2979
7.379.1.1 register_log_words() . . . 2979
7.380 v3/src/word_source/include/logical_words.h File Reference . . . 2980
7.380.1 Function Documentation . . . 2981
7.380.1.1 register_logical_words() . . . 2981
7.381 v3/src/word_source/include/mama_forth_words.h File Reference . . . 2983
7.381.1 Detailed Description . . . 2984
7.381.2 Function Documentation . . . 2984
7.381.2.1 register_mama_forth_words() . . . 2984
7.382 v3/src/word_source/include/memory_words.h File Reference . . . 2984
7.382.1 Function Documentation . . . 2985
7.382.1.1 register_memory_words() . . . 2985
7.383 v3/src/word_source/include/mixed_arithmetic_words.h File Reference . . . 2986
7.383.1 Function Documentation . . . 2987
7.383.1.1 register_mixed_arithmetic_words() . . . 2987
7.384 v3/src/word_source/include/physics_benchmark_words.h File Reference . . . 2988
7.384.1 Function Documentation . . . 2989
7.384.1.1 register_physics_benchmark_words() . . . 2989
7.385 v3/src/word_source/include/physics_diagnostic_words.h File Reference . . . 2990
7.385.1 Function Documentation . . . 2991
7.385.1.1 forth_PHYSICS_BURN() . . . 2991

7.385.1.2 forth_PHYSICS_CALC_KNOBS() . . . 2992
7.385.1.3 forth_PHYSICS_SHOW_FEEDBACK() . . . 2993
7.385.1.4 forth_PHYSICS_WORD_METRICS() . . . 2994
7.385.1.5 register_physics_diagnostic_words() . . . 2995
7.386 v3/src/word_source/include/physics_freeze_words.h File Reference . . . 2995
7.386.1 Function Documentation . . . 2996
7.386.1.1 register_physics_freeze_words() . . . 2996
7.387 v3/src/word_source/include/physics_pipelining_diagnostic_words.h File Reference . . . 2997
7.387.1 Function Documentation . . . 2998
7.387.1.1 register_physics_pipelining_diagnostic_words() . . . 2998
7.388 v3/src/word_source/include/q48_words.h File Reference . . . 2999
7.388.1 Function Documentation . . . 3000
7.388.1.1 register_q48_words() . . . 3000
7.389 v3/src/word_source/include/return_stack_words.h File Reference . . . 3002
7.389.1 Function Documentation . . . 3003
7.389.1.1 register_return_stack_words() . . . 3003
7.390 v3/src/word_source/include/scroll_words.h File Reference . . . 3004
7.390.1 Function Documentation . . . 3004
7.390.1.1 register_scroll_words() . . . 3004
7.391 v3/src/word_source/include/stack_words.h File Reference . . . 3005
7.391.1 Function Documentation . . . 3006
7.391.1.1 register_stack_words() . . . 3006
7.392 v3/src/word_source/include/starforth_words.h File Reference . . . 3007
7.392.1 Function Documentation . . . 3009
7.392.1.1 register_starforth_words() . . . 3009
7.392.1.2 starforth_word_entropy_fetch() . . . 3011
7.392.1.3 starforth_word_entropy_store() . . . 3011
7.392.1.4 starforth_word_init() . . . 3011
7.392.1.5 starforth_word_paren_dash() . . . 3012
7.392.1.6 starforth_word_random() . . . 3013
7.392.1.7 starforth_word_reset_entropy() . . . 3014
7.392.1.8 starforth_word_seed() . . . 3014
7.392.1.9 starforth_word_top_words() . . . 3015
7.392.1.10 starforth_word_wait() . . . 3016
7.392.1.11 starforth_word_word_entropy() . . . 3016
7.393 v3/src/word_source/include/string_words.h File Reference . . . 3017
7.393.1 Detailed Description . . . 3018
7.393.2 Function Documentation . . . 3018
7.393.2.1 register_string_words() . . . 3018
7.394 v3/src/word_source/include/system_words.h File Reference . . . 3020
7.394.1 Function Documentation . . . 3021
7.394.1.1 register_system_words() . . . 3021

7.395 v3/src/word_source/include/ttf_words.h File Reference . . . 3023
7.395.1 Function Documentation . . . 3023
7.395.1.1 register_ttf_words() . . . 3023
7.396 v3/src/word_source/include/vocabulary_words.h File Reference . . . 3024
7.396.1 Function Documentation . . . 3025
7.396.1.1 register_vocabulary_words() . . . 3025
7.396.1.2 vocabulary_create_vocabulary_direct() . . . 3026
7.396.1.3 vocabulary_word_definitions() . . . 3027
7.396.1.4 vocabulary_word_forth() . . . 3028
7.397 v3/src/word_source/inference_words.c File Reference . . . 3029
7.397.1 Function Documentation . . . 3030
7.397.1.1 array_ptr() . . . 3030
7.397.1.2 bucket_posterior() . . . 3031
7.397.1.3 cache_posterior() . . . 3031
7.397.1.4 infer_word_bayes_bucket_lower() . . . 3032
7.397.1.5 infer_word_bayes_bucket_mean() . . . 3032
7.397.1.6 infer_word_bayes_bucket_upper() . . . 3033
7.397.1.7 infer_word_bayes_cache_lower() . . . 3033
7.397.1.8 infer_word_bayes_cache_mean() . . . 3034
7.397.1.9 infer_word_bayes_cache_upper() . . . 3034
7.397.1.10 infer_word_decay_fetch() . . . 3035
7.397.1.11 infer_word_decay_slope() . . . 3035
7.397.1.12 infer_word_early_exit_fetch() . . . 3036
7.397.1.13 infer_word_fit_fetch() . . . 3036
7.397.1.14 infer_word_l8_apply() . . . 3037
7.397.1.15 infer_word_l8_mode() . . . 3037
7.397.1.16 infer_word_l8_table_force() . . . 3038
7.397.1.17 infer_word_l8_update() . . . 3038
7.397.1.18 infer_word_q_variance() . . . 3039
7.397.1.19 infer_word_run() . . . 3039
7.397.1.20 infer_word_variance_fetch() . . . 3040
7.397.1.21 infer_word_window_diversity() . . . 3041
7.397.1.22 infer_word_window_fetch() . . . 3042
7.397.1.23 infer_word_window_width() . . . 3042
7.397.1.24 q48_pop_inf() . . . 3043
7.397.1.25 q48_push_inf() . . . 3044
7.397.1.26 q48_to_dbl() . . . 3044
7.397.1.27 register_inference_words() . . . 3044
7.398 v3/src/word_source/io_words.c File Reference . . . 3045
7.398.1 Function Documentation . . . 3046
7.398.1.1 io_runtime_do_string() . . . 3046
7.398.1.2 io_word_cr() . . . 3047

7.398.1.3 io_word_dot_quote() . . . 3048
7.398.1.4 io_word_emit() . . . 3049
7.398.1.5 io_word_key() . . . 3049
7.398.1.6 io_word_question_terminal() . . . 3050
7.398.1.7 io_word_space() . . . 3051
7.398.1.8 io_word_spaces() . . . 3052
7.398.1.9 io_word_type() . . . 3053
7.398.1.10 register_io_words() . . . 3054
7.399 v3/src/word_source/keyboard_words.c File Reference . . . 3055
7.399.1 Function Documentation . . . 3056
7.399.1.1 kbw_alt_tab() . . . 3056
7.399.1.2 kbw_debug() . . . 3057
7.399.1.3 kbw_key_event() . . . 3058
7.399.1.4 kbw_scan() . . . 3058
7.399.1.5 kbw_vdebug() . . . 3059
7.399.1.6 kbw_vevent() . . . 3059
7.399.1.7 register_keyboard_words() . . . 3060
7.399.1.8 sk_key_event_poll() . . . 3060
7.399.2 Variable Documentation . . . 3061
7.399.2.1 g_i8042_isr_count . . . 3061
7.399.2.2 g_spurious_count . . . 3062
7.399.2.3 g_virtio_input_isr_count . . . 3062
7.400 v3/src/word_source/lifecycle_words_hosted.c File Reference . . . 3062
7.400.1 Macro Definition Documentation . . . 3063
7.400.1.1 NAME_MAX_LEN . . . 3063
7.400.2 Function Documentation . . . 3063
7.400.2.1 extract_name() . . . 3063
7.400.2.2 hosted_lifecycle_birth() . . . 3064
7.400.2.3 hosted_lifecycle_kill() . . . 3065
7.400.2.4 hosted_lifecycle_pause() . . . 3065
7.400.2.5 hosted_lifecycle_resume() . . . 3066
7.400.2.6 hosted_lifecycle_use() . . . 3066
7.400.2.7 register_lifecycle_words() . . . 3067
7.401 v3/src/word_source/log_words.c File Reference . . . 3068
7.401.1 Function Documentation . . . 3069
7.401.1.1 log_do_debug() . . . 3069
7.401.1.2 log_do_emit() . . . 3070
7.401.1.3 log_do_error() . . . 3071
7.401.1.4 log_do_info() . . . 3071
7.401.1.5 log_do_test() . . . 3072
7.401.1.6 log_do_warn() . . . 3072
7.401.1.7 log_emit_string() . . . 3073

7.401.1.8 log_str_emit() . . . 3073
7.401.1.9 log_word_debug() . . . 3074
7.401.1.10 log_word_debug_quote() . . . 3075
7.401.1.11 log_word_debug_str() . . . 3075
7.401.1.12 log_word_error() . . . 3076
7.401.1.13 log_word_error_quote() . . . 3076
7.401.1.14 log_word_error_str() . . . 3077
7.401.1.15 log_word_get_level() . . . 3078
7.401.1.16 log_word_info() . . . 3078
7.401.1.17 log_word_info_quote() . . . 3079
7.401.1.18 log_word_info_str() . . . 3080
7.401.1.19 log_word_set_level() . . . 3080
7.401.1.20 log_word_test() . . . 3081
7.401.1.21 log_word_test_quote() . . . 3081
7.401.1.22 log_word_test_str() . . . 3082
7.401.1.23 log_word_warn() . . . 3083
7.401.1.24 log_word_warn_quote() . . . 3083
7.401.1.25 log_word_warn_str() . . . 3084
7.401.1.26 register_log_words() . . . 3084
7.402 v3/src/word_source/logical_words.c File Reference . . . 3085
7.402.1 Macro Definition Documentation . . . 3087
7.402.1.1 FORTH_FALSE . . . 3087
7.402.1.2 FORTH_TRUE . . . 3087
7.402.2 Function Documentation . . . 3087
7.402.2.1 logical_word_and() . . . 3087
7.402.2.2 logical_word_equals() . . . 3088
7.402.2.3 logical_word_false() . . . 3088
7.402.2.4 logical_word_greater_equal() . . . 3089
7.402.2.5 logical_word_greater_than() . . . 3090
7.402.2.6 logical_word_invert() . . . 3090
7.402.2.7 logical_word_less_equal() . . . 3091
7.402.2.8 logical_word_less_than() . . . 3091
7.402.2.9 logical_word_lshift() . . . 3092
7.402.2.10 logical_word_not() . . . 3092
7.402.2.11 logical_word_not_equals() . . . 3093
7.402.2.12 logical_word_or() . . . 3093
7.402.2.13 logical_word_rshift() . . . 3094
7.402.2.14 logical_word_true() . . . 3094
7.402.2.15 logical_word_u_greater_than() . . . 3095
7.402.2.16 logical_word_u_less_than() . . . 3095
7.402.2.17 logical_word_within() . . . 3096
7.402.2.18 logical_word_xor() . . . 3096

7.402.2.19 logical_word_zero_equals() . . . 3097
7.402.2.20 logical_word_zero_greater() . . . 3097
7.402.2.21 logical_word_zero_less() . . . 3098
7.402.2.22 logical_word_zero_not_equal() . . . 3098
7.402.2.23 register_logical_words() . . . 3099
7.403 v3/src/word_source/memory_words.c File Reference . . . 3100
7.403.1 Function Documentation . . . 3102
7.403.1.1 memory_word_2fetch() . . . 3102
7.403.1.2 memory_word_2store() . . . 3102
7.403.1.3 memory_word_cells() . . . 3103
7.403.1.4 memory_word_cfetch() . . . 3104
7.403.1.5 memory_word_cstore() . . . 3105
7.403.1.6 memory_word_erase() . . . 3106
7.403.1.7 memory_word_fetch() . . . 3107
7.403.1.8 memory_word_fill() . . . 3108
7.403.1.9 memory_word_minus_store() . . . 3109
7.403.1.10 memory_word_move() . . . 3110
7.403.1.11 memory_word_plus_store() . . . 3111
7.403.1.12 memory_word_store() . . . 3112
7.403.1.13 register_memory_words() . . . 3113
7.404 v3/src/word_source/mixed_arithmetic_words.c File Reference . . . 3114
7.404.1 Function Documentation . . . 3115
7.404.1.1 mixed_math_word_m_minus() . . . 3115
7.404.1.2 mixed_math_word_m_plus() . . . 3116
7.404.1.3 mixed_math_word_m_slash_mod() . . . 3117
7.404.1.4 mixed_math_word_m_star() . . . 3118
7.404.1.5 mixed_math_word_mod() . . . 3119
7.404.1.6 mixed_math_word_slash_mod() . . . 3120
7.404.1.7 mixed_math_word_star_slash() . . . 3120
7.404.1.8 mixed_math_word_star_slash_mod() . . . 3121
7.404.1.9 register_mixed_arithmetic_words() . . . 3122
7.405 v3/src/word_source/physics_benchmark_words.c File Reference . . . 3123
7.405.1 Function Documentation . . . 3124
7.405.1.1 forth_BENCH_DICT_LOOKUP() . . . 3124
7.405.1.2 forth_PHYSICS_BAYESIAN_REPORT() . . . 3125
7.405.1.3 forth_PHYSICS_BUILD_INFO() . . . 3126
7.405.1.4 forth_PHYSICS_CACHE_STATS() . . . 3127
7.405.1.5 forth_PHYSICS_RESET_STATS() . . . 3128
7.405.1.6 forth_PHYSICS_TOGGLE_CACHE() . . . 3128
7.405.1.7 register_physics_benchmark_words() . . . 3129
7.405.2 Variable Documentation . . . 3130
7.405.2.1 sf_fc_count . . . 3130

7.405.2.2 sf_fc_list . . . 3130
7.406 v3/src/word_source/physics_diagnostic_words.c File Reference . . . 3131
7.406.1 Function Documentation . . . 3131
7.406.1.1 forth_PHYSICS_BURN() . . . 3131
7.406.1.2 forth_PHYSICS_CALC_KNOBS() . . . 3132
7.406.1.3 forth_PHYSICS_SHOW_FEEDBACK() . . . 3133
7.406.1.4 forth_PHYSICS_WORD_METRICS() . . . 3134
7.406.1.5 register_physics_diagnostic_words() . . . 3135
7.407 v3/src/word_source/physics_freeze_words.c File Reference . . . 3135
7.407.1 Function Documentation . . . 3136
7.407.1.1 forth_ALL_HEATS() . . . 3136
7.407.1.2 forth_DECAY_RATE_FETCH() . . . 3137
7.407.1.3 forth_FREEZE_CRITICAL() . . . 3138
7.407.1.4 forth_FREEZE_WORD() . . . 3139
7.407.1.5 forth_FROZEN_QUERY() . . . 3140
7.407.1.6 forth_HEAT_FETCH() . . . 3140
7.407.1.7 forth_HEAT_STORE() . . . 3141
7.407.1.8 forth_SHOW_HEAT() . . . 3142
7.407.1.9 forth_UNFREEZE_WORD() . . . 3143
7.407.1.10 register_physics_freeze_words() . . . 3144
7.408 v3/src/word_source/physics_pipelining_diagnostic_words.c File Reference . . . 3145
7.408.1 Function Documentation . . . 3146
7.408.1.1 forth_PIPELINING_ANALYZE_WORD() . . . 3146
7.408.1.2 forth_PIPELINING_ENABLE() . . . 3146
7.408.1.3 forth_PIPELINING_RESET_ALL() . . . 3147
7.408.1.4 forth_PIPELINING_SHOW_STATS() . . . 3148
7.408.1.5 forth_PIPELINING_SHOW_TOP_TRANSITIONS() . . . 3148
7.408.1.6 forth_PIPELINING_STATS() . . . 3149
7.408.1.7 register_physics_pipelining_diagnostic_words() . . . 3150
7.409 v3/src/word_source/q48_16_words.c File Reference . . . 3151
7.409.1 Function Documentation . . . 3152
7.409.1.1 q48_cos_approx() . . . 3152
7.409.1.2 q48_div() . . . 3152
7.409.1.3 q48_exp_approx() . . . 3153
7.409.1.4 q48_from_double() . . . 3154
7.409.1.5 q48_is_valid() . . . 3154
7.409.1.6 q48_log_approx() . . . 3154
7.409.1.7 q48_mul() . . . 3155
7.409.1.8 q48_reduce_angle() . . . 3155
7.409.1.9 q48_sin_approx() . . . 3157
7.409.1.10 q48_sqrt_approx() . . . 3157
7.409.1.11 q48_to_double() . . . 3158

7.409.1.12 q48_to_string() . . . 3158
7.410 v3/src/word_source/q48_words.c File Reference . . . 3159
7.410.1 Function Documentation . . . 3160
7.410.1.1 q48_pop() . . . 3160
7.410.1.2 q48_push() . . . 3161
7.410.1.3 q48_word_abs() . . . 3162
7.410.1.4 q48_word_add() . . . 3162
7.410.1.5 q48_word_cos() . . . 3163
7.410.1.6 q48_word_div() . . . 3164
7.410.1.7 q48_word_eq() . . . 3164
7.410.1.8 q48_word_exp() . . . 3165
7.410.1.9 q48_word_from_int() . . . 3166
7.410.1.10 q48_word_gt() . . . 3166
7.410.1.11 q48_word_log() . . . 3167
7.410.1.12 q48_word_lt() . . . 3168
7.410.1.13 q48_word_max() . . . 3168
7.410.1.14 q48_word_min() . . . 3169
7.410.1.15 q48_word_mul() . . . 3170
7.410.1.16 q48_word_neg() . . . 3170
7.410.1.17 q48_word_one() . . . 3171
7.410.1.18 q48_word_print() . . . 3171
7.410.1.19 q48_word_scale() . . . 3172
7.410.1.20 q48_word_sin() . . . 3173
7.410.1.21 q48_word_sqrt() . . . 3173
7.410.1.22 q48_word_sub() . . . 3174
7.410.1.23 q48_word_to_int() . . . 3175
7.410.1.24 q48_word_zero() . . . 3175
7.410.1.25 q48_word_zero_eq() . . . 3176
7.410.1.26 register_q48_words() . . . 3177
7.411 v3/src/word_source/return_stack_words.c File Reference . . . 3179
7.411.1 Detailed Description . . . 3179
7.411.2 Function Documentation . . . 3180
7.411.2.1 register_return_stack_words() . . . 3180
7.411.2.2 return_stack_word_r_fetch() . . . 3181
7.411.2.3 return_stack_word_r_from() . . . 3182
7.411.2.4 return_stack_word_to_r() . . . 3183
7.412 v3/src/word_source/scroll_words.c File Reference . . . 3184
7.412.1 Function Documentation . . . 3184
7.412.1.1 register_scroll_words() . . . 3184
7.413 v3/src/word_source/stack_words.c File Reference . . . 3185
7.413.1 Function Documentation . . . 3187
7.413.1.1 register_stack_words() . . . 3187

7.413.1.2 stack_word_depth() . . . 3188
7.413.1.3 stack_word_drop() . . . 3188
7.413.1.4 stack_word_dup() . . . 3189
7.413.1.5 stack_word_minus_rot() . . . 3190
7.413.1.6 stack_word_over() . . . 3190
7.413.1.7 stack_word_pick() . . . 3191
7.413.1.8 stack_word_question_dup() . . . 3192
7.413.1.9 stack_word_roll() . . . 3193
7.413.1.10 stack_word_rot() . . . 3193
7.413.1.11 stack_word_swap() . . . 3194
7.414 v3/src/word_source/starforth_words.c File Reference . . . 3195
7.414.1 Macro Definition Documentation . . . 3196
7.414.1.1 STARFORTH_CHECK_ARENA . . . 3196
7.414.2 Function Documentation . . . 3197
7.414.2.1 prng_next() . . . 3197
7.414.2.2 register_starforth_words() . . . 3197
7.414.2.3 starforth_word_execution_heat_fetch() . . . 3199
7.414.2.4 starforth_word_execution_heat_store() . . . 3199
7.414.2.5 starforth_word_heartbeat_ticks() . . . 3200
7.414.2.6 starforth_word_init() . . . 3201
7.414.2.7 starforth_word_paren_dash() . . . 3202
7.414.2.8 starforth_word_random() . . . 3203
7.414.2.9 starforth_word_reset_execution_heat() . . . 3204
7.414.2.10 starforth_word_seed() . . . 3204
7.414.2.11 starforth_word_top_words() . . . 3206
7.414.2.12 starforth_word_version() . . . 3207
7.414.2.13 starforth_word_wait() . . . 3208
7.414.2.14 starforth_word_word_execution_heat() . . . 3209
7.414.2.15 starforth_word_zuse_authenticate() . . . 3209
7.414.2.16 starforth_word_zuse_cert_installed_query() . . . 3210
7.414.2.17 starforth_word_zuse_pubkey_fetch() . . . 3210
7.414.2.18 starforth_word_zuse_session() . . . 3211
7.414.3 Variable Documentation . . . 3211
7.414.3.1 g_prng_state . . . 3211
7.415 v3/src/word_source/string_words.c File Reference . . . 3211
7.415.1 Macro Definition Documentation . . . 3213
7.415.1.1 WORD_SCRATCH_CAP . . . 3213
7.415.2 Function Documentation . . . 3213
7.415.2.1 convert_string_to_number() . . . 3213
7.415.2.2 ensure_input() . . . 3213
7.415.2.3 register_string_words() . . . 3214
7.415.2.4 string_runtime_s_quote() . . . 3217

7.415.2.5 string_word_bl() . . . 3217
7.415.2.6 string_word_blank() . . . 3218
7.415.2.7 string_word_bracket_literal() . . . 3218
7.415.2.8 string_word_bracket_tick() . . . 3219
7.415.2.9 string_word_cmove() . . . 3219
7.415.2.10 string_word_cmove_greater() . . . 3220
7.415.2.11 string_word_compare() . . . 3221
7.415.2.12 string_word_convert() . . . 3221
7.415.2.13 string_word_count() . . . 3222
7.415.2.14 string_word_enclose() . . . 3223
7.415.2.15 string_word_expect() . . . 3224
7.415.2.16 string_word_literal() . . . 3224
7.415.2.17 string_word_minus_trailing() . . . 3225
7.415.2.18 string_word_number() . . . 3225
7.415.2.19 string_word_query() . . . 3226
7.415.2.20 string_word_s_quote() . . . 3226
7.415.2.21 string_word_scan() . . . 3227
7.415.2.22 string_word_search() . . . 3228
7.415.2.23 string_word_skip() . . . 3228
7.415.2.24 string_word_source() . . . 3229
7.415.2.25 string_word_span() . . . 3230
7.415.2.26 string_word_tib() . . . 3230
7.415.2.27 string_word_to_in() . . . 3231
7.415.2.28 string_word_word() . . . 3231
7.415.2.29 vaddr_from_ptr() . . . 3232
7.415.3 Variable Documentation . . . 3232
7.415.3.1 word_scratch_addr . . . 3232
7.416 v3/src/word_source/system_words.c File Reference . . . 3233
7.416.1 Function Documentation . . . 3234
7.416.1.1 forth_backslash() . . . 3234
7.416.1.2 forth_paren() . . . 3234
7.416.1.3 print_word_name() . . . 3235
7.416.1.4 register_system_words() . . . 3235
7.416.1.5 reset_vm_state() . . . 3237
7.416.1.6 runtime_abortq() . . . 3237
7.416.1.7 set_forth_79_compliance() . . . 3238
7.416.1.8 system_is_running() . . . 3238
7.416.1.9 system_word_79_standard() . . . 3238
7.416.1.10 system_word_abort() . . . 3239
7.416.1.11 system_word_abort_quote() . . . 3239
7.416.1.12 system_word_bye() . . . 3240
7.416.1.13 system_word_cold() . . . 3240

7.416.1.14 system_word_execute() . . . 3241
7.416.1.15 system_word_nop() . . . 3242
7.416.1.16 system_word_page() . . . 3242
7.416.1.17 system_word_quit() . . . 3243
7.416.1.18 system_word_reboot() . . . 3243
7.416.1.19 system_word_save_system() . . . 3244
7.416.1.20 system_word_see() . . . 3244
7.416.1.21 system_word_vlist() . . . 3245
7.416.1.22 system_word_warm() . . . 3246
7.416.1.23 system_word_words() . . . 3247
7.416.2 Variable Documentation . . . 3247
7.416.2.1 forth_79_standard . . . 3247
7.416.2.2 system_running . . . 3248
7.417 v3/src/word_source/ttf_words.c File Reference . . . 3248
7.417.1 Function Documentation . . . 3248
7.417.1.1 register_ttf_words() . . . 3248
7.418 v3/src/word_source/vocabulary_words.c File Reference . . . 3249
7.418.1 Function Documentation . . . 3250
7.418.1.1 fc_free() . . . 3250
7.418.1.2 fc_rebuild() . . . 3251
7.418.1.3 init_vocabulary_system() . . . 3252
7.418.1.4 register_vocabulary_words() . . . 3253
7.418.1.5 vm_vocabulary_find_word() . . . 3254
7.418.1.6 vocab_find_word() . . . 3256
7.418.1.7 vocab_safe_len() . . . 3256
7.418.1.8 vocab_sync_vm_vars() . . . 3257
7.418.1.9 vocabulary_create_vocabulary_direct() . . . 3258
7.418.1.10 vocabulary_debug_print_hex() . . . 3258
7.418.1.11 vocabulary_select_runtime() . . . 3259
7.418.1.12 vocabulary_word_context() . . . 3259
7.418.1.13 vocabulary_word_current() . . . 3260
7.418.1.14 vocabulary_word_definitions() . . . 3261
7.418.1.15 vocabulary_word_forth() . . . 3262
7.418.1.16 vocabulary_word_order() . . . 3263
7.418.1.17 vocabulary_word_paren_find() . . . 3263
7.418.1.18 vocabulary_word_vocabulary() . . . 3264

Index **3267**

Chapter 1

include/

Public headers for the hosted StarForth VM (src/). Kernel-only headers live under include/starkernel/.

Core VM

- vm.h — the VM struct and all core types (DictEntry, DictPhysics, stacks, dictionary state); the central header most other files include.
- vm_api.h — external VM API surface.
- vm_host.h, vm_debug.h, startup.h, version.h — host integration, debug utilities, startup sequencing, version string.
- cli.h, repl.h, io.h, log.h — CLI parsing, REPL loop, I/O, logging.
- word_registry.h — word registration system shared by every src/word_source/*.c file.
- compudynamics.h — the generic compudynamics module: tuning-word/config lookups (cd_tuning_word(), cd_tuning_vm()) consumed by ssm_jacquard.c.

Memory / blocks

- memory_management.h, dictionary_management.h — dictionary allocator and search.
- block_subsystem.h, blkcfg.h, blkio.h, blkio_factory.h — logical→ physical block mapper and pluggable block I/O backends (file/RAM).
- platform_alloc.h, platform_lock.h, platform_time.h — platform abstraction shims (hosted vs. kernel allocation/locking/timing).

Physics-driven adaptive runtime (7 feedback loops)

- physics_runtime.h — main physics coordinator.
- physics_hotwords_cache.h — Loop #1, execution-heat hot-words cache.
- physics_metadata.h — per-word DictPhysics metadata tracking.
- physics_pipelining_metrics.h — Loop #4, word-transition prediction.
- physics_execution_hooks.h — execution instrumentation hook points.

- rolling_window_of_truth.h, rolling_window_knobs.h — Loop #2 circular execution-history buffer and its tuning knobs.
- inference_engine.h — Loops #5/#6, window-width and decay-slope statistical inference.
- dictionary_heat_optimization.h — Loop #1 heat counters.
- ssm_jacquard.h — L8 Jacquard steady-state mode selector.
- doe_metrics.h — Design of Experiments (2^7 factorial) metrics.
- profiler.h — performance profiling hooks.

Arithmetic / codegen

- q48_16.h — Q48.16 deterministic fixed-point arithmetic (used instead of IEEE-754 float specifically to keep cross-architecture behavior bit-identical).
- math_portable.h — portable math helpers.
- arch_detect.h, starforth_config.h — architecture detection and the build-flag fallback-default layer (used when a .c file is compiled by hand without make).
- vm_asm_opt.h, vm_asm_opt_arm64.h, vm_asm_opt_riscv64.h, vm_inner_interp_asm.h, vm_inner_interp_arm64.h, vm_inner_interp_riscv64.h — per-architecture assembler-optimized inner interpreter (USE_ASM_OPT=1).

See include/starkernel/README.md for the bare-metal kernel headers.

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Architecture Detection . . . 19
Unified API Wrappers . . . 19
Runtime Architecture Detection . . . 19
Build Information . . . 20
VM Input Operations . . . 20
Stack Operations . . . 25
Fast Primitive Word Implementations . . . 37
Control Flow Stack . . . 45
Runtime Branch Helpers . . . 48
FORTH-79 Control Flow Words . . . 70
FORTH-79 Defining Words . . . 70
FORTH-79 Dictionary Manipulation Words . . . 70
FORTH-79 Dictionary & Compilation Words . . . 72
FORTH-79 Line Editor Words . . . 72
Framebuffer Words . . . 72
Keyboard Words . . . 73
Log Words . . . 74
FORTH-79 Logical Operations . . . 75
FORTH-79 Memory Operations . . . 75
FORTH-79 Return Stack Operations . . . 76
REPL Scrollback Words . . . 76
FORTH-79 System & Environment Words . . . 76
TrueType Text Words . . . 77
FORTH-79 Vocabulary System Words . . . 77

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

__attribute__
    loapic.c - I/O APIC driver (amd64) . . . 79
_EFI_BLOCK_IO_PROTOCOL . . . 84
_EFI_BOOT_SERVICES . . . 85
_EFI_FILE_PROTOCOL . . . 90
_EFI_GRAPHICS_OUTPUT_PROTOCOL . . . 92
_EFI_RUNTIME_SERVICES . . . 94
_EFI_SIMPLE_FILE_SYSTEM_PROTOCOL . . . 96
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL . . . 97
_EFI_SYSTEM_TABLE . . . 99
artemis_sig_t . . . 101
BayesianLatencyPosterior
    Bayesian statistics structure for inference and credible intervals All values stored in Q48.16 fixed-point (64-bit signed integer) representing nanoseconds with 2^-16 ≈ 0.0000153 ns precision . . . 102
blk_bam_entry_t . . . 104
blk_dev_slot . . . 105
blk_meta_t . . . 108
blk_reloc_entry_t . . . 111
blk_volume_meta_t . . . 111
blkcfg_t . . . 115
blkio_dev . . . 116
blkio_file_state_t . . . 118
blkio_info_t . . . 119
blkio_params_t . . . 120
blkio_ram_state_t . . . 120
blkio_vtable_t . . . 121
BlkioUsbState . . . 123
BlockBuffer
    Block buffer structure maintaining buffer state . . . 124
BootDisk . . . 125
BootInfo . . . 126
cache_slot_t . . . 128
CapsuleDirHeader . . . 130
CapsuleNameEntry . . . 131
CapsuleRunRecord . . . 132

CapsuleSigEntry . . . 134
CDConfigEntry . . . 134
CDConfigTable . . . 136
CDRegimeMetrics . . . 138
CDTuning . . . 138
cf_item_t . . . 141
CLIConfig
Configuration structure from command-line arguments Consolidates all parsed CLI options into a single, cleanly organized struct . . . 141
ControlFlowState . . . 144
der_span_t . . . 146
DiagEvent . . . 147
DictEntry . . . 147
DictPhysics . . . 150
DoeMetrics
Complete metrics snapshot for a single DoE run . . . 152
ed_point_t . . . 161
EFI_BLOCK_IO_MEDIA . . . 162
EFI_CONFIGURATION_TABLE . . . 163
EFI_DEVICE_PATH_PROTOCOL . . . 164
EFI_FILE_INFO . . . 165
EFI_GRAPHICS_OUTPUT_MODE_INFORMATION . . . 167
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE . . . 168
EFI_GUID . . . 170
EFI_LOADED_IMAGE_PROTOCOL . . . 170
EFI_MEMORY_DESCRIPTOR . . . 172
EFI_PIXEL_BITMASK . . . 173
EFI_TABLE_HEADER . . . 174
EFI_TIME . . . 175
Elf64_Ehdr . . . 177
Elf64_Phdr . . . 179
Elf64_Rel . . . 180
Elf64_Rela . . . 181
Elf64_Shdr . . . 181
Elf64_Sym . . . 183
ExecResult . . . 184
FbState . . . 185
fdt_header_t . . . 186
fdt_uart_t . . . 188
fe25519 . . . 189
FILE . . . 189
FramebufferInfo . . . 190
gdt_entry
Arch.c (amd64) - Architecture abstractions for StarKernel on x86_64 . . . 191
gdtr . . . 192
gpt_part_dev_t . . . 193
gpt_part_t . . . 194
heap_block . . . 195
HeartbeatSnapshot . . . 196
HeartbeatState . . . 197
HeartbeatTickSnapshot . . . 202
HeartbeatWorker . . . 204
homeblocks_sig_t . . . 205
HotwordsCache_s . . . 207
HotwordsStats . . . 208
idt_entry . . . 213
idtr . . . 214
InferenceInputs . . . 215

InferenceOutputs . . . 217
KernelArgs . . . 218
kmalloc_prefix_t . . . 220
kmalloc_stats_t
Kmalloc.h - Kernel heap allocator interface . . . 220
log_region_ctrl_t . . . 221
log_slot_t . . . 223
MemoryProfile . . . 224
ParityPacket
ParityPacket - Summary of VM state for comparison . . . 225
PciDevice . . . 227
physics_analytics_event_header
Event header written into the analytics ring buffer . . . 228
physics_analytics_header
Analytics heap header seen by both producer (VM) and consumer (HOLA) . . . 229
physics_analytics_heap
Simple view onto the analytics heap layout . . . 233
physics_host_snapshot
Snapshot of scheduler-adjacent host signals . . . 236
physics_runtime_state . . . 242
physics_seed_definition . . . 243
PhysicsSnapshot . . . 244
PipelineGlobalMetrics . . . 246
pmm_stats_t . . . 247
Profiler . . . 248
ProfileScope . . . 251
ProfileTimer
Timer for measuring execution duration . . . 251
reloc_block
Reloc_stub.c - Provide a minimal .reloc section for PE/COFF EFI loader . . . 253
rolling_window_view_t
Immutable snapshot view of a RollingWindowOfTruth . . . 254
RollingWindowOfTruth . . . 255
rpi5_range_t . . . 258
Rpi5FramebufferInfo
Rpi5_mailbox.h - VideoCore mailbox property-interface framebuffer setup, for the native (non-↩ UEFI) Raspberry Pi 5 boot path (FABRIC-3.md ?IV.3 item 3) . . . 259
sbiret_t . . . 260
sf_alloc_stats . . . 261
sf_mutex . . . 262
sf_time_backend_t . . . 262
sha512_ctx_t . . . 264
sigaction . . . 265
sk_blk_attach_pending_t . . . 266
sk_usb_ctrl_t . . . 267
sk_v4_drive_t . . . 268
sk_v4_marker_t . . . 270
SpeedupEstimate
Calculate speedup estimate with credible interval All values in Q48.16 fixed-point format (64-bit signed integer) Probabilities are in range [0, Q48_SCALE] representing [0%, 100%] . . . 271
ssm_config_t . . . 272
ssm_l8_metrics_t . . . 274
ssm_l8_state_t . . . 275
TestCase
Structure representing a single test case . . . 276
TestModule
Structure representing a test module . . . 278

TestStats
Structure holding test execution statistics . . . 280
time_trust_state . . . 281
time_window . . . 283
timer_calibration_record . . . 283
timespec . . . 285
timeval . . . 286
timezone . . . 286
ttf_bitmap_t
A caller-owned 8-bit-per-pixel bitmap . . . 287
ttf_flat_point_t . . . 288
ttf_font_t
Parsed font handle . . . 288
ttf_glyph_header_t
Raw glyf record header, per ?27.6/4.3.7's "done when" clause . . . 291
ttf_outline_t
Simple- and composite-glyph outline, flattened to one point list plus per-contour end indices (TrueType convention: contour_ends[c] is the index of the LAST point of contour c, inclusive; points are shared across contours only in the sense that contour c+1 starts right after contour↩ _ends[c]) . . . 293
ttf_point_t
One outline point, in raw font design units (NOT scaled by unitsPerEm — that's the caller's job, same convention as ttf_glyph_header_t's bbox), expressed in Q48.16 . . . 294
ttf_raster_cache_slot_t . . . 295
ttf_raster_cache_t . . . 297
usb_bot_cbw_t . . . 298
usb_bot_csw_t . . . 299
usb_setup_packet_t . . . 300
user_identity_seed_t . . . 301
VirtBlkReq . . . 303
VirtBlkState . . . 304
VirtInputState . . . 306
VirtioCommonCfg . . . 309
VirtioInputEvent . . . 311
VirtqAvail . . . 312
VirtqDesc . . . 313
VirtqUsed . . . 314
VirtqUsedElem . . . 315
VirtRngState . . . 316
VM
Main virtual machine state container . . . 318
vm_node . . . 338
vm_physics_node . . . 339
VMFleetTouchSample . . . 340
VMFleetWindow . . . 340
VMHostServices
VMHostServices - pluggable host operations . . . 341
vmm_page_info_t . . . 344
VMPhysics . . . 345
VMRegistryEntry . . . 345
VocabularyState . . . 348
VT100State . . . 350
wirebind_live_entry_t . . . 353
WordContract . . . 354
WordProfile
Execution statistics for a single word . . . 355
WordStats
Statistics for individual word execution . . . 357

WordTestSuite
Collection of test cases for a single Forth word . . . 359
WordTransitionMetrics
SPECULATION_THRESHOLD_Q48: Minimum probability (Q48.16) to speculate on a word transition . . . 360
xhci_cap_regs_t . . . 365
xhci_dev_t . . . 367
xhci_ep_ctx32_t . . . 378
xhci_input_ctrl_ctx32_t . . . 379
xhci_intr_regs_t . . . 379
xhci_msc_slot_t . . . 381
xhci_op_regs_t . . . 387
xhci_port_regs_t . . . 389
xhci_runtime_regs_t . . . 390
xhci_slot_ctx32_t . . . 390
xhci_trb_t . . . 391
XXHash64State
Streaming state structure . . . 392
zuse_cert_devblock_t . . . 394
zuse_genesis_marker_t . . . 395
zuse_genesis_marker_v2_t . . . 396

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

kernel/include/starkernel/apic.h . . . 397
kernel/include/starkernel/arch.h . . . 409
kernel/include/starkernel/artemis_sig.h . . . 417
kernel/include/starkernel/blkio_usb.h . . . 423
kernel/include/starkernel/boot_info_offsets.h . . . 425
kernel/include/starkernel/capsule.h . . . 425
kernel/include/starkernel/capsule_birth.h . . . 435
kernel/include/starkernel/capsule_blocks.h . . . 453
kernel/include/starkernel/capsule_console.h . . . 455
kernel/include/starkernel/capsule_generated.h . . . 455
kernel/include/starkernel/capsule_loader.h . . . 456
kernel/include/starkernel/capsule_mint.h . . . 462
kernel/include/starkernel/capsule_run.h . . . 463
kernel/include/starkernel/capsule_runcap.h . . . 471
kernel/include/starkernel/capsule_sig.h . . . 471
kernel/include/starkernel/capsule_vm_physics.h . . . 474
kernel/include/starkernel/capsule_vm_switch_signal.h . . . 483
kernel/include/starkernel/capsule_wirebind.h . . . 483
kernel/include/starkernel/capsule_zuse_boot.h . . . 484
kernel/include/starkernel/cmdline.h . . . 484
kernel/include/starkernel/console.h . . . 487
kernel/include/starkernel/doe_log.h . . . 507
kernel/include/starkernel/ed25519.h . . . 512
kernel/include/starkernel/elf64.h . . . 516
kernel/include/starkernel/elf_loader.h . . . 526
kernel/include/starkernel/fdt.h . . . 528
kernel/include/starkernel/fdt_uart.h . . . 542
kernel/include/starkernel/fe25519.h . . . 544
kernel/include/starkernel/framebuffer.h . . . 553
kernel/include/starkernel/gpt.h . . . 609
kernel/include/starkernel/hal_memory.h . . . 616
kernel/include/starkernel/homeblocks_sig.h . . . 618
kernel/include/starkernel/i8042.h . . . 622
kernel/include/starkernel/ioapic.h . . . 627
kernel/include/starkernel/kbd_input.h . . . 631

kernel/include/starkernel/kernel_args.h . . . 634
kernel/include/starkernel/kmalloc.h . . . 636
kernel/include/starkernel/log.h . . . 1804
kernel/include/starkernel/log_attrib.h . . . 642
kernel/include/starkernel/log_region.h . . . 642
kernel/include/starkernel/pci.h . . . 647
kernel/include/starkernel/pie_hidden.h . . . 659
kernel/include/starkernel/plic.h . . . 659
kernel/include/starkernel/pmm.h . . . 664
kernel/include/starkernel/q48_16.h . . . 1918
kernel/include/starkernel/repl.h . . . 1943
kernel/include/starkernel/rng.h . . . 669
kernel/include/starkernel/rpi5_dtb.h . . . 672
kernel/include/starkernel/rpi5_mailbox.h . . . 674
kernel/include/starkernel/rpi5_mmu.h . . . 676
kernel/include/starkernel/rpi5_native_boot.h . . . 679
kernel/include/starkernel/rpi5_native_entry.h . . . 681
kernel/include/starkernel/scalar25519.h . . . 682
kernel/include/starkernel/session.h . . . 686
kernel/include/starkernel/sha512.h . . . 686
kernel/include/starkernel/timer.h . . . 689
kernel/include/starkernel/ttf.h . . . 706
kernel/include/starkernel/uefi.h . . . 717
kernel/include/starkernel/usb_kbd.h . . . 734
kernel/include/starkernel/user_identity_seed.h . . . 738
kernel/include/starkernel/virtio_blk.h . . . 758
kernel/include/starkernel/virtio_input.h . . . 761
kernel/include/starkernel/virtio_rng.h . . . 764
kernel/include/starkernel/vm_identity.h . . . 777
kernel/include/starkernel/vm_log_buffer.h . . . 777
kernel/include/starkernel/vm_uuid.h . . . 777
kernel/include/starkernel/vmm.h . . . 777
kernel/include/starkernel/vt100.h . . . 784
kernel/include/starkernel/x509_ed25519.h . . . 796
kernel/include/starkernel/xhci.h . . . 801
kernel/include/starkernel/xhci_driver.h . . . 820
kernel/include/starkernel/xxhash64.h . . . 848
kernel/include/starkernel/zuse_cert_devblock.h . . . 853
kernel/include/starkernel/zuse_genesis_marker.h . . . 855
kernel/include/starkernel/freestanding/assert.h . . . 564
kernel/include/starkernel/freestanding/ctype.h . . . 565
kernel/include/starkernel/freestanding/errno.h . . . 570
kernel/include/starkernel/freestanding/inttypes.h . . . 571
kernel/include/starkernel/freestanding/math.h . . . 574
kernel/include/starkernel/freestanding/sched.h . . . 576
kernel/include/starkernel/freestanding/signal.h . . . 577
kernel/include/starkernel/freestanding/stdio.h . . . 579
kernel/include/starkernel/freestanding/stdlib.h . . . 592
kernel/include/starkernel/freestanding/string.h . . . 598
kernel/include/starkernel/freestanding/time.h . . . 606
kernel/include/starkernel/freestanding/sys/time.h . . . 605
kernel/include/starkernel/freestanding/sys/types.h . . . 607
kernel/include/starkernel/hal/hal.h . . . 616
kernel/include/starkernel/v4/sk_v4.h . . . 740
kernel/include/starkernel/v4/sk_v4_grow.h . . . 742
kernel/include/starkernel/v4/sk_v4_identity.h . . . 744
kernel/include/starkernel/v4/sk_v4_marker.h . . . 754
kernel/include/starkernel/v4/sk_v4_mint.h . . . 756

kernel/include/starkernel/vm/arena.h . . . 767
kernel/include/starkernel/vm/kernel_hermes.h . . . 768
kernel/include/starkernel/vm/parity.h . . . 768
kernel/include/starkernel/vm/stadium.h . . . 775
kernel/include/starkernel/vm/stadium_blocks.h . . . 775
kernel/include/starkernel/vm/stadium_words.h . . . 776
kernel/include/starkernel/vm/switch.h . . . 776
kernel/include/starkernel/vm/bootstrap/sk_vm_bootstrap.h . . . 767
kernel/src/artemis_sig.c . . . 1061
kernel/src/doe_log.c . . . 1231
kernel/src/gpt.c . . . 1237
kernel/src/heartbeat.c . . . 1403
kernel/src/homeblocks_sig.c . . . 1410
kernel/src/kernel_main.c . . . 1412
kernel/src/log_region.c . . . 1427
kernel/src/repl.c . . . 2442
kernel/src/arch/aarch64/apic.c . . . 907
kernel/src/arch/aarch64/arch.c . . . 949
kernel/src/arch/aarch64/interrupts.c . . . 967
kernel/src/arch/aarch64/rpi5_dtb.c . . . 856
kernel/src/arch/aarch64/rpi5_mailbox.c . . . 860
kernel/src/arch/aarch64/rpi5_mmu.c . . . 871
kernel/src/arch/aarch64/rpi5_native_boot.c . . . 878
kernel/src/arch/aarch64/timer.c . . . 1006
kernel/src/arch/amd64/apic.c . . . 922
kernel/src/arch/amd64/arch.c . . . 955
kernel/src/arch/amd64/i8042.c . . . 891
kernel/src/arch/amd64/interrupts.c . . . 977
kernel/src/arch/amd64/ioapic.c . . . 899
kernel/src/arch/amd64/timer.c . . . 1012
kernel/src/arch/riscv64/apic.c . . . 939
kernel/src/arch/riscv64/arch.c . . . 961
kernel/src/arch/riscv64/interrupts.c . . . 989
kernel/src/arch/riscv64/plic.c . . . 996
kernel/src/arch/riscv64/timer.c . . . 1052
kernel/src/boot/cmdline.c . . . 1065
kernel/src/boot/elf_loader.c . . . 1071
kernel/src/boot/reloc_stub.c . . . 1078
kernel/src/boot/uefi_loader.c . . . 1081
kernel/src/capsule/capsule_birth.c . . . 1089
kernel/src/capsule/capsule_blocks.c . . . 1111
kernel/src/capsule/capsule_console.c . . . 1111
kernel/src/capsule/capsule_find.c . . . 1114
kernel/src/capsule/capsule_loader.c . . . 1117
kernel/src/capsule/capsule_mint.c . . . 1127
kernel/src/capsule/capsule_run.c . . . 1136
kernel/src/capsule/capsule_runcap.c . . . 1146
kernel/src/capsule/capsule_sig.c . . . 1147
kernel/src/capsule/capsule_validate.c . . . 1151
kernel/src/capsule/capsule_vm_hooks.c . . . 1153
kernel/src/capsule/capsule_vm_physics.c . . . 1158
kernel/src/capsule/capsule_vm_switch_signal.c . . . 1171
kernel/src/capsule/capsule_wirebind.c . . . 1171
kernel/src/capsule/capsule_zuse_boot.c . . . 1183
kernel/src/capsule/mama_forth_words.c
Mama FORTH vocabulary implementation for kernel capsule system (M7.1) . . . 1192
kernel/src/capsule/vm_identity.c . . . 1193
kernel/src/capsule/vm_uuid.c . . . 1195

kernel/src/crypto/ed25519.c . . . 1195
kernel/src/crypto/fe25519.c . . . 1206
kernel/src/crypto/scalar25519.c . . . 1215
kernel/src/crypto/sha512.c . . . 1220
kernel/src/crypto/x509_ed25519.c . . . 1224
kernel/src/hal/console.c . . . 1250
kernel/src/hal/fdt.c . . . 1284
kernel/src/hal/fdt_uart.c . . . 1306
kernel/src/hal/font_8x16.c . . . 1310
kernel/src/hal/framebuffer.c . . . 1312
kernel/src/hal/hal.c . . . 1323
kernel/src/hal/host_services.c . . . 1323
kernel/src/hal/kbd_input.c . . . 1323
kernel/src/hal/memory.c . . . 1332
kernel/src/hal/ttf.c . . . 1334
kernel/src/hal/vt100.c . . . 1356
kernel/src/hash/xxhash64.c . . . 1395
kernel/src/math/q48_16.c . . . 1433
kernel/src/memory/kmalloc.c . . . 1441
kernel/src/memory/pmm.c . . . 1457
kernel/src/memory/vmm.c . . . 1468
kernel/src/pci/pci.c . . . 1482
kernel/src/rng/rng.c . . . 1500
kernel/src/usb/blkio_usb.c . . . 1503
kernel/src/usb/usb_kbd.c . . . 1509
kernel/src/usb/xhci.c . . . 1515
kernel/src/v4/sk_v4.c . . . 1560
kernel/src/v4/sk_v4_grow.c . . . 1568
kernel/src/v4/sk_v4_identity.c . . . 1572
kernel/src/v4/sk_v4_marker.c . . . 1597
kernel/src/v4/sk_v4_mint.c . . . 1600
kernel/src/virtio/virtio_blk.c . . . 1604
kernel/src/virtio/virtio_input.c . . . 1615
kernel/src/virtio/virtio_rng.c . . . 1625
kernel/src/vm/alloc_kernel.c . . . 2400
kernel/src/vm/arena.c . . . 1633
kernel/src/vm/kernel_hermes.c . . . 1633
kernel/src/vm/parity.c . . . 1633
kernel/src/vm/q48_stubs.c . . . 1646
kernel/src/vm/session.c . . . 1648
kernel/src/vm/stadium.c . . . 1648
kernel/src/vm/stadium_blocks.c . . . 1648
kernel/src/vm/stadium_words.c . . . 1648
kernel/src/vm/switch.c . . . 1649
kernel/src/vm/vm_bootstrap.c . . . 2719
kernel/src/vm/vm_core.c . . . 1649
kernel/src/vm/vm_internal.h . . . 2734
kernel/src/vm/vm_log_buffer.c . . . 1694
kernel/src/vm/vm_runtime.c . . . 1694
kernel/src/vm/bootstrap/sk_vm_bootstrap.c . . . 1633
kernel/src/vm/host/shim.c . . . 1633
v3/include/arch_detect.h
Architecture Detection and Unified API for StarForth . . . 1708
v3/include/blkcfg.h . . . 1716
v3/include/blkio.h . . . 1719
v3/include/blkio_factory.h . . . 1724
v3/include/block_subsystem.h . . . 1727
v3/include/cli.h . . . 1759

v3/include/compudynamics.h . . . 1763
v3/include/dictionary_heat_optimization.h . . . 1770
v3/include/dictionary_management.h . . . 1780
v3/include/doe_metrics.h
Design of Experiments (DoE) metrics collection API . . . 1782
v3/include/inference_engine.h . . . 1791
v3/include/io.h . . . 1800
v3/include/log.h . . . 1806
v3/include/math_portable.h . . . 1812
v3/include/memory_management.h
Memory management functions for the StarForth virtual machine . . . 1816
v3/include/physics_execution_hooks.h . . . 1820
v3/include/physics_hotwords_cache.h . . . 1826
v3/include/physics_metadata.h . . . 1845
v3/include/physics_pipelining_metrics.h . . . 1857
v3/include/physics_runtime.h . . . 1874
v3/include/platform_alloc.h . . . 1882
v3/include/platform_io.h . . . 1891
v3/include/platform_lock.h . . . 1892
v3/include/platform_time.h . . . 1897
v3/include/profiler.h . . . 1904
v3/include/q48_16.h . . . 1926
v3/include/repl.h . . . 1958
v3/include/rolling_window_knobs.h . . . 1961
v3/include/rolling_window_of_truth.h . . . 1962
v3/include/ssm_jacquard.h . . . 1977
v3/include/starforth_config.h . . . 1988
v3/include/startup.h . . . 1998
v3/include/version.h . . . 2000
v3/include/vm.h . . . 2001
v3/include/vm_api.h
StarForth Virtual Machine Public API . . . 2077
v3/include/vm_asm_opt.h . . . 2078
v3/include/vm_asm_opt_arm64.h . . . 2080
v3/include/vm_asm_opt_riscv64.h . . . 2087
v3/include/vm_debug.h . . . 2092
v3/include/vm_host.h . . . 2095
v3/include/vm_inner_interp_arm64.h . . . 2100
v3/include/vm_inner_interp_asm.h . . . 2102
v3/include/vm_inner_interp_riscv64.h . . . 2112
v3/include/word_registry.h . . . 2120
v3/src/blkio_factory.c . . . 2124
v3/src/blkio_file.c . . . 2132
v3/src/blkio_ram.c . . . 2141
v3/src/block_subsystem.c . . . 2148
v3/src/cli.c . . . 2214
v3/src/compudynamics.c . . . 2217
v3/src/dictionary_heat_optimization.c . . . 2226
v3/src/dictionary_management.c . . . 2235
v3/src/doe_metrics.c
Design of Experiments metrics collection implementation . . . 2261
v3/src/heartbeat_export.c . . . 2270
v3/src/inference_engine.c . . . 2272
v3/src/io.c . . . 2288
v3/src/log.c . . . 2292
v3/src/main.c . . . 2301
v3/src/math_portable.c . . . 2312
v3/src/memory_management.c . . . 2316

v3/src/physics_execution_hooks.c . . . 2321
v3/src/physics_hotwords_cache.c . . . 2326
v3/src/physics_metadata.c . . . 2348
v3/src/physics_pipelining_metrics.c . . . 2362
v3/src/physics_runtime.c . . . 2380
v3/src/profiler.c
Performance profiling implementation for StarForth . . . 2424
v3/src/repl.c . . . 2468
v3/src/rolling_window_of_truth.c . . . 2471
v3/src/ssm_jacquard.c . . . 2494
v3/src/stack_management.c
Implementation of stack operations for the Forth virtual machine . . . 2506
v3/src/vm.c . . . 2694
v3/src/vm_api.c . . . 2712
v3/src/vm_bootstrap.c . . . 2726
v3/src/vm_debug.c . . . 2728
v3/src/vm_internal.h . . . 2746
v3/src/vm_time.c . . . 2765
v3/src/word_registry.c . . . 2773
v3/src/platform/alloc_host.c . . . 2397
v3/src/platform/alloc_kernel.c . . . 2409
v3/src/platform/platform_init.c . . . 2418
v3/src/platform/threading.c . . . 2419
v3/src/platform/l4re/time.c . . . 2414
v3/src/platform/linux/io.c . . . 2291
v3/src/platform/linux/time.c . . . 2415
v3/src/test_runner/test_common.c . . . 2651
v3/src/test_runner/test_contracts.c . . . 2674
v3/src/test_runner/test_runner.c . . . 2686
v3/src/test_runner/include/test_common.h
Common test utilities and structures for StarForth test runner . . . 2512
v3/src/test_runner/include/test_contracts.h . . . 2534
v3/src/test_runner/include/test_runner.h . . . 2545
v3/src/test_runner/modules/acl_words_test.c . . . 2580
v3/src/test_runner/modules/arithmetic_words_test.c . . . 2599
v3/src/test_runner/modules/block_words_test.c . . . 2601
v3/src/test_runner/modules/break_me_tests.c . . . 2603
v3/src/test_runner/modules/control_words_test.c . . . 2607
v3/src/test_runner/modules/defining_words_tests.c . . . 2609
v3/src/test_runner/modules/dictionary_manipulation_words_test.c . . . 2611
v3/src/test_runner/modules/dictionary_words_test.c . . . 2613
v3/src/test_runner/modules/double_words_test.c . . . 2615
v3/src/test_runner/modules/format_words_test.c . . . 2616
v3/src/test_runner/modules/inference_words_test.c . . . 2618
v3/src/test_runner/modules/integration_tests.c
Full program integration tests for StarForth . . . 2620
v3/src/test_runner/modules/io_words_test.c . . . 2623
v3/src/test_runner/modules/logical_words_test.c . . . 2625
v3/src/test_runner/modules/mama_forth_words_test.c
Test module for Mama FORTH vocabulary (M7.1 Capsule System) . . . 2626
v3/src/test_runner/modules/memory_words_test.c . . . 2629
v3/src/test_runner/modules/mixed_arithmetic_words_test.c . . . 2631
v3/src/test_runner/modules/physics_freeze_words_test.c . . . 2633
v3/src/test_runner/modules/q48_words_test.c . . . 2635
v3/src/test_runner/modules/return_stack_words_test.c . . . 2636
v3/src/test_runner/modules/stack_words_test.c
Test module for Forth stack manipulation words . . . 2638
v3/src/test_runner/modules/starforth_words_test.c . . . 2641

v3/src/test_runner/modules/stress_tests.c
FORTH-79 Standard Comprehensive Stress Testing Module . . . 2643
v3/src/test_runner/modules/string_words_test.c . . . 2645
v3/src/test_runner/modules/system_words_test.c . . . 2647
v3/src/test_runner/modules/vocabulary_words_test.c . . . 2649
v3/src/word_source/acl_words.c . . . 2777
v3/src/word_source/arithmetic_words.c . . . 2792
v3/src/word_source/block_words.c . . . 2806
v3/src/word_source/control_words.c . . . 2834
v3/src/word_source/defer_words.c . . . 2836
v3/src/word_source/defining_words.c . . . 2841
v3/src/word_source/dictionary_heat_diagnostic_words.c . . . 2860
v3/src/word_source/dictionary_manipulation_words.c . . . 2866
v3/src/word_source/dictionary_words.c . . . 2879
v3/src/word_source/double_words.c . . . 2888
v3/src/word_source/editor_words.c . . . 2902
v3/src/word_source/format_words.c . . . 2909
v3/src/word_source/framebuffer_words.c . . . 2926
v3/src/word_source/inference_words.c . . . 3029
v3/src/word_source/io_words.c . . . 3045
v3/src/word_source/keyboard_words.c . . . 3055
v3/src/word_source/lifecycle_words_hosted.c . . . 3062
v3/src/word_source/log_words.c . . . 3068
v3/src/word_source/logical_words.c . . . 3085
v3/src/word_source/memory_words.c . . . 3100
v3/src/word_source/mixed_arithmetic_words.c . . . 3114
v3/src/word_source/physics_benchmark_words.c . . . 3123
v3/src/word_source/physics_diagnostic_words.c . . . 3131
v3/src/word_source/physics_freeze_words.c . . . 3135
v3/src/word_source/physics_pipelining_diagnostic_words.c . . . 3145
v3/src/word_source/q48_16_words.c . . . 3151
v3/src/word_source/q48_words.c . . . 3159
v3/src/word_source/return_stack_words.c
Implementation of FORTH-79 standard return stack words . . . 3179
v3/src/word_source/scroll_words.c . . . 3184
v3/src/word_source/stack_words.c . . . 3185
v3/src/word_source/starforth_words.c . . . 3195
v3/src/word_source/string_words.c . . . 3211
v3/src/word_source/system_words.c . . . 3233
v3/src/word_source/ttf_words.c . . . 3248
v3/src/word_source/vocabulary_words.c . . . 3249
v3/src/word_source/include/acl_words.h . . . 2930
v3/src/word_source/include/arithmetic_words.h . . . 2933
v3/src/word_source/include/block_words.h
FORTH-79 Block and Buffer System Implementation . . . 2936
v3/src/word_source/include/control_words.h . . . 2948
v3/src/word_source/include/defer_words.h . . . 2949
v3/src/word_source/include/defining_words.h . . . 2951
v3/src/word_source/include/dictionary_heat_diagnostic_words.h . . . 2955
v3/src/word_source/include/dictionary_manipulation_words.h . . . 2957
v3/src/word_source/include/dictionary_words.h . . . 2959
v3/src/word_source/include/double_words.h . . . 2961
v3/src/word_source/include/editor_words.h . . . 2964
v3/src/word_source/include/format_words.h . . . 2966
v3/src/word_source/include/framebuffer_words.h . . . 2968
v3/src/word_source/include/inference_words.h . . . 2970
v3/src/word_source/include/io_words.h . . . 2972
v3/src/word_source/include/keyboard_words.h . . . 2975

v3/src/word_source/include/log_words.h . . . 2978
v3/src/word_source/include/logical_words.h . . . 2980
v3/src/word_source/include/mama_forth_words.h
Mama FORTH vocabulary words for kernel capsule system (M7.1) . . . 2983
v3/src/word_source/include/memory_words.h . . . 2984
v3/src/word_source/include/mixed_arithmetic_words.h . . . 2986
v3/src/word_source/include/physics_benchmark_words.h . . . 2988
v3/src/word_source/include/physics_diagnostic_words.h . . . 2990
v3/src/word_source/include/physics_freeze_words.h . . . 2995
v3/src/word_source/include/physics_pipelining_diagnostic_words.h . . . 2997
v3/src/word_source/include/q48_words.h . . . 2999
v3/src/word_source/include/return_stack_words.h . . . 3002
v3/src/word_source/include/scroll_words.h . . . 3004
v3/src/word_source/include/stack_words.h . . . 3005
v3/src/word_source/include/starforth_words.h . . . 3007
v3/src/word_source/include/string_words.h
FORTH-79 Standard string and text processing words implementation . . . 3017
v3/src/word_source/include/system_words.h . . . 3020
v3/src/word_source/include/ttf_words.h . . . 3023
v3/src/word_source/include/vocabulary_words.h . . . 3024

Chapter 5

Topic Documentation

5.1 Architecture Detection

Automatic detection of target architecture and inclusion of optimization headers.

Automatic detection of target architecture and inclusion of optimization headers.

This module automatically detects the target architecture at compile time and includes the appropriate assembly optimization headers.

Supported architectures:

- x86_64 (AMD64, Intel 64)
- ARM64 (AArch64, ARMv8-A)

Unsupported architectures fall back to pure C implementations.

5.2 Unified API Wrappers

Consistent API macros across all architectures.

Consistent API macros across all architectures.

These macros provide a consistent API across all architectures by automatically dispatching to the appropriate optimized or fallback implementation based on the target architecture and available optimizations.

5.3 Runtime Architecture Detection

Runtime detection of CPU features.

Runtime detection of CPU features.

Functions for detecting available CPU features at runtime, such as SIMD support (SSE4.2, AVX, NEON) and other architecture-specific capabilities.

5.4 Build Information

Functions for querying build configuration.

Functions for querying build configuration.

Provides information about the build configuration, including target architecture, enabled optimizations, and threading model.

5.5 VM Input Operations

Public API for managing VM input operations including Terminal Input Buffer (TIB), input pointer (>IN), and input span (SPAN) handling.

Functions

- int vm_input_ensure (VM *vm)

  Ensure input buffers are allocated.
- unsigned char * vm_input_tib (VM *vm)

  Get pointer to terminal input buffer (TIB)
- cell_t * vm_input_in (VM *vm)

  Get pointer to input offset variable.
- cell_t * vm_input_span (VM *vm)

  Get pointer to input span variable.
- void vm_input_load_line (VM *vm, const char *src, size_t n)

  Load input line into TIB.
- void vm_input_source (VM *vm, cell_t *out_addr, cell_t *out_len)

  Get current input source buffer info.
- cell_t vm_addr_from_ptr (VM *vm, void *p)

  Convert native pointer to Forth memory address.

5.5.1 Detailed Description

Public API for managing VM input operations including Terminal Input Buffer (TIB), input pointer (>IN), and input span (SPAN) handling.

5.5.2 Function Documentation

5.5.2.1 vm_addr_from_ptr()

```
cell_t vm_addr_from_ptr (
            VM * vm,
            void * p )
```

Convert native pointer to Forth memory address.

Parameters

vm	VM instance
p	Native pointer to convert

Returns

Forth memory address (offset from VM memory base)

Converts a native C pointer to a Forth memory address by calculating offset from VM memory base. Returns 0 for invalid pointers.

References base, and VM::memory.

5.5.2.2 vm_input_ensure()

```
int vm_input_ensure (
            VM * vm )
```

Ensure input buffers are allocated.

Parameters

vm	VM instance

Returns

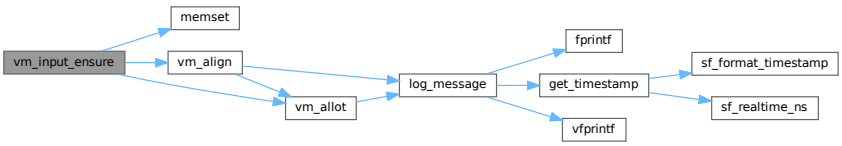
0 on success, 1 on error

Allocates TIB buffer and input variables if not already allocated

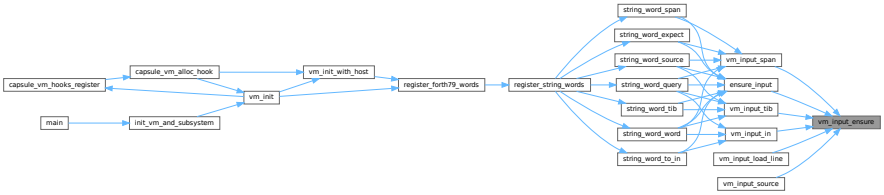
References VM::error, VM::in_var, INPUT_BUFFER_SIZE, memset(), VM::span_var, VM::tib_buf, VM::tib_cap, vm_align(), and vm_allot().

Referenced by ensure_input(), vm_input_in(), vm_input_load_line(), vm_input_source(), vm_input_span(), and vm_input_tib().

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.3 vm_input_in()

```
cell_t * vm_input_in (
            VM * vm )
```

Get pointer to input offset variable.

Parameters

vm	VM instance

Returns

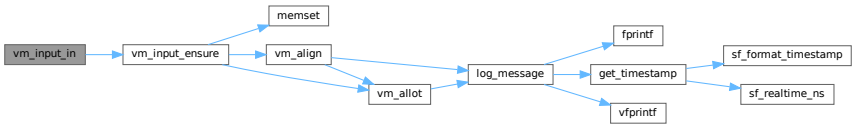
Pointer to input offset or NULL on error

Returns pointer to VM's input offset variable after ensuring it exists

References VM::in_var, and vm_input_ensure().

Referenced by string_word_query(), string_word_to_in(), and string_word_word().

Here is the call graph for this function:



Here is the caller graph for this function:

5.5.2.4 vm_input_load_line()

```
void vm_input_load_line (
            VM * vm,
            const char * src,
            size_t n )
```

Load input line into TIB.

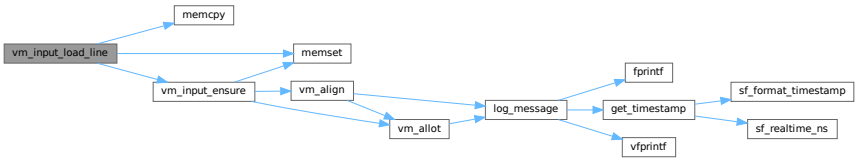
Parameters

vm	VM instance
src	Source string to load
n	Number of characters to load

Loads up to n characters from src into TIB, zero-pads remainder Resets input offset and updates span. Handles NULL src gracefully.

References VM::in_var, memcpy(), memset(), VM::span_var, VM::tib_buf, VM::tib_cap, and vm_input_ensure().

Here is the call graph for this function:



5.5.2.5 vm_input_source()

```
void vm_input_source (
            VM * vm,
            cell_t * out_addr,
            cell_t * out_len )
```

Get current input source buffer info.

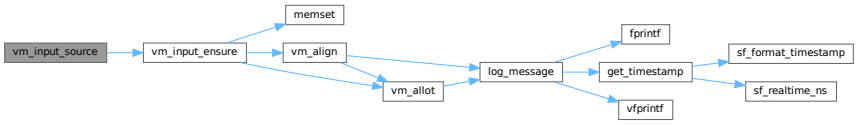
Parameters

vm	VM instance
out_addr	Output parameter for buffer address
out_len	Output parameter for buffer length

Returns current input buffer address and length through parameters Both parameters are optional (can be NULL)

References VM::span_var, VM::tib_buf, and vm_input_ensure().

Here is the call graph for this function:



5.5.2.6 vm_input_span()

```
cell_t * vm_input_span (
            VM * vm )
```

Get pointer to input span variable.

Parameters

vm	VM instance

Returns

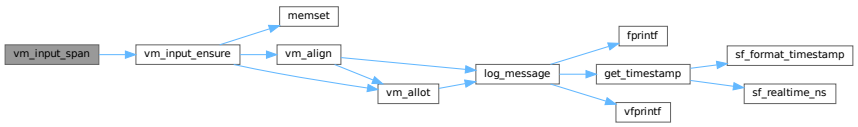
Pointer to span variable or NULL on error

Returns pointer to VM's input span variable after ensuring it exists

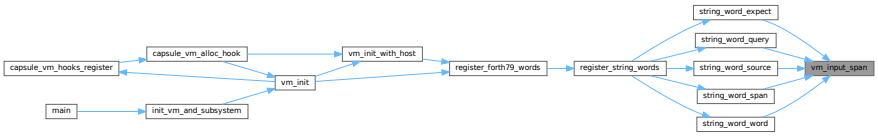
References VM::span_var, and vm_input_ensure().

Referenced by string_word_expect(), string_word_query(), string_word_source(), string_word_span(), and string_word_word().

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.7 vm_input_tib()

```
unsigned char * vm_input_tib (
            VM * vm )
```

Get pointer to terminal input buffer (TIB)

Parameters

vm	VM instance

Returns

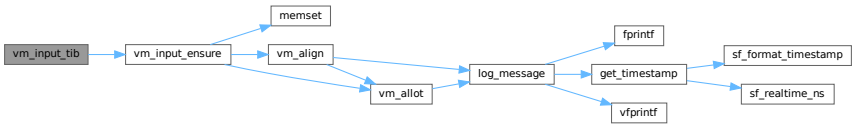
Pointer to TIB or NULL on error

Returns pointer to VM's terminal input buffer after ensuring it exists

References VM::tib_buf, and vm_input_ensure().

Referenced by string_word_query(), string_word_source(), string_word_tib(), and string_word_word().

Here is the call graph for this function:



Here is the caller graph for this function:

5.6 Stack Operations

Optimized stack manipulation operations.

Functions

- static void vm_push_asm (VM *vm, cell_t value)
  Optimized data stack push operation.
- static cell_t vm_pop_asm (VM *vm)
  Optimized data stack pop operation.
- static void vm_rpush_asm (VM *vm, cell_t value)
  Optimized return stack push operation.
- static cell_t vm_rpop_asm (VM *vm)
  Optimized return stack pop operation.
- static int vm_add_check_overflow (cell_t a, cell_t b, cell_t *result)
  Performs addition with hardware overflow detection.
- static int vm_sub_check_overflow (cell_t a, cell_t b, cell_t *result)
  Performs subtraction with hardware overflow detection.
- static void vm_mul_double (cell_t a, cell_t b, cell_t *hi, cell_t *lo)
  Performs double-width multiplication.
- static void vm_divmod (cell_t dividend, cell_t divisor, cell_t *quotient, cell_t *remainder)
  Performs division with remainder.
- static int vm_strcmp_asm (const char *s1, const char *s2, size_t len)
  Hardware-accelerated string comparison.
- static void vm_memcpy_asm (void *dest, const void *src, size_t len)
  Hardware-accelerated memory copy.
- static void vm_memzero_asm (void *dest, size_t len)
  Hardware-accelerated memory zero.
- static cell_t vm_min_asm (cell_t a, cell_t b)
  Branchless minimum calculation.
- static cell_t vm_max_asm (cell_t a, cell_t b)
  Branchless maximum calculation.
- static void vm_cpuid (uint32_t leaf, uint32_t *eax, uint32_t *ebx, uint32_t *ecx, uint32_t *edx)
  Wrapper for CPUID instruction.
- static int vm_has_sse42 (void)
  Checks for SSE4.2 support.

5.6.1 Detailed Description

Optimized stack manipulation operations.

Standard FORTH-79 stack manipulation words.

Critical optimizations for vm_push, vm_pop, vm_rpush, vm_rpop

Impact: EXTREME - These are called on nearly every Forth word execution

Benefits:

- Eliminates function call overhead (inline)
- Reduces branch mispredictions with conditional moves
- Keeps frequently-used values in registers
- Uses lea for pointer arithmetic (no flags affected)

L4Re Compatibility: YES - Pure computational code, no syscalls

The following FORTH words are implemented for stack manipulation:

- DROP (n –) Remove top stack item
- DUP (n – n n) Duplicate top stack item

- ?DUP (n – n n | n – 0) Duplicate if non-zero
- SWAP (n1 n2 – n2 n1) Exchange top two stack items
- OVER (n1 n2 – n1 n2 n1) Copy second stack item to top
- ROT (n1 n2 n3 – n2 n3 n1) Rotate top three stack items
- -ROT (n1 n2 n3 – n3 n1 n2) Reverse rotate top three items
- DEPTH (– n) Return number of stack items
- PICK (n – stack[n]) Copy nth stack item to top
- ROLL (n –) Move nth stack item to top

5.6.2 Function Documentation

5.6.2.1 vm_add_check_overflow()

```
static int vm_add_check_overflow (
            cell_t a,
            cell_t b,
            cell_t * result )  [inline], [static]
```

Performs addition with hardware overflow detection.

Parameters

a	First operand
b	Second operand
result	Pointer to store result

Returns

1 if overflow occurred, 0 otherwise

5.6.2.2 vm_cpuid()

```
static void vm_cpuid (
            uint32_t leaf,
            uint32_t * eax,
            uint32_t * ebx,
            uint32_t * ecx,
            uint32_t * edx )  [inline], [static]
```

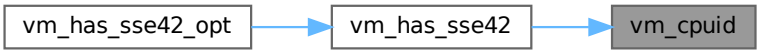
Wrapper for CPUID instruction.

Parameters

leaf	CPUID leaf number
eax	Pointer to store EAX result
ebx	Pointer to store EBX result
ecx	Pointer to store ECX result
edx	Pointer to store EDX result

Referenced by vm_has_sse42().

Here is the caller graph for this function:



5.6.2.3 vm_divmod()

```
static void vm_divmod (
             cell_t dividend,
             cell_t divisor,
             cell_t * quotient,
             cell_t * remainder )  [inline], [static]
```

Performs division with remainder.

Parameters

dividend	Dividend value
divisor	Divisor value
quotient	Pointer to store quotient
remainder	Pointer to store remainder

5.6.2.4 vm_has_sse42()

```
static int vm_has_sse42 (
             void  )  [inline], [static]
```

Checks for SSE4.2 support.

Returns

1 if SSE4.2 is supported, 0 otherwise

References vm_cpuid().

Referenced by vm_has_sse42_opt().

Here is the call graph for this function:

Here is the caller graph for this function:

5.6.2.5 vm_max_asm()

```
static cell_t vm_max_asm (
            cell_t a,
            cell_t b )  [inline], [static]
```

Branchless maximum calculation.

Parameters

a	First value
b	Second value

Returns

Larger of a and b

5.6.2.6 vm_memcpy_asm()

```
static void vm_memcpy_asm (
            void * dest,
```

```
            const void * src,
            size_t len )  [inline], [static]
```

Hardware-accelerated memory copy.

Parameters

dest	Destination buffer
src	Source buffer
len	Number of bytes to copy

5.6.2.7 vm_memzero_asm()

```
static void vm_memzero_asm (
            void * dest,
            size_t len )  [inline], [static]
```

Hardware-accelerated memory zero.

Parameters

dest	Buffer to zero
len	Number of bytes to zero

5.6.2.8 vm_min_asm()

```
static cell_t vm_min_asm (
            cell_t a,
            cell_t b )  [inline], [static]
```

Branchless minimum calculation.

Parameters

a	First value
b	Second value

Returns

Smaller of a and b

5.6.2.9 vm_mul_double()

```
static void vm_mul_double (
            cell_t a,
            cell_t b,
            cell_t * hi,
            cell_t * lo )  [inline], [static]
```

Performs double-width multiplication.

Parameters

a	First operand
b	Second operand
hi	Pointer to store high 64 bits
lo	Pointer to store low 64 bits

5.6.2.10 vm_pop_asm()

```
static cell_t vm_pop_asm (
            VM * vm )  [inline], [static]
```

Optimized data stack pop operation.

Parameters

vm	Pointer to VM context

Returns

Top value from stack, 0 on underflow

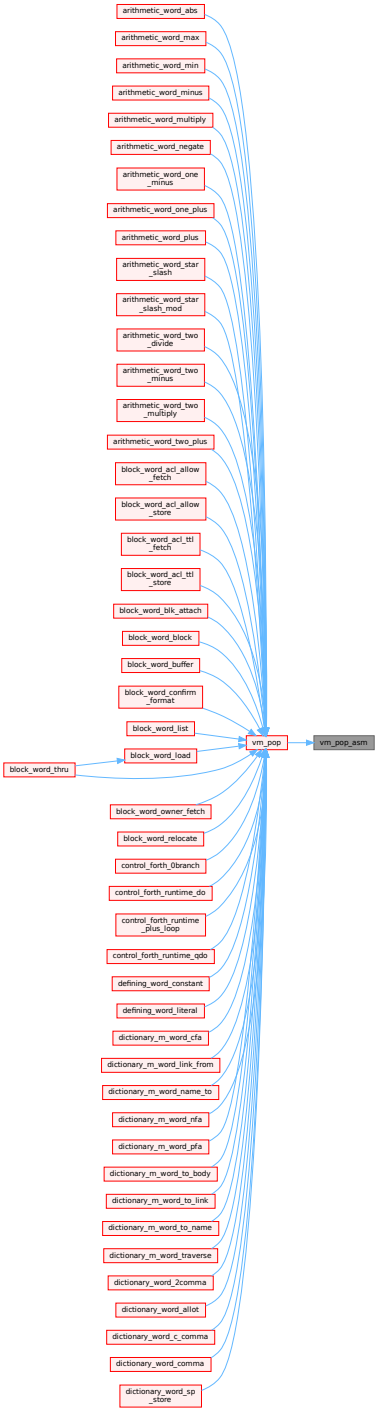
Note

Sets vm->error on stack underflow

References VM::data_stack, VM::dsp, and VM::error.

Referenced by vm_pop().

Here is the caller graph for this function:



5.6.2.11 vm_push_asm()

```
static void vm_push_asm (
            VM * vm,
            cell_t value )  [inline], [static]
```

Optimized data stack push operation.

Parameters

vm	Pointer to VM context
value	Value to push onto stack

Returns

void

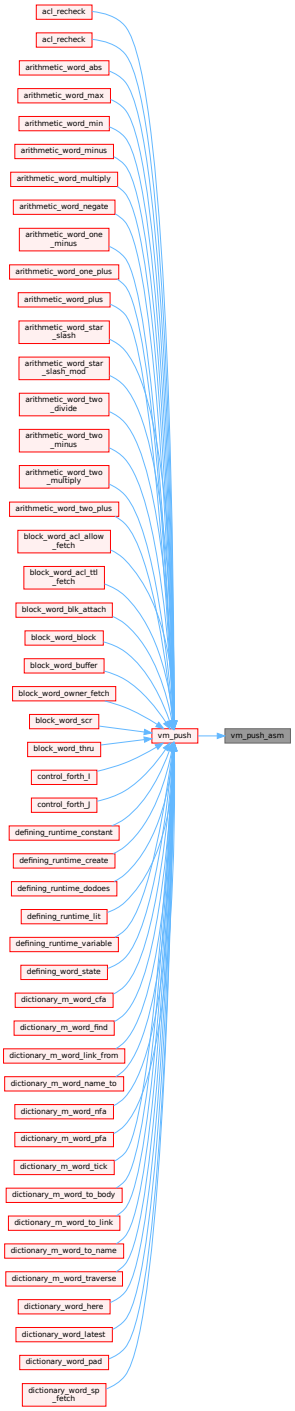
Note

Sets vm->error on stack overflow

References VM::data_stack, VM::dsp, and VM::error.

Referenced by vm_push().

Here is the caller graph for this function:



5.6.2.12 vm_rpop_asm()

```
static cell_t vm_rpop_asm (
            VM * vm )  [inline], [static]
```

Optimized return stack pop operation.

Parameters

vm	Pointer to VM context

Returns

Top value from return stack, 0 on underflow

Note

Sets vm->error on stack underflow

References VM::error, VM::return_stack, and VM::rsp.

Referenced by vm_rpop().

Here is the caller graph for this function:

5.6.2.13 vm_rpush_asm()

```
static void vm_rpush_asm (
            VM * vm,
            cell_t value )  [inline], [static]
```

Optimized return stack push operation.

Parameters

vm	Pointer to VM context
value	Value to push onto return stack

Returns

void

Note

Sets vm->error on stack overflow

References VM::error, VM::return_stack, and VM::rsp.

Referenced by vm_rpush().

Here is the caller graph for this function:

5.6.2.14 vm_strcmp_asm()

```
static int vm_strcmp_asm (
            const char * s1,
            const char * s2,
            size_t len )  [inline], [static]
```

Hardware-accelerated string comparison.

Parameters

s1	First string
s2	Second string
len	Length to compare

Returns

0 if equal, non-zero otherwise

5.6.2.15 vm_sub_check_overflow()

```
static int vm_sub_check_overflow (
            cell_t a,
            cell_t b,
            cell_t * result )  [inline], [static]
```

Performs subtraction with hardware overflow detection.

Parameters

a	First operand (minuend)
b	Second operand (subtrahend)
result	Pointer to store result

Returns

1 if overflow occurred, 0 otherwise

5.7 Fast Primitive Word Implementations

Register-based stack primitives optimized for ARM64.

Macros

- #define PRIM_DUP_ARM64()

  DUP primitive - duplicates top of stack item.
- #define PRIM_DROP_ARM64()
- #define PRIM_SWAP_ARM64()
- #define PRIM_OVER_ARM64()
- #define PRIM_ROT_ARM64()
- #define PRIM_PLUS_ARM64()
- #define PRIM_MINUS_ARM64()
- #define PRIM_STAR_ARM64()
- #define PRIM_SLASH_ARM64()
- #define PRIM_MOD_ARM64()
- #define PRIM_AND_ARM64()
- #define PRIM_OR_ARM64()
- #define PRIM_XOR_ARM64()
- #define PRIM_INVERT_ARM64()
- #define PRIM_NEGATE_ARM64()
- #define PRIM_FETCH_ARM64()
- #define PRIM_STORE_ARM64()
- #define PRIM_C_FETCH_ARM64()
- #define PRIM_C_STORE_ARM64()
- #define PRIM_TO_R_ARM64()
- #define PRIM_R_FROM_ARM64()
- #define PRIM_R_FETCH_ARM64()
- #define PRIM_2DUP_ARM64()
- #define PRIM_2DROP_ARM64()
- #define PRIM_ZERO_EQUALS_ARM64()
- #define PRIM_ZERO_LESS_ARM64()
- #define PRIM_EQUALS_ARM64()
- #define PRIM_LESS_ARM64()
- #define PRIM_GREATER_ARM64()
- #define PRIM_BRANCH_ARM64()
- #define PRIM_ZBRANCH_ARM64()
- #define PRIM_EXECUTE_ARM64()

5.7.1 Detailed Description

Register-based stack primitives optimized for ARM64.

Key optimization: TOS stays in x23, avoiding memory traffic

5.7.2 Macro Definition Documentation

5.7.2.1 PRIM_2DROP_ARM64

```
#define PRIM_2DROP_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x23, [x21, #-8]\n\t"     /* Load new TOS */ \
        "sub     x21, x21, #16\n\t"       /* DSP -= 2 */ \
        ::: "x21", "x23", "memory" \
    )
```

5.7.2.2 PRIM_2DUP_ARM64

```
#define PRIM_2DUP_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21]\n\t"           /* Load NOS */ \
        "stp     x0, x23, [x21, #8]\n\t"  /* Store pair */ \
        "add     x21, x21, #16\n\t"       /* DSP += 2 */ \
        ::: "x0", "x21", "memory" \
    )
```

5.7.2.3 PRIM_AND_ARM64

```
#define PRIM_AND_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t" \
        "and     x23, x23, x0\n\t" \
        ::: "x0", "x21", "x23" \
    )
```

5.7.2.4 PRIM_BRANCH_ARM64

```
#define PRIM_BRANCH_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x20]\n\t"           /* Load offset */ \
        "add     x20, x20, x0\n\t"        /* IP += offset */ \
        ::: "x0", "x20", "memory" \
    )
```

5.7.2.5 PRIM_C_FETCH_ARM64

```
#define PRIM_C_FETCH_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x19]\n\t" \
        "ldrb    w23, [x0, x23]\n\t"      /* Load byte, zero-extend */ \
        ::: "x0", "x23", "memory" \
    )
```

5.7.2.6 PRIM_C_STORE_ARM64

```
#define PRIM_C_STORE_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t" \
        "ldr     x1, [x19]\n\t" \
        "strb    w23, [x1, x0]\n\t"       /* Store byte */ \
        "ldr     x23, [x21]\n\t" \
        ::: "x0", "x1", "x21", "x23", "memory" \
    )
```

5.7.2.7 PRIM_DROP_ARM64

```
#define PRIM_DROP_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x23, [x21], #-8\n\t"     /* Load new TOS, decrement DSP */ \
        ::: "x21", "x23", "memory" \
    )
```

5.7.2.8 PRIM_DUP_ARM64

```
#define PRIM_DUP_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "str     x23, [x21, #8]!\n\t"     /* Store TOS, advance DSP */ \
        ::: "x21", "memory" \
    )
```

DUP primitive - duplicates top of stack item.

5.7.2.9 PRIM_EQUALS_ARM64

```
#define PRIM_EQUALS_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t" \
        "cmp     x0, x23\n\t" \
        "cset    x23, eq\n\t" \
        "neg     x23, x23\n\t" \
        ::: "x0", "x21", "x23", "cc" \
    )
```

5.7.2.10 PRIM_EXECUTE_ARM64

```
#define PRIM_EXECUTE_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x23, [x21], #-8\n\t"     /* Load new TOS */ \
        "br      x0\n\t"                  /* Branch to xt */ \
        ::: "x0", "x21", "x23", "memory" \
    )
```

5.7.2.11 PRIM_FETCH_ARM64

```
#define PRIM_FETCH_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x19]\n\t"           /* Load VM->memory */ \
        "ldr     x23, [x0, x23]\n\t"      /* TOS = memory[TOS] */ \
        ::: "x0", "x23", "memory" \
    )
```

5.7.2.12 PRIM_GREATER_ARM64

```
#define PRIM_GREATER_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t" \
        "cmp     x0, x23\n\t" \
        "cset    x23, gt\n\t" \
        "neg     x23, x23\n\t" \
        ::: "x0", "x21", "x23", "cc" \
    )
```

5.7.2.13 PRIM_INVERT_ARM64

```
#define PRIM_INVERT_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "mvn     x23, x23\n\t" \
        ::: "x23" \
    )
```

5.7.2.14 PRIM_LESS_ARM64

```
#define PRIM_LESS_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t" \
        "cmp     x0, x23\n\t" \
        "cset    x23, lt\n\t" \
        "neg     x23, x23\n\t" \
        ::: "x0", "x21", "x23", "cc" \
    )
```

5.7.2.15 PRIM_MINUS_ARM64

```
#define PRIM_MINUS_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t"      /* Load NOS, decrement DSP */ \
        "sub     x23, x0, x23\n\t"        /* TOS = NOS - TOS */ \
        ::: "x0", "x21", "x23" \
    )
```

5.7.2.16 PRIM_MOD_ARM64

```
#define PRIM_MOD_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t"      /* Load NOS, decrement DSP */ \
        "sdiv    x1, x0, x23\n\t"         /* quotient = NOS / TOS */ \
        "msub    x23, x1, x23, x0\n\t"    /* TOS = NOS - quotient * TOS */ \
        ::: "x0", "x1", "x21", "x23" \
    )
```

5.7.2.17 PRIM_NEGATE_ARM64

```
#define PRIM_NEGATE_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "neg     x23, x23\n\t" \
        ::: "x23" \
    )
```

5.7.2.18 PRIM_OR_ARM64

```
#define PRIM_OR_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t" \
        "orr     x23, x23, x0\n\t" \
        ::: "x0", "x21", "x23" \
    )
```

5.7.2.19 PRIM_OVER_ARM64

```
#define PRIM_OVER_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21]\n\t"           /* Load NOS */ \
        "str     x23, [x21, #8]!\n\t"     /* Store TOS, advance DSP */ \
        "mov     x23, x0\n\t"             /* TOS = NOS */ \
        ::: "x0", "x21", "x23", "memory" \
    )
```

5.7.2.20 PRIM_PLUS_ARM64

```
#define PRIM_PLUS_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t"      /* Load NOS, decrement DSP */ \
        "add     x23, x23, x0\n\t"        /* TOS = TOS + NOS */ \
        ::: "x0", "x21", "x23" \
    )
```

5.7.2.21 PRIM_R_FETCH_ARM64

```
#define PRIM_R_FETCH_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "str     x23, [x21, #8]!\n\t"     /* Store TOS, advance DSP */ \
        "ldr     x23, [x22, #-8]\n\t"     /* Load from RS (don't pop) */ \
        ::: "x21", "x23", "memory" \
    )
```

5.7.2.22 PRIM_R_FROM_ARM64

```
#define PRIM_R_FROM_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "str     x23, [x21, #8]!\n\t"     /* Store TOS, advance DSP */ \
        "ldr     x23, [x22, #-8]!\n\t"    /* Load from RS, decrement RSP */ \
        ::: "x21", "x22", "x23", "memory" \
    )
```

5.7.2.23 PRIM_ROT_ARM64

```
#define PRIM_ROT_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21]\n\t"           /* Load item2 */ \
        "ldr     x1, [x21, #-8]\n\t"      /* Load item3 */ \
        "str     x1, [x21]\n\t"           /* item3 -> NOS */ \
        "str     x23, [x21, #-8]\n\t"     /* TOS -> item3 */ \
        "mov     x23, x0\n\t"             /* item2 -> TOS */ \
        ::: "x0", "x1", "x23", "memory" \
    )
```

5.7.2.24 PRIM_SLASH_ARM64

```
#define PRIM_SLASH_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t"      /* Load NOS (dividend), decrement DSP */ \
        "sdiv    x23, x0, x23\n\t"        /* TOS = NOS / TOS */ \
        ::: "x0", "x21", "x23" \
    )
```

5.7.2.25 PRIM_STAR_ARM64

```
#define PRIM_STAR_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t"      /* Load NOS, decrement DSP */ \
        "mul     x23, x23, x0\n\t"        /* TOS = TOS * NOS */ \
        ::: "x0", "x21", "x23" \
    )
```

5.7.2.26 PRIM_STORE_ARM64

```
#define PRIM_STORE_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t"      /* Load address, decrement DSP */ \
        "ldr     x1, [x19]\n\t"           /* Load VM->memory */ \
        "str     x23, [x1, x0]\n\t"       /* memory[addr] = value */ \
        "ldr     x23, [x21]\n\t"          /* Load new TOS */ \
        ::: "x0", "x1", "x21", "x23", "memory" \
    )
```

5.7.2.27 PRIM_SWAP_ARM64

```
#define PRIM_SWAP_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21]\n\t"           /* Load NOS */ \
        "str     x23, [x21]\n\t"          /* Store TOS to NOS position */ \
        "mov     x23, x0\n\t"             /* TOS = old NOS */ \
        ::: "x0", "x23", "memory" \
    )
```

5.7.2.28 PRIM_TO_R_ARM64

```
#define PRIM_TO_R_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "str     x23, [x22], #8\n\t"      /* Store to RS, advance RSP */ \
        "ldr     x23, [x21], #-8\n\t"     /* Load new TOS, decrement DSP */ \
        ::: "x21", "x22", "x23", "memory" \
    )
```

5.7.2.29 PRIM_XOR_ARM64

```
#define PRIM_XOR_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x21], #-8\n\t" \
        "eor     x23, x23, x0\n\t" \
        ::: "x0", "x21", "x23" \
    )
```

5.7.2.30 PRIM_ZBRANCH_ARM64

```
#define PRIM_ZBRANCH_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x20]\n\t"           /* Load offset */ \
        "cmp     x23, #0\n\t" \
        "add     x1, x20, x0\n\t"         /* Calculate target */ \
        "add     x2, x20, #8\n\t"         /* Calculate skip */ \
        "csel    x20, x1, x2, eq\n\t"     /* Select target if zero */ \
        "ldr     x23, [x21], #-8\n\t"     /* Load new TOS */ \
        ::: "x0", "x1", "x2", "x20", "x21", "x23", "cc", "memory" \
    )
```

5.7.2.31 PRIM_ZERO_EQUALS_ARM64

```
#define PRIM_ZERO_EQUALS_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "cmp     x23, #0\n\t" \
        "cset    x23, eq\n\t"             /* Set to 1 if zero, else 0 */ \
        "neg     x23, x23\n\t"            /* Convert to -1 (true flag) */ \
        ::: "x23", "cc" \
    )
```

5.7.2.32 PRIM_ZERO_LESS_ARM64

```
#define PRIM_ZERO_LESS_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "cmp     x23, #0\n\t" \
        "cset    x23, lt\n\t" \
        "neg     x23, x23\n\t" \
        ::: "x23", "cc" \
    )
```

5.8 Control Flow Stack

Compile-time control flow stack for tracking branch targets and loop structures.

Functions

- static void cf_epoch_sync (VM *vm)
- static int cf_push_item (VM *vm, cf_tag_t tag, size_t mark)
- static int cf_pop_item (VM *vm, cf_item_t *out)
- static int cf_peek_item (VM *vm, cf_item_t *out)

5.8.1 Detailed Description

Compile-time control flow stack for tracking branch targets and loop structures.

The types (cf_tag_t, cf_item_t, ControlFlowState) and the CF_STACK_MAX constant live in include/vm.h and the storage itself lives per-VM at vm->cf, so two VMs compiling colon definitions concurrently never share nesting state (see include/vm.h's ControlFlowState comment).

5.8.2 Function Documentation

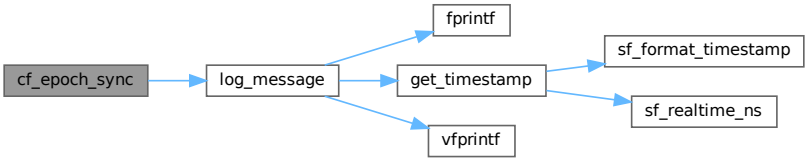
5.8.2.1 cf_epoch_sync()

```
static void cf_epoch_sync (
            VM * vm )  [inline], [static]
```

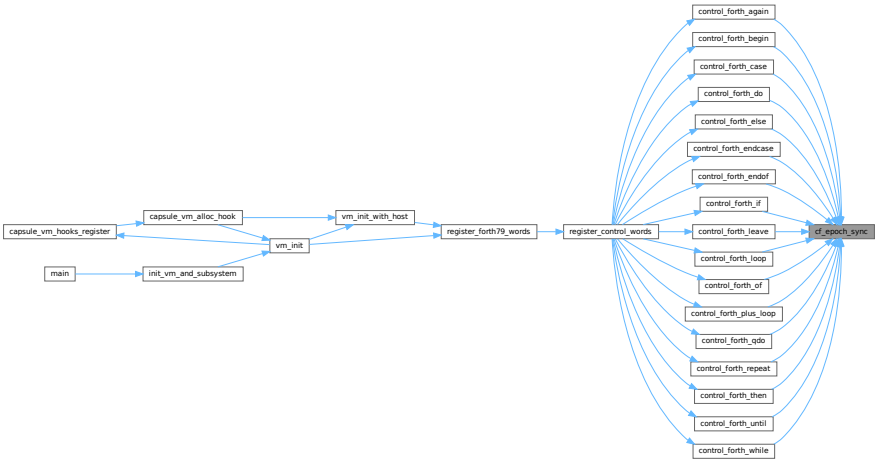
References VM::cf, ControlFlowState::cf_last_mode, ControlFlowState::cf_sp, LOG_DEBUG, log_message(), and VM::mode.

Referenced by control_forth_again(), control_forth_begin(), control_forth_case(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_if(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_then(), control_forth_until(), and control_forth_while().

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.2.2 cf_peek_item()

```
static int cf_peek_item (
            VM * vm,
            cf_item_t * out )  [inline], [static]
```

References VM::cf, ControlFlowState::cf_sp, and ControlFlowState::cf_stack.

Referenced by control_forth_else(), control_forth_loop(), control_forth_plus_loop(), and control_forth_while().

Here is the caller graph for this function:

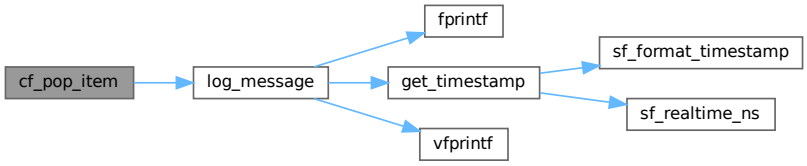
5.8.2.3 cf_pop_item()

```
static int cf_pop_item (
            VM * vm,
            cf_item_t * out )  [inline], [static]
```

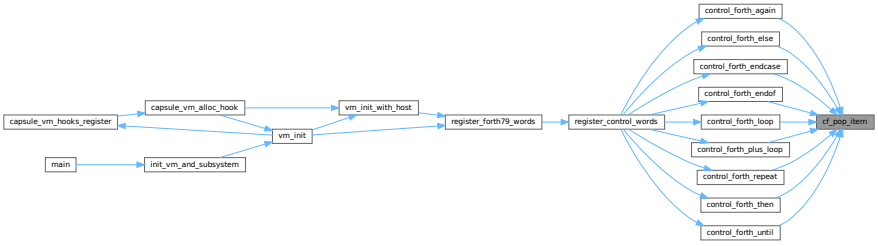
References VM::cf, ControlFlowState::cf_sp, ControlFlowState::cf_stack, LOG_ERROR, and log_message().

Referenced by control_forth_again(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_loop(), control_forth_plus_loop(), control_forth_repeat(), control_forth_then(), and control_forth_until().

Here is the call graph for this function:



Here is the caller graph for this function:



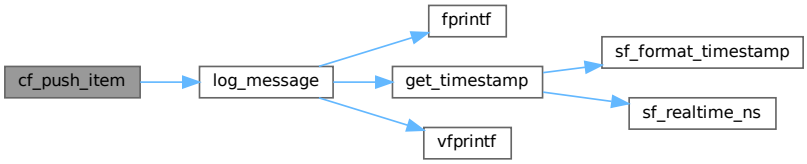
5.8.2.4 cf_push_item()

```
static int cf_push_item (
            VM * vm,
            cf_tag_t tag,
            size_t mark )  [inline], [static]
```

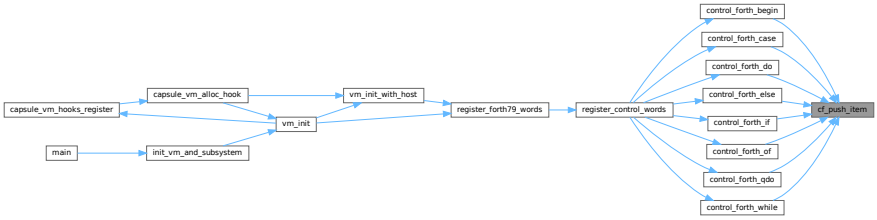
References cf_item_t::addr, VM::cf, ControlFlowState::cf_sp, ControlFlowState::cf_stack, CF_STACK_MAX, LOG_ERROR, log_message(), and cf_item_t::tag.

Referenced by control_forth_begin(), control_forth_case(), control_forth_do(), control_forth_else(), control_forth_if(), control_forth_of(), control_forth_qdo(), and control_forth_while().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9 Runtime Branch Helpers

Runtime support for control flow operations.

Functions

- static void control_forth_UNLOOP (VM *vm)
- static void control_forth_if (VM *vm)
- static void control_forth_else (VM *vm)
- static void control_forth_then (VM *vm)
- static void control_forth_begin (VM *vm)
- static void control_forth_until (VM *vm)
- static void control_forth_again (VM *vm)
- static void control_forth_while (VM *vm)
- static void control_forth_repeat (VM *vm)
- static void control_forth_qdo (VM *vm)
- static void control_forth_do (VM *vm)
- static void control_forth_leave (VM *vm)
- static void control_forth_loop (VM *vm)
- static void control_forth_plus_loop (VM *vm)
- static void control_forth_case (VM *vm)
- static void control_forth_of (VM *vm)
- static void control_forth_endof (VM *vm)
- static void control_forth_endcase (VM *vm)
- void register_control_words (VM *vm)

  Registers all control flow words with the Forth virtual machine.

Internal Runtime Functions

Forward declarations for internal runtime helpers and control flow primitives

- static void control_forth_branch (VM *vm)
  Execute unconditional branch.
- static void control_forth_0branch (VM *vm)
  Execute conditional branch based on top of stack.
- static void control_forth_runtime_qdo (VM *vm)
  Runtime handler for ?DO loops.
- static void control_forth_runtime_do (VM *vm)
  Runtime handler for DO loops.
- static void control_forth_runtime_loop (VM *vm)
  Runtime handler for LOOP.
- static void control_forth_runtime_plus_loop (VM *vm)
  Runtime handler for +LOOP.
- static void control_forth_runtime_leave (VM *vm)
  Runtime handler for LEAVE.
- static void control_forth_I (VM *vm)
  Get current loop index.
- static void control_forth_J (VM *vm)
  Get outer loop index.
- static void control_forth_EXIT (VM *vm)
  Exit from current definition.

5.9.1 Detailed Description

Runtime support for control flow operations.

All functions operate on the per-step return IP (top of RS). All offsets are in BYTES.

5.9.2 Function Documentation

5.9.2.1 control_forth_0branch()

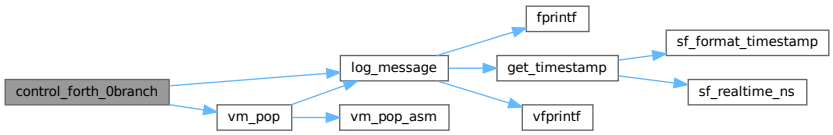
```
static void control_forth_0branch (
            VM * vm )  [static]
```

Execute conditional branch based on top of stack.

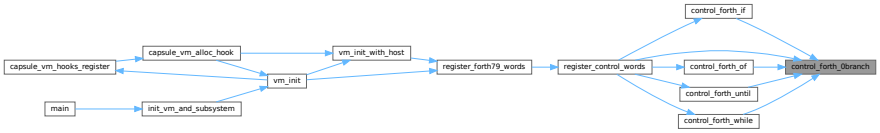
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, and vm_pop().

Referenced by control_forth_if(), control_forth_of(), control_forth_until(), control_forth_while(), and register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



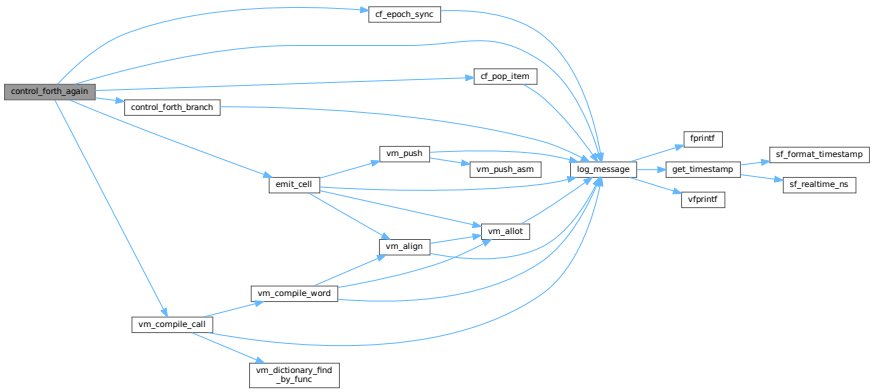
5.9.2.2 control_forth_again()

```
static void control_forth_again (
            VM * vm )  [static]
```

References cf_item_t::addr, CF_BEGIN, cf_epoch_sync(), cf_pop_item(), control_forth_branch(), emit_cell(), VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

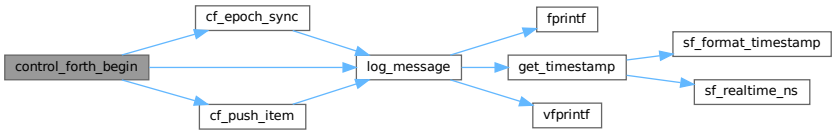
5.9.2.3 control_forth_begin()

```
static void control_forth_begin (
            VM * vm )  [static]
```

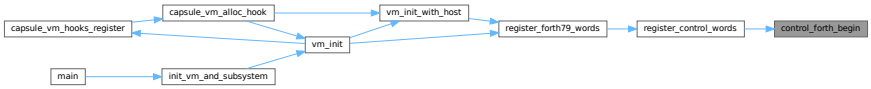
References CF_BEGIN, cf_epoch_sync(), cf_push_item(), VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, and MODE_COMPILE.

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.4 control_forth_branch()

```
static void control_forth_branch (
            VM * vm )  [static]
```

Execute unconditional branch.

Runtime unconditional branch.

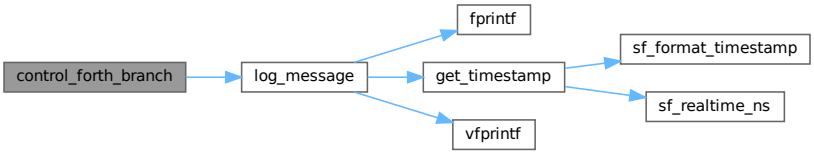
Parameters

vm	Virtual machine state

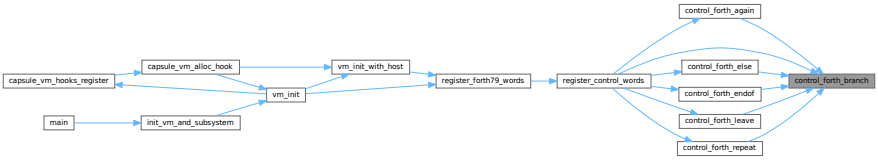
References VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::return_stack, and VM::rsp.

Referenced by control_forth_again(), control_forth_else(), control_forth_endof(), control_forth_leave(), control_forth_repeat(), and register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



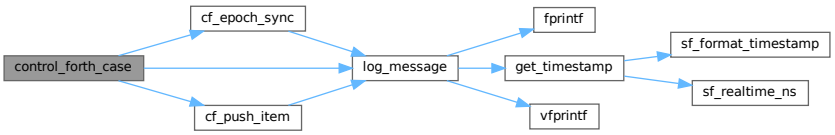
5.9.2.5 control_forth_case()

```
static void control_forth_case (
            VM * vm )  [static]
```

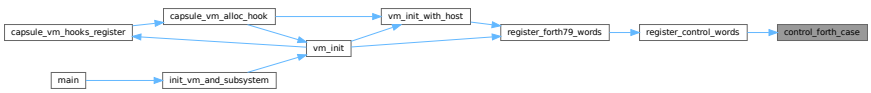
References VM::cf, CF_CASE, cf_epoch_sync(), cf_push_item(), CF_STACK_MAX, ControlFlowState::endof_mark_sp, ControlFlowState::endof_mark_stack, ControlFlowState::endof_sp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, and MODE_COMPILE.

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



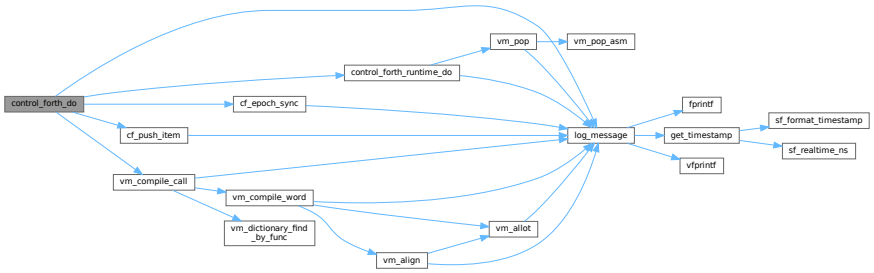
5.9.2.6 control_forth_do()

```
static void control_forth_do (
            VM * vm )  [static]
```

References VM::cf, CF_DO, cf_epoch_sync(), cf_push_item(), control_forth_runtime_do(), VM::error, VM::here, ControlFlowState::leave_mark_sp, ControlFlowState::leave_mark_stack, ControlFlowState::leave_sp, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

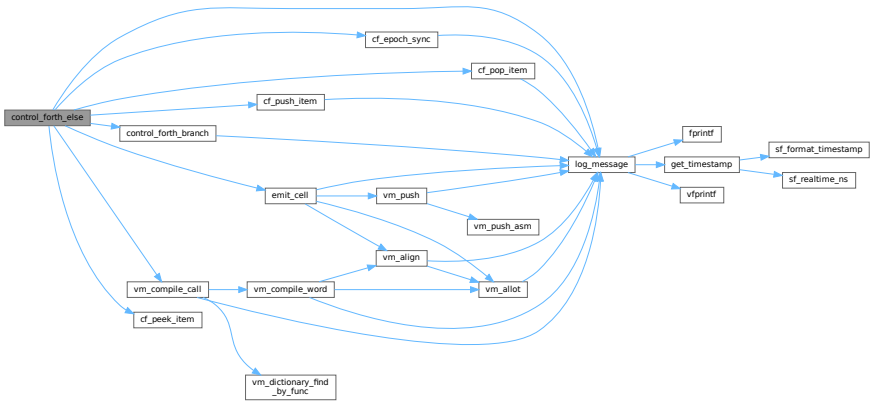
5.9.2.7 control_forth_else()

```
static void control_forth_else (
            VM * vm )  [static]
```

References cf_item_t::addr, CF_ELSE, cf_epoch_sync(), CF_IF, cf_peek_item(), cf_pop_item(), cf_push_item(), control_forth_branch(), emit_cell(), VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), VM::memory, VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

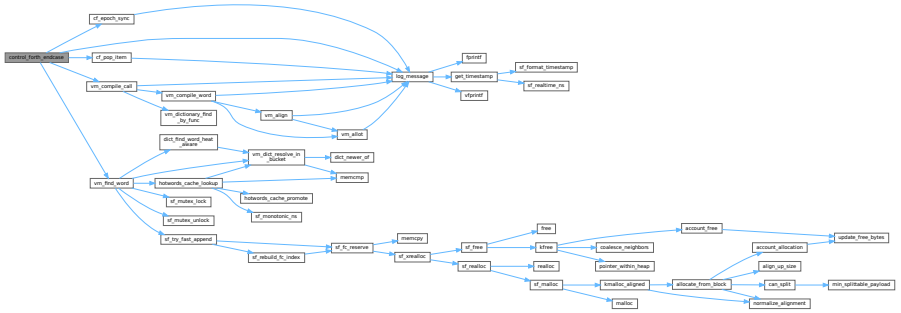
5.9.2.8 control_forth_endcase()

```
static void control_forth_endcase (
            VM * vm )  [static]
```

References VM::cf, CF_CASE, cf_epoch_sync(), cf_pop_item(), ControlFlowState::endof_addrs, ControlFlowState::endof_mark_sp, ControlFlowState::endof_mark_stack, ControlFlowState::endof_sp, VM::error, DictEntry::func, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), VM::memory, VM::mode, MODE_COMPILE, cf_item_t::tag, vm_compile_call(), and vm_find_word().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

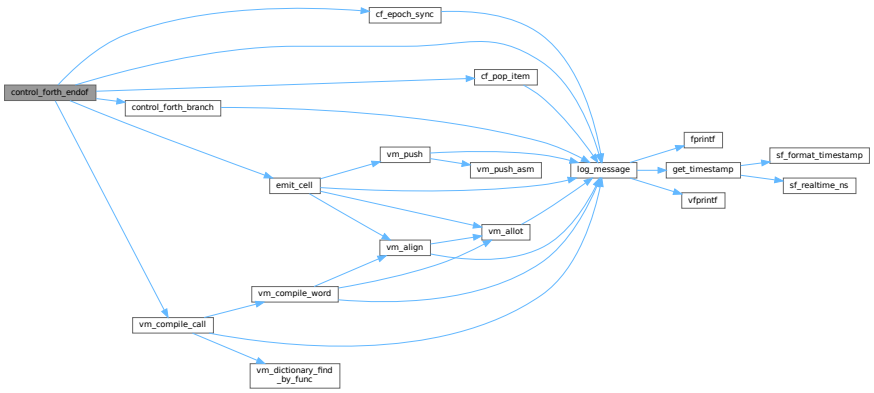
5.9.2.9 control_forth_endof()

```
static void control_forth_endof (
            VM * vm )  [static]
```

References cf_item_t::addr, VM::cf, cf_epoch_sync(), CF_OF, cf_pop_item(), CF_STACK_MAX, control_forth_branch(), emit_cell(), ControlFlowState::endof_addrs, ControlFlowState::endof_sp, VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), VM::memory, VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

5.9.2.10 control_forth_EXIT()

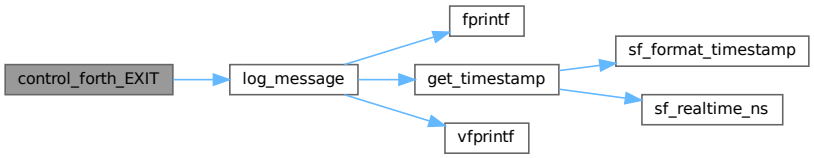
```
static void control_forth_EXIT (
            VM * vm )  [static]
```

Exit from current definition.

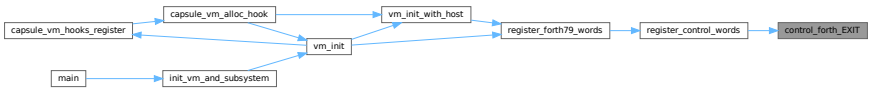
References VM::error, VM::exit_colon, LOG_DEBUG, LOG_ERROR, log_message(), and VM::rsp.

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.11 control_forth_I()

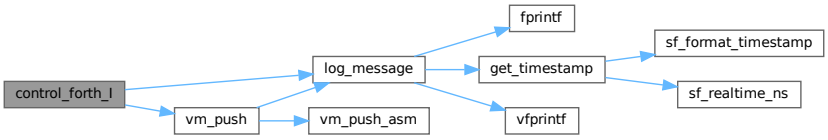
```
static void control_forth_I (
            VM * vm )  [static]
```

Get current loop index.

References VM::error, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, and vm_push().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

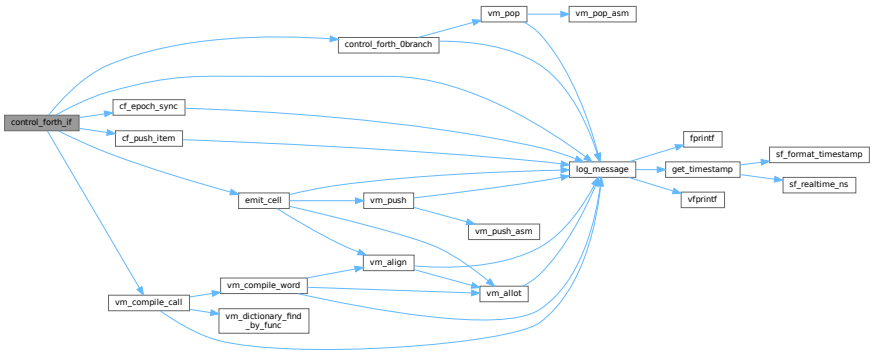
5.9.2.12 control_forth_if()

```
static void control_forth_if (
            VM * vm )  [static]
```

References cf_epoch_sync(), CF_IF, cf_push_item(), control_forth_0branch(), emit_cell(), VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

5.9.2.13 control_forth_J()

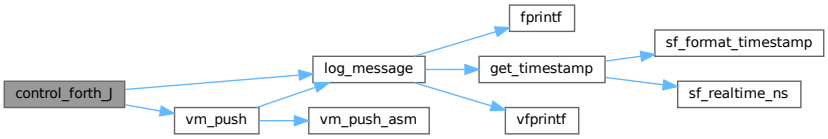
```
static void control_forth_J (
            VM * vm )  [static]
```

Get outer loop index.

References VM::error, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, and vm_push().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

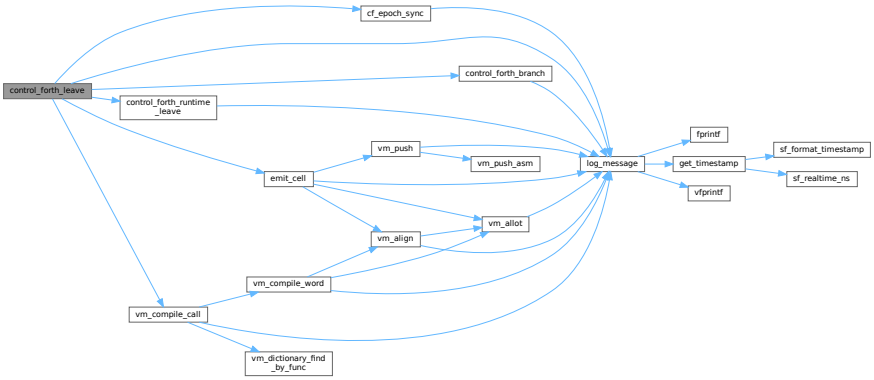
5.9.2.14 control_forth_leave()

```
static void control_forth_leave (
            VM * vm )  [static]
```

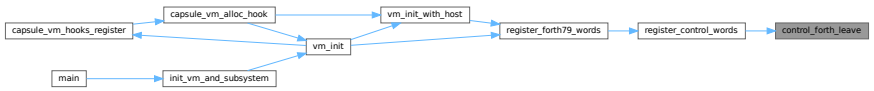
References VM::cf, CF_DO, cf_epoch_sync(), ControlFlowState::cf_sp, ControlFlowState::cf_stack, CF_STACK_MAX, control_forth_branch(), control_forth_runtime_leave(), emit_cell(), VM::error, ControlFlowState::leave_addrs, ControlFlowState::leave_mark_sp, ControlFlowState::leave_sp, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



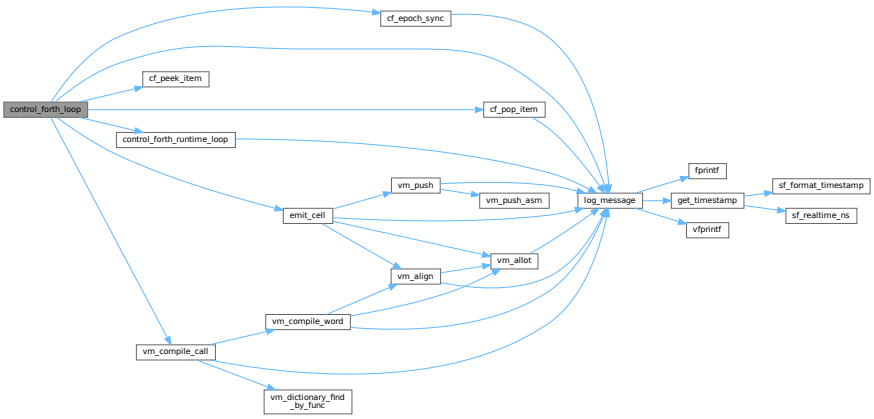
5.9.2.15 control_forth_loop()

```
static void control_forth_loop (
            VM * vm )  [static]
```

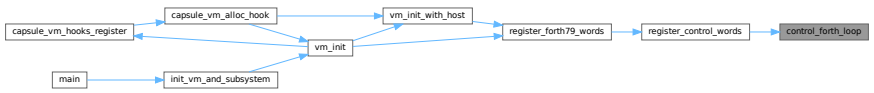
References cf_item_t::addr, VM::cf, CF_DO, cf_epoch_sync(), cf_peek_item(), cf_pop_item(), CF_WHILE, control_forth_runtime_loop(), emit_cell(), VM::error, VM::here, ControlFlowState::leave_addrs, ControlFlowState::leave_mark_sp, ControlFlowState::leave_mark_stack, ControlFlowState::leave_sp, LOG_DEBUG, LOG_ERROR, log_message(), VM::memory, VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



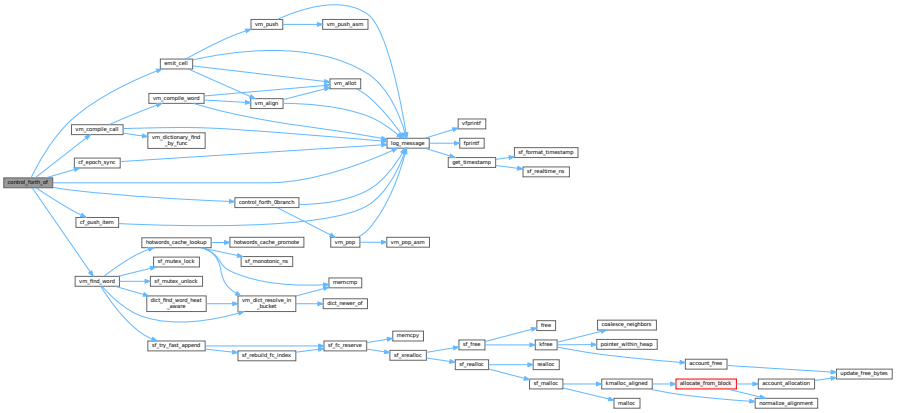
5.9.2.16 control_forth_of()

```
static void control_forth_of (
            VM * vm )  [static]
```

References VM::cf, CF_CASE, cf_epoch_sync(), CF_OF, cf_push_item(), ControlFlowState::cf_sp, ControlFlowState::cf_stack, control_forth_0branch(), emit_cell(), VM::error, DictEntry::func, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, cf_item_t::tag, vm_compile_call(), and vm_find_word().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

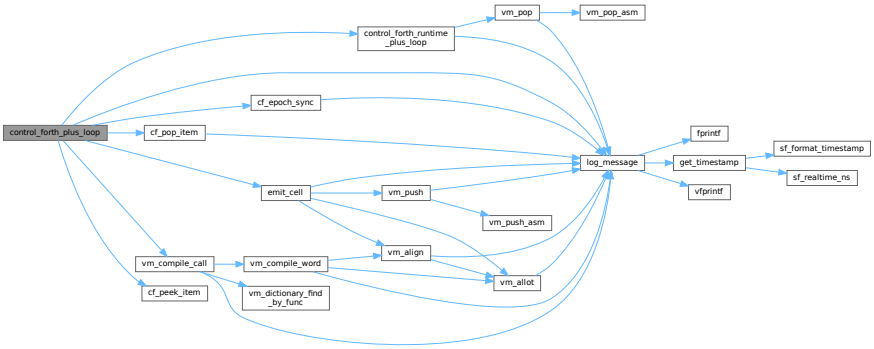
5.9.2.17 control_forth_plus_loop()

```
static void control_forth_plus_loop (
            VM * vm )  [static]
```

References cf_item_t::addr, VM::cf, CF_DO, cf_epoch_sync(), cf_peek_item(), cf_pop_item(), CF_WHILE, control_forth_runtime_plus_loop(), emit_cell(), VM::error, VM::here, ControlFlowState::leave_addrs, ControlFlowState::leave_mark_sp, ControlFlowState::leave_mark_stack, ControlFlowState::leave_sp, LOG_DEBUG, LOG_ERROR, log_message(), VM::memory, VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

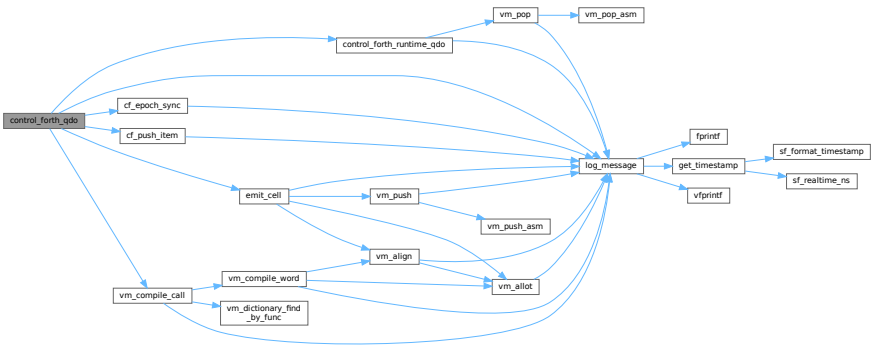
5.9.2.18 control_forth_qdo()

```
static void control_forth_qdo (
            VM * vm )  [static]
```

References VM::cf, CF_DO, cf_epoch_sync(), cf_push_item(), CF_WHILE, control_forth_runtime_qdo(), emit_cell(), VM::error, VM::here, ControlFlowState::leave_mark_sp, ControlFlowState::leave_mark_stack, ControlFlowState::leave_sp, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

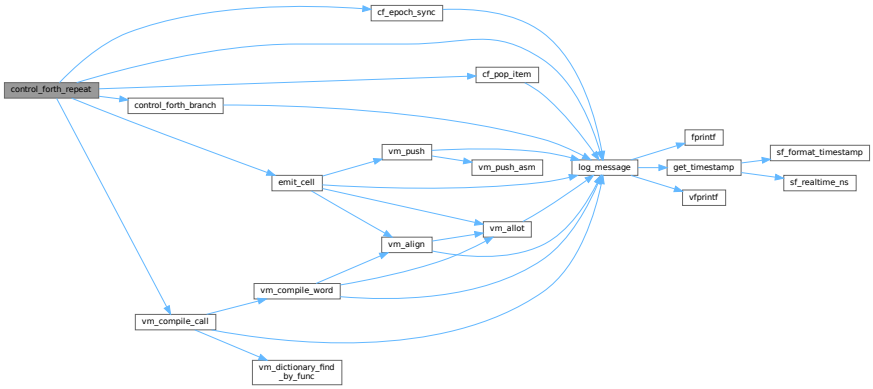
5.9.2.19 control_forth_repeat()

```
static void control_forth_repeat (
            VM * vm )  [static]
```

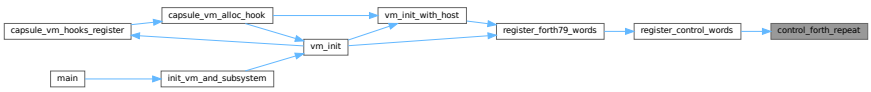
References cf_item_t::addr, CF_BEGIN, cf_epoch_sync(), cf_pop_item(), CF_WHILE, control_forth_branch(), emit_cell(), VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), VM::memory, VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.20 control_forth_runtime_do()

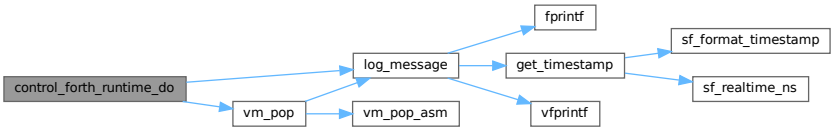
```
static void control_forth_runtime_do (
            VM * vm )  [static]
```

Runtime handler for DO loops.

References VM::dsp, VM::error, limit, LOG_DEBUG, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, STACK_SIZE, and vm_pop().

Referenced by control_forth_do(), and register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

5.9.2.21 control_forth_runtime_leave()

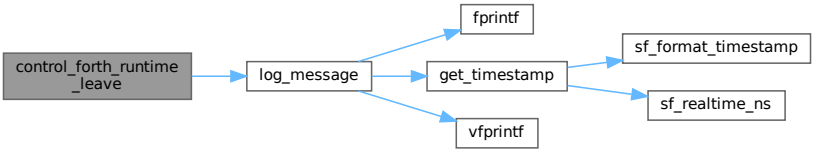
```
static void control_forth_runtime_leave (
            VM * vm )  [static]
```

Runtime handler for LEAVE.

References VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::return_stack, and VM::rsp.

Referenced by control_forth_leave(), and register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

5.9.2.22 control_forth_runtime_loop()

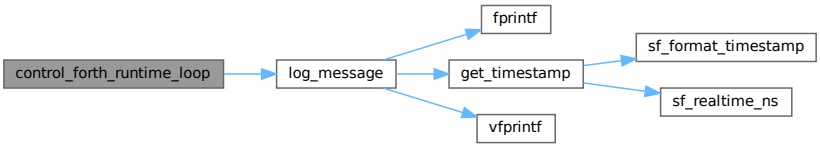
```
static void control_forth_runtime_loop (
            VM * vm )  [static]
```

Runtime handler for LOOP.

References VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::return_stack, and VM::rsp.

Referenced by control_forth_loop(), and register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

5.9.2.23 control_forth_runtime_plus_loop()

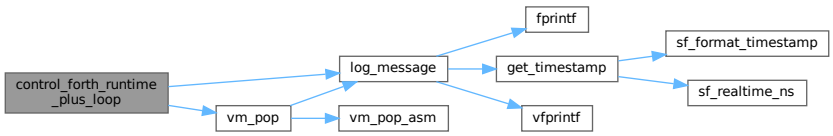
```
static void control_forth_runtime_plus_loop (
            VM * vm )  [static]
```

Runtime handler for +LOOP.

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, and vm_pop().

Referenced by control_forth_plus_loop(), and register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

5.9.2.24 control_forth_runtime_qdo()

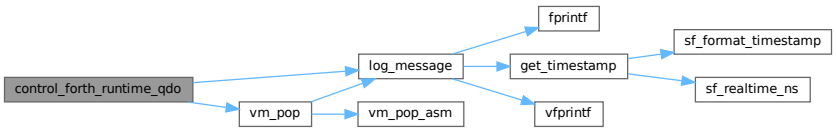
```
static void control_forth_runtime_qdo (
            VM * vm )  [static]
```

Runtime handler for ?DO loops.

References VM::dsp, VM::error, limit, LOG_DEBUG, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, STACK_SIZE, and vm_pop().

Referenced by control_forth_qdo(), and register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

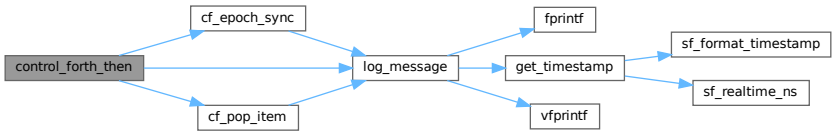
5.9.2.25 control_forth_then()

```
static void control_forth_then (
            VM * vm )  [static]
```

References cf_item_t::addr, CF_ELSE, cf_epoch_sync(), CF_IF, cf_pop_item(), VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), VM::memory, VM::mode, MODE_COMPILE, and cf_item_t::tag.

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

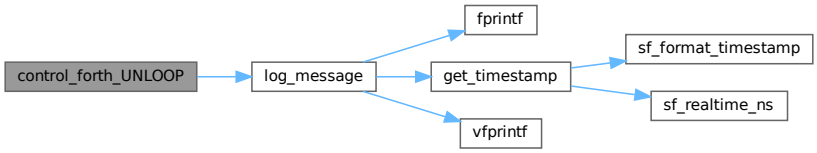
5.9.2.26 control_forth_UNLOOP()

```
static void control_forth_UNLOOP (
            VM * vm )  [static]
```

References VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::return_stack, and VM::rsp.

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:

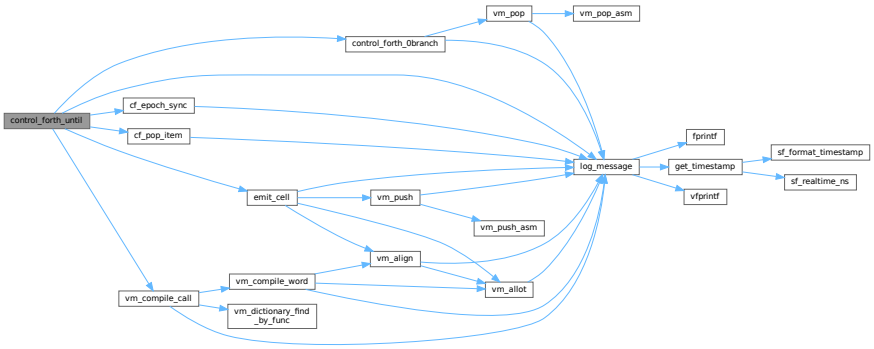
5.9.2.27 control_forth_until()

```
static void control_forth_until (
            VM * vm )  [static]
```

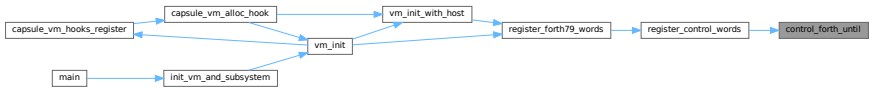
References cf_item_t::addr, CF_BEGIN, cf_epoch_sync(), cf_pop_item(), control_forth_0branch(), emit_cell(), VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



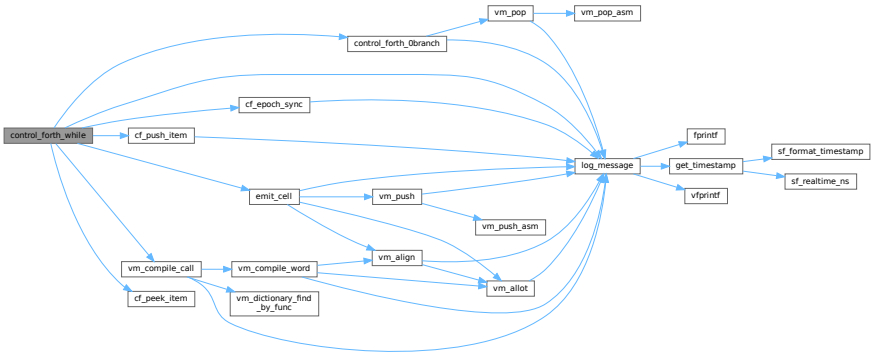
5.9.2.28 control_forth_while()

```
static void control_forth_while (
            VM * vm )  [static]
```

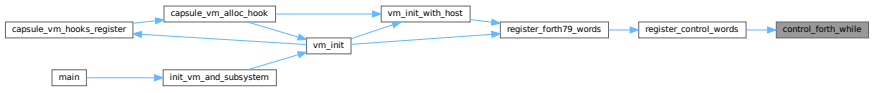
References CF_BEGIN, cf_epoch_sync(), cf_peek_item(), cf_push_item(), CF_WHILE, control_forth_0branch(), emit_cell(), VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, cf_item_t::tag, and vm_compile_call().

Referenced by register_control_words().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.2.29 register_control_words()

```
void register_control_words (
            VM * vm )
```

Registers all control flow words with the Forth virtual machine.

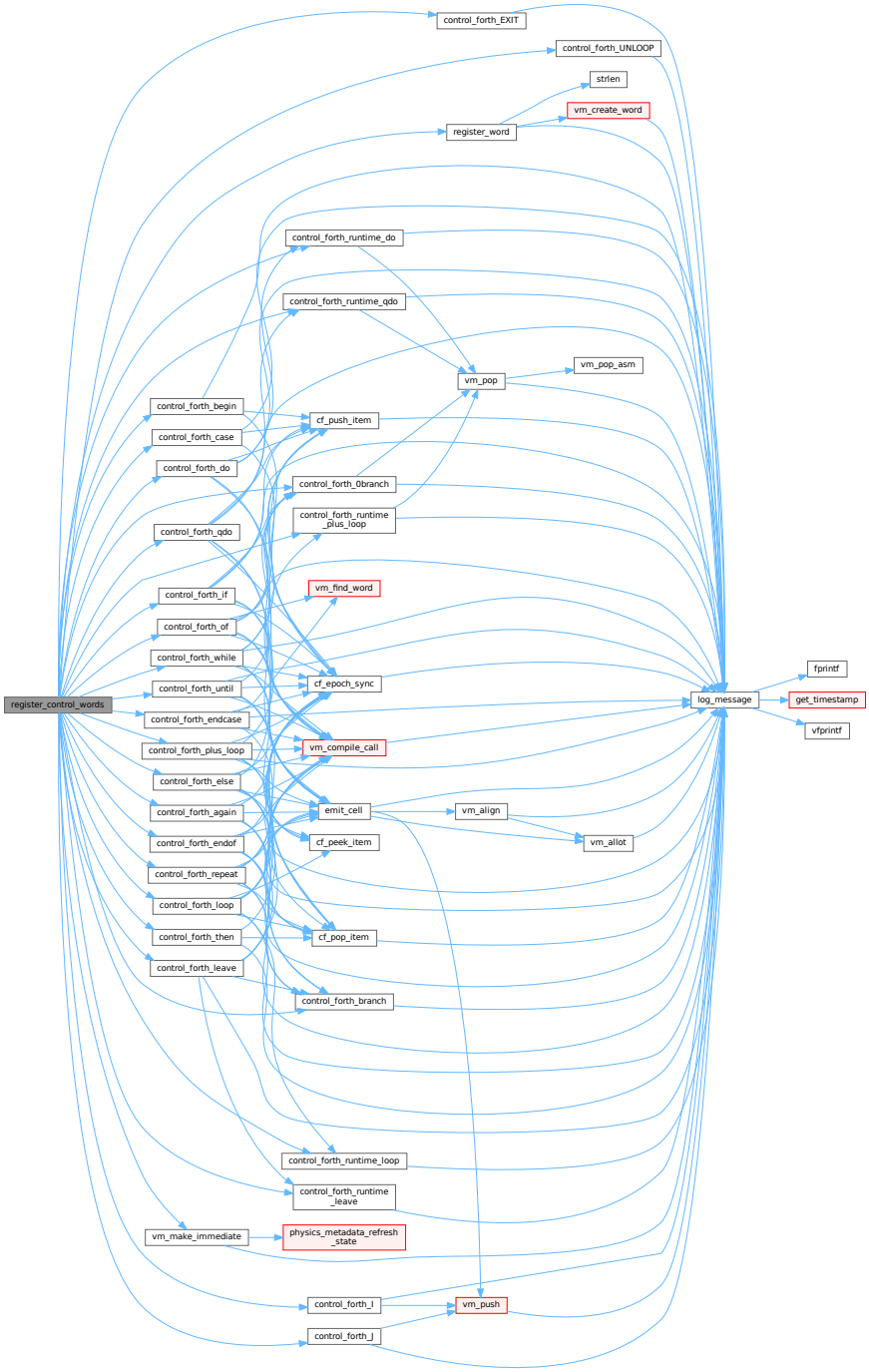
Parameters

vm	Pointer to the Forth virtual machine instance

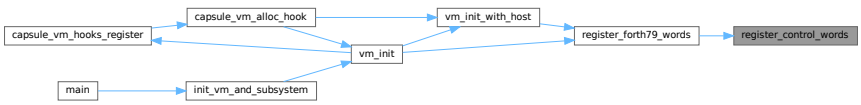
References control_forth_0branch(), control_forth_again(), control_forth_begin(), control_forth_branch(), control_forth_case(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_EXIT(), control_forth_I(), control_forth_if(), control_forth_J(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_runtime_do(), control_forth_runtime_leave(), control_forth_runtime_loop(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), control_forth_then(), control_forth_UNLOOP(), control_forth_until(), control_forth_while(), register_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



5.10 FORTH-79 Control Flow Words

Standard FORTH-79 control flow words implementation.

Standard FORTH-79 control flow words implementation.

5.11 FORTH-79 Defining Words

Standard FORTH-79 defining words.

Standard FORTH-79 defining words.

These words are used for creating new dictionary entries and defining words:

- : (–) Start colon definition
- ; (–) End colon definition
- CREATE (–) Create dictionary entry
- DOES> (–) Define runtime behavior
- <BUILDS (--) Create word with <BUILDS/DOES>
- CONSTANT (n –) Define constant
- VARIABLE (–) Define variable
- 2CONSTANT (d –) Define double constant
- 2VARIABLE (–) Define double variable
- USER (n –) Define user variable
- IMMEDIATE (–) Make most recent word immediate
- [COMPILE] (–) Force compilation of immediate word
- COMPILE (–) Compile inline code
- FORGET (–) Remove word and all after it

5.12 FORTH-79 Dictionary Manipulation Words

Standard FORTH-79 words for dictionary manipulation.

Standard FORTH-79 words for dictionary manipulation.

[(–)

Enter interpretation mode. Switches the system to interpret incoming words.

] (–)

Enter compilation mode. Switches the system to compile incoming words.

STATE (– addr)

Variable containing current compilation state. Returns address of STATE variable.

SMUDGE (–)

Toggle smudge bit of latest defined word. Used during word definition.

>BODY (xt – addr)

Convert execution token to parameter field address.

>NAME (xt – addr)

Convert execution token to name field address.

NAME> (addr – xt)

Convert name field address to execution token.

>LINK (addr – addr)

Get link field address from name field address.

LINK> (addr – addr)

Get next word's address from link field.

CFA (addr – xt)

Get code field address from name field address.

LFA (addr – addr)

Get link field address from name field address.

NFA (addr – addr)

Get name field address from any field address.

PFA (addr – addr)

Get parameter field address from name field address.

TRAVERSE (addr n – addr)

Move through name field by n characters.

INTERPRET (–)

Text interpreter - process input stream.

5.13 FORTH-79 Dictionary & Compilation Words

Dictionary and compilation words as defined in FORTH-79 Standard.

Dictionary and compilation words as defined in FORTH-79 Standard.

@word HERE (– addr) Returns address of next free dictionary space @word ALLOT (n –) Allocates n bytes in dictionary @word , (n –) Compiles n into dictionary @word C, (c –) Compiles character into dictionary @word 2, (d –) Compiles double into dictionary @word PAD (– addr) Returns address of temporary text buffer @word SP! (addr –) Sets data stack pointer @word SP@ (– addr) Gets data stack pointer @word LATEST (– addr) Returns address of most recent definition

5.14 FORTH-79 Line Editor Words

Line editor commands implementation.

Line editor commands implementation.

5.15 Framebuffer Words

Raw hardware-boundary FORTH words for the Console drawing fabric (FABRIC-0.md item 4.3.3).

Raw hardware-boundary FORTH words for the Console drawing fabric (FABRIC-0.md item 4.3.3).

Deliberately raster-native – no Cartesian awareness, no policy. That lives in capsules/fabric.4th instead, per the compose-in-FORTH-first rule.

Kernel-only: no-op on hosted builds (no framebuffer exists there).

PLOT (x y color –)

Write one pixel at raster coordinates (top-left origin, Y-down). Out-of-bounds writes are silently ignored (inherited from fb_put_pixel).

FB-WIDTH (– n)

FB-HEIGHT (– n)

Push the framebuffer's pixel width / height.

5.16 Keyboard Words

Raw hardware-boundary FORTH word for verifying the interrupt-driven keyboard path (FABRIC-0.md item 4.3.5).

Raw hardware-boundary FORTH word for verifying the interrupt-driven keyboard path (FABRIC-0.md item 4.3.5).

Deliberately a diagnostic peek at the raw scancode ring buffer – no set-2 translation, no REPL wiring. Those belong to the later REPL keyboard-input work noted in FABRIC-0.md.

Kernel-only, amd64-only today (i8042 is amd64 hardware); no-op elsewhere so dictionary parity across all three architectures is unaffected.

KBD-SCAN (– c -1 | 0)

Pop one raw scancode off the interrupt-fed ring buffer. Pushes the scancode and -1 (true) if one was available, or just 0 (false) if the buffer was empty.

KBD-DEBUG (– isr_count spurious_count)

Standing diagnostic: count of real keyboard IRQs serviced and count of LAPIC spurious-vector interrupts, both since boot.

VKBD-EVENT (– code value -1 | 0)

Items 4.3.5c (riscv64) / 4.3.5e (aarch64). Pop one decoded EV_KEY event (Linux input-event code/value pair) off the virtio-input interrupt-fed ring buffer. Kernel-only, riscv64/aarch64 (virtio-keyboard-pci); no-op on amd64 (i8042 hardware instead, see KBD-SCAN).

VKBD-DEBUG (– isr_count)

Standing diagnostic: count of virtio-input ISR invocations since boot.

KEY-EVENT (– keycode pressed -1 | 0)

Item 4.3.5f convergence checkpoint. One shared event shape across all three architectures: keycode in the Linux input keycode namespace (identical numbering to amd64's raw XT Set-1 scancode for the non-extended key range, by historical design, not translated arbitrarily – see the implementation's own doc comment), pressed is 1 or 0. This is the word the later REPL keyboard-input work (M8) is expected to build on; KBD-SCAN/VKBD-EVENT remain as lower-level per-device diagnostics.

ALT+TAB (–)

FABRIC-0.md item 4.4y-revised. Programmatic equivalent of the physical Alt+TAB graphics/text toggle – calls the same state-machine transition (console_fb_toggle_graphics() / vt100_toggle_graphics()) the keyboard interception in repl.c uses, so there is exactly one place the toggle actually happens.

5.17 Log Words

FORTH words for log-level control and structured string emission.

FORTH words for log-level control and structured string emission.

Level constants (– n)

Push the corresponding LogLevel integer onto the stack. LOG-ERROR (– 0) LOG-WARN (– 1) LOG-INFO (– 2) LOG-TEST (– 3) LOG-DEBUG (– 4)

LOG-LEVEL! (n –)

Set the current log level to n. Values outside 0–4 clamp to LOG-DEBUG.

LOG-LEVEL@ (– n)

Push the current log level.

LOG-INFO" ("ccc" --) IMMEDIATE @par LOG-WARN" ("ccc<quote>" –) IMMEDIATE

LOG-ERROR" ("ccc" --) IMMEDIATE @par LOG-TEST" ("ccc<quote>" –) IMMEDIATE

LOG-DEBUG" ("ccc" –) IMMEDIATE Parse the string literal up to the closing double-quote and emit it via log_message() at the corresponding level. W

DEBUG messages are free when the level is INFO.

LOG-INFO-STR (c-addr u –)

LOG-WARN-STR (c-addr u –)

LOG-ERROR-STR (c-addr u –)

LOG-TEST-STR (c-addr u –)

LOG-DEBUG-STR (c-addr u –)

Consume a counted string from the stack (as produced by S") and emit it via log_message() at the corresponding level. @par Runtime words (not for direct use) (do-log-error) (do-log-warn) (do-log-info) (do-log-test) (do-log-debug) Threaded-code runtime helpers compiled by the LOG-∗" immediates.

5.18 FORTH-79 Logical Operations

FORTH-79 Standard logical and comparison operations.

FORTH-79 Standard logical and comparison operations.

Implementation of the following FORTH-79 words:

- AND (n1 n2 – n3) Bitwise AND
- OR (n1 n2 – n3) Bitwise OR
- XOR (n1 n2 – n3) Bitwise XOR
- NOT (n1 – n2) Bitwise NOT
- 0= (n – flag) True if n is zero
- 0< (n – flag) True if n is negative
- 0> (n – flag) True if n is positive
- = (n1 n2 – flag) True if n1 equals n2
- < (n1 n2 – flag) True if n1 < n2
- > (n1 n2 – flag) True if n1 > n2
- U< (u1 u2 – flag) True if u1 < u2 (unsigned)
- U> (u1 u2 – flag) True if u1 > u2 (unsigned)
- WITHIN (n low high – flag) True if low <= n < high

5.19 FORTH-79 Memory Operations

Memory operation words defined in FORTH-79 standard.

Memory operation words defined in FORTH-79 standard.

Available operations:

- ! (n addr –) Store n at addr
- @ (addr – n) Fetch n from addr
- C! (c addr –) Store character at addr
- C@ (addr – c) Fetch character from addr
- +! (n addr –) Add n to value at addr
- 2! (d addr –) Store double at addr
- 2@ (addr – d) Fetch double from addr
- FILL (addr u c –) Fill u bytes at addr with c
- MOVE (addr1 addr2 u –) Move u bytes from addr1 to addr2
- CMOVE (addr1 addr2 u –) Move u bytes, low to high addresses
- CMOVE> (addr1 addr2 u –) Move u bytes, high to low addresses
- ERASE (addr u –) Fill u bytes at addr with zeros

5.20 FORTH-79 Return Stack Operations

5.21 REPL Scrollback Words

`SCROLL-BACK/SCROLL-FWD`, the REPL scrollback controls (FABRIC-0.md item 4.4q).

`SCROLL-BACK/SCROLL-FWD`, the REPL scrollback controls (FABRIC-0.md item 4.4q).

Kernel-only; no-op on hosted builds (no framebuffer console there).

SCROLL-BACK (n –)

Move the scrollback view back `n` lines (toward older output) and redraw. Clamped at the oldest stored line – no error on over-request, it just stops there.

SCROLL-FWD (n –)

Move the scrollback view forward `n` lines (toward live output) and redraw. Clamped at the live view.

5.22 FORTH-79 System & Environment Words

System and environment control words for FORTH-79.

System and environment control words for FORTH-79.

The following words provide system and environment control functionality:

COLD (–)

Performs a cold start of the system, reinitializing all system variables

WARM (–)

Performs a warm start of the system, preserving some system state

BYE (–)

Exits the FORTH system and returns to the host operating system

SAVE-SYSTEM (–)

Saves the current system image to disk

WORDS (–)

Lists all words in the current vocabulary

VLIST (–)

Lists all words in all vocabularies

?STACK (–)

Checks the stack depth and reports any stack errors

PAGE (–)

Clears the screen or outputs a form feed

NOP (–)

Performs no operation

79-STANDARD (–)

Ensures FORTH-79 standard compliance

5.23 TrueType Text Words

`TTF-TEXT`, the TrueType-backed text entry point (FABRIC-0.md item 4.3.7e).

`TTF-TEXT`, the TrueType-backed text entry point (FABRIC-0.md item 4.3.7e).

Kernel-only; no-op on hosted builds (no framebuffer/capsule system exists there).

TTF-TEXT (c-addr u x y size color –)

Render a UTF-8 string via the v1 default font capsule (`fonts:JetBrainsMono-Regular.ttf`, loaded and cached lazily on first use). (x,y) is raster pixel space (top-left origin, Y-down) – NOT capsules/fabric.4th's Cartesian Y-up convention used by the stroke font's `TEXT` (4.3.6f); the two are deliberately different coordinate systems, not interchangeable. (x,y) is the first glyph's baseline-left origin. `size` is capped at `TTF_CACHE_MAX_SIZE_PX` (see ttf.h). Advances proportionally per glyph via the font's real hmtx advance width, scaled to pixels.

5.24 FORTH-79 Vocabulary System Words

Standard vocabulary manipulation words.

Standard vocabulary manipulation words.

VOCABULARY (–)

Create new vocabulary

DEFINITIONS (–)

Set current vocabulary

CONTEXT (– addr)

Variable: search vocabulary pointer

CURRENT (– addr)

Variable: definition vocabulary pointer

FORTH (–)

Select FORTH vocabulary

ONLY (–)

Set minimal search order

ALSO (–)

Duplicate top of search order

ORDER (–)

Display search order

PREVIOUS (–)

Remove top of search order

(FIND) (addr – addr flag)

Find word in dictionary (primitive)

Chapter 6

Data Structure Documentation

6.1 __attribute__ Struct Reference

ioapic.c - I/O APIC driver (amd64)

Data Fields

- uint8_t signature [8]
- uint8_t checksum
- uint8_t oem_id [6]
- uint8_t revision
- uint32_t rsdt_address
- uint32_t length
- uint64_t xsdt_address
- uint8_t extended_checksum
- uint8_t reserved [3]
- uint8_t oem_table_id [8]
- uint32_t oem_revision
- uint32_t creator_id
- uint32_t creator_revision
- AcpiHeader hdr
- uint32_t local_apic_address
- uint32_t flags
- uint8_t type
- uint8_t length
- uint8_t io_apic_id
- uint32_t io_apic_address
- uint32_t gsi_base
- uint8_t bus
- uint8_t source
- uint32_t gsi
- uint16_t flags
- uint8_t address_space_id
- uint8_t register_bit_width
- uint8_t register_bit_offset
- uint8_t access_size
- uint64_t address
- uint64_t base_address
- uint16_t segment_group
- uint8_t start_bus
- uint8_t end_bus
- uint32_t reserved
- uint64_t reserved

6.1.1 Detailed Description

ioapic.c - I/O APIC driver (amd64)

Item 4.3.5 (FABRIC-0.md ?27.5). MADT parsing mirrors pci.c's RSDP -> XSDT -> table-by-signature walk (the two files don't share a header for this — same duplication pci.c already has relative to a hypothetical shared acpi.c, not introduced fresh here).

6.1.2 Field Documentation

6.1.2.1 access_size

```
uint8_t __attribute__::access_size
```

6.1.2.2 address

```
uint64_t __attribute__::address
```

6.1.2.3 address_space_id

```
uint8_t __attribute__::address_space_id
```

6.1.2.4 base_address

```
uint64_t __attribute__::base_address
```

6.1.2.5 bus

```
uint8_t __attribute__::bus
```

6.1.2.6 checksum

```
uint8_t __attribute__::checksum
```

6.1.2.7 creator_id

```
uint32_t __attribute__::creator_id
```

6.1.2.8 creator_revision

```
uint32_t __attribute__::creator_revision
```

6.1.2.9 end_bus

```
uint8_t __attribute__::end_bus
```

6.1.2.10 extended_checksum

```
uint8_t __attribute__::extended_checksum
```

6.1.2.11 flags [1/2]

```
uint32_t __attribute__::flags
```

6.1.2.12 flags [2/2]

```
uint16_t __attribute__::flags
```

6.1.2.13 gsi

```
uint32_t __attribute__::gsi
```

6.1.2.14 gsi_base

```
uint32_t __attribute__::gsi_base
```

6.1.2.15 hdr

```
AcpiHeader __attribute__::hdr
```

6.1.2.16 io_apic_address

```
uint32_t __attribute__::io_apic_address
```

6.1.2.17 io_apic_id

```
uint8_t __attribute__::io_apic_id
```

6.1.2.18 length [1/2]

```
uint32_t __attribute__::length
```

6.1.2.19 length [2/2]

```
uint8_t __attribute__::length
```

6.1.2.20 local_apic_address

```
uint32_t __attribute__::local_apic_address
```

6.1.2.21 oem_id

```
uint8_t __attribute__::oem_id
```

6.1.2.22 oem_revision

```
uint32_t __attribute__::oem_revision
```

6.1.2.23 oem_table_id

```
uint8_t __attribute__::oem_table_id
```

6.1.2.24 register_bit_offset

```
uint8_t __attribute__::register_bit_offset
```

6.1.2.25 register_bit_width

```
uint8_t __attribute__::register_bit_width
```

6.1.2.26 reserved [1/3]

```
uint8_t __attribute__::reserved
```

6.1.2.27 reserved [2/3]

```
uint32_t __attribute__::reserved
```

6.1.2.28 reserved [3/3]

```
uint64_t __attribute__::reserved
```

6.1.2.29 revision

```
uint8_t __attribute__::revision
```

6.1.2.30 rsdt_address

```
uint32_t __attribute__::rsdt_address
```

6.1.2.31 segment_group

```
uint16_t __attribute__::segment_group
```

6.1.2.32 signature

```
uint8_t __attribute__::signature
```

6.1.2.33 source

```
uint8_t __attribute__::source
```

6.1.2.34 start_bus

```
uint8_t __attribute__::start_bus
```

6.1.2.35 type

```
uint8_t __attribute__::type
```

6.1.2.36 xsdt_address

```
uint64_t __attribute__::xsdt_address
```

Referenced by fadt_find_pm_tmr_port(), and parse_madt().

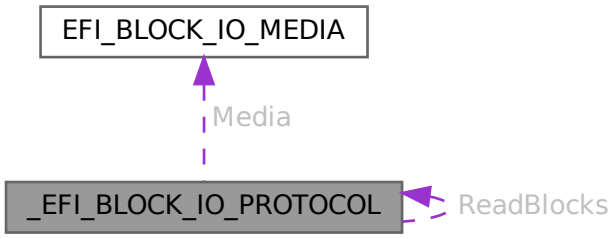
The documentation for this struct was generated from the following files:

- kernel/src/arch/amd64/ioapic.c
- kernel/src/arch/amd64/timer.c
- kernel/src/pci/pci.c

6.2 _EFI_BLOCK_IO_PROTOCOL Struct Reference

`#include <uefi.h>`

Collaboration diagram for _EFI_BLOCK_IO_PROTOCOL:



Data Fields

- UINT64 Revision
- EFI_BLOCK_IO_MEDIA * Media
- void * Reset
- EFI_BLOCK_READ ReadBlocks
- void * WriteBlocks
- void * FlushBlocks

6.2.1 Field Documentation

6.2.1.1 FlushBlocks

`void* _EFI_BLOCK_IO_PROTOCOL::FlushBlocks`

6.2.1.2 Media

`EFI_BLOCK_IO_MEDIA* _EFI_BLOCK_IO_PROTOCOL::Media`

6.2.1.3 ReadBlocks

`EFI_BLOCK_READ _EFI_BLOCK_IO_PROTOCOL::ReadBlocks`

6.2.1.4 Reset

`void* _EFI_BLOCK_IO_PROTOCOL::Reset`

6.2.1.5 Revision

```
UINT64 _EFI_BLOCK_IO_PROTOCOL::Revision
```

6.2.1.6 WriteBlocks

```
void* _EFI_BLOCK_IO_PROTOCOL::WriteBlocks
```

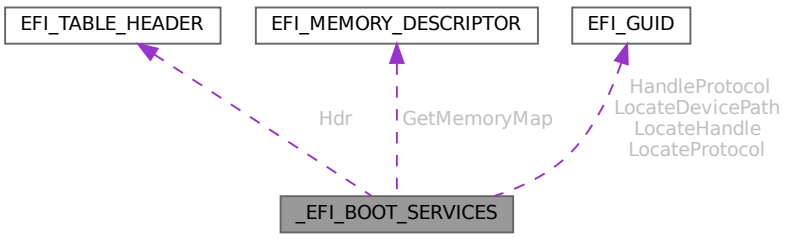
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.3 _EFI_BOOT_SERVICES Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for _EFI_BOOT_SERVICES:



Data Fields

- EFI_TABLE_HEADER Hdr
- EFI_RAISE_TPL RaiseTPL
- EFI_RESTORE_TPL RestoreTPL
- EFI_ALLOCATE_PAGES AllocatePages
- EFI_FREE_PAGES FreePages
- EFI_GET_MEMORY_MAP GetMemoryMap
- EFI_ALLOCATE_POOL AllocatePool
- EFI_FREE_POOL FreePool
- void * CreateEvent
- void * SetTimer
- void * WaitForEvent
- void * SignalEvent
- void * CloseEvent
- void * CheckEvent
- void * InstallProtocolInterface
- void * ReinstallProtocolInterface
- void * UninstallProtocolInterface

- EFI_HANDLE_PROTOCOL HandleProtocol
- void * Reserved
- void * RegisterProtocolNotify
- EFI_LOCATE_HANDLE LocateHandle
- EFI_LOCATE_DEVICE_PATH LocateDevicePath
- void * InstallConfigurationTable
- void * LoadImage
- void * StartImage
- void * Exit
- void * UnloadImage
- EFI_EXIT_BOOT_SERVICES ExitBootServices
- void * GetNextMonotonicCount
- EFI_STALL Stall
- void * SetWatchdogTimer
- void * ConnectController
- void * DisconnectController
- void * OpenProtocol
- void * CloseProtocol
- void * OpenProtocolInformation
- void * ProtocolsPerHandle
- void * LocateHandleBuffer
- EFI_LOCATE_PROTOCOL LocateProtocol

6.3.1 Field Documentation

6.3.1.1 AllocatePages

```
EFI_ALLOCATE_PAGES _EFI_BOOT_SERVICES::AllocatePages
```

Referenced by efi_main(), and load_kernel_from_esp().

6.3.1.2 AllocatePool

```
EFI_ALLOCATE_POOL _EFI_BOOT_SERVICES::AllocatePool
```

6.3.1.3 CheckEvent

```
void* _EFI_BOOT_SERVICES::CheckEvent
```

6.3.1.4 CloseEvent

```
void* _EFI_BOOT_SERVICES::CloseEvent
```

6.3.1.5 CloseProtocol

```
void* _EFI_BOOT_SERVICES::CloseProtocol
```

6.3.1.6 ConnectController

```
void* _EFI_BOOT_SERVICES::ConnectController
```

6.3.1.7 CreateEvent

```
void* _EFI_BOOT_SERVICES::CreateEvent
```

6.3.1.8 DisconnectController

```
void* _EFI_BOOT_SERVICES::DisconnectController
```

6.3.1.9 Exit

```
void* _EFI_BOOT_SERVICES::Exit
```

6.3.1.10 ExitBootServices

```
EFI_EXIT_BOOT_SERVICES _EFI_BOOT_SERVICES::ExitBootServices
```

Referenced by efi_main().

6.3.1.11 FreePages

```
EFI_FREE_PAGES _EFI_BOOT_SERVICES::FreePages
```

6.3.1.12 FreePool

```
EFI_FREE_POOL _EFI_BOOT_SERVICES::FreePool
```

6.3.1.13 GetMemoryMap

```
EFI_GET_MEMORY_MAP _EFI_BOOT_SERVICES::GetMemoryMap
```

Referenced by efi_main().

6.3.1.14 GetNextMonotonicCount

```
void* _EFI_BOOT_SERVICES::GetNextMonotonicCount
```

6.3.1.15 HandleProtocol

```
EFI_HANDLE_PROTOCOL _EFI_BOOT_SERVICES::HandleProtocol
```

Referenced by efi_main(), load_cfg_from_esp(), and load_kernel_from_esp().

6.3.1.16 Hdr

```
EFI_TABLE_HEADER _EFI_BOOT_SERVICES::Hdr
```

6.3.1.17 InstallConfigurationTable

```
void* _EFI_BOOT_SERVICES::InstallConfigurationTable
```

6.3.1.18 InstallProtocolInterface

```
void* _EFI_BOOT_SERVICES::InstallProtocolInterface
```

6.3.1.19 LoadImage

```
void* _EFI_BOOT_SERVICES::LoadImage
```

6.3.1.20 LocateDevicePath

```
EFI_LOCATE_DEVICE_PATH _EFI_BOOT_SERVICES::LocateDevicePath
```

6.3.1.21 LocateHandle

```
EFI_LOCATE_HANDLE _EFI_BOOT_SERVICES::LocateHandle
```

6.3.1.22 LocateHandleBuffer

```
void* _EFI_BOOT_SERVICES::LocateHandleBuffer
```

6.3.1.23 LocateProtocol

```
EFI_LOCATE_PROTOCOL _EFI_BOOT_SERVICES::LocateProtocol
```

Referenced by efi_main().

6.3.1.24 OpenProtocol

```
void* _EFI_BOOT_SERVICES::OpenProtocol
```

6.3.1.25 OpenProtocolInformation

```
void* _EFI_BOOT_SERVICES::OpenProtocolInformation
```

6.3.1.26 ProtocolsPerHandle

```
void* _EFI_BOOT_SERVICES::ProtocolsPerHandle
```

6.3.1.27 RaiseTPL

```
EFI_RAISE_TPL _EFI_BOOT_SERVICES::RaiseTPL
```

6.3.1.28 RegisterProtocolNotify

```
void* _EFI_BOOT_SERVICES::RegisterProtocolNotify
```

6.3.1.29 ReinstallProtocolInterface

```
void* _EFI_BOOT_SERVICES::ReinstallProtocolInterface
```

6.3.1.30 Reserved

```
void* _EFI_BOOT_SERVICES::Reserved
```

6.3.1.31 RestoreTPL

```
EFI_RESTORE_TPL _EFI_BOOT_SERVICES::RestoreTPL
```

6.3.1.32 SetTimer

```
void* _EFI_BOOT_SERVICES::SetTimer
```

6.3.1.33 SetWatchdogTimer

```
void* _EFI_BOOT_SERVICES::SetWatchdogTimer
```

6.3.1.34 SignalEvent

```
void* _EFI_BOOT_SERVICES::SignalEvent
```

6.3.1.35 Stall

```
EFI_STALL _EFI_BOOT_SERVICES::Stall
```

Referenced by debug_checkpoint().

6.3.1.36 StartImage

```
void* _EFI_BOOT_SERVICES::StartImage
```

6.3.1.37 UninstallProtocolInterface

```
void* _EFI_BOOT_SERVICES::UninstallProtocolInterface
```

6.3.1.38 UnloadImage

```
void* _EFI_BOOT_SERVICES::UnloadImage
```

6.3.1.39 WaitForEvent

```
void* _EFI_BOOT_SERVICES::WaitForEvent
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.4 _EFI_FILE_PROTOCOL Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for _EFI_FILE_PROTOCOL:

Data Fields

- UINT64 Revision
- EFI_FILE_OPEN Open
- EFI_FILE_CLOSE Close
- VOID * Delete
- EFI_FILE_READ Read
- VOID * Write
- VOID * GetPosition
- VOID * SetPosition
- EFI_FILE_GET_INFO GetInfo
- VOID * SetInfo
- VOID * Flush

6.4.1 Field Documentation

6.4.1.1 Close

```
EFI_FILE_CLOSE _EFI_FILE_PROTOCOL::Close
```

Referenced by load_cfg_from_esp(), and load_kernel_from_esp().

6.4.1.2 Delete

```
VOID* _EFI_FILE_PROTOCOL::Delete
```

6.4.1.3 Flush

```
VOID* _EFI_FILE_PROTOCOL::Flush
```

6.4.1.4 GetInfo

```
EFI_FILE_GET_INFO _EFI_FILE_PROTOCOL::GetInfo
```

Referenced by load_kernel_from_esp().

6.4.1.5 GetPosition

```
VOID* _EFI_FILE_PROTOCOL::GetPosition
```

6.4.1.6 Open

```
EFI_FILE_OPEN _EFI_FILE_PROTOCOL::Open
```

Referenced by load_cfg_from_esp(), and load_kernel_from_esp().

6.4.1.7 Read

```
EFI_FILE_READ _EFI_FILE_PROTOCOL::Read
```

Referenced by load_cfg_from_esp(), and load_kernel_from_esp().

6.4.1.8 Revision

```
UINT64 _EFI_FILE_PROTOCOL::Revision
```

6.4.1.9 SetInfo

```
VOID* _EFI_FILE_PROTOCOL::SetInfo
```

6.4.1.10 SetPosition

```
VOID* _EFI_FILE_PROTOCOL::SetPosition
```

6.4.1.11 Write

```
VOID* _EFI_FILE_PROTOCOL::Write
```

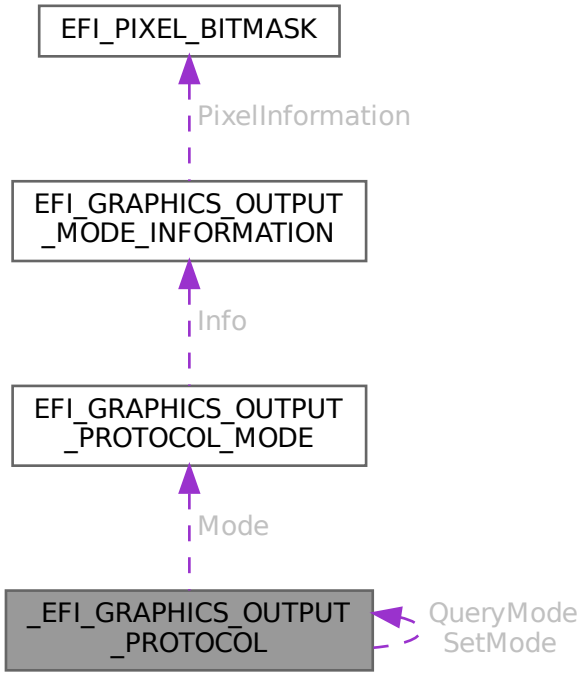
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.5 _EFI_GRAPHICS_OUTPUT_PROTOCOL Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for _EFI_GRAPHICS_OUTPUT_PROTOCOL:



Data Fields

- EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE QueryMode
- EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE SetMode
- void * Blt
- EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE * Mode

6.5.1 Field Documentation

6.5.1.1 Blt

```
void* _EFI_GRAPHICS_OUTPUT_PROTOCOL::Blt
```

6.5.1.2 Mode

```
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE* _EFI_GRAPHICS_OUTPUT_PROTOCOL::Mode
```

Referenced by efi_main().

6.5.1.3 QueryMode

EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE _EFI_GRAPHICS_OUTPUT_PROTOCOL::QueryMode

6.5.1.4 SetMode

EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE _EFI_GRAPHICS_OUTPUT_PROTOCOL::SetMode

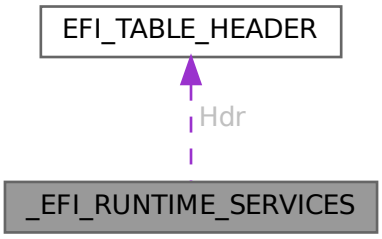
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.6 _EFI_RUNTIME_SERVICES Struct Reference

#include <uefi.h>

Collaboration diagram for _EFI_RUNTIME_SERVICES:



Data Fields

- EFI_TABLE_HEADER Hdr
- void * GetTime
- void * SetTime
- void * GetWakeupTime
- void * SetWakeupTime
- void * SetVirtualAddressMap
- void * ConvertPointer
- void * GetVariable
- void * GetNextVariableName
- void * SetVariable
- void * GetNextHighMonotonicCount
- void * ResetSystem

6.6.1 Field Documentation

6.6.1.1 ConvertPointer

```
void* _EFI_RUNTIME_SERVICES::ConvertPointer
```

6.6.1.2 GetNextHighMonotonicCount

```
void* _EFI_RUNTIME_SERVICES::GetNextHighMonotonicCount
```

6.6.1.3 GetNextVariableName

```
void* _EFI_RUNTIME_SERVICES::GetNextVariableName
```

6.6.1.4 GetTime

```
void* _EFI_RUNTIME_SERVICES::GetTime
```

6.6.1.5 GetVariable

```
void* _EFI_RUNTIME_SERVICES::GetVariable
```

Referenced by efi_main(), and system_word_reboot().

6.6.1.6 GetWakeupTime

```
void* _EFI_RUNTIME_SERVICES::GetWakeupTime
```

6.6.1.7 Hdr

```
EFI_TABLE_HEADER _EFI_RUNTIME_SERVICES::Hdr
```

6.6.1.8 ResetSystem

```
void* _EFI_RUNTIME_SERVICES::ResetSystem
```

Referenced by system_word_reboot().

6.6.1.9 SetTime

```
void* _EFI_RUNTIME_SERVICES::SetTime
```

6.6.1.10 SetVariable

```
void* _EFI_RUNTIME_SERVICES::SetVariable
```

Referenced by efi_main(), kernel_main_deep(), and system_word_reboot().

6.6.1.11 SetVirtualAddressMap

```
void* _EFI_RUNTIME_SERVICES::SetVirtualAddressMap
```

6.6.1.12 SetWakeupTime

```
void* _EFI_RUNTIME_SERVICES::SetWakeupTime
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.7 _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL:

Data Fields

- UINT64 Revision
- EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME OpenVolume

6.7.1 Field Documentation

6.7.1.1 OpenVolume

```
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL::OpenVolume
```

Referenced by load_cfg_from_esp(), and load_kernel_from_esp().

6.7.1.2 Revision

```
UINT64 _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL::Revision
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.8 _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL:

Data Fields

- EFI_TEXT_RESET Reset
- EFI_TEXT_STRING OutputString
- void * TestString
- void * QueryMode
- void * SetMode
- void * SetAttribute
- void * ClearScreen
- void * SetCursorPosition
- void * EnableCursor
- void * Mode

6.8.1 Field Documentation

6.8.1.1 ClearScreen

```
void* _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::ClearScreen
```

6.8.1.2 EnableCursor

```
void* _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::EnableCursor
```

6.8.1.3 Mode

```
void* _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::Mode
```

6.8.1.4 OutputString

```
EFI_TEXT_STRING _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::OutputString
```

Referenced by debug_checkpoint(), and efi_main().

6.8.1.5 QueryMode

```
void* _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::QueryMode
```

6.8.1.6 Reset

```
EFI_TEXT_RESET _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::Reset
```

Referenced by efi_main().

6.8.1.7 SetAttribute

```
void* _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::SetAttribute
```

6.8.1.8 SetCursorPosition

```
void* _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::SetCursorPosition
```

6.8.1.9 SetMode

```
void* _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::SetMode
```

6.8.1.10 TestString

```
void* _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::TestString
```

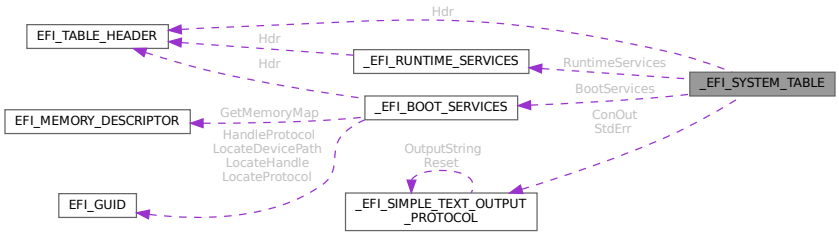
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.9 _EFI_SYSTEM_TABLE Struct Reference

`#include <uefi.h>`

Collaboration diagram for _EFI_SYSTEM_TABLE:



Data Fields

- EFI_TABLE_HEADER Hdr
- CHAR16 * FirmwareVendor
- UINT32 FirmwareRevision
- EFI_HANDLE ConsoleInHandle
- void * ConIn
- EFI_HANDLE ConsoleOutHandle
- EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL * ConOut
- EFI_HANDLE StandardErrorHandle
- EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL * StdErr
- EFI_RUNTIME_SERVICES * RuntimeServices
- EFI_BOOT_SERVICES * BootServices
- UINTN NumberOfTableEntries
- void * ConfigurationTable

6.9.1 Field Documentation

6.9.1.1 BootServices

```
EFI_BOOT_SERVICES* _EFI_SYSTEM_TABLE::BootServices
```

Referenced by debug_checkpoint(), and efi_main().

6.9.1.2 ConfigurationTable

```
void* _EFI_SYSTEM_TABLE::ConfigurationTable
```

Referenced by efi_main().

6.9.1.3 ConIn

```
void* _EFI_SYSTEM_TABLE::ConIn
```

6.9.1.4 ConOut

```
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL* _EFI_SYSTEM_TABLE::ConOut
```

Referenced by debug_checkpoint(), and efi_main().

6.9.1.5 ConsoleInHandle

```
EFI_HANDLE _EFI_SYSTEM_TABLE::ConsoleInHandle
```

6.9.1.6 ConsoleOutHandle

```
EFI_HANDLE _EFI_SYSTEM_TABLE::ConsoleOutHandle
```

6.9.1.7 FirmwareRevision

```
UINT32 _EFI_SYSTEM_TABLE::FirmwareRevision
```

6.9.1.8 FirmwareVendor

```
CHAR16* _EFI_SYSTEM_TABLE::FirmwareVendor
```

6.9.1.9 Hdr

```
EFI_TABLE_HEADER _EFI_SYSTEM_TABLE::Hdr
```

6.9.1.10 NumberOfTableEntries

```
UINTN _EFI_SYSTEM_TABLE::NumberOfTableEntries
```

Referenced by efi_main().

6.9.1.11 RuntimeServices

```
EFI_RUNTIME_SERVICES* _EFI_SYSTEM_TABLE::RuntimeServices
```

Referenced by efi_main().

6.9.1.12 StandardErrorHandle

```
EFI_HANDLE _EFI_SYSTEM_TABLE::StandardErrorHandle
```

6.9.1.13 StdErr

```
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL* _EFI_SYSTEM_TABLE::StdErr
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.10 artemis_sig_t Struct Reference

```
#include <artemis_sig.h>
```

Data Fields

- uint64_t magic
- uint8_t disk_uuid [16]
- uint64_t genesis_time_ns
- uint64_t metadata_devblocks
- uint32_t log_region_offset
- uint32_t log_region_devblocks
- uint64_t hdr_crc
- uint8_t _pad [4096 -(8+16+8+8+4+4+8)]

6.10.1 Field Documentation

6.10.1.1 _pad

```
uint8_t artemis_sig_t::_pad[4096 -( 8+ 16+ 8+ 8+ 4+4+ 8)]
```

6.10.1.2 disk_uuid

```
uint8_t artemis_sig_t::disk_uuid[16]
```

Referenced by artemis_sig_genesis_stamp().

6.10.1.3 genesis_time_ns

```
uint64_t artemis_sig_t::genesis_time_ns
```

Referenced by artemis_sig_genesis_stamp().

6.10.1.4 hdr_crc

```
uint64_t artemis_sig_t::hdr_crc
```

Referenced by artemis_sig_check(), and artemis_sig_genesis_stamp().

6.10.1.5 log_region_devblocks

```
uint32_t artemis_sig_t::log_region_devblocks
```

Referenced by artemis_sig_genesis_stamp().

6.10.1.6 log_region_offset

```
uint32_t artemis_sig_t::log_region_offset
```

Referenced by artemis_sig_genesis_stamp().

6.10.1.7 magic

```
uint64_t artemis_sig_t::magic
```

Referenced by artemis_sig_check(), and artemis_sig_genesis_stamp().

6.10.1.8 metadata_devblocks

```
uint64_t artemis_sig_t::metadata_devblocks
```

Referenced by artemis_sig_genesis_stamp().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/artemis_sig.h

6.11 BayesianLatencyPosterior Struct Reference

Bayesian statistics structure for inference and credible intervals All values stored in Q48.16 fixed-point (64-bit signed integer) representing nanoseconds with 2^-16 ≈ 0.0000153 ns precision.

```
#include <physics_hotwords_cache.h>
```

Data Fields

- int64_t mean_ns_q48
- int64_t stddev_ns_q48
- int64_t median_ns_q48
- int64_t credible_lower_95
- int64_t credible_upper_95
- int64_t credible_lower_99
- int64_t credible_upper_99
- uint64_t sample_count

6.11.1 Detailed Description

Bayesian statistics structure for inference and credible intervals All values stored in Q48.16 fixed-point (64-bit signed integer) representing nanoseconds with 2^-16 ≈ 0.0000153 ns precision.

6.11.2 Field Documentation

6.11.2.1 credible_lower_95

```
int64_t BayesianLatencyPosterior::credible_lower_95
```

Referenced by calculate_posterior(), and hotwords_bayesian_report().

6.11.2.2 credible_lower_99

```
int64_t BayesianLatencyPosterior::credible_lower_99
```

Referenced by calculate_posterior(), and hotwords_bayesian_report().

6.11.2.3 credible_upper_95

```
int64_t BayesianLatencyPosterior::credible_upper_95
```

Referenced by calculate_posterior(), and hotwords_bayesian_report().

6.11.2.4 credible_upper_99

```
int64_t BayesianLatencyPosterior::credible_upper_99
```

Referenced by calculate_posterior(), and hotwords_bayesian_report().

6.11.2.5 mean_ns_q48

```
int64_t BayesianLatencyPosterior::mean_ns_q48
```

Referenced by calculate_posterior(), hotwords_bayesian_report(), and hotwords_speedup_estimate().

6.11.2.6 median_ns_q48

```
int64_t BayesianLatencyPosterior::median_ns_q48
```

Referenced by calculate_posterior().

6.11.2.7 sample_count

```
uint64_t BayesianLatencyPosterior::sample_count
```

Referenced by calculate_posterior(), and hotwords_bayesian_report().

6.11.2.8 stddev_ns_q48

```
int64_t BayesianLatencyPosterior::stddev_ns_q48
```

Referenced by calculate_posterior(), hotwords_bayesian_report(), and hotwords_speedup_estimate().

The documentation for this struct was generated from the following file:

- v3/include/physics_hotwords_cache.h

6.12 blk_bam_entry_t Struct Reference

```
#include <block_subsystem.h>
```

Data Fields

- uint8_t allocated
- uint8_t dirty

6.12.1 Field Documentation

6.12.1.1 allocated

```
uint8_t blk_bam_entry_t::allocated
```

Referenced by bam_flush_to_disk(), bam_sync_from_disk(), blk_allocate(), blk_get_buffer(), blk_is_allocated(), blk_mark_allocated(), blk_mark_free(), blk_update(), and devblock_is_free().

6.12.1.2 dirty

```
uint8_t blk_bam_entry_t::dirty
```

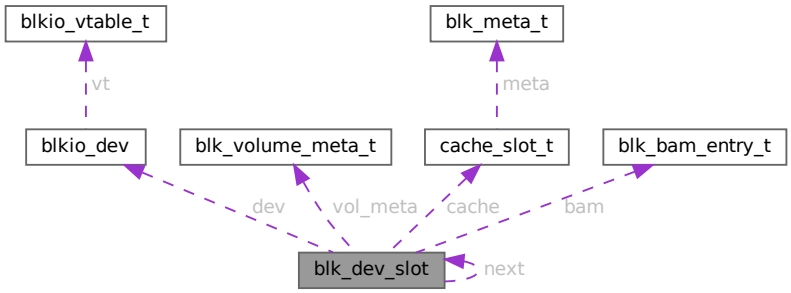
Referenced by blk_flush(), blk_get_buffer(), and blk_update().

The documentation for this struct was generated from the following file:

- v3/include/block_subsystem.h

6.13 blk_dev_slot Struct Reference

Collaboration diagram for blk_dev_slot:



Data Fields

- uint32_t start_lbn
- uint32_t user_blocks
- uint8_t * raw_base
- struct blkio_dev * dev
- uint32_t total_blkio_blocks_1k
- uint32_t devblock_base_4k
- blk_volume_meta_t vol_meta
- uint8_t vol_meta_dirty
- uint8_t bam_dirty
- uint8_t format_state
- cache_slot_t cache [8]
- blk_bam_entry_t * bam
- uint8_t has_ever_claimed
- uint32_t migration_scan_lbn
- blk_dev_slot_t * next

6.13.1 Field Documentation

6.13.1.1 bam

```
blk_bam_entry_t* blk_dev_slot::bam
```

Referenced by bam_flush_to_disk(), bam_sync_from_disk(), blk_allocate(), blk_flush(), blk_format_or_load_disk(), blk_get_buffer(), blk_is_allocated(), blk_mark_allocated(), blk_mark_free(), blk_subsys_add_raw_device(), blk_subsys_attach_device(), blk_subsys_detach_device(), blk_subsys_shutdown(), blk_update(), and devblock_is_free().

6.13.1.2 bam_dirty

```
uint8_t blk_dev_slot::bam_dirty
```

Referenced by bam_flush_to_disk(), bam_sync_from_disk(), blk_allocate(), blk_commit_format(), blk_mark_allocated(), blk_mark_free(), and blk_update().

6.13.1.3 cache

```
cache_slot_t blk_dev_slot::cache[8]
```

Referenced by blk_flush(), and cache_get_slot().

6.13.1.4 dev

```
struct blkio_dev* blk_dev_slot::dev
```

Referenced by bam_flush_to_disk(), bam_sync_from_disk(), blk_allocate(), blk_commit_format(), blk_flush(), blk_meta_zone_read(), blk_meta_zone_write(), blk_migration_idle_check(), blk_subsys_attach_device(), blk_subsys_confirm_format(), blk_subsys_detach_device(), blk_subsys_shutdown(), cache_writeback(), first_disk_slot(), read_devblock_4k(), read_header_4k(), reloc_flush_to_disk(), reloc_load_from_disk(), slot_by_dev(), write_devblock_4k(), and write_header_4k().

6.13.1.5 devblock_base_4k

```
uint32_t blk_dev_slot::devblock_base_4k
```

Referenced by blk_compute_fresh_geometry(), blk_format_or_load_disk(), and devblock4k_to_lba1k().

6.13.1.6 format_state

```
uint8_t blk_dev_slot::format_state
```

Referenced by blk_allocate(), blk_commit_format(), blk_format_or_load_disk(), blk_get_buffer(), blk_mark_allocated(), blk_mark_free(), blk_subsys_attach_device(), blk_subsys_shutdown(), and blk_update().

6.13.1.7 has_ever_claimed

```
uint8_t blk_dev_slot::has_ever_claimed
```

Referenced by blk_migration_idle_check_slot(), and blk_set_meta().

6.13.1.8 migration_scan_lbn

```
uint32_t blk_dev_slot::migration_scan_lbn
```

Referenced by blk_migration_idle_check_slot().

6.13.1.9 next

```
blk_dev_slot_t* blk_dev_slot::next
```

Referenced by blk_allocate(), blk_flush(), blk_migration_idle_check(), blk_subsys_detach_device(), blk_subsys_shutdown(), chain_append(), first_disk_slot(), lbn_to_slot(), and slot_by_dev().

6.13.1.10 raw_base

```
uint8_t* blk_dev_slot::raw_base
```

Referenced by blk_allocate(), blk_flush(), blk_get_buffer(), blk_get_device_free_blocks(), blk_get_meta(), blk_mark_allocated(), blk_mark_free(), blk_set_meta(), blk_subsys_add_raw_device(), and blk_update().

6.13.1.11 start_lbn

```
uint32_t blk_dev_slot::start_lbn
```

Referenced by blk_allocate(), blk_compute_fresh_geometry(), blk_firsttouch_claim(), blk_flush(), blk_get_buffer(), blk_get_device_range(), blk_get_first_disk_range(), blk_is_allocated(), blk_mark_allocated(), blk_mark_free(), blk_migration_idle_check_slot(), blk_owner_has_claim(), blk_subsys_add_raw_device(), blk_subsys_attach_device(), blk_subsys_detach_device(), blk_update(), devblock_claimed_by_lbn(), devblock_is_free(), lbn_to_slot(), and lbn_to_slot_pbn().

6.13.1.12 total_blkio_blocks_1k

```
uint32_t blk_dev_slot::total_blkio_blocks_1k
```

Referenced by blk_compute_fresh_geometry(), blk_format_or_load_disk(), blk_subsys_attach_device(), read_devblock_4k(), and write_devblock_4k().

6.13.1.13 user_blocks

```
uint32_t blk_dev_slot::user_blocks
```

Referenced by bam_flush_to_disk(), bam_pages_used(), bam_sync_from_disk(), blk_allocate(), blk_compute_fresh_geometry(), blk_firsttouch_claim(), blk_flush(), blk_format_or_load_disk(), blk_get_device_range(), blk_get_first_disk_range(), blk_migration_idle_check_slot(), blk_owner_has_claim(), blk_subsys_add_raw_device(), blk_subsys_attach_device(), blk_subsys_detach_device(), devblock_is_free(), and lbn_to_slot().

6.13.1.14 vol_meta

```
blk_volume_meta_t blk_dev_slot::vol_meta
```

Referenced by bam_flush_to_disk(), bam_pages_used(), bam_sync_from_disk(), blk_allocate(), blk_commit_format(), blk_compute_fresh_geometry(), blk_format_or_load_disk(), blk_get_device_free_blocks(), blk_get_volume_meta(), blk_mark_allocated(), blk_mark_free(), blk_meta_zone_read(), blk_meta_zone_write(), blk_set_volume_meta(), blk_subsys_attach_device(), blk_subsys_detach_device(), blk_subsys_shutdown(), blk_update(), reloc_flush_to_disk(), and reloc_load_from_disk().

6.13.1.15 vol_meta_dirty

```
uint8_t blk_dev_slot::vol_meta_dirty
```

Referenced by blk_allocate(), blk_mark_allocated(), blk_mark_free(), blk_set_volume_meta(), blk_subsys_attach_device(), blk_subsys_shutdown(), and blk_update().

The documentation for this struct was generated from the following file:

- v3/src/block_subsystem.c

6.14 blk_meta_t Struct Reference

`#include <block_subsystem.h>`

Data Fields

- uint64_t checksum
- uint64_t magic
- uint64_t created_time
- uint64_t modified_time
- uint64_t flags
- uint64_t write_count
- uint64_t content_type
- uint64_t encoding
- uint64_t content_length
- uint64_t reserved1
- uint64_t entropy [4]
- uint64_t hash [4]
- uint8_t owner_fp [8]
- uint8_t acl_allow
- uint32_t acl_ttl
- uint8_t acl_reserved [3]
- uint64_t reserved_future
- uint64_t prev_block
- uint64_t next_block
- uint64_t parent_block
- uint64_t chain_length
- uint64_t app_data [15]
- uint8_t padding [5]

6.14.1 Field Documentation

6.14.1.1 acl_allow

`uint8_t blk_meta_t::acl_allow`

Referenced by blk_meta_relocate_devblock().

6.14.1.2 acl_reserved

`uint8_t blk_meta_t::acl_reserved[3]`

6.14.1.3 acl_ttl

`uint32_t blk_meta_t::acl_ttl`

Referenced by blk_meta_relocate_devblock().

6.14.1.4 app_data

```
uint64_t blk_meta_t::app_data[15]
```

6.14.1.5 chain_length

```
uint64_t blk_meta_t::chain_length
```

6.14.1.6 checksum

```
uint64_t blk_meta_t::checksum
```

Referenced by blk_update().

6.14.1.7 content_length

```
uint64_t blk_meta_t::content_length
```

6.14.1.8 content_type

```
uint64_t blk_meta_t::content_type
```

6.14.1.9 created_time

```
uint64_t blk_meta_t::created_time
```

Referenced by blk_update().

6.14.1.10 encoding

```
uint64_t blk_meta_t::encoding
```

6.14.1.11 entropy

```
uint64_t blk_meta_t::entropy[4]
```

6.14.1.12 flags

```
uint64_t blk_meta_t::flags
```

Referenced by blk_meta_relocate_devblock(), and devblock_is_free().

6.14.1.13 hash

```
uint64_t blk_meta_t::hash[4]
```

6.14.1.14 magic

```
uint64_t blk_meta_t::magic
```

Referenced by blk_update(), and cache_load_devblock().

6.14.1.15 modified_time

```
uint64_t blk_meta_t::modified_time
```

Referenced by blk_update().

6.14.1.16 next_block

```
uint64_t blk_meta_t::next_block
```

6.14.1.17 owner_fp

```
uint8_t blk_meta_t::owner_fp[8]
```

Referenced by blk_meta_relocate_devblock(), and devblock_is_free().

6.14.1.18 padding

```
uint8_t blk_meta_t::padding[5]
```

6.14.1.19 parent_block

```
uint64_t blk_meta_t::parent_block
```

6.14.1.20 prev_block

```
uint64_t blk_meta_t::prev_block
```

6.14.1.21 reserved1

```
uint64_t blk_meta_t::reserved1
```

6.14.1.22 reserved_future

```
uint64_t blk_meta_t::reserved_future
```

6.14.1.23 write_count

```
uint64_t blk_meta_t::write_count
```

The documentation for this struct was generated from the following file:

- v3/include/block_subsystem.h

6.15 blk_reloc_entry_t Struct Reference

Data Fields

- uint32_t home_lbn
- uint32_t actual_lbn

6.15.1 Field Documentation

6.15.1.1 actual_lbn

```
uint32_t blk_reloc_entry_t::actual_lbn
```

Referenced by resolve_lbn().

6.15.1.2 home_lbn

```
uint32_t blk_reloc_entry_t::home_lbn
```

The documentation for this struct was generated from the following file:

- v3/src/block_subsystem.c

6.16 blk_volume_meta_t Struct Reference

```
#include <block_subsystem.h>
```

Data Fields

- uint32_t magic
- uint32_t version
- uint32_t total_volumes
- uint32_t flags
- char label [64]
- uint64_t total_devblocks
- uint32_t bam_start
- uint32_t bam_devblocks
- uint32_t devblock_base
- uint64_t tracked_blocks
- uint64_t total_blocks
- uint64_t free_blocks
- uint64_t first_free
- uint64_t last_allocated
- uint32_t reserved_disk_lo
- uint32_t reserved_ram_lo
- uint64_t created_time
- uint64_t mounted_time
- uint64_t hdr_crc
- uint32_t reloc_start
- uint32_t reloc_devblocks
- uint32_t meta_fence_blocks
- uint8_t _pad [4096 -(4+4+4+4+64+4+8+4+4+4+8+8+8+8+8+4+4+8+8+8+4+4+4)]

6.16.1 Field Documentation

6.16.1.1 _pad

```
uint8_t blk_volume_meta_t::_pad[4096 -( 4+4+ 4+4+64+ 4+ 8+ 4+4+4+ 8+8+8+ 8+8+ 4+4+ 8+8+ 8+
4+4+ 4)]
```

6.16.1.2 bam_devblocks

```
uint32_t blk_volume_meta_t::bam_devblocks
```

Referenced by bam_cover_devblocks(), bam_flush_to_disk(), bam_pages_used(), bam_sync_from_disk(), blk_commit_format(), blk_compute_fresh_geometry(), blk_format_or_load_disk(), blk_subsys_attach_device(), and compute_totals_from_B().

6.16.1.3 bam_start

```
uint32_t blk_volume_meta_t::bam_start
```

Referenced by bam_flush_to_disk(), bam_sync_from_disk(), blk_commit_format(), and blk_compute_fresh_geometry().

6.16.1.4 created_time

```
uint64_t blk_volume_meta_t::created_time
```

Referenced by blk_compute_fresh_geometry().

6.16.1.5 devblock_base

```
uint32_t blk_volume_meta_t::devblock_base
```

Referenced by blk_compute_fresh_geometry(), blk_format_or_load_disk(), and blk_subsys_attach_device().

6.16.1.6 first_free

```
uint64_t blk_volume_meta_t::first_free
```

Referenced by blk_allocate(), blk_compute_fresh_geometry(), and blk_mark_free().

6.16.1.7 flags

```
uint32_t blk_volume_meta_t::flags
```

Referenced by blk_firsttouch_claim(), blk_flags_get(), blk_flags_set(), blk_migration_idle_check_slot(), blk_owner_has_claim(), blk_set_meta(), and devblock_claimed_by_lbn().

6.16.1.8 free_blocks

```
uint64_t blk_volume_meta_t::free_blocks
```

Referenced by blk_allocate(), blk_compute_fresh_geometry(), blk_get_device_free_blocks(), blk_mark_allocated(), blk_mark_free(), blk_subsys_attach_device(), and blk_update().

6.16.1.9 hdr_crc

```
uint64_t blk_volume_meta_t::hdr_crc
```

6.16.1.10 label

```
char blk_volume_meta_t::label[64]
```

Referenced by blk_compute_fresh_geometry(), blk_subsys_attach_device(), and blk_subsys_detach_device().

6.16.1.11 last_allocated

```
uint64_t blk_volume_meta_t::last_allocated
```

Referenced by blk_allocate(), and blk_compute_fresh_geometry().

6.16.1.12 magic

```
uint32_t blk_volume_meta_t::magic
```

Referenced by blk_compute_fresh_geometry(), blk_format_or_load_disk(), and blk_get_meta().

6.16.1.13 meta_fence_blocks

```
uint32_t blk_volume_meta_t::meta_fence_blocks
```

Referenced by bam_cover_devblocks(), blk_compute_fresh_geometry(), blk_meta_zone_read(), blk_meta_zone_write(), blk_subsys_attach_device(), and compute_totals_from_B().

6.16.1.14 mounted_time

```
uint64_t blk_volume_meta_t::mounted_time
```

Referenced by blk_subsys_attach_device().

6.16.1.15 reloc_devblocks

```
uint32_t blk_volume_meta_t::reloc_devblocks
```

Referenced by bam_cover_devblocks(), blk_commit_format(), blk_compute_fresh_geometry(), compute_totals_from_B(), reloc_flush_to_disk(), and reloc_load_from_disk().

6.16.1.16 reloc_start

```
uint32_t blk_volume_meta_t::reloc_start
```

Referenced by blk_commit_format(), blk_compute_fresh_geometry(), reloc_flush_to_disk(), and reloc_load_from_disk().

6.16.1.17 reserved_disk_lo

```
uint32_t blk_volume_meta_t::reserved_disk_lo
```

Referenced by blk_compute_fresh_geometry().

6.16.1.18 reserved_ram_lo

```
uint32_t blk_volume_meta_t::reserved_ram_lo
```

Referenced by blk_compute_fresh_geometry().

6.16.1.19 total_blocks

```
uint64_t blk_volume_meta_t::total_blocks
```

Referenced by blk_compute_fresh_geometry(), blk_format_or_load_disk(), blk_get_device_free_blocks(), blk_subsys_attach_device(), and compute_totals_from_B().

6.16.1.20 total_devblocks

```
uint64_t blk_volume_meta_t::total_devblocks
```

Referenced by blk_compute_fresh_geometry(), blk_format_or_load_disk(), blk_meta_zone_read(), blk_meta_zone_write(), blk_subsys_attach_device(), and compute_totals_from_B().

6.16.1.21 total_volumes

```
uint32_t blk_volume_meta_t::total_volumes
```

Referenced by blk_compute_fresh_geometry().

6.16.1.22 tracked_blocks

```
uint64_t blk_volume_meta_t::tracked_blocks
```

Referenced by compute_totals_from_B().

6.16.1.23 version

```
uint32_t blk_volume_meta_t::version
```

Referenced by blk_compute_fresh_geometry(), and blk_format_or_load_disk().

The documentation for this struct was generated from the following file:

- v3/include/block_subsystem.h

6.17 blkcfg_t Struct Reference

```
#include <blkcfg.h>
```

Data Fields

- blkcfg_backend_t backend
- uint32_t fbs
- uint8_t read_only
- uint64_t size_bytes
- uint32_t blocks
- char file_path [4096]

6.17.1 Field Documentation

6.17.1.1 backend

```
blkcfg_backend_t blkcfg_t::backend
```

6.17.1.2 blocks

```
uint32_t blkcfg_t::blocks
```

6.17.1.3 fbs

```
uint32_t blkcfg_t::fbs
```

6.17.1.4 file_path

```
char blkcfg_t::file_path[4096]
```

6.17.1.5 read_only

```
uint8_t blkcfg_t::read_only
```

6.17.1.6 size_bytes

```
uint64_t blkcfg_t::size_bytes
```

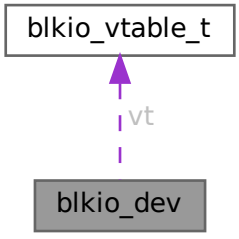
The documentation for this struct was generated from the following file:

- v3/include/blkcfg.h

6.18 blkio_dev Struct Reference

```
#include <blkio.h>
```

Collaboration diagram for blkio_dev:



Data Fields

- const blkio_vtable_t * vt
- uint32_t forth_block_size
- uint32_t total_blocks
- void * state
- uint8_t is_open

6.18.1 Field Documentation

6.18.1.1 forth_block_size

```
uint32_t blkio_dev::forth_block_size
```

Referenced by blkio_info(), blkio_open(), file_open(), gpt_read(), gpt_write_one(), part_open(), ram_open(), usb_blk_open(), and vblk_open().

6.18.1.2 is_open

```
uint8_t blkio_dev::is_open
```

Referenced by blkio_close(), blkio_flush(), blkio_info(), blkio_open(), blkio_read(), and blkio_write().

6.18.1.3 state

```
void* blkio_dev::state
```

Referenced by blkio_open(), file_close(), file_flush(), file_info(), file_open(), file_read(), file_write(), part_flush(), part_info(), part_open(), part_read(), part_write(), ram_close(), ram_info(), ram_open(), ram_read(), ram_write(), usb_blk_info(), usb_blk_open(), usb_blk_read(), usb_blk_write(), vblk_info(), vblk_open(), vblk_read(), and vblk_write().

6.18.1.4 total_blocks

```
uint32_t blkio_dev::total_blocks
```

Referenced by blkio_info(), blkio_open(), blkio_read(), blkio_write(), capsule_wirebind_overflow_idle_check(), file_open(), file_read(), file_write(), gpt_grow_last(), gpt_part_open(), gpt_read(), gpt_write_one(), part_open(), ram_open(), sk_v4_grow_stick(), sk_v4_identity_offer_wait(), usb_blk_open(), and vblk_open().

6.18.1.5 vt

```
const blkio_vtable_t* blkio_dev::vt
```

Referenced by blkio_close(), blkio_flush(), blkio_info(), blkio_open(), blkio_read(), blkio_write(), gpt_grow_last(), gpt_read(), gpt_write_one(), part_flush(), and part_info().

The documentation for this struct was generated from the following file:

- v3/include/blkio.h

6.19 blkio_file_state_t Struct Reference

Data Fields

- int fd
- char path [4096]
- uint32_t total_blocks
- uint32_t fbs
- uint8_t read_only

6.19.1 Field Documentation

6.19.1.1 fbs

```
uint32_t blkio_file_state_t::fbs
```

Referenced by blkio_file_init_state(), file_info(), file_open(), file_read(), and file_write().

6.19.1.2 fd

```
int blkio_file_state_t::fd
```

Referenced by blkio_file_init_state(), file_close(), file_flush(), file_info(), file_open(), file_read(), and file_write().

6.19.1.3 path

```
char blkio_file_state_t::path[4096]
```

Referenced by blkio_file_init_state(), and file_open().

6.19.1.4 read_only

```
uint8_t blkio_file_state_t::read_only
```

Referenced by blkio_file_init_state(), file_info(), file_open(), and file_write().

6.19.1.5 total_blocks

```
uint32_t blkio_file_state_t::total_blocks
```

Referenced by blkio_file_init_state(), file_info(), and file_open().

The documentation for this struct was generated from the following file:

- v3/src/blkio_file.c

6.20 blkio_info_t Struct Reference

`#include <blkio.h>`

Data Fields

- uint32_t forth_block_size
- uint32_t total_blocks
- uint32_t phys_sector_size
- uint64_t phys_size_bytes
- uint8_t read_only

6.20.1 Field Documentation

6.20.1.1 forth_block_size

`uint32_t blkio_info_t::forth_block_size`

Referenced by blkio_info(), file_info(), init_blkio(), ram_info(), usb_blk_info(), and vblk_info().

6.20.1.2 phys_sector_size

`uint32_t blkio_info_t::phys_sector_size`

Referenced by file_info(), gpt_read(), gpt_write_one(), ram_info(), usb_blk_info(), and vblk_info().

6.20.1.3 phys_size_bytes

`uint64_t blkio_info_t::phys_size_bytes`

Referenced by file_info(), init_blkio(), part_info(), ram_info(), usb_blk_info(), and vblk_info().

6.20.1.4 read_only

`uint8_t blkio_info_t::read_only`

Referenced by file_info(), init_blkio(), ram_info(), usb_blk_info(), and vblk_info().

6.20.1.5 total_blocks

`uint32_t blkio_info_t::total_blocks`

Referenced by artemis_sig_start_fblock(), blk_subsys_attach_device(), blkio_info(), file_info(), init_blkio(), part_info(), ram_info(), sk_v4_grow_artemis(), usb_blk_info(), and vblk_info().

The documentation for this struct was generated from the following file:

- v3/include/blkio.h

6.21 blkio_params_t Struct Reference

`#include <blkio.h>`

Data Fields

- uint32_t forth_block_size
- uint32_t total_blocks
- void * opaque

6.21.1 Field Documentation

6.21.1.1 forth_block_size

`uint32_t blkio_params_t::forth_block_size`

Referenced by blkio_factory_open(), blkio_open(), blkio_usb_open_msc(), capsule_mint_identity_scratch(), gpt_part_open(), and virtio_blk_find_artemis().

6.21.1.2 opaque

`void* blkio_params_t::opaque`

Referenced by blkio_usb_open_msc(), file_open(), gpt_part_open(), part_open(), ram_open(), usb_blk_open(), and virtio_blk_find_artemis().

6.21.1.3 total_blocks

`uint32_t blkio_params_t::total_blocks`

Referenced by blkio_open(), blkio_usb_open_msc(), gpt_part_open(), and virtio_blk_find_artemis().

The documentation for this struct was generated from the following file:

- v3/include/blkio.h

6.22 blkio_ram_state_t Struct Reference

Data Fields

- uint8_t * base
- uint32_t total_blocks
- uint32_t fbs
- uint8_t read_only

6.22.1 Field Documentation

6.22.1.1 base

```
uint8_t* blkio_ram_state_t::base
```

Referenced by blkio_ram_init_state(), ram_open(), ram_read(), and ram_write().

6.22.1.2 fbs

```
uint32_t blkio_ram_state_t::fbs
```

Referenced by blkio_ram_init_state(), ram_info(), ram_open(), ram_read(), and ram_write().

6.22.1.3 read_only

```
uint8_t blkio_ram_state_t::read_only
```

Referenced by blkio_ram_init_state(), ram_info(), and ram_write().

6.22.1.4 total_blocks

```
uint32_t blkio_ram_state_t::total_blocks
```

Referenced by blkio_ram_init_state(), in_bounds(), ram_info(), and ram_open().

The documentation for this struct was generated from the following file:

- v3/src/blkio_ram.c

6.23 blkio_vtable_t Struct Reference

```
#include <blkio.h>
```

Data Fields

- int(∗ open)(struct blkio_dev ∗dev, const blkio_params_t ∗p)
- int(∗ close)(struct blkio_dev ∗dev)
- int(∗ read)(struct blkio_dev ∗dev, uint32_t fblock, void ∗dst)
- int(∗ write)(struct blkio_dev ∗dev, uint32_t fblock, const void ∗src)
- int(∗ flush)(struct blkio_dev ∗dev)
- int(∗ info)(struct blkio_dev ∗dev, blkio_info_t ∗out)

6.23.1 Field Documentation

6.23.1.1 close

```
int(* blkio_vtable_t::close) (struct blkio_dev *dev)
```

Referenced by blkio_close(), and blkio_open().

6.23.1.2 flush

```
int(* blkio_vtable_t::flush) (struct blkio_dev *dev)
```

Referenced by blkio_flush(), blkio_open(), gpt_grow_last(), gpt_write_one(), and part_flush().

6.23.1.3 info

```
int(* blkio_vtable_t::info) (struct blkio_dev *dev, blkio_info_t *out)
```

Referenced by blkio_info(), blkio_open(), gpt_read(), gpt_write_one(), and part_info().

6.23.1.4 open

```
int(* blkio_vtable_t::open) (struct blkio_dev *dev, const blkio_params_t *p)
```

Referenced by blkio_open().

6.23.1.5 read

```
int(* blkio_vtable_t::read) (struct blkio_dev *dev, uint32_t fblock, void *dst)
```

Referenced by blkio_open(), and blkio_read().

6.23.1.6 write

```
int(* blkio_vtable_t::write) (struct blkio_dev *dev, uint32_t fblock, const void *src)
```

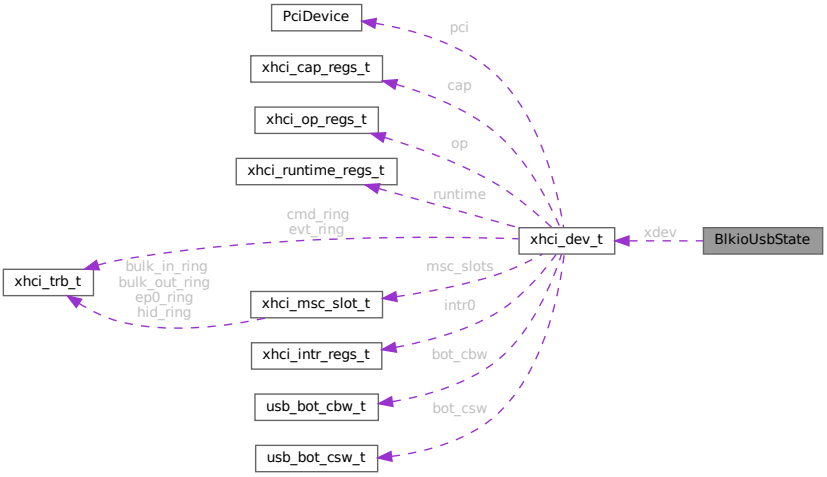
Referenced by blkio_open(), and blkio_write().

The documentation for this struct was generated from the following file:

- v3/include/blkio.h

6.24 BlkioUsbState Struct Reference

Collaboration diagram for BlkioUsbState:



Data Fields

- xhci_dev_t * xdev
- uint32_t slot_id
- uint32_t scsi_block_size
- uint32_t scsi_blocks_per_fblock
- uint32_t total_forth_blocks

6.24.1 Field Documentation

6.24.1.1 scsi_block_size

```
uint32_t BlkioUsbState::scsi_block_size
```

Referenced by blkio_usb_open_msc(), usb_blk_info(), usb_blk_read(), and usb_blk_write().

6.24.1.2 scsi_blocks_per_fblock

```
uint32_t BlkioUsbState::scsi_blocks_per_fblock
```

Referenced by blkio_usb_open_msc(), usb_blk_read(), and usb_blk_write().

6.24.1.3 slot_id

```
uint32_t BlkioUsbState::slot_id
```

Referenced by blkio_usb_open_msc(), usb_blk_read(), and usb_blk_write().

6.24.1.4 total_forth_blocks

```
uint32_t BlkioUsbState::total_forth_blocks
```

Referenced by blkio_usb_open_msc(), usb_blk_info(), usb_blk_open(), usb_blk_read(), and usb_blk_write().

6.24.1.5 xdev

```
xhci_dev_t* BlkioUsbState::xdev
```

Referenced by blkio_usb_open_msc(), usb_blk_read(), and usb_blk_write().

The documentation for this struct was generated from the following file:

- kernel/src/usb/blkio_usb.c

6.25 BlockBuffer Struct Reference

Block buffer structure maintaining buffer state.

```
#include <block_words.h>
```

Data Fields

- unsigned char data [1024]
  Block content buffer.
- int block_number
  Currently loaded block number (-1 = empty)
- int dirty
  Modified flag for write-back.
- int accessed
  LRU counter for buffer management.
- buffer_type_t type
  Buffer type for storage mechanism.
- uintptr_t hw_address
  Future: Hardware backing address.

6.25.1 Detailed Description

Block buffer structure maintaining buffer state.

6.25.2 Field Documentation

6.25.2.1 accessed

```
int BlockBuffer::accessed
```

LRU counter for buffer management.

6.25.2.2 block_number

```
int BlockBuffer::block_number
```

Currently loaded block number (-1 = empty)

6.25.2.3 data

```
unsigned char BlockBuffer::data[1024]
```

Block content buffer.

6.25.2.4 dirty

```
int BlockBuffer::dirty
```

Modified flag for write-back.

6.25.2.5 hw_address

```
uintptr_t BlockBuffer::hw_address
```

Future: Hardware backing address.

6.25.2.6 type

```
buffer_type_t BlockBuffer::type
```

Buffer type for storage mechanism.

The documentation for this struct was generated from the following file:

- v3/src/word_source/include/block_words.h

6.26 BootDisk Struct Reference

```
#include <uefi.h>
```

Data Fields

- uint8_t gpt
- uint8_t is_stick
- uint8_t disk_guid [16]

6.26.1 Field Documentation

6.26.1.1 disk_guid

```
uint8_t BootDisk::disk_guid[16]
```

Referenced by sk_v4_is_this_stick().

6.26.1.2 gpt

```
uint8_t BootDisk::gpt
```

6.26.1.3 is_stick

```
uint8_t BootDisk::is_stick
```

Referenced by kernel_main_deep(), and sk_v4_is_this_stick().

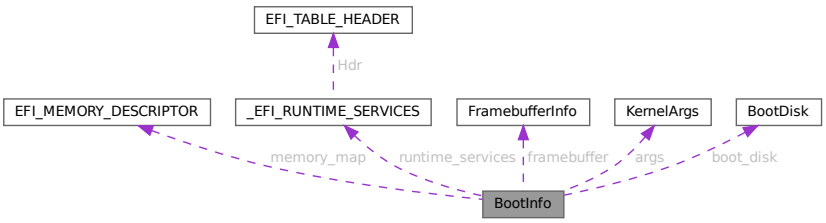
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.27 BootInfo Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for BootInfo:



Data Fields

- EFI_MEMORY_DESCRIPTOR * memory_map
- UINTN memory_map_size
- UINTN memory_map_descriptor_size
- EFI_RUNTIME_SERVICES * runtime_services
- void * acpi_table
- void * dtb
- FramebufferInfo framebuffer
- UINT8 uefi_boot_services_exited
- void * kernel_stack_base
- uint64_t kernel_stack_size
- KernelArgs args
- BootDisk boot_disk

6.27.1 Field Documentation

6.27.1.1 acpi_table

```
void* BootInfo::acpi_table
```

Referenced by efi_main(), kernel_main(), kernel_main_deep(), rpi5_native_boot(), running_under_hypervisor(), and timer_init().

6.27.1.2 args

```
KernelArgs BootInfo::args
```

Referenced by efi_main(), kernel_main(), kernel_main_deep(), and rpi5_native_boot().

6.27.1.3 boot_disk

```
BootDisk BootInfo::boot_disk
```

Referenced by efi_main(), and kernel_main_deep().

6.27.1.4 dtb

```
void* BootInfo::dtb
```

Referenced by apic_init(), efi_main(), kernel_main(), rpi5_native_boot(), running_under_hypervisor(), and timer_init().

6.27.1.5 framebuffer

```
FramebufferInfo BootInfo::framebuffer
```

Referenced by efi_main(), rpi5_native_boot(), and vmm_init().

6.27.1.6 kernel_stack_base

```
void* BootInfo::kernel_stack_base
```

Referenced by efi_main(), and rpi5_native_boot().

6.27.1.7 kernel_stack_size

```
uint64_t BootInfo::kernel_stack_size
```

Referenced by efi_main(), and rpi5_native_boot().

6.27.1.8 memory_map

```
EFI_MEMORY_DESCRIPTOR* BootInfo::memory_map
```

Referenced by efi_main(), pmm_init(), print_boot_info(), rpi5_native_boot(), and vmm_init().

6.27.1.9 memory_map_descriptor_size

```
UINTN BootInfo::memory_map_descriptor_size
```

Referenced by efi_main(), pmm_init(), print_boot_info(), rpi5_native_boot(), and vmm_init().

6.27.1.10 memory_map_size

```
UINTN BootInfo::memory_map_size
```

Referenced by efi_main(), pmm_init(), print_boot_info(), rpi5_native_boot(), and vmm_init().

6.27.1.11 runtime_services

```
EFI_RUNTIME_SERVICES* BootInfo::runtime_services
```

Referenced by efi_main(), kernel_main(), and rpi5_native_boot().

6.27.1.12 uefi_boot_services_exited

```
UINT8 BootInfo::uefi_boot_services_exited
```

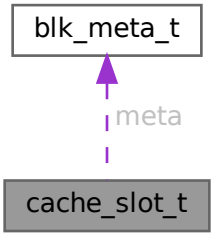
Referenced by efi_main(), and rpi5_native_boot().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.28 cache_slot_t Struct Reference

Collaboration diagram for cache_slot_t:



Data Fields

- uint32_t devblock
- uint8_t data [BLK_DEVICE_SECTOR]
- blk_meta_t meta [BLK_PACK_RATIO]
- uint8_t valid
- uint8_t loaded
- uint8_t dirty
- uint8_t meta_dirty

6.28.1 Field Documentation

6.28.1.1 data

```
uint8_t cache_slot_t::data[BLK_DEVICE_SECTOR]
```

Referenced by blk_get_buffer(), blk_update(), cache_load_devblock(), and cache_writeback().

6.28.1.2 devblock

```
uint32_t cache_slot_t::devblock
```

Referenced by blk_flush(), cache_get_slot(), and cache_writeback().

6.28.1.3 dirty

```
uint8_t cache_slot_t::dirty
```

Referenced by blk_flush(), blk_get_buffer(), blk_update(), and cache_writeback().

6.28.1.4 loaded

```
uint8_t cache_slot_t::loaded
```

Referenced by cache_load_devblock().

6.28.1.5 meta

```
blk_meta_t cache_slot_t::meta[BLK_PACK_RATIO]
```

Referenced by blk_get_meta(), blk_set_meta(), blk_update(), cache_load_devblock(), and cache_writeback().

6.28.1.6 meta_dirty

```
uint8_t cache_slot_t::meta_dirty
```

Referenced by blk_flush(), blk_set_meta(), blk_update(), and cache_writeback().

6.28.1.7 valid

```
uint8_t cache_slot_t::valid
```

Referenced by blk_flush(), and cache_get_slot().

The documentation for this struct was generated from the following file:

- v3/src/block_subsystem.c

6.29 CapsuleDirHeader Struct Reference

```
#include <capsule.h>
```

Data Fields

- uint64_t magic
- uint64_t arena_base
- uint64_t arena_size
- uint32_t desc_count
- uint32_t desc_capacity
- uint32_t name_count
- uint32_t reserved
- uint64_t dir_hash

6.29.1 Field Documentation

6.29.1.1 arena_base

```
uint64_t CapsuleDirHeader::arena_base
```

Referenced by capsule_console_birth(), capsule_runcap_birth(), and kernel_main_deep().

6.29.1.2 arena_size

```
uint64_t CapsuleDirHeader::arena_size
```

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_console_birth(), capsule_exec_init(), capsule_run_experiment(), capsule_runcap_birth(), and kernel_main_deep().

6.29.1.3 desc_capacity

```
uint32_t CapsuleDirHeader::desc_capacity
```

Referenced by capsule_console_birth(), capsule_runcap_birth(), and kernel_main_deep().

6.29.1.4 desc_count

```
uint32_t CapsuleDirHeader::desc_count
```

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_console_birth(), capsule_find_by_id(), capsule_find_by_name(), capsule_find_mama_init(), capsule_run_experiment(), capsule_runcap_birth(), and kernel_main_deep().

6.29.1.5 dir_hash

```
uint64_t CapsuleDirHeader::dir_hash
```

Referenced by capsule_console_birth(), capsule_runcap_birth(), and kernel_main_deep().

6.29.1.6 magic

```
uint64_t CapsuleDirHeader::magic
```

Referenced by capsule_console_birth(), capsule_runcap_birth(), and kernel_main_deep().

6.29.1.7 name_count

```
uint32_t CapsuleDirHeader::name_count
```

Referenced by capsule_console_birth(), capsule_runcap_birth(), and kernel_main_deep().

6.29.1.8 reserved

```
uint32_t CapsuleDirHeader::reserved
```

Referenced by kernel_main_deep().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/capsule.h

6.30 CapsuleNameEntry Struct Reference

```
#include <capsule.h>
```

Data Fields

- char name [512]

6.30.1 Field Documentation

6.30.1.1 name

```
char CapsuleNameEntry::name[512]
```

Referenced by capsule_console_birth(), and capsule_runcap_birth().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/capsule.h

6.31 CapsuleRunRecord Struct Reference

```
#include <capsule_run.h>
```

Data Fields

- uint64_t run_id
- VMUuid vm_id
- uint32_t reserved
- uint64_t capsule_id
- uint64_t capsule_hash
- uint64_t pre_dict_hash
- uint64_t post_dict_hash
- uint64_t started_ns
- uint64_t ended_ns
- uint32_t result_code
- uint32_t flags

6.31.1 Field Documentation

6.31.1.1 capsule_hash

```
uint64_t CapsuleRunRecord::capsule_hash
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.2 capsule_id

```
uint64_t CapsuleRunRecord::capsule_id
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.3 ended_ns

```
uint64_t CapsuleRunRecord::ended_ns
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.4 flags

```
uint32_t CapsuleRunRecord::flags
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.5 post_dict_hash

```
uint64_t CapsuleRunRecord::post_dict_hash
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.6 pre_dict_hash

```
uint64_t CapsuleRunRecord::pre_dict_hash
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.7 reserved

```
uint32_t CapsuleRunRecord::reserved
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.8 result_code

```
uint32_t CapsuleRunRecord::result_code
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.9 run_id

```
uint64_t CapsuleRunRecord::run_id
```

Referenced by capsule_run_experiment(), capsule_run_log_init(), and capsule_run_log_record().

6.31.1.10 started_ns

```
uint64_t CapsuleRunRecord::started_ns
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

6.31.1.11 vm_id

```
VMUuid CapsuleRunRecord::vm_id
```

Referenced by capsule_run_experiment(), and capsule_run_log_init().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/capsule_run.h

6.32 CapsuleSigEntry Struct Reference

```
#include <capsule.h>
```

Data Fields

- uint8_t sig [64]
- uint8_t has_sig
- uint8_t _pad [7]

6.32.1 Field Documentation

6.32.1.1 _pad

```
uint8_t CapsuleSigEntry::_pad[7]
```

6.32.1.2 has_sig

```
uint8_t CapsuleSigEntry::has_sig
```

6.32.1.3 sig

```
uint8_t CapsuleSigEntry::sig[64]
```

Referenced by capsule_verify_signature().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/capsule.h

6.33 CDConfigEntry Struct Reference

```
#include <compudynamics.h>
```

Data Fields

- uint8_t config_bits
- uint8_t has_prev
- uint8_t _pad [2]
- uint32_t benefit_score
- uint32_t tick_count
- uint32_t prev_window
- uint32_t prev_locality_q16

6.33.1 Field Documentation

6.33.1.1 _pad

```
uint8_t CDConfigEntry::_pad[2]
```

Referenced by cd_config_table_seed().

6.33.1.2 benefit_score

```
uint32_t CDConfigEntry::benefit_score
```

Referenced by cd_config_table_seed().

6.33.1.3 config_bits

```
uint8_t CDConfigEntry::config_bits
```

Referenced by cd_config_table_seed(), cd_config_table_tick(), ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), ssm_l8_update_table(), and vm_heartbeat_update_l8().

6.33.1.4 has_prev

```
uint8_t CDConfigEntry::has_prev
```

Referenced by cd_config_table_seed().

6.33.1.5 prev_locality_q16

```
uint32_t CDConfigEntry::prev_locality_q16
```

Referenced by cd_config_table_seed().

6.33.1.6 prev_window

```
uint32_t CDConfigEntry::prev_window
```

Referenced by cd_config_table_seed().

6.33.1.7 tick_count

```
uint32_t CDConfigEntry::tick_count
```

Referenced by cd_config_table_seed().

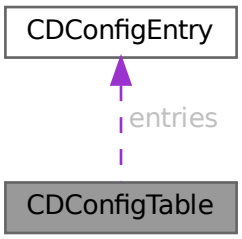
The documentation for this struct was generated from the following file:

- v3/include/compudynamics.h

6.34 CDConfigTable Struct Reference

```
#include <compudynamics.h>
```

Collaboration diagram for CDConfigTable:



Data Fields

- CDConfigEntry * entries
- uint32_t * regime_scores
- uint32_t * regime_ticks
- uint32_t * total_regime_ticks
- uint32_t num_configs
- uint32_t num_regimes
- uint8_t current_config
- uint8_t current_regime

6.34.1 Field Documentation

6.34.1.1 current_config

```
uint8_t CDConfigTable::current_config
```

Referenced by cd_config_table_force(), cd_config_table_seed(), cd_config_table_tick(), ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), ssm_l8_update_table(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.34.1.2 current_regime

```
uint8_t CDConfigTable::current_regime
```

Referenced by cd_config_table_seed(), cd_config_table_tick(), l8_regime_modulation_q16(), ssm_l8_tick_score_and_select(), ssm_l8_update_table(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.34.1.3 entries

```
CDConfigEntry* CDConfigTable::entries
```

Referenced by cd_config_table_force(), cd_config_table_free(), cd_config_table_init(), cd_config_table_seed(), cd_config_table_tick(), ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), ssm_l8_update_table(), and vm_heartbeat_update_l8().

6.34.1.4 num_configs

```
uint32_t CDConfigTable::num_configs
```

Referenced by cd_config_table_force(), cd_config_table_init(), cd_config_table_seed(), cd_config_table_tick(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.34.1.5 num_regimes

```
uint32_t CDConfigTable::num_regimes
```

Referenced by cd_config_table_init(), and cd_config_table_seed().

6.34.1.6 regime_scores

```
uint32_t* CDConfigTable::regime_scores
```

Referenced by cd_config_table_free(), cd_config_table_init(), cd_config_table_seed(), cd_config_table_tick(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.34.1.7 regime_ticks

```
uint32_t* CDConfigTable::regime_ticks
```

Referenced by cd_config_table_free(), cd_config_table_init(), cd_config_table_seed(), and cd_config_table_tick().

6.34.1.8 total_regime_ticks

```
uint32_t* CDConfigTable::total_regime_ticks
```

Referenced by cd_config_table_free(), cd_config_table_init(), cd_config_table_seed(), and cd_config_table_tick().

The documentation for this struct was generated from the following file:

- v3/include/compudynamics.h

6.35 CDRegimeMetrics Struct Reference

`#include <compudynamics.h>`

Data Fields

- uint32_t diversity_q16
- uint32_t volatility_q16
- uint32_t locality_q16

6.35.1 Field Documentation

6.35.1.1 diversity_q16

`uint32_t CDRegimeMetrics::diversity_q16`

Referenced by cd_classify_ids().

6.35.1.2 locality_q16

`uint32_t CDRegimeMetrics::locality_q16`

Referenced by cd_classify_ids(), and ssm_l8_update_table().

6.35.1.3 volatility_q16

`uint32_t CDRegimeMetrics::volatility_q16`

Referenced by cd_classify_ids().

The documentation for this struct was generated from the following file:

- v3/include/compudynamics.h

6.36 CDTuning Struct Reference

`#include <compudynamics.h>`

Data Fields

- uint32_t id_window_depth
- uint32_t locality_lookback
- uint32_t diversity_high_threshold_q16
- uint32_t volatility_high_threshold_q16
- uint32_t locality_high_threshold_q16
- uint32_t num_configs
- uint32_t num_regimes
- uint32_t score_max
- uint32_t score_min
- uint32_t score_decay_factor_q16
- uint32_t ucb_k
- uint32_t reward_gain
- uint32_t joint_weight_q16
- uint32_t anova_weight_q16

6.36.1 Field Documentation

6.36.1.1 anova_weight_q16

```
uint32_t CDTuning::anova_weight_q16
```

Referenced by cd_config_table_tick(), cd_tuning_vm(), and cd_tuning_word().

6.36.1.2 diversity_high_threshold_q16

```
uint32_t CDTuning::diversity_high_threshold_q16
```

Referenced by cd_tuning_vm(), and cd_tuning_word().

6.36.1.3 id_window_depth

```
uint32_t CDTuning::id_window_depth
```

Referenced by cd_tuning_vm(), cd_tuning_word(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.36.1.4 joint_weight_q16

```
uint32_t CDTuning::joint_weight_q16
```

Referenced by cd_config_table_tick(), cd_tuning_vm(), and cd_tuning_word().

6.36.1.5 locality_high_threshold_q16

```
uint32_t CDTuning::locality_high_threshold_q16
```

Referenced by cd_tuning_vm(), cd_tuning_word(), and ssm_l8_update_table().

6.36.1.6 locality_lookback

```
uint32_t CDTuning::locality_lookback
```

Referenced by cd_tuning_vm(), cd_tuning_word(), and ssm_l8_update_table().

6.36.1.7 num_configs

```
uint32_t CDTuning::num_configs
```

Referenced by cd_config_table_init(), cd_tuning_vm(), cd_tuning_word(), and ssm_l8_seed_from_doe().

6.36.1.8 num_regimes

```
uint32_t CDTuning::num_regimes
```

Referenced by cd_config_table_init(), cd_tuning_vm(), and cd_tuning_word().

6.36.1.9 reward_gain

```
uint32_t CDTuning::reward_gain
```

Referenced by cd_config_table_tick(), cd_tuning_vm(), and cd_tuning_word().

6.36.1.10 score_decay_factor_q16

```
uint32_t CDTuning::score_decay_factor_q16
```

Referenced by cd_config_table_tick(), cd_tuning_vm(), and cd_tuning_word().

6.36.1.11 score_max

```
uint32_t CDTuning::score_max
```

Referenced by cd_config_table_tick(), cd_tuning_vm(), cd_tuning_word(), and ssm_l8_seed_from_doe().

6.36.1.12 score_min

```
uint32_t CDTuning::score_min
```

Referenced by cd_config_table_tick(), cd_tuning_vm(), and cd_tuning_word().

6.36.1.13 ucb_k

```
uint32_t CDTuning::ucb_k
```

Referenced by cd_config_table_tick(), cd_tuning_vm(), and cd_tuning_word().

6.36.1.14 volatility_high_threshold_q16

```
uint32_t CDTuning::volatility_high_threshold_q16
```

Referenced by cd_tuning_vm(), and cd_tuning_word().

The documentation for this struct was generated from the following file:

- v3/include/compudynamics.h

6.37 cf_item_t Struct Reference

```
#include <vm.h>
```

Data Fields

- size_t addr
- cf_tag_t tag

6.37.1 Field Documentation

6.37.1.1 addr

```
size_t cf_item_t::addr
```

Referenced by cf_push_item(), control_forth_again(), control_forth_else(), control_forth_endof(), control_forth_loop(), control_forth_plus_loop(), control_forth_repeat(), control_forth_then(), and control_forth_until().

6.37.1.2 tag

```
cf_tag_t cf_item_t::tag
```

Referenced by cf_push_item(), control_forth_again(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_repeat(), control_forth_then(), control_forth_until(), and control_forth_while().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.38 CLIConfig Struct Reference

Configuration structure from command-line arguments Consolidates all parsed CLI options into a single, cleanly organized struct.

```
#include <cli.h>
```

Data Fields

- int run_tests
- int break_me
- int fail_fast
- int benchmark
- int benchmark_iterations
- int doe_experiment
- HeartbeatLogMode heartbeat_log_mode
- int script_mode
- LogLevel log_level
- int log_level_explicitly_set
- const char * disk_img_path
- uint32_t ram_disk_mb
- int run_tests_flag_observed

6.38.1 Detailed Description

Configuration structure from command-line arguments Consolidates all parsed CLI options into a single, cleanly organized struct.

6.38.2 Field Documentation

6.38.2.1 benchmark

```
int CLIConfig::benchmark
```

Referenced by cli_parse(), and main().

6.38.2.2 benchmark_iterations

```
int CLIConfig::benchmark_iterations
```

Referenced by cli_parse(), and run_benchmark_mode().

6.38.2.3 break_me

```
int CLIConfig::break_me
```

Referenced by cli_parse(), main(), and run_optional_tests().

6.38.2.4 disk_img_path

```
const char* CLIConfig::disk_img_path
```

Referenced by cli_parse(), and init_blkio().

6.38.2.5 doe_experiment

```
int CLIConfig::doe_experiment
```

Referenced by cli_parse(), and main().

6.38.2.6 fail_fast

```
int CLIConfig::fail_fast
```

Referenced by cli_parse(), and main().

6.38.2.7 heartbeat_log_mode

```
HeartbeatLogMode CLIConfig::heartbeat_log_mode
```

Referenced by cli_parse().

6.38.2.8 log_level

```
LogLevel CLIConfig::log_level
```

Referenced by cli_parse(), and main().

6.38.2.9 log_level_explicitly_set

```
int CLIConfig::log_level_explicitly_set
```

Referenced by cli_parse(), and main().

6.38.2.10 ram_disk_mb

```
uint32_t CLIConfig::ram_disk_mb
```

Referenced by cli_parse(), and init_blkio().

6.38.2.11 run_tests

```
int CLIConfig::run_tests
```

Referenced by cli_parse().

6.38.2.12 run_tests_flag_observed

```
int CLIConfig::run_tests_flag_observed
```

Referenced by cli_parse(), and main().

6.38.2.13 script_mode

```
int CLIConfig::script_mode
```

Referenced by cli_parse(), and main().

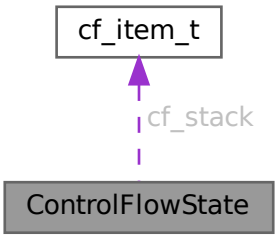
The documentation for this struct was generated from the following file:

- v3/include/cli.h

6.39 ControlFlowState Struct Reference

```
#include <vm.h>
```

Collaboration diagram for ControlFlowState:



Data Fields

- cf_item_t cf_stack [64]
- int cf_sp
- int cf_last_mode
- size_t leave_addrs [64]
- int leave_sp
- int leave_mark_stack [64]
- int leave_mark_sp
- size_t endof_addrs [64]
- int endof_sp
- int endof_mark_stack [64]
- int endof_mark_sp

6.39.1 Field Documentation

6.39.1.1 cf_last_mode

```
int ControlFlowState::cf_last_mode
```

Referenced by cf_epoch_sync(), vm_init(), and vm_init_with_host().

6.39.1.2 cf_sp

```
int ControlFlowState::cf_sp
```

Referenced by cf_epoch_sync(), cf_peek_item(), cf_pop_item(), cf_push_item(), control_forth_leave(), control_forth_of(), vm_init(), and vm_init_with_host().

6.39.1.3 cf_stack

```
cf_item_t ControlFlowState::cf_stack[64]
```

Referenced by cf_peek_item(), cf_pop_item(), cf_push_item(), control_forth_leave(), and control_forth_of().

6.39.1.4 endof_addrs

```
size_t ControlFlowState::endof_addrs[64]
```

Referenced by control_forth_endcase(), and control_forth_endof().

6.39.1.5 endof_mark_sp

```
int ControlFlowState::endof_mark_sp
```

Referenced by control_forth_case(), control_forth_endcase(), vm_init(), and vm_init_with_host().

6.39.1.6 endof_mark_stack

```
int ControlFlowState::endof_mark_stack[64]
```

Referenced by control_forth_case(), and control_forth_endcase().

6.39.1.7 endof_sp

```
int ControlFlowState::endof_sp
```

Referenced by control_forth_case(), control_forth_endcase(), control_forth_endof(), vm_init(), and vm_init_with_host().

6.39.1.8 leave_addrs

```
size_t ControlFlowState::leave_addrs[64]
```

Referenced by control_forth_leave(), control_forth_loop(), and control_forth_plus_loop().

6.39.1.9 leave_mark_sp

```
int ControlFlowState::leave_mark_sp
```

Referenced by control_forth_do(), control_forth_leave(), control_forth_loop(), control_forth_plus_loop(), control_forth_qdo(), vm_init(), and vm_init_with_host().

6.39.1.10 leave_mark_stack

```
int ControlFlowState::leave_mark_stack[64]
```

Referenced by control_forth_do(), control_forth_loop(), control_forth_plus_loop(), and control_forth_qdo().

6.39.1.11 leave_sp

```
int ControlFlowState::leave_sp
```

Referenced by control_forth_do(), control_forth_leave(), control_forth_loop(), control_forth_plus_loop(), control_forth_qdo(), vm_init(), and vm_init_with_host().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.40 der_span_t Struct Reference

Data Fields

- const uint8_t * p
- size_t len

6.40.1 Field Documentation

6.40.1.1 len

```
size_t der_span_t::len
```

Referenced by der_next(), x509_extract_ed25519_pubkey(), x509_extract_serial(), and x509_verify_signature().

6.40.1.2 p

```
const uint8_t* der_span_t::p
```

Referenced by der_next(), x509_extract_ed25519_pubkey(), x509_extract_serial(), and x509_verify_signature().

The documentation for this struct was generated from the following file:

- kernel/src/crypto/x509_ed25519.c

6.41 DiagEvent Struct Reference

Data Fields

- uint16_t code
- uint32_t value

6.41.1 Field Documentation

6.41.1.1 code

```
uint16_t DiagEvent::code
```

Referenced by diag_push(), and virtio_input_pop_event().

6.41.1.2 value

```
uint32_t DiagEvent::value
```

Referenced by diag_push(), and virtio_input_pop_event().

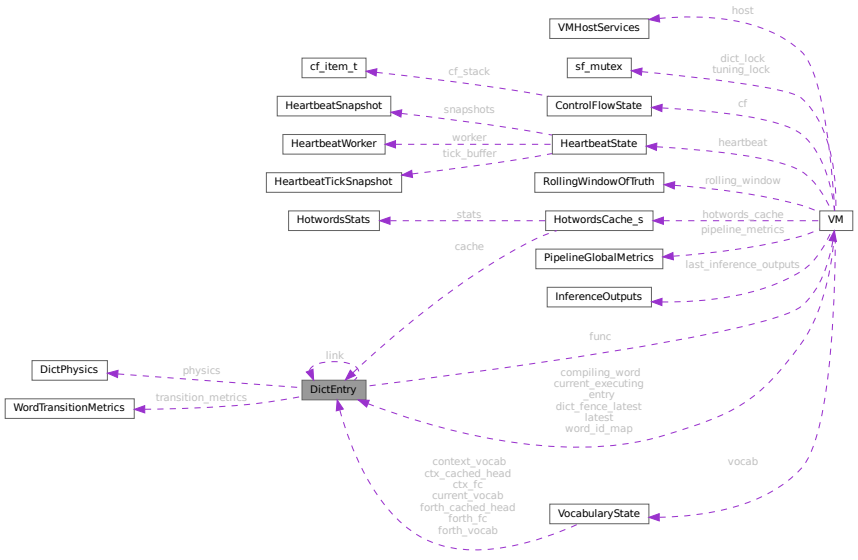
The documentation for this struct was generated from the following file:

- kernel/src/virtio/virtio_input.c

6.42 DictEntry Struct Reference

```
#include <vm.h>
```

Collaboration diagram for DictEntry:



Data Fields

- struct DictEntry * link
- word_func_t func
- cell_t flags
- uint8_t name_len
- cell_t execution_heat
- uint8_t acl_default
- uint32_t acl_ttl
- uint8_t acl_allow
- uint8_t acl_mode
- uint8_t acl_pinned
- uint32_t word_id
- DictPhysics physics
- WordTransitionMetrics * transition_metrics
- char name [ ]

6.42.1 Field Documentation

6.42.1.1 acl_allow

```
uint8_t DictEntry::acl_allow
```

Referenced by acl_inherit_entry(), dictionary_snapshot_acl_from_parent(), execute_colon_word(), forth_acl_allow_fetch(), forth_acl_allow_store(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_inherit(), test_acl_init_primitives(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_bypa and vm_exit_compile_mode().

6.42.1.2 acl_default

```
uint8_t DictEntry::acl_default
```

Referenced by sk_dict_canonical_hash().

6.42.1.3 acl_mode

```
uint8_t DictEntry::acl_mode
```

Referenced by acl_inherit_entry(), dictionary_snapshot_acl_from_parent(), forth_acl_mode_fetch(), forth_acl_mode_store(), kernel_main_deep(), test_acl_inherit(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), and test_acl_ttl_hot_path().

6.42.1.4 acl_pinned

```
uint8_t DictEntry::acl_pinned
```

Referenced by acl_inherit_entry(), dictionary_snapshot_acl_from_parent(), forth_acl_allow_store(), forth_acl_mode_store(), forth_acl_pin(), forth_acl_pinned_query(), forth_acl_ttl_store(), kernel_main_deep(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_inherit(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_blocks_shadow(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), and test_emergency_console_bypass().

6.42.1.5 acl_ttl

```
uint32_t DictEntry::acl_ttl
```

Referenced by acl_inherit_entry(), dictionary_snapshot_acl_from_parent(), execute_colon_word(), forth_acl_ttl_fetch(), forth_acl_ttl_store(), test_acl_deny_aborts(), test_acl_inherit(), test_acl_init_primitives(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_bypass(), and vm_exit_compile_mode().

6.42.1.6 execution_heat

```
cell_t DictEntry::execution_heat
```

Referenced by compare_dict_entry_heat_desc(), execute_colon_word(), forth_acl_heat_fetch(), forth_ALL_HEATS(), forth_PHYSICS_BURN(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_SHOW_FEEDBACK(), forth_PHYSICS_WORD_METR hotwords_cache_lookup(), infer_word_run(), physics_on_lookup(), physics_post_execute(), physics_pre_execute(), starforth_word_init(), test_acl_adaptive_ttl(), test_acl_read_accessors(), and vm_interpret_word().

6.42.1.7 flags

```
cell_t DictEntry::flags
```

Referenced by defining_word_immediate(), dictionary_m_word_hidden(), dictionary_word_forget(), forth_ALL_HEATS(), hotwords_cache_lookup(), sk_dict_canonical_hash(), system_word_vlist(), vm_dict_resolve_in_bucket(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_hide_word(), vm_interpret_word(), vm_make_immediate(), vm_pin_execution_heat(), vm_smudge_word(), vm_unpin_execution_heat(), vocab_find_word(), and vocabulary_word_paren_find().

6.42.1.8 func

```
word_func_t DictEntry::func
```

Referenced by acl_recheck(), acl_recheck(), control_forth_endcase(), control_forth_of(), defer_runtime(), defining_runtime_does_rt(), execute_colon_word(), forth_PHYSICS_BURN(), hash_colon_body(), sk_dict_canonical_hash(), starforth_word_init(), system_word_execute(), vm_create_word(), word_defer_fetch(), and word_is().

6.42.1.9 link

```
struct DictEntry* DictEntry::link
```

Referenced by dictionary_snapshot_acl_from_parent(), dictionary_word_forget(), find_entry_from_name(), infer_word_run(), sf_try_fast_append(), system_word_words(), vm_cleanup(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

6.42.1.10 name

```
char DictEntry::name[]
```

Referenced by defer_runtime(), defining_word_immediate(), dictionary_snapshot_acl_from_parent(), dictionary_word_forget(), execute_colon_word(), fc_rebuild(), forth_ALL_HEATS(), forth_PHYSICS_BURN(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_SHOW_FEEDBACK(), forth_PHYSICS_WORD_METRICS(), hotwords_cache_evict_name(), hotwords_cache_lookup(), sf_rebuild_fc_index(), sf_try_fast_append(), sk_dict_canonical_hash(), system_word_see(), vm_debug_dump_state(), vm_dict_resolve_in_bucket(), vm_exit_compile_mode(), vm_hide_word(), vm_make_immediate(), vm_smudge_word(), vocab_find_word(), and vocabulary_word_order().

6.42.1.11 name_len

```
uint8_t DictEntry::name_len
```

Referenced by defer_runtime(), defining_word_immediate(), dictionary_snapshot_acl_from_parent(), dictionary_word_forget(), execute_colon_word(), forth_ALL_HEATS(), hotwords_cache_evict_name(), hotwords_cache_lookup(), profiler_generate_report(), profiler_print_hotspots(), sk_dict_canonical_hash(), system_word_see(), system_word_vlist(), vm_dict_resolve_in_bucket(), vm_exit_compile_mode(), vm_hide_word(), vm_make_immediate(), vm_smudge_word(), vocab_find_word(), and vocabulary_word_order().

6.42.1.12 physics

```
DictPhysics DictEntry::physics
```

Referenced by execute_colon_word(), forth_PHYSICS_BURN(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_SHOW_FEEDBACK(), forth_PHYSICS_WORD_METRICS(), physics_on_lookup(), physics_pre_execute(), vm_interpret_word(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

6.42.1.13 transition_metrics

```
WordTransitionMetrics* DictEntry::transition_metrics
```

Referenced by dictionary_word_forget(), execute_colon_word(), physics_pre_execute(), rolling_window_seed_pipelining_context(), and vm_cleanup().

6.42.1.14 word_id

```
uint32_t DictEntry::word_id
```

Referenced by execute_colon_word(), forth_acl_word_id(), hash_colon_body(), physics_pre_execute(), sk_dict_canonical_hash(), sk_parity_collect(), test_acl_read_accessors(), vm_interpret_word(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.43 DictPhysics Struct Reference

```
#include <vm.h>
```

Data Fields

- uint16_t temperature_q8
- uint64_t last_active_ns
- uint64_t last_decay_ns
- uint64_t last_decay_tick
- uint32_t mass_bytes
- uint32_t avg_latency_ns
- uint8_t state_flags
- uint8_t acl_hint
- uint16_t pubsub_mask

6.43.1 Field Documentation

6.43.1.1 acl_hint

```
uint8_t DictPhysics::acl_hint
```

6.43.1.2 avg_latency_ns

```
uint32_t DictPhysics::avg_latency_ns
```

Referenced by forth_PHYSICS_SHOW_FEEDBACK(), and forth_PHYSICS_WORD_METRICS().

6.43.1.3 last_active_ns

```
uint64_t DictPhysics::last_active_ns
```

Referenced by execute_colon_word(), forth_PHYSICS_BURN(), forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_SHOW_FEEDBA
forth_PHYSICS_WORD_METRICS(), physics_on_lookup(), physics_pre_execute(), and vm_interpret_word().

6.43.1.4 last_decay_ns

```
uint64_t DictPhysics::last_decay_ns
```

Referenced by execute_colon_word(), physics_on_lookup(), physics_pre_execute(), vm_interpret_word(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

6.43.1.5 last_decay_tick

```
uint64_t DictPhysics::last_decay_tick
```

Referenced by execute_colon_word(), physics_on_lookup(), physics_pre_execute(), vm_interpret_word(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

6.43.1.6 mass_bytes

```
uint32_t DictPhysics::mass_bytes
```

Referenced by forth_PHYSICS_SHOW_FEEDBACK(), and forth_PHYSICS_WORD_METRICS().

6.43.1.7 pubsub_mask

```
uint16_t DictPhysics::pubsub_mask
```

6.43.1.8 state_flags

```
uint8_t DictPhysics::state_flags
```

6.43.1.9 temperature_q8

```
uint16_t DictPhysics::temperature_q8
```

Referenced by forth_PHYSICS_BURN(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_SHOW_FEEDBACK(), and forth_PHYSICS_WORD_METRICS().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.44 DoeMetrics Struct Reference

Complete metrics snapshot for a single DoE run.

```
#include <doe_metrics.h>
```

Data Fields

- uint32_t total_lookups
- uint64_t cache_hits
- double cache_hit_percent
- uint64_t bucket_hits
- double bucket_hit_percent
- int64_t cache_hit_latency_ns
- int64_t cache_hit_stddev_ns
- int64_t bucket_search_latency_ns
- int64_t bucket_search_stddev_ns
- uint64_t context_predictions_total
- uint64_t context_correct
- double context_accuracy_percent
- uint64_t cache_promotions
- uint64_t cache_demotions
- double window_diversity_percent
- uint32_t window_final_size_bytes
- uint32_t rolling_window_width
- uint32_t actual_window_size
- uint64_t total_executions
- uint64_t window_variance_q48
- double decay_slope
- uint64_t total_heat
- uint64_t hot_word_count
- uint64_t stale_word_count
- double stale_word_ratio
- double avg_word_heat
- uint64_t tick_count
- uint64_t tick_target_ns
- uint64_t inference_run_count
- uint64_t early_exit_count
- double prefetch_accuracy_percent
- uint64_t prefetch_attempts

- uint64_t prefetch_hits
- uint64_t window_tuning_checks
- uint32_t final_effective_window_size
- int64_t vm_workload_duration_ns_q48
- uint64_t total_runtime_ms
- uint64_t words_executed
- uint64_t dictionary_lookups
- uint64_t memory_allocated_bytes
- double speedup_vs_baseline
- double ci_lower_95
- double ci_upper_95
- int64_t cpu_temp_delta_c_q48
- int64_t cpu_freq_delta_mhz_q48
- uint32_t decay_rate_q16
- uint32_t decay_min_interval_ns
- uint32_t rolling_window_size
- uint32_t adaptive_shrink_rate
- uint32_t heat_cache_demotion_threshold
- int enable_loop_1_heat_tracking
- int enable_loop_2_rolling_window
- int enable_loop_3_linear_decay
- int enable_loop_4_pipelining
- int enable_loop_5_window_inference
- int enable_loop_6_decay_inference
- int enable_loop_7_adaptive_heartrate
- int enable_hotwords_cache
- int enable_pipelining

6.44.1 Detailed Description

Complete metrics snapshot for a single DoE run.

Represents all 35 metrics collected per test iteration, mirroring the CSV schema. All values extracted directly from VM structures (no string parsing).

6.44.2 Field Documentation

6.44.2.1 actual_window_size

```
uint32_t DoeMetrics::actual_window_size
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.2 adaptive_shrink_rate

```
uint32_t DoeMetrics::adaptive_shrink_rate
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.3 avg_word_heat

```
double DoeMetrics::avg_word_heat
```

Referenced by metrics_from_vm(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.4 bucket_hit_percent

```
double DoeMetrics::bucket_hit_percent
```

Referenced by extract_cache_metrics(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.5 bucket_hits

```
uint64_t DoeMetrics::bucket_hits
```

Referenced by extract_cache_metrics(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.6 bucket_search_latency_ns

```
int64_t DoeMetrics::bucket_search_latency_ns
```

Referenced by extract_cache_metrics(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.7 bucket_search_stddev_ns

```
int64_t DoeMetrics::bucket_search_stddev_ns
```

Referenced by extract_cache_metrics(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.8 cache_demotions

```
uint64_t DoeMetrics::cache_demotions
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.9 cache_hit_latency_ns

```
int64_t DoeMetrics::cache_hit_latency_ns
```

Referenced by extract_cache_metrics(), metrics_print_text(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.10 cache_hit_percent

```
double DoeMetrics::cache_hit_percent
```

Referenced by extract_cache_metrics(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.11 cache_hit_stddev_ns

```
int64_t DoeMetrics::cache_hit_stddev_ns
```

Referenced by extract_cache_metrics(), metrics_print_text(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.12 cache_hits

```
uint64_t DoeMetrics::cache_hits
```

Referenced by extract_cache_metrics(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.13 cache_promotions

```
uint64_t DoeMetrics::cache_promotions
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.14 ci_lower_95

```
double DoeMetrics::ci_lower_95
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.15 ci_upper_95

```
double DoeMetrics::ci_upper_95
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.16 context_accuracy_percent

```
double DoeMetrics::context_accuracy_percent
```

Referenced by metrics_from_vm(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.17 context_correct

```
uint64_t DoeMetrics::context_correct
```

Referenced by metrics_from_vm(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.18 context_predictions_total

```
uint64_t DoeMetrics::context_predictions_total
```

Referenced by metrics_from_vm(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.19 cpu_freq_delta_mhz_q48

```
int64_t DoeMetrics::cpu_freq_delta_mhz_q48
```

Referenced by metrics_from_vm(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.20 cpu_temp_delta_c_q48

```
int64_t DoeMetrics::cpu_temp_delta_c_q48
```

Referenced by metrics_from_vm(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.21 decay_min_interval_ns

```
uint32_t DoeMetrics::decay_min_interval_ns
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.22 decay_rate_q16

```
uint32_t DoeMetrics::decay_rate_q16
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.23 decay_slope

```
double DoeMetrics::decay_slope
```

Referenced by metrics_from_vm(), metrics_print_text(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.24 dictionary_lookups

```
uint64_t DoeMetrics::dictionary_lookups
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.25 early_exit_count

```
uint64_t DoeMetrics::early_exit_count
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.26 enable_hotwords_cache

```
int DoeMetrics::enable_hotwords_cache
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.27 enable_loop_1_heat_tracking

```
int DoeMetrics::enable_loop_1_heat_tracking
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.28 enable_loop_2_rolling_window

```
int DoeMetrics::enable_loop_2_rolling_window
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.29 enable_loop_3_linear_decay

```
int DoeMetrics::enable_loop_3_linear_decay
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.30 enable_loop_4_pipelining

```
int DoeMetrics::enable_loop_4_pipelining
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.31 enable_loop_5_window_inference

```
int DoeMetrics::enable_loop_5_window_inference
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.32 enable_loop_6_decay_inference

```
int DoeMetrics::enable_loop_6_decay_inference
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.33 enable_loop_7_adaptive_heartrate

```
int DoeMetrics::enable_loop_7_adaptive_heartrate
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.34 enable_pipelining

```
int DoeMetrics::enable_pipelining
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.35 final_effective_window_size

```
uint32_t DoeMetrics::final_effective_window_size
```

Referenced by metrics_from_vm(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.36 heat_cache_demotion_threshold

```
uint32_t DoeMetrics::heat_cache_demotion_threshold
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.37 hot_word_count

```
uint64_t DoeMetrics::hot_word_count
```

Referenced by metrics_from_vm(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.38 inference_run_count

```
uint64_t DoeMetrics::inference_run_count
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.39 memory_allocated_bytes

```
uint64_t DoeMetrics::memory_allocated_bytes
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.40 prefetch_accuracy_percent

```
double DoeMetrics::prefetch_accuracy_percent
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.41 prefetch_attempts

```
uint64_t DoeMetrics::prefetch_attempts
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.42 prefetch_hits

```
uint64_t DoeMetrics::prefetch_hits
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.43 rolling_window_size

```
uint32_t DoeMetrics::rolling_window_size
```

Referenced by metrics_from_vm(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.44 rolling_window_width

```
uint32_t DoeMetrics::rolling_window_width
```

Referenced by metrics_from_vm(), metrics_print_text(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.45 speedup_vs_baseline

```
double DoeMetrics::speedup_vs_baseline
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.46 stale_word_count

```
uint64_t DoeMetrics::stale_word_count
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.47 stale_word_ratio

```
double DoeMetrics::stale_word_ratio
```

Referenced by metrics_from_vm(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.48 tick_count

```
uint64_t DoeMetrics::tick_count
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.49 tick_target_ns

```
uint64_t DoeMetrics::tick_target_ns
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.50 total_executions

```
uint64_t DoeMetrics::total_executions
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.51 total_heat

```
uint64_t DoeMetrics::total_heat
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.52 total_lookups

```
uint32_t DoeMetrics::total_lookups
```

Referenced by metrics_from_vm(), metrics_print_text(), and metrics_write_csv_row().

6.44.2.53 total_runtime_ms

```
uint64_t DoeMetrics::total_runtime_ms
```

Referenced by metrics_from_vm(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.54 vm_workload_duration_ns_q48

```
int64_t DoeMetrics::vm_workload_duration_ns_q48
```

Referenced by metrics_from_vm(), metrics_print_text(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.55 window_diversity_percent

```
double DoeMetrics::window_diversity_percent
```

Referenced by metrics_from_vm(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

6.44.2.56 window_final_size_bytes

```
uint32_t DoeMetrics::window_final_size_bytes
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.57 window_tuning_checks

```
uint64_t DoeMetrics::window_tuning_checks
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.58 window_variance_q48

```
uint64_t DoeMetrics::window_variance_q48
```

Referenced by metrics_from_vm(), and metrics_write_csv_row().

6.44.2.59 words_executed

```
uint64_t DoeMetrics::words_executed
```

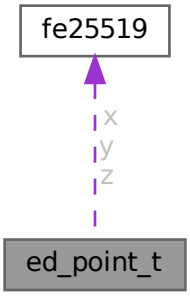
Referenced by metrics_from_vm(), metrics_write_csv_row(), and metrics_write_james_law_csv_row().

The documentation for this struct was generated from the following file:

- v3/include/doe_metrics.h

6.45 ed_point_t Struct Reference

Collaboration diagram for ed_point_t:



Data Fields

- fe25519 x
- fe25519 y
- fe25519 z

6.45.1 Field Documentation

6.45.1.1 x

```
fe25519 ed_point_t::x
```

Referenced by base_point(), ed25519_verify(), point_add(), point_compress(), point_decompress(), and point_identity().

6.45.1.2 y

```
fe25519 ed_point_t::y
```

Referenced by base_point(), ed25519_verify(), point_add(), point_compress(), point_decompress(), and point_identity().

6.45.1.3 z

```
fe25519 ed_point_t::z
```

Referenced by base_point(), ed25519_verify(), point_add(), point_compress(), point_decompress(), and point_identity().

The documentation for this struct was generated from the following file:

- kernel/src/crypto/ed25519.c

6.46 EFI_BLOCK_IO_MEDIA Struct Reference

```
#include <uefi.h>
```

Data Fields

- UINT32 MediaId
- BOOLEAN RemovableMedia
- BOOLEAN MediaPresent
- BOOLEAN LogicalPartition
- BOOLEAN ReadOnly
- BOOLEAN WriteCaching
- UINT32 BlockSize
- UINT32 IoAlign
- UINT64 LastBlock

6.46.1 Field Documentation

6.46.1.1 BlockSize

```
UINT32 EFI_BLOCK_IO_MEDIA::BlockSize
```

6.46.1.2 IoAlign

```
UINT32 EFI_BLOCK_IO_MEDIA::IoAlign
```

6.46.1.3 LastBlock

```
UINT64 EFI_BLOCK_IO_MEDIA::LastBlock
```

6.46.1.4 LogicalPartition

```
BOOLEAN EFI_BLOCK_IO_MEDIA::LogicalPartition
```

6.46.1.5 MediaId

```
UINT32 EFI_BLOCK_IO_MEDIA::MediaId
```

6.46.1.6 MediaPresent

```
BOOLEAN EFI_BLOCK_IO_MEDIA::MediaPresent
```

6.46.1.7 ReadOnly

```
BOOLEAN EFI_BLOCK_IO_MEDIA::ReadOnly
```

6.46.1.8 RemovableMedia

```
BOOLEAN EFI_BLOCK_IO_MEDIA::RemovableMedia
```

6.46.1.9 WriteCaching

```
BOOLEAN EFI_BLOCK_IO_MEDIA::WriteCaching
```

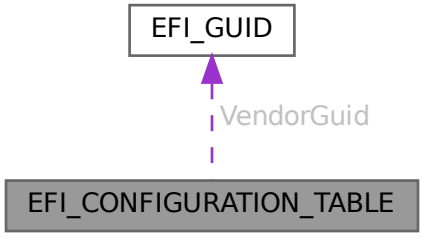
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.47 EFI_CONFIGURATION_TABLE Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for EFI_CONFIGURATION_TABLE:



Data Fields

- EFI_GUID VendorGuid
- void * VendorTable

6.47.1 Field Documentation

6.47.1.1 VendorGuid

```
EFI_GUID EFI_CONFIGURATION_TABLE::VendorGuid
```

6.47.1.2 VendorTable

```
void* EFI_CONFIGURATION_TABLE::VendorTable
```

Referenced by efi_main().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.48 EFI_DEVICE_PATH_PROTOCOL Struct Reference

```
#include <uefi.h>
```

Data Fields

- UINT8 Type
- UINT8 SubType
- UINT8 Length [2]

6.48.1 Field Documentation

6.48.1.1 Length

```
UINT8 EFI_DEVICE_PATH_PROTOCOL::Length[2]
```

6.48.1.2 SubType

```
UINT8 EFI_DEVICE_PATH_PROTOCOL::SubType
```

6.48.1.3 Type

```
UINT8 EFI_DEVICE_PATH_PROTOCOL::Type
```

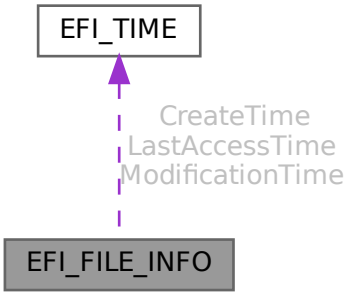
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.49 EFI_FILE_INFO Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for EFI_FILE_INFO:



Data Fields

- UINT64 Size
- UINT64 FileSize
- UINT64 PhysicalSize
- EFI_TIME CreateTime
- EFI_TIME LastAccessTime
- EFI_TIME ModificationTime
- UINT64 Attribute
- CHAR16 FileName [256]

6.49.1 Field Documentation

6.49.1.1 Attribute

```
UINT64 EFI_FILE_INFO::Attribute
```

6.49.1.2 CreateTime

```
EFI_TIME EFI_FILE_INFO::CreateTime
```

6.49.1.3 FileName

```
CHAR16 EFI_FILE_INFO::FileName[256]
```

6.49.1.4 FileSize

```
UINT64 EFI_FILE_INFO::FileSize
```

Referenced by load_kernel_from_esp().

6.49.1.5 LastAccessTime

```
EFI_TIME EFI_FILE_INFO::LastAccessTime
```

6.49.1.6 ModificationTime

```
EFI_TIME EFI_FILE_INFO::ModificationTime
```

6.49.1.7 PhysicalSize

```
UINT64 EFI_FILE_INFO::PhysicalSize
```

6.49.1.8 Size

```
UINT64 EFI_FILE_INFO::Size
```

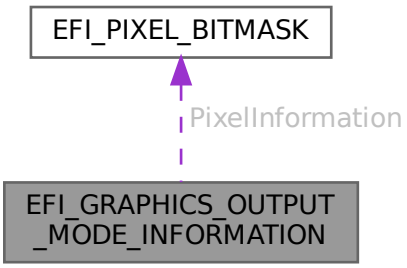
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.50 EFI_GRAPHICS_OUTPUT_MODE_INFORMATION Struct Reference

`#include <uefi.h>`

Collaboration diagram for EFI_GRAPHICS_OUTPUT_MODE_INFORMATION:



Data Fields

- UINT32 Version
- UINT32 HorizontalResolution
- UINT32 VerticalResolution
- EFI_GRAPHICS_PIXEL_FORMAT PixelFormat
- EFI_PIXEL_BITMASK PixelInformation
- UINT32 PixelsPerScanLine

6.50.1 Field Documentation

6.50.1.1 HorizontalResolution

```
UINT32 EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::HorizontalResolution
```

Referenced by efi_main().

6.50.1.2 PixelFormat

```
EFI_GRAPHICS_PIXEL_FORMAT EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::PixelFormat
```

Referenced by efi_main().

6.50.1.3 PixelInformation

```
EFI_PIXEL_BITMASK EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::PixelInformation
```

6.50.1.4 PixelsPerScanLine

```
UINT32 EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::PixelsPerScanLine
```

Referenced by efi_main().

6.50.1.5 Version

```
UINT32 EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::Version
```

6.50.1.6 VerticalResolution

```
UINT32 EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::VerticalResolution
```

Referenced by efi_main().

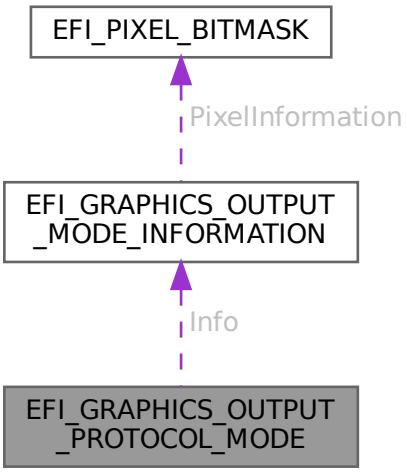
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.51 EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE Struct Reference

```
#include <uefi.h>
```

Collaboration diagram for EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE:



Data Fields

- UINT32 MaxMode
- UINT32 Mode
- EFI_GRAPHICS_OUTPUT_MODE_INFORMATION * Info
- UINTN SizeOfInfo
- EFI_PHYSICAL_ADDRESS FrameBufferBase
- UINTN FrameBufferSize

6.51.1 Field Documentation

6.51.1.1 FrameBufferBase

```
EFI_PHYSICAL_ADDRESS EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::FrameBufferBase
```

Referenced by efi_main().

6.51.1.2 FrameBufferSize

```
UINTN EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::FrameBufferSize
```

Referenced by efi_main().

6.51.1.3 Info

```
EFI_GRAPHICS_OUTPUT_MODE_INFORMATION* EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::Info
```

Referenced by efi_main().

6.51.1.4 MaxMode

```
UINT32 EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::MaxMode
```

6.51.1.5 Mode

```
UINT32 EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::Mode
```

6.51.1.6 SizeOfInfo

```
UINTN EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::SizeOfInfo
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.52 EFI_GUID Struct Reference

`#include <uefi.h>`

Data Fields

- UINT32 Data1
- UINT16 Data2
- UINT16 Data3
- UINT8 Data4 [8]

6.52.1 Field Documentation

6.52.1.1 Data1

`UINT32 EFI_GUID::Data1`

Referenced by guid_equals().

6.52.1.2 Data2

`UINT16 EFI_GUID::Data2`

Referenced by guid_equals().

6.52.1.3 Data3

`UINT16 EFI_GUID::Data3`

Referenced by guid_equals().

6.52.1.4 Data4

`UINT8 EFI_GUID::Data4[8]`

Referenced by guid_equals().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.53 EFI_LOADED_IMAGE_PROTOCOL Struct Reference

`#include <uefi.h>`

Data Fields

- UINT32 Revision
- EFI_HANDLE ParentHandle
- void * SystemTable
- EFI_HANDLE DeviceHandle
- VOID * FilePath
- VOID * Reserved
- UINT32 LoadOptionsSize
- VOID * LoadOptions
- VOID * ImageBase
- UINT64 ImageSize
- EFI_MEMORY_TYPE ImageCodeType
- EFI_MEMORY_TYPE ImageDataType
- VOID * Unload

6.53.1 Field Documentation

6.53.1.1 DeviceHandle

```
EFI_HANDLE EFI_LOADED_IMAGE_PROTOCOL::DeviceHandle
```

Referenced by load_cfg_from_esp(), and load_kernel_from_esp().

6.53.1.2 FilePath

```
VOID* EFI_LOADED_IMAGE_PROTOCOL::FilePath
```

6.53.1.3 ImageBase

```
VOID* EFI_LOADED_IMAGE_PROTOCOL::ImageBase
```

6.53.1.4 ImageCodeType

```
EFI_MEMORY_TYPE EFI_LOADED_IMAGE_PROTOCOL::ImageCodeType
```

6.53.1.5 ImageDataType

```
EFI_MEMORY_TYPE EFI_LOADED_IMAGE_PROTOCOL::ImageDataType
```

6.53.1.6 ImageSize

```
UINT64 EFI_LOADED_IMAGE_PROTOCOL::ImageSize
```

6.53.1.7 LoadOptions

```
VOID* EFI_LOADED_IMAGE_PROTOCOL::LoadOptions
```

Referenced by efi_main().

6.53.1.8 LoadOptionsSize

```
UINT32 EFI_LOADED_IMAGE_PROTOCOL::LoadOptionsSize
```

Referenced by efi_main().

6.53.1.9 ParentHandle

```
EFI_HANDLE EFI_LOADED_IMAGE_PROTOCOL::ParentHandle
```

6.53.1.10 Reserved

```
VOID* EFI_LOADED_IMAGE_PROTOCOL::Reserved
```

6.53.1.11 Revision

```
UINT32 EFI_LOADED_IMAGE_PROTOCOL::Revision
```

6.53.1.12 SystemTable

```
void* EFI_LOADED_IMAGE_PROTOCOL::SystemTable
```

6.53.1.13 Unload

```
VOID* EFI_LOADED_IMAGE_PROTOCOL::Unload
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.54 EFI_MEMORY_DESCRIPTOR Struct Reference

```
#include <uefi.h>
```

Data Fields

- UINT32 Type
- UINT64 PhysicalStart
- UINT64 VirtualStart
- UINT64 NumberOfPages
- UINT64 Attribute

6.54.1 Field Documentation

6.54.1.1 Attribute

```
UINT64 EFI_MEMORY_DESCRIPTOR::Attribute
```

Referenced by add_memmap_entry().

6.54.1.2 NumberOfPages

```
UINT64 EFI_MEMORY_DESCRIPTOR::NumberOfPages
```

Referenced by add_memmap_entry(), pmm_init(), print_boot_info(), and vmm_init().

6.54.1.3 PhysicalStart

```
UINT64 EFI_MEMORY_DESCRIPTOR::PhysicalStart
```

Referenced by add_memmap_entry(), pmm_init(), and vmm_init().

6.54.1.4 Type

```
UINT32 EFI_MEMORY_DESCRIPTOR::Type
```

Referenced by add_memmap_entry(), pmm_init(), print_boot_info(), and vmm_init().

6.54.1.5 VirtualStart

```
UINT64 EFI_MEMORY_DESCRIPTOR::VirtualStart
```

Referenced by add_memmap_entry().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.55 EFI_PIXEL_BITMASK Struct Reference

```
#include <uefi.h>
```

Data Fields

- UINT32 RedMask
- UINT32 GreenMask
- UINT32 BlueMask
- UINT32 ReservedMask

6.55.1 Field Documentation

6.55.1.1 BlueMask

```
UINT32 EFI_PIXEL_BITMASK::BlueMask
```

6.55.1.2 GreenMask

```
UINT32 EFI_PIXEL_BITMASK::GreenMask
```

6.55.1.3 RedMask

```
UINT32 EFI_PIXEL_BITMASK::RedMask
```

6.55.1.4 ReservedMask

```
UINT32 EFI_PIXEL_BITMASK::ReservedMask
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.56 EFI_TABLE_HEADER Struct Reference

```
#include <uefi.h>
```

Data Fields

- UINT64 Signature
- UINT32 Revision
- UINT32 HeaderSize
- UINT32 CRC32
- UINT32 Reserved

6.56.1 Field Documentation

6.56.1.1 CRC32

```
UINT32 EFI_TABLE_HEADER::CRC32
```

6.56.1.2 HeaderSize

```
UINT32 EFI_TABLE_HEADER::HeaderSize
```

6.56.1.3 Reserved

```
UINT32 EFI_TABLE_HEADER::Reserved
```

6.56.1.4 Revision

```
UINT32 EFI_TABLE_HEADER::Revision
```

6.56.1.5 Signature

```
UINT64 EFI_TABLE_HEADER::Signature
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.57 EFI_TIME Struct Reference

```
#include <uefi.h>
```

Data Fields

- UINT16 Year
- UINT8 Month
- UINT8 Day
- UINT8 Hour
- UINT8 Minute
- UINT8 Second
- UINT8 Pad1
- UINT32 Nanosecond
- INT16 TimeZone
- UINT8 Daylight
- UINT8 Pad2

6.57.1 Field Documentation

6.57.1.1 Day

```
UINT8 EFI_TIME::Day
```

6.57.1.2 Daylight

```
UINT8 EFI_TIME::Daylight
```

6.57.1.3 Hour

```
UINT8 EFI_TIME::Hour
```

6.57.1.4 Minute

```
UINT8 EFI_TIME::Minute
```

6.57.1.5 Month

```
UINT8 EFI_TIME::Month
```

6.57.1.6 Nanosecond

```
UINT32 EFI_TIME::Nanosecond
```

6.57.1.7 Pad1

```
UINT8 EFI_TIME::Pad1
```

6.57.1.8 Pad2

```
UINT8 EFI_TIME::Pad2
```

6.57.1.9 Second

```
UINT8 EFI_TIME::Second
```

6.57.1.10 TimeZone

```
INT16 EFI_TIME::TimeZone
```

6.57.1.11 Year

```
UINT16 EFI_TIME::Year
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.58 Elf64_Ehdr Struct Reference

```
#include <elf64.h>
```

Data Fields

- unsigned char e_ident [16]
- Elf64_Half e_type
- Elf64_Half e_machine
- Elf64_Word e_version
- Elf64_Addr e_entry
- Elf64_Off e_phoff
- Elf64_Off e_shoff
- Elf64_Word e_flags
- Elf64_Half e_ehsize
- Elf64_Half e_phentsize
- Elf64_Half e_phnum
- Elf64_Half e_shentsize
- Elf64_Half e_shnum
- Elf64_Half e_shstrndx

6.58.1 Field Documentation

6.58.1.1 e_ehsize

```
Elf64_Half Elf64_Ehdr::e_ehsize
```

6.58.1.2 e_entry

```
Elf64_Addr Elf64_Ehdr::e_entry
```

Referenced by elf_load_kernel().

6.58.1.3 e_flags

```
Elf64_Word Elf64_Ehdr::e_flags
```

6.58.1.4 e_ident

```
unsigned char Elf64_Ehdr::e_ident[16]
```

Referenced by elf_validate_header().

6.58.1.5 e_machine

```
Elf64_Half Elf64_Ehdr::e_machine
```

Referenced by elf_validate_header().

6.58.1.6 e_phentsize

```
Elf64_Half Elf64_Ehdr::e_phentsize
```

6.58.1.7 e_phnum

```
Elf64_Half Elf64_Ehdr::e_phnum
```

Referenced by elf_load_segments().

6.58.1.8 e_phoff

```
Elf64_Off Elf64_Ehdr::e_phoff
```

Referenced by elf_load_segments().

6.58.1.9 e_shentsize

```
Elf64_Half Elf64_Ehdr::e_shentsize
```

6.58.1.10 e_shnum

```
Elf64_Half Elf64_Ehdr::e_shnum
```

Referenced by elf_apply_relocations().

6.58.1.11 e_shoff

```
Elf64_Off Elf64_Ehdr::e_shoff
```

Referenced by elf_apply_relocations().

6.58.1.12 e_shstrndx

```
Elf64_Half Elf64_Ehdr::e_shstrndx
```

6.58.1.13 e_type

```
Elf64_Half Elf64_Ehdr::e_type
```

Referenced by elf_load_kernel(), and elf_validate_header().

6.58.1.14 e_version

```
Elf64_Word Elf64_Ehdr::e_version
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/elf64.h

6.59 Elf64_Phdr Struct Reference

```
#include <elf64.h>
```

Data Fields

- Elf64_Word p_type
- Elf64_Word p_flags
- Elf64_Off p_offset
- Elf64_Addr p_vaddr
- Elf64_Addr p_paddr
- Elf64_Xword p_filesz
- Elf64_Xword p_memsz
- Elf64_Xword p_align

6.59.1 Field Documentation

6.59.1.1 p_align

```
Elf64_Xword Elf64_Phdr::p_align
```

6.59.1.2 p_filesz

```
Elf64_Xword Elf64_Phdr::p_filesz
```

6.59.1.3 p_flags

```
Elf64_Word Elf64_Phdr::p_flags
```

6.59.1.4 p_memsz

```
Elf64_Xword Elf64_Phdr::p_memsz
```

6.59.1.5 p_offset

```
Elf64_Off Elf64_Phdr::p_offset
```

Referenced by elf_load_segments().

6.59.1.6 p_paddr

```
Elf64_Addr Elf64_Phdr::p_paddr
```

6.59.1.7 p_type

```
Elf64_Word Elf64_Phdr::p_type
```

6.59.1.8 p_vaddr

```
Elf64_Addr Elf64_Phdr::p_vaddr
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/elf64.h

6.60 Elf64_Rel Struct Reference

```
#include <elf64.h>
```

Data Fields

- Elf64_Addr r_offset
- Elf64_Xword r_info

6.60.1 Field Documentation

6.60.1.1 r_info

```
Elf64_Xword Elf64_Rel::r_info
```

6.60.1.2 r_offset

```
Elf64_Addr Elf64_Rel::r_offset
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/elf64.h

6.61 Elf64_Rela Struct Reference

```
#include <elf64.h>
```

Data Fields

- Elf64_Addr r_offset
- Elf64_Xword r_info
- Elf64_Sxword r_addend

6.61.1 Field Documentation

6.61.1.1 r_addend

```
Elf64_Sxword Elf64_Rela::r_addend
```

Referenced by elf_apply_relocations().

6.61.1.2 r_info

```
Elf64_Xword Elf64_Rela::r_info
```

6.61.1.3 r_offset

```
Elf64_Addr Elf64_Rela::r_offset
```

Referenced by elf_apply_relocations().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/elf64.h

6.62 Elf64_Shdr Struct Reference

```
#include <elf64.h>
```

Data Fields

- Elf64_Word sh_name
- Elf64_Word sh_type
- Elf64_Xword sh_flags
- Elf64_Addr sh_addr
- Elf64_Off sh_offset
- Elf64_Xword sh_size
- Elf64_Word sh_link
- Elf64_Word sh_info
- Elf64_Xword sh_addralign
- Elf64_Xword sh_entsize

6.62.1 Field Documentation

6.62.1.1 sh_addr

```
Elf64_Addr Elf64_Shdr::sh_addr
```

6.62.1.2 sh_addralign

```
Elf64_Xword Elf64_Shdr::sh_addralign
```

6.62.1.3 sh_entsize

```
Elf64_Xword Elf64_Shdr::sh_entsize
```

Referenced by elf_apply_relocations().

6.62.1.4 sh_flags

```
Elf64_Xword Elf64_Shdr::sh_flags
```

6.62.1.5 sh_info

```
Elf64_Word Elf64_Shdr::sh_info
```

6.62.1.6 sh_link

```
Elf64_Word Elf64_Shdr::sh_link
```

Referenced by elf_apply_relocations().

6.62.1.7 sh_name

```
Elf64_Word Elf64_Shdr::sh_name
```

6.62.1.8 sh_offset

```
Elf64_Off Elf64_Shdr::sh_offset
```

Referenced by elf_apply_relocations().

6.62.1.9 sh_size

```
Elf64_Xword Elf64_Shdr::sh_size
```

Referenced by elf_apply_relocations().

6.62.1.10 sh_type

```
Elf64_Word Elf64_Shdr::sh_type
```

Referenced by elf_apply_relocations().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/elf64.h

6.63 Elf64_Sym Struct Reference

```
#include <elf64.h>
```

Data Fields

- Elf64_Word st_name
- unsigned char st_info
- unsigned char st_other
- Elf64_Half st_shndx
- Elf64_Addr st_value
- Elf64_Xword st_size

6.63.1 Field Documentation

6.63.1.1 st_info

```
unsigned char Elf64_Sym::st_info
```

6.63.1.2 st_name

```
Elf64_Word Elf64_Sym::st_name
```

6.63.1.3 st_other

```
unsigned char Elf64_Sym::st_other
```

6.63.1.4 st_shndx

```
Elf64_Half Elf64_Sym::st_shndx
```

6.63.1.5 st_size

```
Elf64_Xword Elf64_Sym::st_size
```

6.63.1.6 st_value

```
Elf64_Addr Elf64_Sym::st_value
```

Referenced by elf_apply_relocations().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/elf64.h

6.64 ExecResult Struct Reference

```
#include <test_contracts.h>
```

Data Fields

- cell_t stack [1024]
- int dsp
- int rsp
- int error

6.64.1 Field Documentation

6.64.1.1 dsp

```
int ExecResult::dsp
```

Referenced by assert_heartbeat_safe(), capture_exec_result(), check_physics_transparent(), and exec_results_equal().

6.64.1.2 error

```
int ExecResult::error
```

Referenced by capture_exec_result(), and exec_results_equal().

6.64.1.3 rsp

```
int ExecResult::rsp
```

Referenced by assert_heartbeat_safe(), capture_exec_result(), and exec_results_equal().

6.64.1.4 stack

```
cell_t ExecResult::stack[1024]
```

Referenced by capture_exec_result(), and exec_results_equal().

The documentation for this struct was generated from the following file:

- v3/src/test_runner/include/test_contracts.h

6.65 FbState Struct Reference

Data Fields

- volatile uint32_t * base
- uint32_t stride
- uint32_t width
- uint32_t height
- FbPixelFormat fmt
- uint32_t scale
- int ready

6.65.1 Field Documentation

6.65.1.1 base

```
volatile uint32_t* FbState::base
```

Referenced by fb_draw_glyph(), fb_fill_rect(), fb_init(), fb_put_pixel(), fb_scroll_rect(), and fb_scroll_rows().

6.65.1.2 fmt

```
FbPixelFormat FbState::fmt
```

Referenced by fb_init(), and pack_pixel().

6.65.1.3 height

```
uint32_t FbState::height
```

Referenced by fb_draw_glyph(), fb_draw_orientation_test(), fb_fill_rect(), fb_height(), fb_init(), fb_put_pixel(), fb_scroll_rect(), and fb_scroll_rows().

6.65.1.4 ready

```
int FbState::ready
```

Referenced by fb_draw_glyph(), fb_draw_orientation_test(), fb_fill_rect(), fb_init(), fb_is_available(), fb_put_pixel(), fb_scroll_rect(), and fb_scroll_rows().

6.65.1.5 scale

```
uint32_t FbState::scale
```

Referenced by fb_cell_h(), fb_cell_w(), fb_draw_glyph(), and fb_init().

6.65.1.6 stride

```
uint32_t FbState::stride
```

Referenced by fb_draw_glyph(), fb_fill_rect(), fb_init(), fb_put_pixel(), fb_scroll_rect(), and fb_scroll_rows().

6.65.1.7 width

```
uint32_t FbState::width
```

Referenced by fb_draw_glyph(), fb_draw_orientation_test(), fb_fill_rect(), fb_init(), fb_put_pixel(), fb_scroll_rect(), fb_scroll_rows(), and fb_width().

The documentation for this struct was generated from the following file:

- kernel/src/hal/framebuffer.c

6.66 fdt_header_t Struct Reference

Data Fields

- uint32_t magic
- uint32_t totalsize
- uint32_t off_dt_struct
- uint32_t off_dt_strings
- uint32_t off_mem_rsvmap
- uint32_t version
- uint32_t last_comp_version
- uint32_t boot_cpuid_phys
- uint32_t size_dt_strings
- uint32_t size_dt_struct

6.66.1 Field Documentation

6.66.1.1 boot_cpuid_phys

```
uint32_t fdt_header_t::boot_cpuid_phys
```

6.66.1.2 last_comp_version

```
uint32_t fdt_header_t::last_comp_version
```

6.66.1.3 magic

```
uint32_t fdt_header_t::magic
```

Referenced by fdt_valid().

6.66.1.4 off_dt_strings

```
uint32_t fdt_header_t::off_dt_strings
```

Referenced by fdt_find_node_by_compatible(), fdt_find_node_by_device_type(), fdt_find_prop(), fdt_find_prop_in_node(), and fdt_valid().

6.66.1.5 off_dt_struct

```
uint32_t fdt_header_t::off_dt_struct
```

Referenced by fdt_complete(), fdt_find_child(), fdt_find_node_by_compatible(), fdt_find_node_by_device_type(), fdt_find_node_by_name(), fdt_find_prop(), fdt_find_prop_in_node(), fdt_next_child_node(), fdt_root_node(), and fdt_valid().

6.66.1.6 off_mem_rsvmap

```
uint32_t fdt_header_t::off_mem_rsvmap
```

6.66.1.7 size_dt_strings

```
uint32_t fdt_header_t::size_dt_strings
```

Referenced by fdt_valid().

6.66.1.8 size_dt_struct

```
uint32_t fdt_header_t::size_dt_struct
```

Referenced by fdt_complete(), fdt_find_child(), fdt_find_node_by_compatible(), fdt_find_node_by_device_type(), fdt_find_node_by_name(), fdt_find_prop(), fdt_find_prop_in_node(), fdt_next_child_node(), fdt_root_node(), and fdt_valid().

6.66.1.9 totalsize

```
uint32_t fdt_header_t::totalsize
```

Referenced by fdt_valid().

6.66.1.10 version

```
uint32_t fdt_header_t::version
```

The documentation for this struct was generated from the following file:

- kernel/src/hal/fdt.c

6.67 fdt_uart_t Struct Reference

```
#include <fdt_uart.h>
```

Data Fields

- int kind
  FDT_UART_∗.
- uint64_t base
  address of register 0, as the CPU sees it
- uint32_t reg_shift
  register n is at base + (n << reg_shift)
- uint32_t io_width
  bytes read or written at a time: 1 or 4

6.67.1 Field Documentation

6.67.1.1 base

```
uint64_t fdt_uart_t::base
```

address of register 0, as the CPU sees it

Referenced by console_attach_serial(), and fdt_console_uart().

6.67.1.2 io_width

```
uint32_t fdt_uart_t::io_width
```

bytes read or written at a time: 1 or 4

Referenced by console_attach_serial(), and fdt_console_uart().

6.67.1.3 kind

```
int fdt_uart_t::kind
```

FDT_UART_∗.

Referenced by console_attach_serial(), and fdt_console_uart().

6.67.1.4 reg_shift

```
uint32_t fdt_uart_t::reg_shift
```

register n is at base + (n << reg_shift)

Referenced by console_attach_serial(), and fdt_console_uart().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/fdt_uart.h

6.68 fe25519 Struct Reference

```
#include <fe25519.h>
```

Data Fields

- int64_t v [5]

6.68.1 Field Documentation

6.68.1.1 v

```
int64_t fe25519::v[5]
```

Referenced by fe25519_0(), fe25519_1(), fe25519_add(), fe25519_carry(), fe25519_mul(), fe25519_mul_small(), fe25519_pack(), fe25519_sub(), and fe25519_unpack().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/fe25519.h

6.69 FILE Struct Reference

```
#include <stdio.h>
```

Data Fields

- int _dummy

6.69.1 Field Documentation

6.69.1.1 _dummy

```
int FILE::_dummy
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/freestanding/stdio.h

6.70 FramebufferInfo Struct Reference

```
#include <uefi.h>
```

Data Fields

- void * base
- UINTN size
- UINT32 width
- UINT32 height
- UINT32 pixels_per_scanline
- UINT32 pixel_format

6.70.1 Field Documentation

6.70.1.1 base

```
void* FramebufferInfo::base
```

Referenced by console_fb_init(), efi_main(), fb_init(), rpi5_native_boot(), and vmm_init().

6.70.1.2 height

```
UINT32 FramebufferInfo::height
```

Referenced by efi_main(), fb_init(), and rpi5_native_boot().

6.70.1.3 pixel_format

```
UINT32 FramebufferInfo::pixel_format
```

Referenced by efi_main(), and rpi5_native_boot().

6.70.1.4 pixels_per_scanline

```
UINT32 FramebufferInfo::pixels_per_scanline
```

Referenced by efi_main(), fb_init(), and rpi5_native_boot().

6.70.1.5 size

```
UINTN FramebufferInfo::size
```

Referenced by efi_main(), rpi5_native_boot(), and vmm_init().

6.70.1.6 width

```
UINT32 FramebufferInfo::width
```

Referenced by efi_main(), fb_init(), and rpi5_native_boot().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/uefi.h

6.71 gdt_entry Struct Reference

arch.c (amd64) - Architecture abstractions for StarKernel on x86_64

Data Fields

- uint16_t limit_low
- uint16_t base_low
- uint8_t base_mid
- uint8_t access
- uint8_t granularity
- uint8_t base_high

6.71.1 Detailed Description

arch.c (amd64) - Architecture abstractions for StarKernel on x86_64

6.71.2 Field Documentation

6.71.2.1 access

```
uint8_t gdt_entry::access
```

6.71.2.2 base_high

```
uint8_t gdt_entry::base_high
```

6.71.2.3 base_low

```
uint16_t gdt_entry::base_low
```

6.71.2.4 base_mid

```
uint8_t gdt_entry::base_mid
```

6.71.2.5 granularity

```
uint8_t gdt_entry::granularity
```

6.71.2.6 limit_low

```
uint16_t gdt_entry::limit_low
```

The documentation for this struct was generated from the following file:

- kernel/src/arch/amd64/arch.c

6.72 gdtr Struct Reference

Data Fields

- uint16_t limit
- uint64_t base

6.72.1 Field Documentation

6.72.1.1 base

```
uint64_t gdtr::base
```

Referenced by gdt_init().

6.72.1.2 limit

```
uint16_t gdtr::limit
```

Referenced by gdt_init().

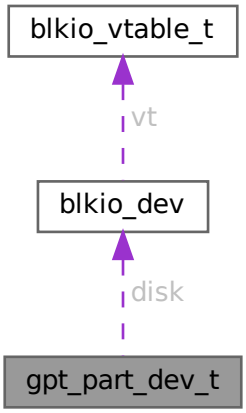
The documentation for this struct was generated from the following file:

- kernel/src/arch/amd64/arch.c

6.73 gpt_part_dev_t Struct Reference

```
#include <gpt.h>
```

Collaboration diagram for gpt_part_dev_t:



Data Fields

- blkio_dev_t * disk
- uint32_t first_fblock
- uint32_t fblocks

6.73.1 Field Documentation

6.73.1.1 disk

```
blkio_dev_t* gpt_part_dev_t::disk
```

Referenced by gpt_part_open(), part_flush(), part_info(), part_read(), and part_write().

6.73.1.2 fblocks

```
uint32_t gpt_part_dev_t::fblocks
```

Referenced by gpt_part_open(), part_info(), part_open(), part_read(), and part_write().

6.73.1.3 first_fblock

```
uint32_t gpt_part_dev_t::first_fblock
```

Referenced by gpt_part_open(), part_read(), and part_write().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/gpt.h

6.74 gpt_part_t Struct Reference

```
#include <gpt.h>
```

Data Fields

- uint8_t type [16]
- uint64_t first_lba
- uint64_t last_lba

6.74.1 Field Documentation

6.74.1.1 first_lba

```
uint64_t gpt_part_t::first_lba
```

Referenced by gpt_part_open(), gpt_read(), and sk_v4_grow_stick().

6.74.1.2 last_lba

```
uint64_t gpt_part_t::last_lba
```

Referenced by gpt_part_open(), and gpt_read().

6.74.1.3 type

```
uint8_t gpt_part_t::type[16]
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/gpt.h

6.75 heap_block Struct Reference

Collaboration diagram for heap_block:

Data Fields

- size_t size
- size_t requested
- int free
- struct heap_block * next
- struct heap_block * prev

6.75.1 Field Documentation

6.75.1.1 free

```
int heap_block::free
```

Referenced by allocate_from_block(), coalesce_neighbors(), kfree(), kmalloc_aligned(), kmalloc_debug_census(), kmalloc_debug_census_bytes(), and kmalloc_init().

6.75.1.2 next

```
struct heap_block* heap_block::next
```

Referenced by allocate_from_block(), coalesce_neighbors(), kmalloc_aligned(), kmalloc_debug_census(), kmalloc_debug_census_bytes(), and kmalloc_init().

6.75.1.3 prev

```
struct heap_block* heap_block::prev
```

Referenced by allocate_from_block(), coalesce_neighbors(), and kmalloc_init().

6.75.1.4 requested

```
size_t heap_block::requested
```

Referenced by allocate_from_block(), kfree(), and kmalloc_init().

6.75.1.5 size

```
size_t heap_block::size
```

Referenced by allocate_from_block(), coalesce_neighbors(), kmalloc_debug_census(), kmalloc_debug_census_bytes(), and kmalloc_init().

The documentation for this struct was generated from the following file:

- kernel/src/memory/kmalloc.c

6.76 HeartbeatSnapshot Struct Reference

```
#include <vm.h>
```

Data Fields

- uint64_t published_tick
- uint64_t published_ns
- uint32_t window_width
- uint32_t actual_window_size
- uint64_t decay_slope_q48
- uint64_t hot_word_count
- uint64_t stale_word_count
- uint64_t total_heat

6.76.1 Field Documentation

6.76.1.1 actual_window_size

```
uint32_t HeartbeatSnapshot::actual_window_size
```

Referenced by vm_heartbeat_publish_snapshot().

6.76.1.2 decay_slope_q48

```
uint64_t HeartbeatSnapshot::decay_slope_q48
```

Referenced by heartbeat_publish_snapshot(), and vm_heartbeat_publish_snapshot().

6.76.1.3 hot_word_count

```
uint64_t HeartbeatSnapshot::hot_word_count
```

Referenced by heartbeat_publish_snapshot(), and vm_heartbeat_publish_snapshot().

6.76.1.4 published_ns

```
uint64_t HeartbeatSnapshot::published_ns
```

Referenced by heartbeat_publish_snapshot(), and vm_heartbeat_publish_snapshot().

6.76.1.5 published_tick

```
uint64_t HeartbeatSnapshot::published_tick
```

Referenced by heartbeat_publish_snapshot(), and vm_heartbeat_publish_snapshot().

6.76.1.6 stale_word_count

```
uint64_t HeartbeatSnapshot::stale_word_count
```

Referenced by heartbeat_publish_snapshot(), and vm_heartbeat_publish_snapshot().

6.76.1.7 total_heat

```
uint64_t HeartbeatSnapshot::total_heat
```

Referenced by heartbeat_publish_snapshot(), and vm_heartbeat_publish_snapshot().

6.76.1.8 window_width

```
uint32_t HeartbeatSnapshot::window_width
```

Referenced by heartbeat_publish_snapshot(), and vm_heartbeat_publish_snapshot().

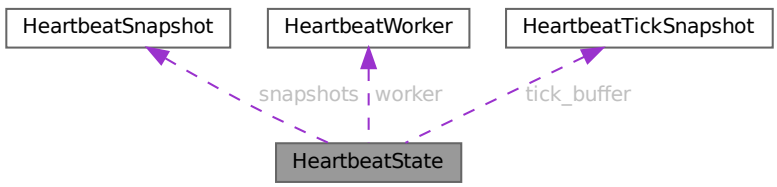
The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.77 HeartbeatState Struct Reference

```
#include <vm.h>
```

Collaboration diagram for HeartbeatState:



Data Fields

- uint64_t tick_count
- uint64_t last_inference_tick
- uint32_t check_counter
- int heartbeat_enabled
- uint64_t tick_target_ns
- volatile uint32_t snapshot_index
- HeartbeatSnapshot snapshots [2]
- HeartbeatWorker * worker
- uint64_t inference_run_count
- uint64_t early_exit_count
- uint64_t words_executed
- uint64_t dictionary_lookups
- HeartbeatTickSnapshot * tick_buffer
- uint32_t tick_buffer_size
- uint64_t tick_buffer_write_index
- uint64_t tick_count_total
- uint64_t run_start_ns
- uint32_t tick_number_offset
- double bucket_sum_K
- double bucket_sum_K_squared
- double bucket_sum_window_variance
- double bucket_sum_heat_variance
- uint64_t bucket_mode_transitions
- uint32_t bucket_collapse_flag
- uint32_t bucket_tick_count
- uint64_t m5_time_trust
- uint64_t m5_variance

6.77.1 Field Documentation

6.77.1.1 bucket_collapse_flag

```
uint32_t HeartbeatState::bucket_collapse_flag
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.2 bucket_mode_transitions

```
uint64_t HeartbeatState::bucket_mode_transitions
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.3 bucket_sum_heat_variance

```
double HeartbeatState::bucket_sum_heat_variance
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.4 bucket_sum_K

```
double HeartbeatState::bucket_sum_K
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.5 bucket_sum_K_squared

```
double HeartbeatState::bucket_sum_K_squared
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.6 bucket_sum_window_variance

```
double HeartbeatState::bucket_sum_window_variance
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.7 bucket_tick_count

```
uint32_t HeartbeatState::bucket_tick_count
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.8 check_counter

```
uint32_t HeartbeatState::check_counter
```

Referenced by execute_colon_word(), physics_post_execute(), vm_init(), vm_init_with_host(), and vm_interpret().

6.77.1.9 dictionary_lookups

```
uint64_t HeartbeatState::dictionary_lookups
```

Referenced by metrics_from_vm(), and vm_find_word().

6.77.1.10 early_exit_count

```
uint64_t HeartbeatState::early_exit_count
```

Referenced by heartbeat_capture_tick_snapshot(), metrics_from_vm(), and vm_tick_inference_engine().

6.77.1.11 heartbeat_enabled

```
int HeartbeatState::heartbeat_enabled
```

Referenced by vm_heartbeat_run_cycle(), vm_init(), vm_init_with_host(), vm_tick(), and vm_tick_inference_engine().

6.77.1.12 inference_run_count

```
uint64_t HeartbeatState::inference_run_count
```

Referenced by metrics_from_vm(), and vm_tick_inference_engine().

6.77.1.13 last_inference_tick

```
uint64_t HeartbeatState::last_inference_tick
```

Referenced by vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_tick(), and vm_tick_inference_engine().

6.77.1.14 m5_time_trust

```
uint64_t HeartbeatState::m5_time_trust
```

6.77.1.15 m5_variance

```
uint64_t HeartbeatState::m5_variance
```

6.77.1.16 run_start_ns

```
uint64_t HeartbeatState::run_start_ns
```

6.77.1.17 snapshot_index

```
volatile uint32_t HeartbeatState::snapshot_index
```

Referenced by heartbeat_publish_snapshot(), vm_heartbeat_publish_snapshot(), vm_init(), vm_init_with_host(), and vm_snapshot_read().

6.77.1.18 snapshots

```
HeartbeatSnapshot HeartbeatState::snapshots[2]
```

Referenced by heartbeat_publish_snapshot(), vm_heartbeat_publish_snapshot(), and vm_snapshot_read().

6.77.1.19 tick_buffer

```
HeartbeatTickSnapshot* HeartbeatState::tick_buffer
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.20 tick_buffer_size

```
uint32_t HeartbeatState::tick_buffer_size
```

6.77.1.21 tick_buffer_write_index

```
uint64_t HeartbeatState::tick_buffer_write_index
```

Referenced by metrics_write_james_law_csv_row().

6.77.1.22 tick_count

```
uint64_t HeartbeatState::tick_count
```

Referenced by block_word_block(), block_word_buffer(), block_word_update(), execute_colon_word(), heartbeat_capture_tick_snapsh heartbeat_publish_snapshot(), metrics_from_vm(), physics_on_lookup(), physics_pre_execute(), starforth_word_heartbeat_ticks(), vm_heartbeat_publish_snapshot(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_interpret_word(), vm_tick(), vm_tick_apply_background_decay(), vm_tick_apply_background_decay(), and vm_tick_inference_engine().

6.77.1.23 tick_count_total

```
uint64_t HeartbeatState::tick_count_total
```

6.77.1.24 tick_number_offset

```
uint32_t HeartbeatState::tick_number_offset
```

6.77.1.25 tick_target_ns

```
uint64_t HeartbeatState::tick_target_ns
```

Referenced by metrics_from_vm(), perturb_physics(), restore_physics(), save_physics(), vm_init(), vm_init_with_host(), and vm_tick_inference_engine().

6.77.1.26 words_executed

```
uint64_t HeartbeatState::words_executed
```

Referenced by execute_colon_word(), heartbeat_capture_tick_snapshot(), metrics_from_vm(), physics_post_execute(), and vm_interpret_word().

6.77.1.27 worker

```
HeartbeatWorker* HeartbeatState::worker
```

Referenced by execute_colon_word(), heartbeat_thread_main(), physics_post_execute(), vm_cleanup(), vm_heartbeat_start_thread(), vm_heartbeat_stop_thread(), vm_init(), vm_init_with_host(), vm_interpret(), and vm_tick_inference_engine().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.78 HeartbeatTickSnapshot Struct Reference

```
#include <vm.h>
```

Data Fields

- uint32_t tick_number
- uint64_t elapsed_ns
- uint64_t tick_interval_ns
- uint32_t cache_hits_delta
- uint32_t bucket_hits_delta
- uint32_t word_executions_delta
- uint64_t hot_word_count
- double avg_word_heat
- uint32_t window_width
- uint32_t actual_window_size
- uint32_t predicted_label_hits
- double estimated_jitter_ns
- uint8_t l8_mode

6.78.1 Field Documentation

6.78.1.1 actual_window_size

```
uint32_t HeartbeatTickSnapshot::actual_window_size
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.2 avg_word_heat

```
double HeartbeatTickSnapshot::avg_word_heat
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.3 bucket_hits_delta

```
uint32_t HeartbeatTickSnapshot::bucket_hits_delta
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.4 cache_hits_delta

```
uint32_t HeartbeatTickSnapshot::cache_hits_delta
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.5 elapsed_ns

```
uint64_t HeartbeatTickSnapshot::elapsed_ns
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.6 estimated_jitter_ns

```
double HeartbeatTickSnapshot::estimated_jitter_ns
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.7 hot_word_count

```
uint64_t HeartbeatTickSnapshot::hot_word_count
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.8 l8_mode

```
uint8_t HeartbeatTickSnapshot::l8_mode
```

Referenced by metrics_write_james_law_csv_row().

6.78.1.9 predicted_label_hits

```
uint32_t HeartbeatTickSnapshot::predicted_label_hits
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.10 tick_interval_ns

```
uint64_t HeartbeatTickSnapshot::tick_interval_ns
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.11 tick_number

```
uint32_t HeartbeatTickSnapshot::tick_number
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.12 window_width

```
uint32_t HeartbeatTickSnapshot::window_width
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

6.78.1.13 word_executions_delta

```
uint32_t HeartbeatTickSnapshot::word_executions_delta
```

Referenced by doe_log_tick_row(), heartbeat_capture_tick_snapshot(), and heartbeat_emit_tick_row().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.79 HeartbeatWorker Struct Reference

```
#include <vm_internal.h>
```

Data Fields

- uint64_t tick_ns
- int running
- int stop_requested

6.79.1 Field Documentation

6.79.1.1 running

```
int HeartbeatWorker::running
```

Referenced by heartbeat_thread_main().

6.79.1.2 stop_requested

```
int HeartbeatWorker::stop_requested
```

Referenced by heartbeat_thread_main(), vm_cleanup(), and vm_heartbeat_stop_thread().

6.79.1.3 tick_ns

```
uint64_t HeartbeatWorker::tick_ns
```

Referenced by heartbeat_thread_main(), vm_heartbeat_start_thread(), vm_init_with_host(), and vm_tick_inference_engine().

The documentation for this struct was generated from the following files:

- v3/src/vm_internal.h
- kernel/src/vm/vm_internal.h

6.80 homeblocks_sig_t Struct Reference

```
#include <homeblocks_sig.h>
```

Data Fields

- uint64_t magic
- uint8_t drive_uuid [16]
- uint64_t minted_time_ns
- uint64_t metadata_devblocks
- uint32_t cert_offset
- uint32_t cert_devblocks
- uint32_t identity_src_offset
- uint32_t identity_src_devblocks
- uint64_t hdr_crc
- uint8_t _pad [4096 -(8+16+8+8+4+4+4+4+8)]

6.80.1 Field Documentation

6.80.1.1 _pad

```
uint8_t homeblocks_sig_t::_pad[4096 -( 8+ 16+ 8+ 8+ 4+4+ 4+4+ 8)]
```

6.80.1.2 cert_devblocks

```
uint32_t homeblocks_sig_t::cert_devblocks
```

Referenced by capsule_mint_identity(), capsule_wirebind_try_attach(), and capsule_wirebind_verify_cert().

6.80.1.3 cert_offset

```
uint32_t homeblocks_sig_t::cert_offset
```

Referenced by capsule_mint_identity(), capsule_wirebind_try_attach(), and capsule_wirebind_verify_cert().

6.80.1.4 drive_uuid

```
uint8_t homeblocks_sig_t::drive_uuid[16]
```

Referenced by capsule_mint_identity(), and capsule_wirebind_verify_cert().

6.80.1.5 hdr_crc

```
uint64_t homeblocks_sig_t::hdr_crc
```

Referenced by capsule_mint_identity(), and homeblocks_sig_check().

6.80.1.6 identity_src_devblocks

```
uint32_t homeblocks_sig_t::identity_src_devblocks
```

Referenced by capsule_mint_identity(), capsule_runcap_birth(), capsule_wirebind_try_attach(), and capsule_zuse_boot_check_key().

6.80.1.7 identity_src_offset

```
uint32_t homeblocks_sig_t::identity_src_offset
```

Referenced by capsule_mint_identity(), capsule_runcap_birth(), capsule_wirebind_try_attach(), and capsule_zuse_boot_check_key().

6.80.1.8 magic

```
uint64_t homeblocks_sig_t::magic
```

Referenced by capsule_mint_identity(), and homeblocks_sig_check().

6.80.1.9 metadata_devblocks

```
uint64_t homeblocks_sig_t::metadata_devblocks
```

Referenced by capsule_mint_identity().

6.80.1.10 minted_time_ns

```
uint64_t homeblocks_sig_t::minted_time_ns
```

Referenced by capsule_mint_identity().

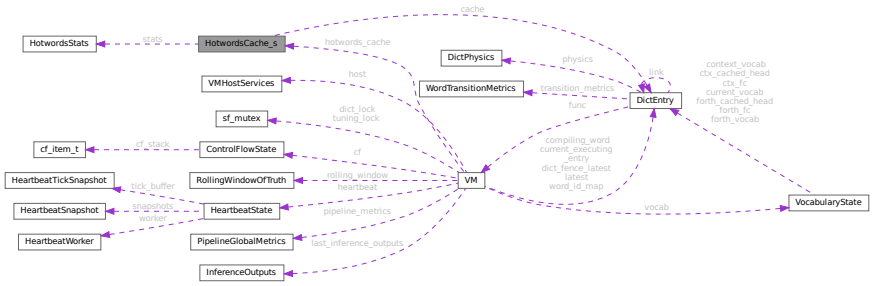
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/homeblocks_sig.h

6.81 HotwordsCache_s Struct Reference

`#include <physics_hotwords_cache.h>`

Collaboration diagram for HotwordsCache_s:



Data Fields

- DictEntry * cache [32]

  Cache entries (pointers to DictEntry)
- uint32_t cache_count

  Number of valid entries in cache.
- uint32_t lru_index

  LRU: last-used index for eviction.
- HotwordsStats stats

  Performance statistics.
- int enabled

  Enabled/disabled flag (runtime toggle)

6.81.1 Field Documentation

6.81.1.1 cache

```
DictEntry* HotwordsCache_s::cache[32]
```

Cache entries (pointers to DictEntry)

Referenced by forth_PHYSICS_CACHE_STATS(), hotwords_cache_evict_entry(), hotwords_cache_evict_name(), hotwords_cache_lookup(), hotwords_cache_promote(), and hotwords_cache_remove_at().

6.81.1.2 cache_count

```
uint32_t HotwordsCache_s::cache_count
```

Number of valid entries in cache.

Referenced by forth_PHYSICS_BUILD_INFO(), forth_PHYSICS_CACHE_STATS(), hotwords_cache_evict_entry(), hotwords_cache_evict_name(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_cache_promote(), and hotwords_cache_remove_at().

6.81.1.3 enabled

```
int HotwordsCache_s::enabled
```

Enabled/disabled flag (runtime toggle)

Referenced by forth_PHYSICS_BUILD_INFO(), forth_PHYSICS_TOGGLE_CACHE(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_cache_set_enabled(), metrics_from_vm(), and physics_pre_execute().

6.81.1.4 lru_index

```
uint32_t HotwordsCache_s::lru_index
```

LRU: last-used index for eviction.

Referenced by hotwords_cache_init(), hotwords_cache_promote(), and hotwords_cache_remove_at().

6.81.1.5 stats

```
HotwordsStats HotwordsCache_s::stats
```

Performance statistics.

Referenced by bucket_posterior(), cache_posterior(), extract_cache_metrics(), forth_BENCH_DICT_LOOKUP(), forth_PHYSICS_BAYESIAN_REPORT(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_RESET_STATS(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_cache_promote(), and metrics_from_vm().

The documentation for this struct was generated from the following file:

- v3/include/physics_hotwords_cache.h

6.82 HotwordsStats Struct Reference

```
#include <physics_hotwords_cache.h>
```

Data Fields

- uint64_t total_lookups

  Total lookups performed.
- uint64_t cache_hits

  Lookups that hit the hot-words cache.
- uint64_t bucket_hits

  Lookups that missed cache but found in bucket.
- uint64_t misses

  Lookups that failed (word not found)
- uint64_t evictions

  Words evicted from cache (LRU)
- uint64_t promotions

  Times a word was promoted to hot-words.
- uint64_t bucket_reorders

  Times bucket was reordered.
- int64_t cache_hit_total_ns_q48

  === 64-bit Fixed-Point Statistics (Q48.16 format) === Q48.16 = 48-bit integer + 16-bit fractional part Precision: 2^-16 ns ≈ 0.0000153 ns Range: ±140 trillion ns (≈ 4.4 years) Allows: Bayesian inference, precise averaging, statistical computation
- int64_t bucket_search_total_ns_q48

  Sum of bucket search latencies (64-bit fixed-point, Q48.16)
- uint64_t cache_hit_samples

  Number of cache hit samples (for averaging)
- uint64_t bucket_search_samples

  Number of bucket search samples (for averaging)
- int64_t min_cache_hit_ns

  Minimum cache hit latency (nanoseconds)
- int64_t max_cache_hit_ns

  Maximum cache hit latency (nanoseconds)
- int64_t min_bucket_search_ns

  Minimum bucket search latency (nanoseconds)
- int64_t max_bucket_search_ns

  Maximum bucket search latency (nanoseconds)
- int64_t cache_hit_variance_sum_q48

  Sum of squared cache hit latencies (for variance/stddev)
- int64_t bucket_search_variance_sum_q48

  Sum of squared bucket search latencies (for variance/stddev)

6.82.1 Field Documentation

6.82.1.1 bucket_hits

```
uint64_t HotwordsStats::bucket_hits
```

Lookups that missed cache but found in bucket.

Referenced by extract_cache_metrics(), forth_PHYSICS_BAYESIAN_REPORT(), hotwords_bayesian_report(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.2 bucket_reorders

```
uint64_t HotwordsStats::bucket_reorders
```

Times bucket was reordered.

Referenced by hotwords_cache_init(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.3 bucket_search_samples

```
uint64_t HotwordsStats::bucket_search_samples
```

Number of bucket search samples (for averaging)

Referenced by extract_cache_metrics(), forth_PHYSICS_BAYESIAN_REPORT(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_bucket_searches(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.4 bucket_search_total_ns_q48

```
int64_t HotwordsStats::bucket_search_total_ns_q48
```

Sum of bucket search latencies (64-bit fixed-point, Q48.16)

Referenced by extract_cache_metrics(), forth_PHYSICS_BAYESIAN_REPORT(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_bucket_searches(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.5 bucket_search_variance_sum_q48

```
int64_t HotwordsStats::bucket_search_variance_sum_q48
```

Sum of squared bucket search latencies (for variance/stddev)

Referenced by extract_cache_metrics(), forth_PHYSICS_BAYESIAN_REPORT(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_bucket_searches(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.6 cache_hit_samples

```
uint64_t HotwordsStats::cache_hit_samples
```

Number of cache hit samples (for averaging)

Referenced by extract_cache_metrics(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_cache_hits(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.7 cache_hit_total_ns_q48

```
int64_t HotwordsStats::cache_hit_total_ns_q48
```

=== 64-bit Fixed-Point Statistics (Q48.16 format) === Q48.16 = 48-bit integer + 16-bit fractional part Precision↩ : 2^-16 ns ≈ 0.0000153 ns Range: ±140 trillion ns (≈ 4.4 years) Allows: Bayesian inference, precise averaging, statistical computation

Sum of cache hit latencies (64-bit fixed-point, Q48.16)

Referenced by extract_cache_metrics(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_cache_hits(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.8 cache_hit_variance_sum_q48

```
int64_t HotwordsStats::cache_hit_variance_sum_q48
```

Sum of squared cache hit latencies (for variance/stddev)

Referenced by extract_cache_metrics(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_cache_hits(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.9 cache_hits

```
uint64_t HotwordsStats::cache_hits
```

Lookups that hit the hot-words cache.

Referenced by extract_cache_metrics(), forth_PHYSICS_BAYESIAN_REPORT(), hotwords_bayesian_report(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.10 evictions

```
uint64_t HotwordsStats::evictions
```

Words evicted from cache (LRU)

Referenced by hotwords_cache_init(), hotwords_cache_promote(), hotwords_stats_print(), hotwords_stats_reset(), and metrics_from_vm().

6.82.1.11 max_bucket_search_ns

```
int64_t HotwordsStats::max_bucket_search_ns
```

Maximum bucket search latency (nanoseconds)

Referenced by forth_PHYSICS_BAYESIAN_REPORT(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_bucket_searches(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.12 max_cache_hit_ns

```
int64_t HotwordsStats::max_cache_hit_ns
```

Maximum cache hit latency (nanoseconds)

Referenced by hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_cache_hits(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.13 min_bucket_search_ns

```
int64_t HotwordsStats::min_bucket_search_ns
```

Minimum bucket search latency (nanoseconds)

Referenced by forth_PHYSICS_BAYESIAN_REPORT(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_bucket_searches(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.14 min_cache_hit_ns

```
int64_t HotwordsStats::min_cache_hit_ns
```

Minimum cache hit latency (nanoseconds)

Referenced by hotwords_cache_init(), hotwords_cache_lookup(), hotwords_posterior_cache_hits(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.15 misses

```
uint64_t HotwordsStats::misses
```

Lookups that failed (word not found)

Referenced by hotwords_bayesian_report(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_stats_print(), and hotwords_stats_reset().

6.82.1.16 promotions

```
uint64_t HotwordsStats::promotions
```

Times a word was promoted to hot-words.

Referenced by hotwords_cache_init(), hotwords_cache_promote(), hotwords_stats_print(), hotwords_stats_reset(), and metrics_from_vm().

6.82.1.17 total_lookups

```
uint64_t HotwordsStats::total_lookups
```

Total lookups performed.

Referenced by extract_cache_metrics(), hotwords_cache_init(), hotwords_cache_lookup(), hotwords_stats_reset(), and metrics_from_vm().

The documentation for this struct was generated from the following file:

- v3/include/physics_hotwords_cache.h

6.83 idt_entry Struct Reference

Data Fields

- uint16_t offset_low
- uint16_t selector
- uint8_t ist
- uint8_t type_attr
- uint16_t offset_mid
- uint32_t offset_high
- uint32_t zero

6.83.1 Field Documentation

6.83.1.1 ist

```
uint8_t idt_entry::ist
```

Referenced by set_idt_entry().

6.83.1.2 offset_high

```
uint32_t idt_entry::offset_high
```

Referenced by set_idt_entry().

6.83.1.3 offset_low

```
uint16_t idt_entry::offset_low
```

Referenced by set_idt_entry().

6.83.1.4 offset_mid

```
uint16_t idt_entry::offset_mid
```

Referenced by set_idt_entry().

6.83.1.5 selector

```
uint16_t idt_entry::selector
```

Referenced by set_idt_entry().

6.83.1.6 type_attr

```
uint8_t idt_entry::type_attr
```

Referenced by set_idt_entry().

6.83.1.7 zero

```
uint32_t idt_entry::zero
```

Referenced by set_idt_entry().

The documentation for this struct was generated from the following file:

- kernel/src/arch/amd64/interrupts.c

6.84 idtr Struct Reference

Data Fields

- uint16_t limit
- uint64_t base

6.84.1 Field Documentation

6.84.1.1 base

```
uint64_t idtr::base
```

6.84.1.2 limit

```
uint16_t idtr::limit
```

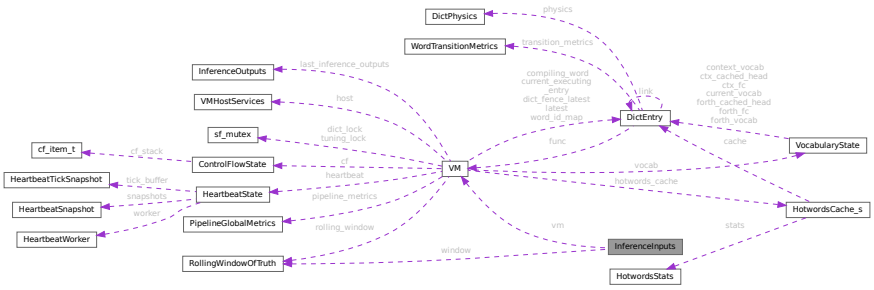
The documentation for this struct was generated from the following file:

- kernel/src/arch/amd64/interrupts.c

6.85 InferenceInputs Struct Reference

```
#include <inference_engine.h>
```

Collaboration diagram for InferenceInputs:



Data Fields

- VM * vm
- RollingWindowOfTruth * window
- uint64_t trajectory_length
- uint64_t prefetch_hits
- uint64_t prefetch_attempts
- uint64_t hot_word_count
- uint64_t stale_word_count
- uint64_t total_heat
- uint32_t word_count
- uint64_t last_total_heat
- uint64_t last_stale_count

6.85.1 Field Documentation

6.85.1.1 hot_word_count

```
uint64_t InferenceInputs::hot_word_count
```

Referenced by infer_word_run().

6.85.1.2 last_stale_count

```
uint64_t InferenceInputs::last_stale_count
```

Referenced by infer_word_run().

6.85.1.3 last_total_heat

```
uint64_t InferenceInputs::last_total_heat
```

Referenced by infer_word_run().

6.85.1.4 prefetch_attempts

```
uint64_t InferenceInputs::prefetch_attempts
```

Referenced by infer_word_run().

6.85.1.5 prefetch_hits

```
uint64_t InferenceInputs::prefetch_hits
```

Referenced by infer_word_run().

6.85.1.6 stale_word_count

```
uint64_t InferenceInputs::stale_word_count
```

Referenced by infer_word_run().

6.85.1.7 total_heat

```
uint64_t InferenceInputs::total_heat
```

Referenced by infer_word_run().

6.85.1.8 trajectory_length

```
uint64_t InferenceInputs::trajectory_length
```

Referenced by infer_word_run().

6.85.1.9 vm

```
VM* InferenceInputs::vm
```

Referenced by infer_word_run(), and inference_engine_run().

6.85.1.10 window

```
RollingWindowOfTruth* InferenceInputs::window
```

Referenced by infer_word_run(), inference_engine_run(), and vm_tick_inference_engine().

6.85.1.11 word_count

```
uint32_t InferenceInputs::word_count
```

Referenced by infer_word_run().

The documentation for this struct was generated from the following file:

- v3/include/inference_engine.h

6.86 InferenceOutputs Struct Reference

```
#include <inference_engine.h>
```

Data Fields

- uint32_t adaptive_window_width
- uint64_t adaptive_decay_slope
- uint64_t window_variance_q48
- uint64_t slope_fit_quality_q48
- uint32_t early_exited
- uint64_t last_check_tick

6.86.1 Field Documentation

6.86.1.1 adaptive_decay_slope

```
uint64_t InferenceOutputs::adaptive_decay_slope
```

Referenced by infer_word_decay_fetch(), inference_engine_run(), inference_outputs_to_string(), inference_outputs_validate(), and vm_tick_inference_engine().

6.86.1.2 adaptive_window_width

```
uint32_t InferenceOutputs::adaptive_window_width
```

Referenced by infer_word_window_fetch(), inference_engine_run(), inference_outputs_to_string(), inference_outputs_validate(), and vm_tick_inference_engine().

6.86.1.3 early_exited

```
uint32_t InferenceOutputs::early_exited
```

Referenced by infer_word_early_exit_fetch(), inference_engine_run(), inference_outputs_to_string(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), and vm_tick_inference_engine().

6.86.1.4 last_check_tick

```
uint64_t InferenceOutputs::last_check_tick
```

6.86.1.5 slope_fit_quality_q48

```
uint64_t InferenceOutputs::slope_fit_quality_q48
```

Referenced by infer_word_fit_fetch(), inference_engine_run(), inference_outputs_to_string(), inference_outputs_validate(), and vm_tick_inference_engine().

6.86.1.6 window_variance_q48

```
uint64_t InferenceOutputs::window_variance_q48
```

Referenced by infer_word_variance_fetch(), inference_engine_run(), inference_outputs_to_string(), metrics_from_vm(), and vm_tick_inference_engine().

The documentation for this struct was generated from the following file:

- v3/include/inference_engine.h

6.87 KernelArgs Struct Reference

```
#include <kernel_args.h>
```

Data Fields

- uint64_t stack_size
- uint64_t heap_size
- int run_doe
- int log_level
- int system
- int screen_boot
- char raw [512]

6.87.1 Field Documentation

6.87.1.1 heap_size

```
uint64_t KernelArgs::heap_size
```

Referenced by cmdline_parse_ascii(), and kernel_main().

6.87.1.2 log_level

```
int KernelArgs::log_level
```

Referenced by cmdline_parse_ascii().

6.87.1.3 raw

```
char KernelArgs::raw[512]
```

Referenced by cmdline_parse_ascii().

6.87.1.4 run_doe

```
int KernelArgs::run_doe
```

Referenced by cmdline_parse_ascii(), and kernel_main_deep().

6.87.1.5 screen_boot

```
int KernelArgs::screen_boot
```

Referenced by cmdline_parse_ascii().

6.87.1.6 stack_size

```
uint64_t KernelArgs::stack_size
```

Referenced by cmdline_parse_ascii(), and efi_main().

6.87.1.7 system

```
int KernelArgs::system
```

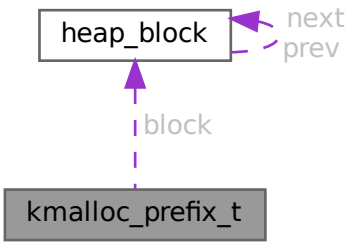
Referenced by cmdline_parse_ascii(), and kernel_main_deep().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/kernel_args.h

6.88 kmalloc_prefix_t Struct Reference

Collaboration diagram for kmalloc_prefix_t:



Data Fields

- uint32_t magic
- heap_block_t * block

6.88.1 Field Documentation

6.88.1.1 block

```
heap_block_t* kmalloc_prefix_t::block
```

Referenced by allocate_from_block(), and kfree().

6.88.1.2 magic

```
uint32_t kmalloc_prefix_t::magic
```

Referenced by allocate_from_block(), and kfree().

The documentation for this struct was generated from the following file:

- kernel/src/memory/kmalloc.c

6.89 kmalloc_stats_t Struct Reference

kmalloc.h - Kernel heap allocator interface

```
#include <kmalloc.h>
```

Data Fields

- uint64_t total_bytes
- uint64_t used_bytes
- uint64_t peak_bytes
- uint64_t free_bytes

6.89.1 Detailed Description

kmalloc.h - Kernel heap allocator interface

6.89.2 Field Documentation

6.89.2.1 free_bytes

```
uint64_t kmalloc_stats_t::free_bytes
```

Referenced by kmalloc_init(), print_heap_stats(), and update_free_bytes().

6.89.2.2 peak_bytes

```
uint64_t kmalloc_stats_t::peak_bytes
```

Referenced by account_allocation(), kmalloc_init(), print_heap_stats(), and sf_alloc_get_stats().

6.89.2.3 total_bytes

```
uint64_t kmalloc_stats_t::total_bytes
```

Referenced by kmalloc_init(), print_heap_stats(), sf_alloc_get_stats(), and update_free_bytes().

6.89.2.4 used_bytes

```
uint64_t kmalloc_stats_t::used_bytes
```

Referenced by account_allocation(), account_free(), kmalloc_init(), print_heap_stats(), sf_alloc_get_stats(), and update_free_bytes().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/kmalloc.h

6.90 log_region_ctrl_t Struct Reference

```
#include <log_region.h>
```

Data Fields

- uint64_t magic
- uint32_t devblocks
- uint32_t head_slot
- uint32_t tail_slot
- uint32_t record_count
- uint64_t hdr_crc
- uint8_t _pad [4096 -(8+4+4+4+4+8)]

6.90.1 Field Documentation

6.90.1.1 _pad

```
uint8_t log_region_ctrl_t::_pad[4096 -(8+4+4+4+4+8)]
```

6.90.1.2 devblocks

```
uint32_t log_region_ctrl_t::devblocks
```

Referenced by log_region_append(), log_region_ctrl_init_fresh(), and log_region_read_recent().

6.90.1.3 hdr_crc

```
uint64_t log_region_ctrl_t::hdr_crc
```

Referenced by log_region_ctrl_read(), and log_region_ctrl_write().

6.90.1.4 head_slot

```
uint32_t log_region_ctrl_t::head_slot
```

Referenced by log_region_append(), and log_region_ctrl_init_fresh().

6.90.1.5 magic

```
uint64_t log_region_ctrl_t::magic
```

Referenced by log_region_ctrl_init_fresh(), and log_region_ctrl_read().

6.90.1.6 record_count

```
uint32_t log_region_ctrl_t::record_count
```

Referenced by log_region_append(), log_region_ctrl_init_fresh(), and log_region_read_recent().

6.90.1.7 tail_slot

```
uint32_t log_region_ctrl_t::tail_slot
```

Referenced by log_region_append(), log_region_ctrl_init_fresh(), and log_region_read_recent().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/log_region.h

6.91 log_slot_t Struct Reference

```
#include <log_region.h>
```

Data Fields

- uint64_t timestamp
- uint16_t msg_len
- uint8_t level
- char source [16u]
- char msg [(1024u - 8u - 2u - 1u - 16u)]

6.91.1 Field Documentation

6.91.1.1 level

```
uint8_t log_slot_t::level
```

Referenced by log_region_append().

6.91.1.2 msg

```
char log_slot_t::msg[(1024u - 8u - 2u - 1u - 16u)]
```

Referenced by log_region_append().

6.91.1.3 msg_len

```
uint16_t log_slot_t::msg_len
```

Referenced by log_region_append().

6.91.1.4 source

```
char log_slot_t::source[16u]
```

Referenced by log_region_append().

6.91.1.5 timestamp

```
uint64_t log_slot_t::timestamp
```

Referenced by log_region_append().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/log_region.h

6.92 MemoryProfile Struct Reference

```
#include <profiler.h>
```

Data Fields

- uint64_t reads
- uint64_t writes
- uint64_t cache_hits
- uint64_t cache_misses
- uint64_t total_bytes_read
- uint64_t total_bytes_written

6.92.1 Field Documentation

6.92.1.1 cache_hits

```
uint64_t MemoryProfile::cache_hits
```

6.92.1.2 cache_misses

```
uint64_t MemoryProfile::cache_misses
```

6.92.1.3 reads

```
uint64_t MemoryProfile::reads
```

6.92.1.4 total_bytes_read

```
uint64_t MemoryProfile::total_bytes_read
```

6.92.1.5 total_bytes_written

```
uint64_t MemoryProfile::total_bytes_written
```

6.92.1.6 writes

```
uint64_t MemoryProfile::writes
```

The documentation for this struct was generated from the following file:

- v3/include/profiler.h

6.93 ParityPacket Struct Reference

ParityPacket - Summary of VM state for comparison.

```
#include <parity.h>
```

Data Fields

- uint32_t word_count
- uint32_t here_offset
- uint32_t latest_word_id
- uint64_t header_hash64
- uint32_t tests_total
- uint32_t tests_passed
- uint32_t tests_failed
- uint32_t tests_skipped
- uint32_t tests_errors
- uint64_t window_hash64
- int bootstrap_result

6.93.1 Detailed Description

ParityPacket - Summary of VM state for comparison.

M7.1a fields are sufficient for bootstrap validation. M7.1b fields are added for POST validation.

6.93.2 Field Documentation

6.93.2.1 bootstrap_result

```
int ParityPacket::bootstrap_result
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.2 header_hash64

```
uint64_t ParityPacket::header_hash64
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.3 here_offset

```
uint32_t ParityPacket::here_offset
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.4 latest_word_id

```
uint32_t ParityPacket::latest_word_id
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.5 tests_errors

```
uint32_t ParityPacket::tests_errors
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.6 tests_failed

```
uint32_t ParityPacket::tests_failed
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.7 tests_passed

```
uint32_t ParityPacket::tests_passed
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.8 tests_skipped

```
uint32_t ParityPacket::tests_skipped
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.9 tests_total

```
uint32_t ParityPacket::tests_total
```

Referenced by sk_parity_collect(), and sk_parity_print().

6.93.2.10 window_hash64

```
uint64_t ParityPacket::window_hash64
```

Referenced by sk_parity_collect().

6.93.2.11 word_count

```
uint32_t ParityPacket::word_count
```

Referenced by sk_parity_collect(), and sk_parity_print().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/vm/parity.h

6.94 PciDevice Struct Reference

```
#include <pci.h>
```

Data Fields

- uint8_t bus
- uint8_t device
- uint8_t function
- uint16_t vendor_id
- uint16_t device_id

6.94.1 Field Documentation

6.94.1.1 bus

```
uint8_t PciDevice::bus
```

Referenced by pci_find_class(), pci_find_first(), pci_read16(), pci_read32(), pci_read8(), pci_write16(), pci_write32(), and pci_write8().

6.94.1.2 device

```
uint8_t PciDevice::device
```

Referenced by pci_find_class(), pci_find_first(), pci_read16(), pci_read32(), pci_read8(), pci_write16(), pci_write32(), and pci_write8().

6.94.1.3 device_id

```
uint16_t PciDevice::device_id
```

Referenced by pci_find_class(), pci_find_first(), and xhci_find_and_map().

6.94.1.4 function

```
uint8_t PciDevice::function
```

Referenced by pci_find_class(), pci_find_first(), pci_read16(), pci_read32(), pci_read8(), pci_write16(), pci_write32(), and pci_write8().

6.94.1.5 vendor_id

```
uint16_t PciDevice::vendor_id
```

Referenced by pci_find_class(), pci_find_first(), and xhci_find_and_map().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/pci.h

6.95 physics_analytics_event_header Struct Reference

Event header written into the analytics ring buffer.

```
#include <physics_runtime.h>
```

Data Fields

- uint32_t channel
  Logical channel identifier.
- uint16_t payload_bytes
  Size of event payload.
- uint16_t reserved
  Reserved for alignment.
- uint64_t timestamp_ns
  Producer timestamp.

6.95.1 Detailed Description

Event header written into the analytics ring buffer.

6.95.2 Field Documentation

6.95.2.1 channel

```
uint32_t physics_analytics_event_header::channel
```

Logical channel identifier.

Referenced by physics_analytics_publish_event().

6.95.2.2 payload_bytes

```
uint16_t physics_analytics_event_header::payload_bytes
```

Size of event payload.

Referenced by physics_analytics_publish_event().

6.95.2.3 reserved

```
uint16_t physics_analytics_event_header::reserved
```

Reserved for alignment.

Referenced by physics_analytics_publish_event().

6.95.2.4 timestamp_ns

```
uint64_t physics_analytics_event_header::timestamp_ns
```

Producer timestamp.

Referenced by physics_analytics_publish_event().

The documentation for this struct was generated from the following file:

- v3/include/physics_runtime.h

6.96 physics_analytics_header Struct Reference

Analytics heap header seen by both producer (VM) and consumer (HOLA).

```
#include <physics_runtime.h>
```

Data Fields

- uint32_t magic
  HOLA_SHARED_MAGIC.
- uint16_t version_major
  Protocol major version.
- uint16_t version_minor
  Protocol minor version.
- uint32_t heap_bytes
  Total heap size.
- uint32_t ring_offset
  Offset to event ring.
- uint32_t ring_bytes
  Event ring length.
- uint32_t summary_offset
  Offset to summary/scratch region.
- uint32_t summary_bytes
  Summary region length.
- uint32_t scratch_offset
  Offset to reserved scratch area.
- uint32_t scratch_bytes
  Scratch area length.
- uint64_t produce_seq
  Monotonic sequence for producers.
- uint64_t consume_seq
  Consumer-visible sequence.
- uint32_t write_offset
  Producer write cursor.
- uint32_t read_offset
  Consumer read cursor.
- uint32_t dropped_events
  Count of dropped events.
- uint32_t flags
  Heap flags (see docs)

6.96.1 Detailed Description

Analytics heap header seen by both producer (VM) and consumer (HOLA).

6.96.2 Field Documentation

6.96.2.1 consume_seq

```
uint64_t physics_analytics_header::consume_seq
```

Consumer-visible sequence.

Referenced by analytics_heap_allocate().

6.96.2.2 dropped_events

```
uint32_t physics_analytics_header::dropped_events
```

Count of dropped events.

Referenced by analytics_heap_allocate(), and physics_analytics_publish_event().

6.96.2.3 flags

```
uint32_t physics_analytics_header::flags
```

Heap flags (see docs)

Referenced by analytics_heap_allocate().

6.96.2.4 heap_bytes

```
uint32_t physics_analytics_header::heap_bytes
```

Total heap size.

Referenced by analytics_heap_allocate().

6.96.2.5 magic

```
uint32_t physics_analytics_header::magic
```

HOLA_SHARED_MAGIC.

Referenced by analytics_heap_allocate().

6.96.2.6 produce_seq

```
uint64_t physics_analytics_header::produce_seq
```

Monotonic sequence for producers.

Referenced by analytics_heap_allocate(), and physics_analytics_publish_event().

6.96.2.7 read_offset

```
uint32_t physics_analytics_header::read_offset
```

Consumer read cursor.

Referenced by analytics_heap_allocate(), and analytics_ring_free().

6.96.2.8 ring_bytes

```
uint32_t physics_analytics_header::ring_bytes
```

Event ring length.

Referenced by analytics_heap_allocate(), analytics_ring_free(), and physics_analytics_publish_event().

6.96.2.9 ring_offset

```
uint32_t physics_analytics_header::ring_offset
```

Offset to event ring.

Referenced by analytics_heap_allocate().

6.96.2.10 scratch_bytes

```
uint32_t physics_analytics_header::scratch_bytes
```

Scratch area length.

Referenced by analytics_heap_allocate().

6.96.2.11 scratch_offset

```
uint32_t physics_analytics_header::scratch_offset
```

Offset to reserved scratch area.

Referenced by analytics_heap_allocate().

6.96.2.12 summary_bytes

```
uint32_t physics_analytics_header::summary_bytes
```

Summary region length.

Referenced by analytics_heap_allocate().

6.96.2.13 summary_offset

```
uint32_t physics_analytics_header::summary_offset
```

Offset to summary/scratch region.

Referenced by analytics_heap_allocate().

6.96.2.14 version_major

```
uint16_t physics_analytics_header::version_major
```

Protocol major version.

Referenced by analytics_heap_allocate().

6.96.2.15 version_minor

```
uint16_t physics_analytics_header::version_minor
```

Protocol minor version.

Referenced by analytics_heap_allocate().

6.96.2.16 write_offset

```
uint32_t physics_analytics_header::write_offset
```

Producer write cursor.

Referenced by analytics_heap_allocate(), analytics_ring_free(), and physics_analytics_publish_event().

The documentation for this struct was generated from the following file:

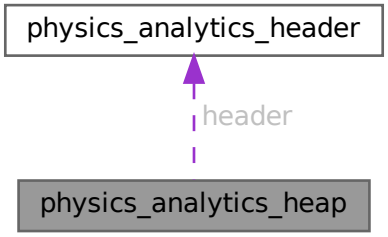
- v3/include/physics_runtime.h

6.97 physics_analytics_heap Struct Reference

Simple view onto the analytics heap layout.

```
#include <physics_runtime.h>
```

Collaboration diagram for physics_analytics_heap:



Data Fields

- uint8_t * base
  Base pointer (owned by runtime)
- size_t bytes
  Total bytes reserved.
- physics_analytics_header_t * header
  Shared header.
- uint8_t * ring
  Event ring base.
- size_t ring_bytes
  Event ring span.
- uint8_t * summary
  Summary/statistics region.
- size_t summary_bytes
  Summary region span.
- uint8_t * scratch
  Scratch/reserved region.
- size_t scratch_bytes
  Scratch region span.

6.97.1 Detailed Description

Simple view onto the analytics heap layout.

6.97.2 Field Documentation

6.97.2.1 base

```
uint8_t* physics_analytics_heap::base
```

Base pointer (owned by runtime)

Referenced by analytics_heap_allocate(), and analytics_heap_release().

6.97.2.2 bytes

```
size_t physics_analytics_heap::bytes
```

Total bytes reserved.

Referenced by analytics_heap_allocate().

6.97.2.3 header

```
physics_analytics_header_t* physics_analytics_heap::header
```

Shared header.

Referenced by analytics_heap_allocate(), and physics_analytics_publish_event().

6.97.2.4 ring

```
uint8_t* physics_analytics_heap::ring
```

Event ring base.

Referenced by analytics_heap_allocate(), and physics_analytics_publish_event().

6.97.2.5 ring_bytes

```
size_t physics_analytics_heap::ring_bytes
```

Event ring span.

Referenced by analytics_heap_allocate().

6.97.2.6 scratch

```
uint8_t* physics_analytics_heap::scratch
```

Scratch/reserved region.

Referenced by analytics_heap_allocate().

6.97.2.7 scratch_bytes

```
size_t physics_analytics_heap::scratch_bytes
```

Scratch region span.

Referenced by analytics_heap_allocate().

6.97.2.8 summary

```
uint8_t* physics_analytics_heap::summary
```

Summary/statistics region.

Referenced by analytics_heap_allocate().

6.97.2.9 summary_bytes

```
size_t physics_analytics_heap::summary_bytes
```

Summary region span.

Referenced by analytics_heap_allocate().

The documentation for this struct was generated from the following file:

- v3/include/physics_runtime.h

6.98 physics_host_snapshot Struct Reference

Snapshot of scheduler-adjacent host signals.

`#include <physics_runtime.h>`

Data Fields

- physics_host_backend_t backend
  Selected backend (POSIX/L4Re/other)
- uint32_t scheduler_policy
  OS-specific scheduler policy identifier.
- uint32_t scheduler_priority
  Nominal thread priority.
- uint32_t scheduler_quantum_ns
  Reported quantum in nanoseconds (0 if unknown)
- uint32_t runnable_threads
  Runnable entities on this CPU set (best effort)
- uint32_t cpu_count
  Online logical CPU count.
- uint32_t load_avg_milli
  Smoothed load average (×1000)
- uint32_t flags
  Backend-specific flags.
- uint32_t psi_cpu_some_avg10_milli
  CPU PSI some avg10 ×1000.
- uint32_t psi_cpu_some_avg60_milli
  CPU PSI some avg60 ×1000.
- uint32_t psi_cpu_some_avg300_milli
  CPU PSI some avg300 ×1000.
- uint32_t psi_cpu_full_avg10_milli
  CPU PSI full avg10 ×1000.
- uint32_t psi_cpu_full_avg60_milli
  CPU PSI full avg60 ×1000.
- uint32_t psi_cpu_full_avg300_milli
  CPU PSI full avg300 ×1000.
- uint32_t psi_io_some_avg10_milli
  IO PSI some avg10 ×1000.
- uint32_t psi_io_some_avg60_milli
  IO PSI some avg60 ×1000.
- uint32_t psi_io_some_avg300_milli
  IO PSI some avg300 ×1000.
- uint32_t psi_io_full_avg10_milli
  IO PSI full avg10 ×1000.
- uint32_t psi_io_full_avg60_milli
  IO PSI full avg60 ×1000.
- uint32_t psi_io_full_avg300_milli
  IO PSI full avg300 ×1000.
- uint32_t psi_mem_some_avg10_milli
  Memory PSI some avg10 ×1000.

- uint32_t psi_mem_some_avg60_milli
  Memory PSI some avg60 ×1000.
- uint32_t psi_mem_some_avg300_milli
  Memory PSI some avg300 ×1000.
- uint64_t monotonic_time_ns
  Monotonic clock reading.
- uint64_t realtime_ns
  Wall-clock timestamp (0 if unavailable)
- uint64_t backend_seq
  Backend-provided sequence counter.
- uint64_t cpu_total_jiffies
  /proc/stat total jiffies snapshot
- uint64_t cpu_idle_jiffies
  /proc/stat idle jiffies snapshot
- uint64_t cgroup_cpu_usage_us
  cgroup v2 cpu.stat usage_usec (0 if unavailable)
- uint64_t cgroup_memory_current_bytes
  cgroup v2 memory.current (0 if unavailable)

6.98.1 Detailed Description

Snapshot of scheduler-adjacent host signals.

Fields are intentionally fixed width to maximise portability and allow the governance repository to consume binary dumps without per-platform quirks.

6.98.2 Field Documentation

6.98.2.1 backend

```
physics_host_backend_t physics_host_snapshot::backend
```

Selected backend (POSIX/L4Re/other)

Referenced by snapshot_posix().

6.98.2.2 backend_seq

```
uint64_t physics_host_snapshot::backend_seq
```

Backend-provided sequence counter.

Referenced by snapshot_posix().

6.98.2.3 cgroup_cpu_usage_us

```
uint64_t physics_host_snapshot::cgroup_cpu_usage_us
```

cgroup v2 cpu.stat usage_usec (0 if unavailable)

Referenced by snapshot_posix().

6.98.2.4 cgroup_memory_current_bytes

```
uint64_t physics_host_snapshot::cgroup_memory_current_bytes
```

cgroup v2 memory.current (0 if unavailable)

Referenced by snapshot_posix().

6.98.2.5 cpu_count

```
uint32_t physics_host_snapshot::cpu_count
```

Online logical CPU count.

Referenced by snapshot_posix().

6.98.2.6 cpu_idle_jiffies

```
uint64_t physics_host_snapshot::cpu_idle_jiffies
```

/proc/stat idle jiffies snapshot

Referenced by snapshot_posix().

6.98.2.7 cpu_total_jiffies

```
uint64_t physics_host_snapshot::cpu_total_jiffies
```

/proc/stat total jiffies snapshot

Referenced by snapshot_posix().

6.98.2.8 flags

```
uint32_t physics_host_snapshot::flags
```

Backend-specific flags.

Referenced by snapshot_posix().

6.98.2.9 load_avg_milli

```
uint32_t physics_host_snapshot::load_avg_milli
```

Smoothed load average ($\times$1000)

Referenced by snapshot_posix().

6.98.2.10 monotonic_time_ns

```
uint64_t physics_host_snapshot::monotonic_time_ns
```

Monotonic clock reading.

Referenced by snapshot_posix().

6.98.2.11 psi_cpu_full_avg10_milli

```
uint32_t physics_host_snapshot::psi_cpu_full_avg10_milli
```

CPU PSI full avg10 ×1000.

Referenced by snapshot_posix().

6.98.2.12 psi_cpu_full_avg300_milli

```
uint32_t physics_host_snapshot::psi_cpu_full_avg300_milli
```

CPU PSI full avg300 ×1000.

Referenced by snapshot_posix().

6.98.2.13 psi_cpu_full_avg60_milli

```
uint32_t physics_host_snapshot::psi_cpu_full_avg60_milli
```

CPU PSI full avg60 ×1000.

Referenced by snapshot_posix().

6.98.2.14 psi_cpu_some_avg10_milli

```
uint32_t physics_host_snapshot::psi_cpu_some_avg10_milli
```

CPU PSI some avg10 ×1000.

Referenced by snapshot_posix().

6.98.2.15 psi_cpu_some_avg300_milli

```
uint32_t physics_host_snapshot::psi_cpu_some_avg300_milli
```

CPU PSI some avg300 ×1000.

Referenced by snapshot_posix().

6.98.2.16 psi_cpu_some_avg60_milli

```
uint32_t physics_host_snapshot::psi_cpu_some_avg60_milli
```

CPU PSI some avg60 ×1000.

Referenced by snapshot_posix().

6.98.2.17 psi_io_full_avg10_milli

```
uint32_t physics_host_snapshot::psi_io_full_avg10_milli
```

IO PSI full avg10 ×1000.

Referenced by snapshot_posix().

6.98.2.18 psi_io_full_avg300_milli

```
uint32_t physics_host_snapshot::psi_io_full_avg300_milli
```

IO PSI full avg300 ×1000.

Referenced by snapshot_posix().

6.98.2.19 psi_io_full_avg60_milli

```
uint32_t physics_host_snapshot::psi_io_full_avg60_milli
```

IO PSI full avg60 ×1000.

Referenced by snapshot_posix().

6.98.2.20 psi_io_some_avg10_milli

```
uint32_t physics_host_snapshot::psi_io_some_avg10_milli
```

IO PSI some avg10 ×1000.

Referenced by snapshot_posix().

6.98.2.21 psi_io_some_avg300_milli

```
uint32_t physics_host_snapshot::psi_io_some_avg300_milli
```

IO PSI some avg300 ×1000.

Referenced by snapshot_posix().

6.98.2.22 psi_io_some_avg60_milli

```
uint32_t physics_host_snapshot::psi_io_some_avg60_milli
```

IO PSI some avg60 ×1000.

Referenced by snapshot_posix().

6.98.2.23 psi_mem_some_avg10_milli

```
uint32_t physics_host_snapshot::psi_mem_some_avg10_milli
```

Memory PSI some avg10 ×1000.

Referenced by snapshot_posix().

6.98.2.24 psi_mem_some_avg300_milli

```
uint32_t physics_host_snapshot::psi_mem_some_avg300_milli
```

Memory PSI some avg300 ×1000.

Referenced by snapshot_posix().

6.98.2.25 psi_mem_some_avg60_milli

```
uint32_t physics_host_snapshot::psi_mem_some_avg60_milli
```

Memory PSI some avg60 ×1000.

Referenced by snapshot_posix().

6.98.2.26 realtime_ns

```
uint64_t physics_host_snapshot::realtime_ns
```

Wall-clock timestamp (0 if unavailable)

Referenced by snapshot_posix().

6.98.2.27 runnable_threads

```
uint32_t physics_host_snapshot::runnable_threads
```

Runnable entities on this CPU set (best effort)

Referenced by snapshot_posix().

6.98.2.28 scheduler_policy

```
uint32_t physics_host_snapshot::scheduler_policy
```

OS-specific scheduler policy identifier.

Referenced by snapshot_posix().

6.98.2.29 scheduler_priority

```
uint32_t physics_host_snapshot::scheduler_priority
```

Nominal thread priority.

Referenced by snapshot_posix().

6.98.2.30 scheduler_quantum_ns

```
uint32_t physics_host_snapshot::scheduler_quantum_ns
```

Reported quantum in nanoseconds (0 if unknown)

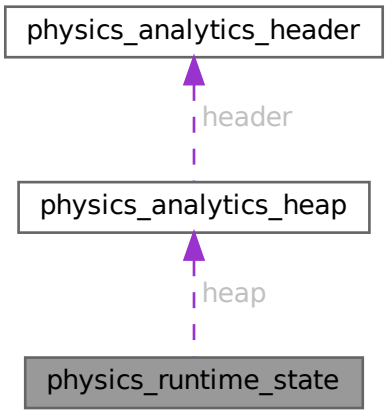
Referenced by snapshot_posix().

The documentation for this struct was generated from the following file:

- v3/include/physics_runtime.h

6.99 physics_runtime_state Struct Reference

Collaboration diagram for physics_runtime_state:



Data Fields

- physics_analytics_heap_t heap
- int heap_ready
- uint64_t backend_seq

6.99.1 Field Documentation

6.99.1.1 backend_seq

```
uint64_t physics_runtime_state::backend_seq
```

Referenced by analytics_heap_allocate(), analytics_heap_release(), and snapshot_posix().

6.99.1.2 heap

```
physics_analytics_heap_t physics_runtime_state::heap
```

Referenced by analytics_heap_allocate(), analytics_heap_release(), physics_analytics_heap_info(), and physics_analytics_publish_event().

6.99.1.3 heap_ready

```
int physics_runtime_state::heap_ready
```

Referenced by analytics_heap_allocate(), analytics_heap_release(), physics_analytics_heap_info(), physics_analytics_publish_event() and physics_host_snapshot().

The documentation for this struct was generated from the following file:

- v3/src/physics_runtime.c

6.100 physics_seed_definition Struct Reference

Data Fields

- const char * name
- uint16_t temperature_q8
- uint32_t avg_latency_ns
- uint8_t acl_hint
- uint8_t pubsub_mask

6.100.1 Field Documentation

6.100.1.1 acl_hint

```
uint8_t physics_seed_definition::acl_hint
```

6.100.1.2 avg_latency_ns

```
uint32_t physics_seed_definition::avg_latency_ns
```

6.100.1.3 name

```
const char* physics_seed_definition::name
```

Referenced by physics_metadata_apply_seed().

6.100.1.4 pubsub_mask

```
uint8_t physics_seed_definition::pubsub_mask
```

6.100.1.5 temperature_q8

```
uint16_t physics_seed_definition::temperature_q8
```

Referenced by physics_metadata_apply_seed().

The documentation for this struct was generated from the following file:

- v3/src/physics_metadata.c

6.101 PhysicsSnapshot Struct Reference

```
#include <test_contracts.h>
```

Data Fields

- uint64_t decay_slope_q48
- uint32_t eff_window_size
- uint64_t tick_target_ns
- cell_t heat_25th
- cell_t heat_50th
- cell_t heat_75th
- uint64_t prefetch_attempts
- uint64_t total_executions

6.101.1 Field Documentation

6.101.1.1 decay_slope_q48

```
uint64_t PhysicsSnapshot::decay_slope_q48
```

Referenced by restore_physics(), and save_physics().

6.101.1.2 eff_window_size

```
uint32_t PhysicsSnapshot::eff_window_size
```

Referenced by restore_physics(), and save_physics().

6.101.1.3 heat_25th

```
cell_t PhysicsSnapshot::heat_25th
```

Referenced by restore_physics(), and save_physics().

6.101.1.4 heat_50th

```
cell_t PhysicsSnapshot::heat_50th
```

Referenced by restore_physics(), and save_physics().

6.101.1.5 heat_75th

```
cell_t PhysicsSnapshot::heat_75th
```

Referenced by restore_physics(), and save_physics().

6.101.1.6 prefetch_attempts

```
uint64_t PhysicsSnapshot::prefetch_attempts
```

Referenced by restore_physics(), and save_physics().

6.101.1.7 tick_target_ns

```
uint64_t PhysicsSnapshot::tick_target_ns
```

Referenced by restore_physics(), and save_physics().

6.101.1.8 total_executions

```
uint64_t PhysicsSnapshot::total_executions
```

Referenced by restore_physics(), and save_physics().

The documentation for this struct was generated from the following file:

- v3/src/test_runner/include/test_contracts.h

6.102 PipelineGlobalMetrics Struct Reference

`#include <vm.h>`

Data Fields

- uint64_t prefetch_attempts
- uint64_t prefetch_hits
- uint64_t window_tuning_checks
- uint32_t last_checked_window_size
- double last_checked_accuracy
- uint32_t suggested_next_size

6.102.1 Field Documentation

6.102.1.1 last_checked_accuracy

`double PipelineGlobalMetrics::last_checked_accuracy`

Referenced by vm_init(), vm_init_with_host(), and vm_tick_window_tuner().

6.102.1.2 last_checked_window_size

`uint32_t PipelineGlobalMetrics::last_checked_window_size`

Referenced by vm_init(), vm_init_with_host(), and vm_tick_window_tuner().

6.102.1.3 prefetch_attempts

`uint64_t PipelineGlobalMetrics::prefetch_attempts`

Referenced by execute_colon_word(), forth_PHYSICS_RESET_STATS(), infer_word_run(), metrics_from_vm(), perturb_physics(), physics_pre_execute(), restore_physics(), save_physics(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_window_tuner().

6.102.1.4 prefetch_hits

`uint64_t PipelineGlobalMetrics::prefetch_hits`

Referenced by execute_colon_word(), forth_PHYSICS_RESET_STATS(), infer_word_run(), metrics_from_vm(), physics_pre_execute(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_window_tuner().

6.102.1.5 suggested_next_size

`uint32_t PipelineGlobalMetrics::suggested_next_size`

Referenced by vm_init(), and vm_init_with_host().

6.102.1.6 window_tuning_checks

```
uint64_t PipelineGlobalMetrics::window_tuning_checks
```

Referenced by forth_PHYSICS_RESET_STATS(), metrics_from_vm(), vm_init(), vm_init_with_host(), and vm_tick_window_tuner().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.103 pmm_stats_t Struct Reference

```
#include <pmm.h>
```

Data Fields

- uint64_t total_pages
- uint64_t used_pages
- uint64_t free_pages
- uint64_t total_bytes
- uint64_t used_bytes
- uint64_t free_bytes

6.103.1 Field Documentation

6.103.1.1 free_bytes

```
uint64_t pmm_stats_t::free_bytes
```

Referenced by pmm_get_stats(), and print_pmm_stats().

6.103.1.2 free_pages

```
uint64_t pmm_stats_t::free_pages
```

Referenced by pmm_get_stats(), and print_pmm_stats().

6.103.1.3 total_bytes

```
uint64_t pmm_stats_t::total_bytes
```

Referenced by pmm_get_stats(), and print_pmm_stats().

6.103.1.4 total_pages

```
uint64_t pmm_stats_t::total_pages
```

Referenced by pmm_get_stats(), and print_pmm_stats().

6.103.1.5 used_bytes

```
uint64_t pmm_stats_t::used_bytes
```

Referenced by pmm_get_stats(), and print_pmm_stats().

6.103.1.6 used_pages

```
uint64_t pmm_stats_t::used_pages
```

Referenced by pmm_get_stats(), and print_pmm_stats().

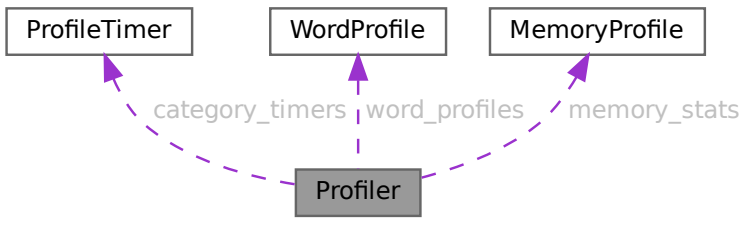
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/pmm.h

6.104 Profiler Struct Reference

```
#include <profiler.h>
```

Collaboration diagram for Profiler:



Data Fields

- ProfileLevel level
- uint64_t profiling_overhead
- uint64_t total_execution_time
- ProfileTimer category_timers [PROF_CATEGORY_COUNT]
- WordProfile * word_profiles
- size_t word_profile_count
- size_t word_profile_capacity
- MemoryProfile memory_stats
- uint64_t instruction_count
- uint64_t branch_count
- uint64_t loop_iterations
- struct {
  uint64_t vm_cycles
  uint64_t dict_lookups
  uint64_t stack_operations
  uint64_t memory_allocations
  uint64_t compile_operations
  } counters

6.104.1 Field Documentation

6.104.1.1 branch_count

```
uint64_t Profiler::branch_count
```

6.104.1.2 category_timers

```
ProfileTimer Profiler::category_timers[PROF_CATEGORY_COUNT]
```

6.104.1.3 compile_operations

```
uint64_t Profiler::compile_operations
```

6.104.1.4 [struct]

```
struct { ... } Profiler::counters
```

6.104.1.5 dict_lookups

```
uint64_t Profiler::dict_lookups
```

6.104.1.6 instruction_count

```
uint64_t Profiler::instruction_count
```

6.104.1.7 level

```
ProfileLevel Profiler::level
```

6.104.1.8 loop_iterations

```
uint64_t Profiler::loop_iterations
```

6.104.1.9 memory_allocations

```
uint64_t Profiler::memory_allocations
```

6.104.1.10 memory_stats

```
MemoryProfile Profiler::memory_stats
```

6.104.1.11 profiling_overhead

```
uint64_t Profiler::profiling_overhead
```

6.104.1.12 stack_operations

```
uint64_t Profiler::stack_operations
```

6.104.1.13 total_execution_time

```
uint64_t Profiler::total_execution_time
```

6.104.1.14 vm_cycles

```
uint64_t Profiler::vm_cycles
```

6.104.1.15 word_profile_capacity

```
size_t Profiler::word_profile_capacity
```

6.104.1.16 word_profile_count

```
size_t Profiler::word_profile_count
```

6.104.1.17 word_profiles

```
WordProfile* Profiler::word_profiles
```

The documentation for this struct was generated from the following file:

- v3/include/profiler.h

6.105 ProfileScope Struct Reference

```
#include <profiler.h>
```

Data Fields

- ProfileCategory category
- const char * word_name

6.105.1 Field Documentation

6.105.1.1 category

```
ProfileCategory ProfileScope::category
```

6.105.1.2 word_name

```
const char* ProfileScope::word_name
```

The documentation for this struct was generated from the following file:

- v3/include/profiler.h

6.106 ProfileTimer Struct Reference

Timer for measuring execution duration.

```
#include <profiler.h>
```

Data Fields

- uint64_t start_time
  Start timestamp.
- uint64_t total_time
  Accumulated execution time.
- uint64_t call_count
  Number of measurements.
- uint64_t min_time
  Minimum recorded time.
- uint64_t max_time
  Maximum recorded time.
- const char * name
  Timer identifier.

6.106.1 Detailed Description

Timer for measuring execution duration.

6.106.2 Field Documentation

6.106.2.1 call_count

```
uint64_t ProfileTimer::call_count
```

Number of measurements.

6.106.2.2 max_time

```
uint64_t ProfileTimer::max_time
```

Maximum recorded time.

6.106.2.3 min_time

```
uint64_t ProfileTimer::min_time
```

Minimum recorded time.

6.106.2.4 name

```
const char* ProfileTimer::name
```

Timer identifier.

6.106.2.5 start_time

```
uint64_t ProfileTimer::start_time
```

Start timestamp.

6.106.2.6 total_time

```
uint64_t ProfileTimer::total_time
```

Accumulated execution time.

The documentation for this struct was generated from the following file:

- v3/include/profiler.h

6.107 reloc_block Struct Reference

reloc_stub.c - Provide a minimal .reloc section for PE/COFF EFI loader

Data Fields

- uint32_t page_rva
- uint32_t block_size
- uint16_t entry

6.107.1 Detailed Description

reloc_stub.c - Provide a minimal .reloc section for PE/COFF EFI loader

Some firmware refuses to load images without a Base Relocation Directory. This stub supplies an empty relocation block to satisfy the loader.

PE/COFF Base Relocation Block header.

Mirrors the `IMAGE_BASE_RELOCATION` layout from the PE specification:

- `page_rva` — RVA of the 4 KB page that the following entries apply to.
- `block_size` — total byte size of this block including the header and all entries (minimum 8 bytes for the header alone).
- `entry` — one or more 16-bit relocation entries packed as TYPE (4 bits) | OFFSET (12 bits); TYPE=0 / OFFSET=0 is a no-op padding entry used to 32-bit-align the block.

Marked `packed` to suppress any padding the compiler might insert between fields; `aligned(4)` ensures the block sits on a 4-byte boundary as required by the PE loader.

6.107.2 Field Documentation

6.107.2.1 block_size

```
uint32_t reloc_block::block_size
```

6.107.2.2 entry

```
uint16_t reloc_block::entry
```

6.107.2.3 page_rva

```
uint32_t reloc_block::page_rva
```

Referenced by __attribute__().

The documentation for this struct was generated from the following file:

- kernel/src/boot/reloc_stub.c

6.108 rolling_window_view_t Struct Reference

Immutable snapshot view of a `RollingWindowOfTruth`.

Data Fields

- const uint32_t * history
- uint32_t window_pos
- uint64_t total_executions
- uint32_t effective_window_size
- int is_warm

6.108.1 Detailed Description

Immutable snapshot view of a `RollingWindowOfTruth`.

Provides a read-only window into one of the two double-buffered snapshot arrays. Reader threads (the heartbeat, pipelining, and diversity subsystems) obtain a `rolling_window_view_t` via `rolling_window_snapshot_view()` rather than accessing the live `RollingWindowOfTruth` fields directly; this avoids TOCTOU races with the writer thread that calls `rolling_window_record_execution()`.

Fields mirror the snapshot-time copies of the corresponding `RollingWindowOfTruth` members:

- `history` — pointer into `snapshot_buffers`[idx]; size `ROLLING_WINDOW_SIZE`.
- `window_pos` — write cursor at snapshot time.
- `total_executions` — monotonic execution count at snapshot time.
- `effective_window_size` — adaptive window width at snapshot time.
- `is_warm` — whether the window had reached $\geq$ 1024 executions at snapshot.

6.108.2 Field Documentation

6.108.2.1 effective_window_size

```
uint32_t rolling_window_view_t::effective_window_size
```

Referenced by rolling_window_measure_diversity_view(), and rolling_window_snapshot_view().

6.108.2.2 history

```
const uint32_t* rolling_window_view_t::history
```

Referenced by rolling_window_count_transition(), rolling_window_export_execution_history(), rolling_window_find_hottest_word(), rolling_window_get_recent_sequence(), rolling_window_measure_diversity_view(), rolling_window_pattern_capture_rate(), rolling_window_run_adaptive_pass(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), and rolling_window_snapshot_view().

6.108.2.3 is_warm

```
int rolling_window_view_t::is_warm
```

Referenced by rolling_window_count_transition(), rolling_window_find_hottest_word(), rolling_window_measure_diversity_view(), rolling_window_run_adaptive_pass(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), rolling_window_snapshot_view(), and rolling_window_stats_string().

6.108.2.4 total_executions

```
uint64_t rolling_window_view_t::total_executions
```

Referenced by rolling_window_export_execution_history(), rolling_window_get_recent_sequence(), rolling_window_measure_diversit rolling_window_pattern_capture_rate(), rolling_window_run_adaptive_pass(), rolling_window_snapshot_view(), and rolling_window_stats_string().

6.108.2.5 window_pos

```
uint32_t rolling_window_view_t::window_pos
```

Referenced by rolling_window_export_execution_history(), rolling_window_get_recent_sequence(), rolling_window_measure_diversit rolling_window_snapshot_view(), and rolling_window_stats_string().

The documentation for this struct was generated from the following file:

- v3/src/rolling_window_of_truth.c

6.109 RollingWindowOfTruth Struct Reference

```
#include <vm.h>
```

Data Fields

- uint32_t * execution_history
- uint32_t * snapshot_buffers [2]
- uint32_t window_pos
- uint64_t total_executions
- int is_warm
- uint32_t effective_window_size
- uint64_t last_pattern_diversity
- uint64_t pattern_diversity_check_count
- volatile uint32_t snapshot_index
- volatile uint32_t snapshot_pending
- uint32_t snapshot_window_pos [2]
- uint64_t snapshot_total_executions [2]
- uint32_t snapshot_effective_window_size [2]
- int snapshot_is_warm [2]
- uint32_t adaptive_check_accumulator
- volatile uint32_t adaptive_pending

6.109.1 Field Documentation

6.109.1.1 adaptive_check_accumulator

```
uint32_t RollingWindowOfTruth::adaptive_check_accumulator
```

Referenced by rolling_window_init(), and rolling_window_record_execution().

6.109.1.2 adaptive_pending

```
volatile uint32_t RollingWindowOfTruth::adaptive_pending
```

Referenced by rolling_window_check_adaptive_shrink(), rolling_window_init(), rolling_window_record_execution(), and rolling_window_service_internal().

6.109.1.3 effective_window_size

```
uint32_t RollingWindowOfTruth::effective_window_size
```

Referenced by extract_heat_trajectory(), heartbeat_capture_tick_snapshot(), heartbeat_publish_snapshot(), metrics_from_vm(), perturb_physics(), restore_physics(), rolling_window_init(), rolling_window_publish_snapshot(), rolling_window_run_adaptive_pass(), run_doe_experiment(), save_physics(), vm_heartbeat_publish_snapshot(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_window_tuner().

6.109.1.4 execution_history

```
uint32_t* RollingWindowOfTruth::execution_history
```

Referenced by rolling_window_cleanup(), rolling_window_init(), rolling_window_publish_snapshot(), rolling_window_record_execution and rolling_window_reset().

6.109.1.5 is_warm

```
int RollingWindowOfTruth::is_warm
```

Referenced by extract_heat_trajectory(), rolling_window_count_transition(), rolling_window_find_hottest_word(), rolling_window_init(), rolling_window_is_warm(), rolling_window_publish_snapshot(), rolling_window_record_execution(), rolling_window_reset(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), run_doe_experiment(), vm_tick_inference_engine(), and vm_tick_window_tuner().

6.109.1.6 last_pattern_diversity

```
uint64_t RollingWindowOfTruth::last_pattern_diversity
```

Referenced by rolling_window_init(), and rolling_window_run_adaptive_pass().

6.109.1.7 pattern_diversity_check_count

```
uint64_t RollingWindowOfTruth::pattern_diversity_check_count
```

Referenced by rolling_window_init(), and rolling_window_run_adaptive_pass().

6.109.1.8 snapshot_buffers

```
uint32_t* RollingWindowOfTruth::snapshot_buffers[2]
```

Referenced by rolling_window_cleanup(), rolling_window_init(), rolling_window_publish_snapshot(), and rolling_window_snapshot_view().

6.109.1.9 snapshot_effective_window_size

```
uint32_t RollingWindowOfTruth::snapshot_effective_window_size[2]
```

Referenced by rolling_window_init(), rolling_window_publish_snapshot(), and rolling_window_snapshot_view().

6.109.1.10 snapshot_index

```
volatile uint32_t RollingWindowOfTruth::snapshot_index
```

Referenced by rolling_window_init(), rolling_window_publish_snapshot(), and rolling_window_snapshot_view().

6.109.1.11 snapshot_is_warm

```
int RollingWindowOfTruth::snapshot_is_warm[2]
```

Referenced by rolling_window_init(), rolling_window_publish_snapshot(), and rolling_window_snapshot_view().

6.109.1.12 snapshot_pending

```
volatile uint32_t RollingWindowOfTruth::snapshot_pending
```

Referenced by rolling_window_init(), rolling_window_publish_snapshot(), rolling_window_publish_snapshot_if_needed(), rolling_window_record_execution(), and rolling_window_reset().

6.109.1.13 snapshot_total_executions

```
uint64_t RollingWindowOfTruth::snapshot_total_executions[2]
```

Referenced by rolling_window_init(), rolling_window_publish_snapshot(), and rolling_window_snapshot_view().

6.109.1.14 snapshot_window_pos

```
uint32_t RollingWindowOfTruth::snapshot_window_pos[2]
```

Referenced by rolling_window_init(), rolling_window_publish_snapshot(), and rolling_window_snapshot_view().

6.109.1.15 total_executions

```
uint64_t RollingWindowOfTruth::total_executions
```

Referenced by heartbeat_capture_tick_snapshot(), infer_word_run(), metrics_from_vm(), perturb_physics(), restore_physics(), rolling_window_init(), rolling_window_publish_snapshot(), rolling_window_record_execution(), rolling_window_reset(), run_doe_experiment(), save_physics(), vm_heartbeat_publish_snapshot(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), and vm_tick_inference_engine().

6.109.1.16 window_pos

```
uint32_t RollingWindowOfTruth::window_pos
```

Referenced by infer_word_run(), rolling_window_init(), rolling_window_publish_snapshot(), rolling_window_record_execution(), rolling_window_reset(), and vm_tick_inference_engine().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.110 rpi5_range_t Struct Reference

Data Fields

- uint64_t addr
- uint64_t size

6.110.1 Field Documentation

6.110.1.1 addr

```
uint64_t rpi5_range_t::addr
```

Referenced by collect_ram_regions(), collect_reserved_ranges(), emit_region_with_carveouts(), and sort_ranges_by_addr().

6.110.1.2 size

```
uint64_t rpi5_range_t::size
```

Referenced by collect_ram_regions(), collect_reserved_ranges(), and emit_region_with_carveouts().

The documentation for this struct was generated from the following file:

- kernel/src/arch/aarch64/rpi5_native_boot.c

6.111 Rpi5FramebufferInfo Struct Reference

rpi5_mailbox.h - VideoCore mailbox property-interface framebuffer setup, for the native (non-UEFI) Raspberry Pi 5 boot path (FABRIC-3.md ?IV.3 item 3).

```
#include <rpi5_mailbox.h>
```

Data Fields

- void * base
- uint64_t size
- uint32_t width
- uint32_t height
- uint32_t pixels_per_scanline
- uint32_t pixel_format

6.111.1 Detailed Description

rpi5_mailbox.h - VideoCore mailbox property-interface framebuffer setup, for the native (non-UEFI) Raspberry Pi 5 boot path (FABRIC-3.md ?IV.3 item 3).

Register layout, message/tag format, and property-tag IDs confirmed against multiple sources before writing `rpi5_mailbox.c` — see that file's own doc comment for exactly which, and what (if anything) remains unverified against real hardware (not in hand until 2026-09-17; this driver has never run on real silicon).

6.111.2 Field Documentation

6.111.2.1 base

```
void* Rpi5FramebufferInfo::base
```

Referenced by rpi5_mailbox_get_framebuffer(), and rpi5_native_boot().

6.111.2.2 height

```
uint32_t Rpi5FramebufferInfo::height
```

Referenced by rpi5_mailbox_get_framebuffer(), and rpi5_native_boot().

6.111.2.3 pixel_format

```
uint32_t Rpi5FramebufferInfo::pixel_format
```

Referenced by rpi5_mailbox_get_framebuffer(), and rpi5_native_boot().

6.111.2.4 pixels_per_scanline

```
uint32_t Rpi5FramebufferInfo::pixels_per_scanline
```

Referenced by rpi5_mailbox_get_framebuffer(), and rpi5_native_boot().

6.111.2.5 size

```
uint64_t Rpi5FramebufferInfo::size
```

Referenced by rpi5_mailbox_get_framebuffer(), and rpi5_native_boot().

6.111.2.6 width

```
uint32_t Rpi5FramebufferInfo::width
```

Referenced by rpi5_mailbox_get_framebuffer(), and rpi5_native_boot().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/rpi5_mailbox.h

6.112 sbiret_t Struct Reference

Data Fields

- long error
- long value

6.112.1 Field Documentation

6.112.1.1 error

```
long sbiret_t::error
```

Referenced by apic_timer_start(), sbi_call1(), and sbi_set_timer().

6.112.1.2 value

```
long sbiret_t::value
```

Referenced by apic_timer_start(), and sbi_call1().

The documentation for this struct was generated from the following file:

- kernel/src/arch/riscv64/apic.c

6.113 sf_alloc_stats Struct Reference

```
#include <platform_alloc.h>
```

Data Fields

- size_t total_bytes
- size_t used_bytes
- size_t peak_bytes
- size_t alloc_count
- size_t free_count

6.113.1 Field Documentation

6.113.1.1 alloc_count

```
size_t sf_alloc_stats::alloc_count
```

Referenced by sf_alloc_get_stats(), sf_alloc_init(), sf_calloc(), sf_malloc(), and sf_realloc().

6.113.1.2 free_count

```
size_t sf_alloc_stats::free_count
```

Referenced by sf_alloc_get_stats(), sf_alloc_init(), and sf_free().

6.113.1.3 peak_bytes

```
size_t sf_alloc_stats::peak_bytes
```

Referenced by sf_alloc_get_stats(), sf_alloc_init(), sf_calloc(), and sf_malloc().

6.113.1.4 total_bytes

```
size_t sf_alloc_stats::total_bytes
```

Referenced by sf_alloc_get_stats(), and sf_alloc_init().

6.113.1.5 used_bytes

```
size_t sf_alloc_stats::used_bytes
```

Referenced by sf_alloc_get_stats(), sf_alloc_init(), sf_calloc(), and sf_malloc().

The documentation for this struct was generated from the following file:

- v3/include/platform_alloc.h

6.114 sf_mutex Struct Reference

```
#include <platform_lock.h>
```

Data Fields

- pthread_mutex_t handle

6.114.1 Field Documentation

6.114.1.1 handle

```
pthread_mutex_t sf_mutex::handle
```

Referenced by sf_mutex_destroy(), sf_mutex_init(), sf_mutex_lock(), and sf_mutex_unlock().

The documentation for this struct was generated from the following file:

- v3/include/platform_lock.h

6.115 sf_time_backend_t Struct Reference

```
#include <platform_time.h>
```

Data Fields

- sf_time_ns_t(* get_monotonic_ns)(void)
  Get monotonic time (for performance measurement)
- sf_time_ns_t(* get_realtime_ns)(void)
  Get wall-clock time.
- int(* set_realtime_ns)(sf_time_ns_t ns_since_epoch)
  Set wall-clock time (may require privileges)
- int(* format_timestamp)(sf_time_ns_t ns_since_epoch, char *buf, int format_24h)
  Format timestamp as human-readable string.
- int(* has_rtc)(void)
  Check if real-time clock is available.

6.115.1 Field Documentation

6.115.1.1 format_timestamp

```
int(* sf_time_backend_t::format_timestamp) (sf_time_ns_t ns_since_epoch, char *buf, int format↩
_24h)
```

Format timestamp as human-readable string.

Parameters

ns_since_epoch	Nanoseconds since Unix epoch
buf	Output buffer (SF_TIME_STAMP_SIZE bytes)
format_24h	Use 24-hour format (1) or 12-hour (0)

Returns

0 on success, -1 on error

Referenced by sf_format_timestamp().

6.115.1.2 get_monotonic_ns

```
sf_time_ns_t(* sf_time_backend_t::get_monotonic_ns) (void)
```

Get monotonic time (for performance measurement)

Returns

Nanoseconds since system boot (never decreases)

Referenced by sf_monotonic_ns().

6.115.1.3 get_realtime_ns

```
sf_time_ns_t(* sf_time_backend_t::get_realtime_ns) (void)
```

Get wall-clock time.

Returns

Nanoseconds since Unix epoch (1970-01-01 00:00:00 UTC) Returns 0 if RTC unavailable

Referenced by sf_realtime_ns().

6.115.1.4 has_rtc

```
int(* sf_time_backend_t::has_rtc) (void)
```

Check if real-time clock is available.

Returns

1 if RTC available, 0 if not

Referenced by sf_has_rtc().

6.115.1.5 set_realtime_ns

```
int(* sf_time_backend_t::set_realtime_ns) (sf_time_ns_t ns_since_epoch)
```

Set wall-clock time (may require privileges)

Parameters

ns_since_epoch	Nanoseconds since Unix epoch

Returns

0 on success, -1 on error

Referenced by sf_set_realtime_ns().

The documentation for this struct was generated from the following file:

- v3/include/platform_time.h

6.116 sha512_ctx_t Struct Reference

`#include <sha512.h>`

Data Fields

- uint64_t state [8]
- uint64_t bitlen
- uint8_t buf [128]
- size_t buf_len

6.116.1 Field Documentation

6.116.1.1 bitlen

`uint64_t sha512_ctx_t::bitlen`

Referenced by sha512_final(), sha512_init(), and sha512_update().

6.116.1.2 buf

`uint8_t sha512_ctx_t::buf[128]`

Referenced by sha512_final(), and sha512_update().

6.116.1.3 buf_len

`size_t sha512_ctx_t::buf_len`

Referenced by sha512_final(), sha512_init(), and sha512_update().

6.116.1.4 state

```
uint64_t sha512_ctx_t::state[8]
```

Referenced by sha512_block(), sha512_final(), and sha512_init().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/sha512.h

6.117 sigaction Struct Reference

```
#include <signal.h>
```

Data Fields

- sighandler_t sa_handler
- int sa_flags

6.117.1 Field Documentation

6.117.1.1 sa_flags

```
int sigaction::sa_flags
```

Referenced by vm_debug_install_signal_handlers().

6.117.1.2 sa_handler

```
sighandler_t sigaction::sa_handler
```

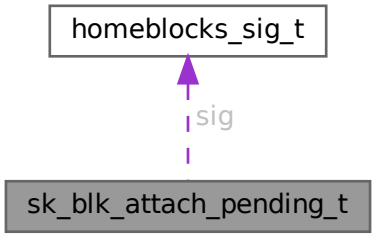
Referenced by vm_debug_install_signal_handlers().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/freestanding/signal.h

6.118 sk_blk_attach_pending_t Struct Reference

Collaboration diagram for sk_blk_attach_pending_t:



Data Fields

- int pending
- homeblocks_sig_result_t sig_rc
- homeblocks_sig_t sig
- int is_artemis

6.118.1 Field Documentation

6.118.1.1 is_artemis

```
int sk_blk_attach_pending_t::is_artemis
```

Referenced by sk_repl_idle(), and sk_word_blk_attach_ack().

6.118.1.2 pending

```
int sk_blk_attach_pending_t::pending
```

Referenced by sk_repl_idle(), and sk_word_blk_attach_ack().

6.118.1.3 sig

```
homeblocks_sig_t sk_blk_attach_pending_t::sig
```

Referenced by sk_repl_idle(), and sk_word_blk_attach_ack().

6.118.1.4 sig_rc

```
homeblocks_sig_result_t sk_blk_attach_pending_t::sig_rc
```

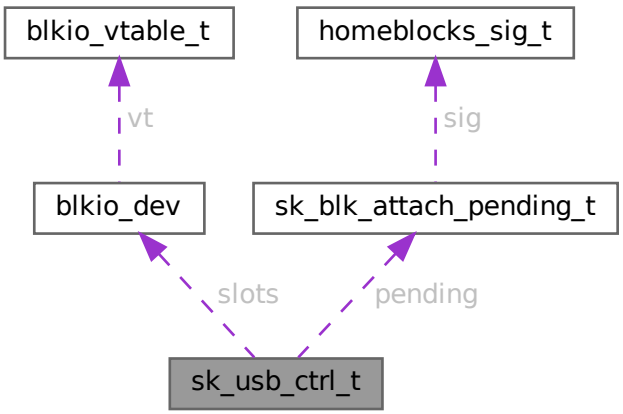
Referenced by sk_repl_idle(), and sk_word_blk_attach_ack().

The documentation for this struct was generated from the following file:

- kernel/src/repl.c

6.119 sk_usb_ctrl_t Struct Reference

Collaboration diagram for sk_usb_ctrl_t:



Data Fields

- blkio_dev_t * slots
- uint32_t count
- sk_blk_attach_pending_t * pending

6.119.1 Field Documentation

6.119.1.1 count

```
uint32_t sk_usb_ctrl_t::count
```

Referenced by sk_repl_idle(), and sk_word_blk_attach_ack().

6.119.1.2 pending

```
sk_blk_attach_pending_t* sk_usb_ctrl_t::pending
```

Referenced by sk_repl_idle(), and sk_word_blk_attach_ack().

6.119.1.3 slots

```
blkio_dev_t* sk_usb_ctrl_t::slots
```

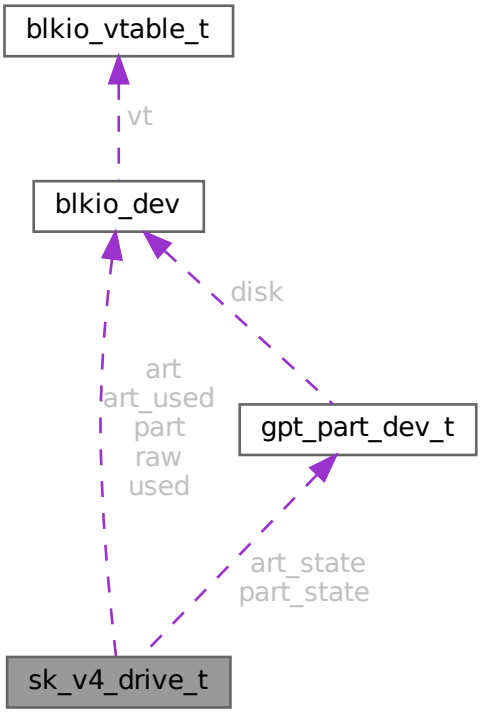
Referenced by sk_repl_idle().

The documentation for this struct was generated from the following file:

- kernel/src/repl.c

6.120 sk_v4_drive_t Struct Reference

Collaboration diagram for sk_v4_drive_t:



Data Fields

- blkio_dev_t raw
- blkio_dev_t part
- gpt_part_dev_t part_state
- blkio_dev_t * used
- blkio_dev_t art
- gpt_part_dev_t art_state
- blkio_dev_t * art_used

6.120.1 Field Documentation

6.120.1.1 art

```
blkio_dev_t sk_v4_drive_t::art
```

Referenced by sk_v4_drive_arrived(), and sk_v4_grow_stick().

6.120.1.2 art_state

```
gpt_part_dev_t sk_v4_drive_t::art_state
```

Referenced by sk_v4_drive_arrived(), and sk_v4_grow_stick().

6.120.1.3 art_used

```
blkio_dev_t* sk_v4_drive_t::art_used
```

Referenced by sk_v4_drive_arrived(), and sk_v4_drive_left().

6.120.1.4 part

```
blkio_dev_t sk_v4_drive_t::part
```

Referenced by sk_v4_drive_arrived(), and sk_v4_real_zuse_arrived().

6.120.1.5 part_state

```
gpt_part_dev_t sk_v4_drive_t::part_state
```

Referenced by sk_v4_drive_arrived(), and sk_v4_real_zuse_arrived().

6.120.1.6 raw

```
blkio_dev_t sk_v4_drive_t::raw
```

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), sk_v4_identity_offer_wait(), and sk_v4_real_zuse_arrived().

6.120.1.7 used

```
blkio_dev_t* sk_v4_drive_t::used
```

Referenced by sk_v4_drive_arrived(), sk_v4_drive_left(), and sk_v4_real_zuse_arrived().

The documentation for this struct was generated from the following file:

- kernel/src/v4/sk_v4_identity.c

6.121 sk_v4_marker_t Struct Reference

```
#include <sk_v4_marker.h>
```

Data Fields

- int present
- uint8_t proxy [32]
- int has_real
- uint8_t real [32]

6.121.1 Field Documentation

6.121.1.1 has_real

```
int sk_v4_marker_t::has_real
```

Referenced by sk_v4_drive_arrived(), sk_v4_drive_left(), sk_v4_marker_read(), sk_v4_marker_write(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

6.121.1.2 present

```
int sk_v4_marker_t::present
```

Referenced by sk_v4_drive_arrived(), sk_v4_drive_left(), sk_v4_genesis(), sk_v4_marker_read(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

6.121.1.3 proxy

```
uint8_t sk_v4_marker_t::proxy[32]
```

Referenced by sk_v4_drive_arrived(), sk_v4_genesis(), sk_v4_marker_read(), and sk_v4_marker_write().

6.121.1.4 real

```
uint8_t sk_v4_marker_t::real[32]
```

Referenced by sk_v4_marker_read(), sk_v4_marker_write(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/v4/sk_v4_marker.h

6.122 SpeedupEstimate Struct Reference

Calculate speedup estimate with credible interval All values in Q48.16 fixed-point format (64-bit signed integer) Probabilities are in range [0, Q48_SCALE] representing [0%, 100%].

```
#include <physics_hotwords_cache.h>
```

Data Fields

- int64_t speedup_factor_q48
- int64_t credible_lower_95_q48
- int64_t credible_upper_95_q48
- int64_t probability_gt_10pct_q48
- int64_t probability_gt_double_q48

6.122.1 Detailed Description

Calculate speedup estimate with credible interval All values in Q48.16 fixed-point format (64-bit signed integer) Probabilities are in range [0, Q48_SCALE] representing [0%, 100%].

6.122.2 Field Documentation

6.122.2.1 credible_lower_95_q48

```
int64_t SpeedupEstimate::credible_lower_95_q48
```

Referenced by hotwords_bayesian_report(), and hotwords_speedup_estimate().

6.122.2.2 credible_upper_95_q48

```
int64_t SpeedupEstimate::credible_upper_95_q48
```

Referenced by hotwords_bayesian_report(), and hotwords_speedup_estimate().

6.122.2.3 probability_gt_10pct_q48

```
int64_t SpeedupEstimate::probability_gt_10pct_q48
```

Referenced by hotwords_bayesian_report(), and hotwords_speedup_estimate().

6.122.2.4 probability_gt_double_q48

```
int64_t SpeedupEstimate::probability_gt_double_q48
```

Referenced by hotwords_bayesian_report(), and hotwords_speedup_estimate().

6.122.2.5 speedup_factor_q48

```
int64_t SpeedupEstimate::speedup_factor_q48
```

Referenced by hotwords_bayesian_report(), and hotwords_speedup_estimate().

The documentation for this struct was generated from the following file:

- v3/include/physics_hotwords_cache.h

6.123 ssm_config_t Struct Reference

```
#include <ssm_jacquard.h>
```

Data Fields

- int L1_heat_tracking
- int L2_rolling_window
- int L3_linear_decay
- int L4_pipelining
- int L5_window_inference
- int L6_decay_inference
- int L7_adaptive_heartrate

6.123.1 Field Documentation

6.123.1.1 L1_heat_tracking

```
int ssm_config_t::L1_heat_tracking
```

Referenced by ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.123.1.2 L2_rolling_window

```
int ssm_config_t::L2_rolling_window
```

Referenced by ssm_apply_mode(), ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.123.1.3 L3_linear_decay

```
int ssm_config_t::L3_linear_decay
```

Referenced by physics_metadata_apply_linear_decay(), ssm_apply_mode(), ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.123.1.4 L4_pipelining

```
int ssm_config_t::L4_pipelining
```

Referenced by physics_pre_execute(), ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.123.1.5 L5_window_inference

```
int ssm_config_t::L5_window_inference
```

Referenced by inference_engine_run(), ssm_apply_mode(), ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.123.1.6 L6_decay_inference

```
int ssm_config_t::L6_decay_inference
```

Referenced by inference_engine_run(), ssm_apply_mode(), ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.123.1.7 L7_adaptive_heartrate

```
int ssm_config_t::L7_adaptive_heartrate
```

Referenced by ssm_apply_mode_from_table(), ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

The documentation for this struct was generated from the following file:

- v3/include/ssm_jacquard.h

6.124 ssm_l8_metrics_t Struct Reference

`#include <ssm_jacquard.h>`

Data Fields

- double entropy
- double cv
- double temporal_decay
- double stability_score
- int inference_ran_this_tick
- int inference_early_exited

6.124.1 Field Documentation

6.124.1.1 cv

`double ssm_l8_metrics_t::cv`

Referenced by infer_word_l8_update(), ssm_l8_update(), ssm_l8_update_table(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.124.1.2 entropy

`double ssm_l8_metrics_t::entropy`

Referenced by infer_word_l8_update(), ssm_l8_update(), ssm_l8_update_table(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.124.1.3 inference_early_exited

`int ssm_l8_metrics_t::inference_early_exited`

Referenced by ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.124.1.4 inference_ran_this_tick

`int ssm_l8_metrics_t::inference_ran_this_tick`

Referenced by ssm_l8_tick_score_and_select(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.124.1.5 stability_score

`double ssm_l8_metrics_t::stability_score`

Referenced by infer_word_l8_update(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.124.1.6 temporal_decay

```
double ssm_l8_metrics_t::temporal_decay
```

Referenced by infer_word_l8_update(), ssm_l8_update(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

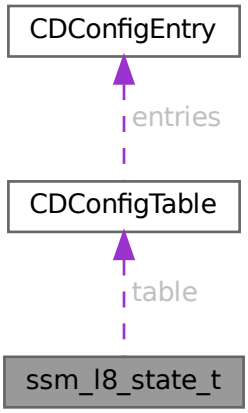
The documentation for this struct was generated from the following file:

- v3/include/ssm_jacquard.h

6.125 ssm_l8_state_t Struct Reference

```
#include <ssm_jacquard.h>
```

Collaboration diagram for ssm_l8_state_t:



Data Fields

- ssm_l8_mode_t current_mode
- uint32_t hysteresis_counter
- ssm_l8_mode_t pending_mode
- CDConfigTable * table

6.125.1 Field Documentation

6.125.1.1 current_mode

```
ssm_l8_mode_t ssm_l8_state_t::current_mode
```

Referenced by infer_word_l8_mode(), ssm_apply_mode(), ssm_l8_init(), ssm_l8_update(), ssm_l8_update_table(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

6.125.1.2 hysteresis_counter

```
uint32_t ssm_l8_state_t::hysteresis_counter
```

Referenced by ssm_l8_init(), and ssm_l8_update().

6.125.1.3 pending_mode

```
ssm_l8_mode_t ssm_l8_state_t::pending_mode
```

Referenced by ssm_l8_init(), and ssm_l8_update().

6.125.1.4 table

```
CDConfigTable* ssm_l8_state_t::table
```

Referenced by l8_regime_modulation_q16(), ssm_apply_mode_from_table(), ssm_l8_force_config(), ssm_l8_free_table(), ssm_l8_init(), ssm_l8_init_table(), ssm_l8_update_table(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

The documentation for this struct was generated from the following file:

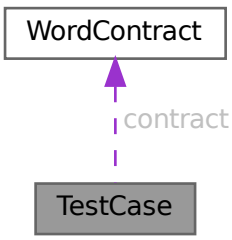
- v3/include/ssm_jacquard.h

6.126 TestCase Struct Reference

Structure representing a single test case.

```
#include <test_common.h>
```

Collaboration diagram for TestCase:



Data Fields

- const char * name
  Name of the test case.
- const char * input
  Input Forth code.
- const char * expected
  Expected output.
- TestType type
  Type of test case.
- int should_error
  Whether test should produce an error.
- int implemented
  Whether the functionality is implemented.
- WordContract contract
  Per-test contract override; zero = inherit from suite/module.

6.126.1 Detailed Description

Structure representing a single test case.

6.126.2 Field Documentation

6.126.2.1 contract

```
WordContract TestCase::contract
```

Per-test contract override; zero = inherit from suite/module.

Referenced by run_test_suite_m().

6.126.2.2 expected

```
const char* TestCase::expected
```

Expected output.

Referenced by run_single_test().

6.126.2.3 implemented

```
int TestCase::implemented
```

Whether the functionality is implemented.

Referenced by run_single_test(), run_test_suite(), and run_test_suite_m().

6.126.2.4 input

```
const char* TestCase::input
```

Input Forth code.

Referenced by run_single_test(), and run_test_suite_m().

6.126.2.5 name

```
const char* TestCase::name
```

Name of the test case.

Referenced by run_single_test(), run_test_suite(), and run_test_suite_m().

6.126.2.6 should_error

```
int TestCase::should_error
```

Whether test should produce an error.

Referenced by run_single_test(), and run_test_suite_m().

6.126.2.7 type

```
TestType TestCase::type
```

Type of test case.

Referenced by run_test_suite_m().

The documentation for this struct was generated from the following file:

- v3/src/test_runner/include/test_common.h

6.127 TestModule Struct Reference

Structure representing a test module.

```
#include <test_runner.h>
```

Data Fields

- const char * module_name
  Name of the test module.
- void * reserved
  Reserved for future use.
- int reserved_int
  Reserved integer value.
- void(* run_module_tests)(VM *vm)
  Function pointer to run module tests.

6.127.1 Detailed Description

Structure representing a test module.

6.127.2 Field Documentation

6.127.2.1 module_name

```
const char* TestModule::module_name
```

Name of the test module.

Referenced by run_all_tests(), run_module_tests(), and run_word_tests().

6.127.2.2 reserved

```
void* TestModule::reserved
```

Reserved for future use.

6.127.2.3 reserved_int

```
int TestModule::reserved_int
```

Reserved integer value.

6.127.2.4 run_module_tests

```
void(* TestModule::run_module_tests) (VM *vm)
```

Function pointer to run module tests.

Referenced by run_module_tests().

The documentation for this struct was generated from the following file:

- v3/src/test_runner/include/test_runner.h

6.128 TestStats Struct Reference

Structure holding test execution statistics.

`#include <test_runner.h>`

Data Fields

- int total_tests
  Total number of tests executed.
- int total_pass
  Number of passed tests.
- int total_fail
  Number of failed tests.
- int total_skip
  Number of skipped tests.
- int total_error
  Number of tests that resulted in errors.

6.128.1 Detailed Description

Structure holding test execution statistics.

6.128.2 Field Documentation

6.128.2.1 total_error

`int TestStats::total_error`

Number of tests that resulted in errors.

Referenced by run_all_tests(), run_stress_tests(), sk_parity_collect(), and test_stats_accumulate().

6.128.2.2 total_fail

`int TestStats::total_fail`

Number of failed tests.

Referenced by run_all_tests(), run_stress_tests(), sk_parity_collect(), and test_stats_accumulate().

6.128.2.3 total_pass

`int TestStats::total_pass`

Number of passed tests.

Referenced by run_all_tests(), run_stress_tests(), sk_parity_collect(), and test_stats_accumulate().

6.128.2.4 total_skip

```
int TestStats::total_skip
```

Number of skipped tests.

Referenced by run_all_tests(), run_stress_tests(), sk_parity_collect(), and test_stats_accumulate().

6.128.2.5 total_tests

```
int TestStats::total_tests
```

Total number of tests executed.

Referenced by run_all_tests(), run_break_me_tests(), run_stress_tests(), sk_parity_collect(), and test_stats_accumulate().

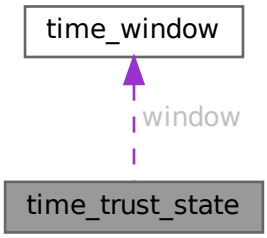
The documentation for this struct was generated from the following file:

- v3/src/test_runner/include/test_runner.h

6.129 time_trust_state Struct Reference

```
#include <timer.h>
```

Collaboration diagram for time_trust_state:



Data Fields

- volatile uint64_t ticks
- uint64_t last_tsc
- uint64_t expected_delta
- TimeWindow window
- q48_16_t variance
- q48_16_t trust
- uint64_t total_samples

6.129.1 Field Documentation

6.129.1.1 expected_delta

```
uint64_t time_trust_state::expected_delta
```

Referenced by heartbeat_init(), and heartbeat_service().

6.129.1.2 last_tsc

```
uint64_t time_trust_state::last_tsc
```

Referenced by heartbeat_init(), and heartbeat_service().

6.129.1.3 ticks

```
volatile uint64_t time_trust_state::ticks
```

Referenced by doe_log_tick_row(), heartbeat_init(), heartbeat_tick(), and heartbeat_ticks().

6.129.1.4 total_samples

```
uint64_t time_trust_state::total_samples
```

Referenced by heartbeat_init(), and heartbeat_service().

6.129.1.5 trust

```
q48_16_t time_trust_state::trust
```

Referenced by doe_log_tick_row(), heartbeat_init(), heartbeat_service(), and heartbeat_trust().

6.129.1.6 variance

```
q48_16_t time_trust_state::variance
```

Referenced by doe_log_tick_row(), heartbeat_init(), heartbeat_service(), and kernel_main_deep().

6.129.1.7 window

```
TimeWindow time_trust_state::window
```

Referenced by heartbeat_init(), and heartbeat_service().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/timer.h

6.130 time_window Struct Reference

`#include <timer.h>`

Data Fields

- int64_t deltas [64]
- uint32_t pos
- uint32_t count

6.130.1 Field Documentation

6.130.1.1 count

`uint32_t time_window::count`

Referenced by heartbeat_init().

6.130.1.2 deltas

`int64_t time_window::deltas[64]`

Referenced by heartbeat_init().

6.130.1.3 pos

`uint32_t time_window::pos`

Referenced by heartbeat_init().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/timer.h

6.131 timer_calibration_record Struct Reference

`#include <timer.h>`

Data Fields

- uint64_t hpet_hz
- uint64_t tsc_hz_mean
- uint64_t pit_hz_mean
- uint64_t cv_hpet_ppm
- uint64_t cv_pit_ppm
- uint64_t diff_ppm
- uint32_t windows_used
- uint8_t converged
- uint8_t vm_mode
- uint8_t trust
- uint8_t reserved [1]

6.131.1 Field Documentation

6.131.1.1 converged

```
uint8_t timer_calibration_record::converged
```

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), and timer_init().

6.131.1.2 cv_hpet_ppm

```
uint64_t timer_calibration_record::cv_hpet_ppm
```

Referenced by converge_tsc_bare_metal(), and init_timer_vm_mode().

6.131.1.3 cv_pit_ppm

```
uint64_t timer_calibration_record::cv_pit_ppm
```

Referenced by converge_tsc_bare_metal(), and init_timer_vm_mode().

6.131.1.4 diff_ppm

```
uint64_t timer_calibration_record::diff_ppm
```

Referenced by converge_tsc_bare_metal(), and init_timer_vm_mode().

6.131.1.5 hpet_hz

```
uint64_t timer_calibration_record::hpet_hz
```

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), and timer_init().

6.131.1.6 pit_hz_mean

```
uint64_t timer_calibration_record::pit_hz_mean
```

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), and timer_init().

6.131.1.7 reserved

```
uint8_t timer_calibration_record::reserved[1]
```

6.131.1.8 trust

```
uint8_t timer_calibration_record::trust
```

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), timer_init(), and timer_now_ns().

6.131.1.9 tsc_hz_mean

```
uint64_t timer_calibration_record::tsc_hz_mean
```

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), and timer_init().

6.131.1.10 vm_mode

```
uint8_t timer_calibration_record::vm_mode
```

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), timer_check_drift_now(), timer_init(), and timer_now_ns().

6.131.1.11 windows_used

```
uint32_t timer_calibration_record::windows_used
```

Referenced by converge_tsc_bare_metal(), and init_timer_vm_mode().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/timer.h

6.132 timespec Struct Reference

```
#include <time.h>
```

Data Fields

- time_t tv_sec
- long tv_nsec

6.132.1 Field Documentation

6.132.1.1 tv_nsec

```
long timespec::tv_nsec
```

Referenced by heartbeat_thread_main(), posix_get_monotonic_ns(), posix_get_realtime_ns(), posix_set_realtime_ns(), and snapshot_posix().

6.132.1.2 tv_sec

```
time_t timespec::tv_sec
```

Referenced by heartbeat_thread_main(), posix_get_monotonic_ns(), posix_get_realtime_ns(), posix_set_realtime_ns(), and snapshot_posix().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/freestanding/time.h

6.133 timeval Struct Reference

```
#include <time.h>
```

Data Fields

- time_t tv_sec
- suseconds_t tv_usec

6.133.1 Field Documentation

6.133.1.1 tv_sec

```
time_t timeval::tv_sec
```

Referenced by sf_terminal_ready().

6.133.1.2 tv_usec

```
suseconds_t timeval::tv_usec
```

Referenced by sf_terminal_ready().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/freestanding/sys/time.h

6.134 timezone Struct Reference

```
#include <time.h>
```

Data Fields

- int tz_minuteswest
- int tz_dsttime

6.134.1 Field Documentation

6.134.1.1 tz_dsttime

```
int timezone::tz_dsttime
```

6.134.1.2 tz_minuteswest

```
int timezone::tz_minuteswest
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/freestanding/sys/time.h

6.135 ttf_bitmap_t Struct Reference

A caller-owned 8-bit-per-pixel bitmap.

```
#include <ttf.h>
```

Data Fields

- uint8_t * pixels
- uint32_t width
- uint32_t height

6.135.1 Detailed Description

A caller-owned 8-bit-per-pixel bitmap.

`ttf_rasterize_glyph()` writes `fill_value` into covered pixels and leaves everything else untouched (it does not clear the buffer first — caller's job, so repeated rasterization into the same bitmap, e.g. for a text run, composites correctly without an extra clear between glyphs).

6.135.2 Field Documentation

6.135.2.1 height

```
uint32_t ttf_bitmap_t::height
```

Referenced by ttf_draw_glyph_cell().

6.135.2.2 pixels

```
uint8_t* ttf_bitmap_t::pixels
```

Referenced by ttf_draw_glyph_cell().

6.135.2.3 width

```
uint32_t ttf_bitmap_t::width
```

Referenced by ttf_draw_glyph_cell().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/ttf.h

6.136 ttf_flat_point_t Struct Reference

Data Fields

- q48_16_t x
- q48_16_t y

6.136.1 Field Documentation

6.136.1.1 x

```
q48_16_t ttf_flat_point_t::x
```

6.136.1.2 y

```
q48_16_t ttf_flat_point_t::y
```

The documentation for this struct was generated from the following file:

- kernel/src/hal/ttf.c

6.137 ttf_font_t Struct Reference

Parsed font handle.

```
#include <ttf.h>
```

Data Fields

- const uint8_t * data
- uint32_t size
- uint32_t head_off
- uint32_t maxp_off
- uint32_t loca_off
- uint32_t loca_len
- uint32_t glyf_off
- uint32_t glyf_len
- uint16_t units_per_em
- int16_t index_to_loc_format
- uint16_t num_glyphs
- uint32_t cmap_subtable_off
- int has_cmap
- uint32_t hmtx_off
- uint16_t num_h_metrics

6.137.1 Detailed Description

Parsed font handle.

Borrows the caller's buffer (does not copy or own it) — the buffer must outlive the ttf_font_t.

6.137.2 Field Documentation

6.137.2.1 cmap_subtable_off

```
uint32_t ttf_font_t::cmap_subtable_off
```

Referenced by ttf_codepoint_to_glyph(), and ttf_parse().

6.137.2.2 data

```
const uint8_t* ttf_font_t::data
```

Referenced by decode_composite_glyph(), decode_simple_glyph(), ttf_codepoint_to_glyph(), ttf_glyph_advance_width(), ttf_glyph_header(), and ttf_parse().

6.137.2.3 glyf_len

```
uint32_t ttf_font_t::glyf_len
```

Referenced by ttf_parse().

6.137.2.4 glyf_off

```
uint32_t ttf_font_t::glyf_off
```

Referenced by ttf_glyph_header(), and ttf_parse().

6.137.2.5 has_cmap

```
int ttf_font_t::has_cmap
```

Referenced by ttf_codepoint_to_glyph(), and ttf_parse().

6.137.2.6 head_off

```
uint32_t ttf_font_t::head_off
```

Referenced by ttf_parse().

6.137.2.7 hmtx_off

```
uint32_t ttf_font_t::hmtx_off
```

Referenced by ttf_glyph_advance_width(), and ttf_parse().

6.137.2.8 index_to_loc_format

```
int16_t ttf_font_t::index_to_loc_format
```

Referenced by ttf_glyph_header(), and ttf_parse().

6.137.2.9 loca_len

```
uint32_t ttf_font_t::loca_len
```

Referenced by ttf_parse().

6.137.2.10 loca_off

```
uint32_t ttf_font_t::loca_off
```

Referenced by ttf_glyph_header(), and ttf_parse().

6.137.2.11 maxp_off

```
uint32_t ttf_font_t::maxp_off
```

Referenced by ttf_parse().

6.137.2.12 num_glyphs

```
uint16_t ttf_font_t::num_glyphs
```

Referenced by ttf_glyph_header(), and ttf_parse().

6.137.2.13 num_h_metrics

```
uint16_t ttf_font_t::num_h_metrics
```

Referenced by ttf_glyph_advance_width(), and ttf_parse().

6.137.2.14 size

```
uint32_t ttf_font_t::size
```

Referenced by in_bounds(), and ttf_parse().

6.137.2.15 units_per_em

```
uint16_t ttf_font_t::units_per_em
```

Referenced by ttf_parse().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/ttf.h

6.138 ttf_glyph_header_t Struct Reference

Raw glyf record header, per ?27.6/4.3.7's "done when" clause.

```
#include <ttf.h>
```

Data Fields

- int16_t num_contours
- int16_t x_min
- int16_t y_min
- int16_t x_max
- int16_t y_max
- uint32_t glyf_offset
- uint32_t glyf_length

6.138.1 Detailed Description

Raw glyf record header, per ?27.6/4.3.7's "done when" clause.

6.138.2 Field Documentation

6.138.2.1 glyf_length

```
uint32_t ttf_glyph_header_t::glyf_length
```

Referenced by decode_glyph_r(), and ttf_glyph_header().

6.138.2.2 glyf_offset

```
uint32_t ttf_glyph_header_t::glyf_offset
```

Referenced by decode_composite_glyph(), decode_simple_glyph(), and ttf_glyph_header().

6.138.2.3 num_contours

```
int16_t ttf_glyph_header_t::num_contours
```

Referenced by decode_glyph_r(), decode_simple_glyph(), and ttf_glyph_header().

6.138.2.4 x_max

```
int16_t ttf_glyph_header_t::x_max
```

Referenced by ttf_glyph_header().

6.138.2.5 x_min

```
int16_t ttf_glyph_header_t::x_min
```

Referenced by ttf_glyph_header().

6.138.2.6 y_max

```
int16_t ttf_glyph_header_t::y_max
```

Referenced by ttf_glyph_header().

6.138.2.7 y_min

```
int16_t ttf_glyph_header_t::y_min
```

Referenced by ttf_glyph_header().

The documentation for this struct was generated from the following file:

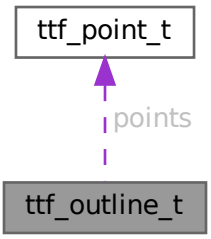
- kernel/include/starkernel/ttf.h

6.139 ttf_outline_t Struct Reference

Simple- and composite-glyph outline, flattened to one point list plus per-contour end indices (TrueType convention: contour_ends[c] is the index of the LAST point of contour c, inclusive; points are shared across contours only in the sense that contour c+1 starts right after contour_ends[c]).

`#include <ttf.h>`

Collaboration diagram for ttf_outline_t:



Data Fields

- ttf_point_t * points
- uint32_t max_points
- uint32_t point_count
- uint16_t * contour_ends
- uint32_t max_contours
- uint32_t contour_count

6.139.1 Detailed Description

Simple- and composite-glyph outline, flattened to one point list plus per-contour end indices (TrueType convention: contour_ends[c] is the index of the LAST point of contour c, inclusive; points are shared across contours only in the sense that contour c+1 starts right after contour_ends[c]).

Caller supplies both backing arrays — this module never allocates.

6.139.2 Field Documentation

6.139.2.1 contour_count

`uint32_t ttf_outline_t::contour_count`

Referenced by decode_simple_glyph(), and ttf_glyph_outline().

6.139.2.2 contour_ends

```
uint16_t* ttf_outline_t::contour_ends
```

Referenced by decode_simple_glyph().

6.139.2.3 max_contours

```
uint32_t ttf_outline_t::max_contours
```

Referenced by decode_simple_glyph().

6.139.2.4 max_points

```
uint32_t ttf_outline_t::max_points
```

Referenced by decode_simple_glyph().

6.139.2.5 point_count

```
uint32_t ttf_outline_t::point_count
```

Referenced by decode_composite_glyph(), decode_simple_glyph(), and ttf_glyph_outline().

6.139.2.6 points

```
ttf_point_t* ttf_outline_t::points
```

Referenced by decode_composite_glyph(), and decode_simple_glyph().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/ttf.h

6.140 ttf_point_t Struct Reference

One outline point, in raw font design units (NOT scaled by unitsPerEm — that's the caller's job, same convention as ttf_glyph_header_t's bbox), expressed in Q48.16.

```
#include <ttf.h>
```

Data Fields

- q48_16_t x
- q48_16_t y
- uint8_t on_curve

6.140.1 Detailed Description

One outline point, in raw font design units (NOT scaled by unitsPerEm — that's the caller's job, same convention as ttf_glyph_header_t's bbox), expressed in Q48.16.

Two's-complement negative values are expected and correct for q48_add/q48_sub and for q48_from_u64-style left-shift conversion; this module never calls q48_mul/q48_div on outline coordinates (see ttf.c's file header comment for why).

6.140.2 Field Documentation

6.140.2.1 on_curve

```
uint8_t ttf_point_t::on_curve
```

Referenced by decode_simple_glyph(), and flatten_contour().

6.140.2.2 x

```
q48_16_t ttf_point_t::x
```

Referenced by decode_composite_glyph(), decode_simple_glyph(), and flatten_contour().

6.140.2.3 y

```
q48_16_t ttf_point_t::y
```

Referenced by decode_composite_glyph(), decode_simple_glyph(), and flatten_contour().

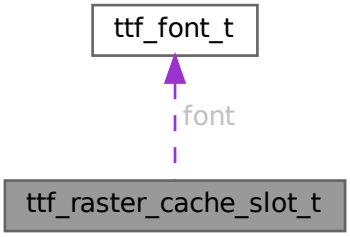
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/ttf.h

6.141 ttf_raster_cache_slot_t Struct Reference

```
#include <ttf.h>
```

Collaboration diagram for ttf_raster_cache_slot_t:



Data Fields

- int valid
- const ttf_font_t * font
- uint32_t codepoint
- uint32_t size_px
- uint32_t width
- uint32_t height
- uint32_t hits
- uint8_t pixels [((64+8 *2) *(64+8 *2))]

6.141.1 Field Documentation

6.141.1.1 codepoint

```
uint32_t ttf_raster_cache_slot_t::codepoint
```

6.141.1.2 font

```
const ttf_font_t* ttf_raster_cache_slot_t::font
```

6.141.1.3 height

```
uint32_t ttf_raster_cache_slot_t::height
```

6.141.1.4 hits

```
uint32_t ttf_raster_cache_slot_t::hits
```

6.141.1.5 pixels

```
uint8_t ttf_raster_cache_slot_t::pixels[((64+ 8 *2) *(64+ 8 *2))]
```

6.141.1.6 size_px

```
uint32_t ttf_raster_cache_slot_t::size_px
```

6.141.1.7 valid

```
int ttf_raster_cache_slot_t::valid
```

6.141.1.8 width

```
uint32_t ttf_raster_cache_slot_t::width
```

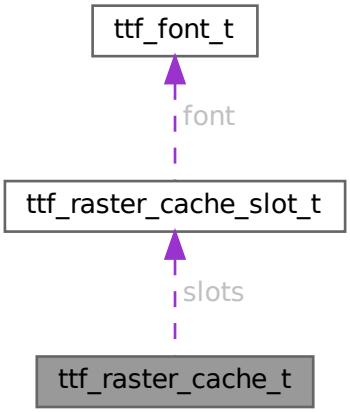
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/ttf.h

6.142 ttf_raster_cache_t Struct Reference

```
#include <ttf.h>
```

Collaboration diagram for ttf_raster_cache_t:



Data Fields

- ttf_raster_cache_slot_t * slots
- uint32_t slot_count
- uint32_t evict_next

6.142.1 Field Documentation

6.142.1.1 evict_next

```
uint32_t ttf_raster_cache_t::evict_next
```

Referenced by ttf_raster_cache_init().

6.142.1.2 slot_count

```
uint32_t ttf_raster_cache_t::slot_count
```

Referenced by ttf_raster_cache_init().

6.142.1.3 slots

```
ttf_raster_cache_slot_t* ttf_raster_cache_t::slots
```

Referenced by ttf_raster_cache_init().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/ttf.h

6.143 usb_bot_cbw_t Struct Reference

```
#include <xhci.h>
```

Data Fields

- uint32_t dCBWSignature
- uint32_t dCBWTag
- uint32_t dCBWDataTransferLength
- uint8_t bmCBWFlags
- uint8_t bCBWLUN
- uint8_t bCBWCBLength
- uint8_t CBWCB [16]

6.143.1 Field Documentation

6.143.1.1 bCBWCBLength

```
uint8_t usb_bot_cbw_t::bCBWCBLength
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), and xhci_bot_send_write10().

6.143.1.2 bCBWLUN

```
uint8_t usb_bot_cbw_t::bCBWLUN
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), and xhci_bot_send_write10().

6.143.1.3 bmCBWFlags

```
uint8_t usb_bot_cbw_t::bmCBWFlags
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), and xhci_bot_send_write10().

6.143.1.4 CBWCB

```
uint8_t usb_bot_cbw_t::CBWCB[16]
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), and xhci_bot_send_write10().

6.143.1.5 dCBWDataTransferLength

```
uint32_t usb_bot_cbw_t::dCBWDataTransferLength
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), and xhci_bot_send_write10().

6.143.1.6 dCBWSignature

```
uint32_t usb_bot_cbw_t::dCBWSignature
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), and xhci_bot_send_write10().

6.143.1.7 dCBWTag

```
uint32_t usb_bot_cbw_t::dCBWTag
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), and xhci_bot_send_write10().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.144 usb_bot_csw_t Struct Reference

```
#include <xhci.h>
```

Data Fields

- uint32_t dCSWSignature
- uint32_t dCSWTag
- uint32_t dCSWDataResidue
- uint8_t bCSWStatus

6.144.1 Field Documentation

6.144.1.1 bCSWStatus

```
uint8_t usb_bot_csw_t::bCSWStatus
```

Referenced by xhci_poll_one().

6.144.1.2 dCSWDataResidue

```
uint32_t usb_bot_csw_t::dCSWDataResidue
```

Referenced by xhci_poll_one().

6.144.1.3 dCSWSignature

```
uint32_t usb_bot_csw_t::dCSWSignature
```

Referenced by xhci_poll_one().

6.144.1.4 dCSWTag

```
uint32_t usb_bot_csw_t::dCSWTag
```

Referenced by xhci_poll_one().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.145 usb_setup_packet_t Struct Reference

```
#include <xhci.h>
```

Data Fields

- uint8_t bmRequestType
- uint8_t bRequest
- uint16_t wValue
- uint16_t wIndex
- uint16_t wLength

6.145.1 Field Documentation

6.145.1.1 bmRequestType

```
uint8_t usb_setup_packet_t::bmRequestType
```

Referenced by xhci_ep0_bot_mass_storage_reset(), xhci_ep0_control_read(), xhci_ep0_control_write_nodata(), xhci_ep0_hid_set_leds(), and xhci_ep0_hid_set_protocol().

6.145.1.2 bRequest

```
uint8_t usb_setup_packet_t::bRequest
```

6.145.1.3 wIndex

```
uint16_t usb_setup_packet_t::wIndex
```

6.145.1.4 wLength

```
uint16_t usb_setup_packet_t::wLength
```

6.145.1.5 wValue

```
uint16_t usb_setup_packet_t::wValue
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.146 user_identity_seed_t Struct Reference

```
#include <user_identity_seed.h>
```

Data Fields

- uint32_t magic
- uint32_t version
- uint8_t seed [32]
- uint8_t pubkey [32]
- char full_name [64u]
- char username [32u]
- char email [64u]
- char phone [24u]
- uint64_t crc
- uint8_t _pad [4096 -(4+4+32+32+64u+32u+64u+24u+8)]

6.146.1 Field Documentation

6.146.1.1 _pad

```
uint8_t user_identity_seed_t::_pad[4096 -(4+4+32+32+ 64u+ 32u+ 64u+ 24u+8)]
```

6.146.1.2 crc

```
uint64_t user_identity_seed_t::crc
```

Referenced by capsule_mint_identity(), capsule_wirebind_try_attach(), capsule_zuse_boot_check_key(), and verify_mint().

6.146.1.3 email

```
char user_identity_seed_t::email[64u]
```

Referenced by capsule_mint_identity().

6.146.1.4 full_name

```
char user_identity_seed_t::full_name[64u]
```

Referenced by capsule_mint_identity().

6.146.1.5 magic

```
uint32_t user_identity_seed_t::magic
```

Referenced by capsule_mint_identity(), capsule_wirebind_try_attach(), capsule_zuse_boot_check_key(), and verify_mint().

6.146.1.6 phone

```
char user_identity_seed_t::phone[24u]
```

Referenced by capsule_mint_identity().

6.146.1.7 pubkey

```
uint8_t user_identity_seed_t::pubkey[32]
```

Referenced by capsule_mint_identity(), and capsule_zuse_boot_check_key().

6.146.1.8 seed

```
uint8_t user_identity_seed_t::seed[32]
```

Referenced by capsule_mint_identity(), and capsule_zuse_boot_check_key().

6.146.1.9 username

```
char user_identity_seed_t::username[32u]
```

Referenced by capsule_mint_identity(), and capsule_wirebind_try_attach().

6.146.1.10 version

```
uint32_t user_identity_seed_t::version
```

Referenced by capsule_mint_identity(), capsule_wirebind_try_attach(), capsule_zuse_boot_check_key(), and verify_mint().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/user_identity_seed.h

6.147 VirtBlkReq Struct Reference

Data Fields

- uint32_t type
- uint32_t reserved
- uint64_t sector

6.147.1 Field Documentation

6.147.1.1 reserved

```
uint32_t VirtBlkReq::reserved
```

Referenced by vblk_io().

6.147.1.2 sector

```
uint64_t VirtBlkReq::sector
```

Referenced by vblk_io().

6.147.1.3 type

```
uint32_t VirtBlkReq::type
```

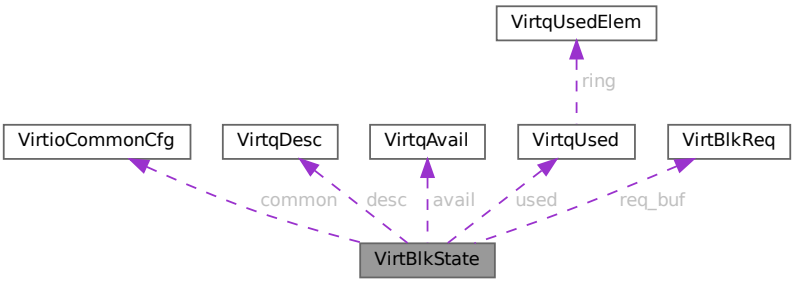
Referenced by vblk_io().

The documentation for this struct was generated from the following file:

- kernel/src/virtio/virtio_blk.c

6.148 VirtBlkState Struct Reference

Collaboration diagram for VirtBlkState:



Data Fields

- VirtioCommonCfg * common
- volatile uint16_t * notify
- uint32_t notify_off_mult
- uint16_t queue_notify_off
- VirtqDesc * desc
- VirtqAvail * avail
- VirtqUsed * used
- uint16_t avail_idx
- uint16_t last_used_idx
- uint64_t capacity_sectors
- uint32_t total_forth_blocks
- VirtBlkReq * req_buf
- uint8_t * data_buf
- uint8_t * status_buf

6.148.1 Field Documentation

6.148.1.1 avail

```
VirtqAvail* VirtBlkState::avail
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.2 avail_idx

```
uint16_t VirtBlkState::avail_idx
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.3 capacity_sectors

```
uint64_t VirtBlkState::capacity_sectors
```

Referenced by vblk_info(), and vblk_init_device().

6.148.1.4 common

```
VirtioCommonCfg* VirtBlkState::common
```

Referenced by vblk_init_device().

6.148.1.5 data_buf

```
uint8_t* VirtBlkState::data_buf
```

Referenced by vblk_init_device(), vblk_read(), and vblk_write().

6.148.1.6 desc

```
VirtqDesc* VirtBlkState::desc
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.7 last_used_idx

```
uint16_t VirtBlkState::last_used_idx
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.8 notify

```
volatile uint16_t* VirtBlkState::notify
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.9 notify_off_mult

```
uint32_t VirtBlkState::notify_off_mult
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.10 queue_notify_off

```
uint16_t VirtBlkState::queue_notify_off
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.11 req_buf

```
VirtBlkReq* VirtBlkState::req_buf
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.12 status_buf

```
uint8_t* VirtBlkState::status_buf
```

Referenced by vblk_init_device(), and vblk_io().

6.148.1.13 total_forth_blocks

```
uint32_t VirtBlkState::total_forth_blocks
```

Referenced by vblk_info(), vblk_init_device(), vblk_open(), vblk_read(), vblk_write(), and virtio_blk_find_artemis().

6.148.1.14 used

```
VirtqUsed* VirtBlkState::used
```

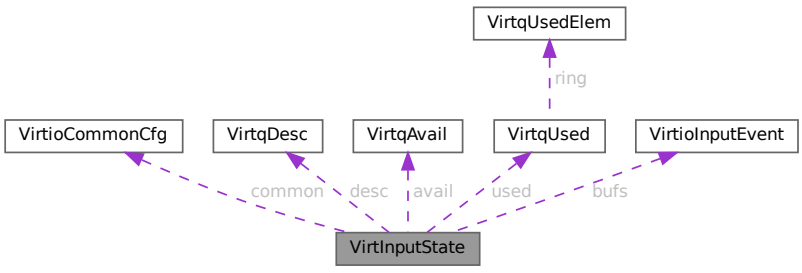
Referenced by vblk_init_device(), and vblk_io().

The documentation for this struct was generated from the following file:

- kernel/src/virtio/virtio_blk.c

6.149 VirtInputState Struct Reference

Collaboration diagram for VirtInputState:



Data Fields

- VirtioCommonCfg * common
- volatile uint16_t * notify
- uint32_t notify_off_mult
- uint16_t queue_notify_off
- volatile uint8_t * isr_status
- VirtqDesc * desc
- VirtqAvail * avail
- VirtqUsed * used
- VirtioInputEvent * bufs
- uint16_t avail_idx
- uint16_t last_used_idx

6.149.1 Field Documentation

6.149.1.1 avail

```
VirtqAvail* VirtInputState::avail
```

Referenced by vinput_init_device(), and virtio_input_isr().

6.149.1.2 avail_idx

```
uint16_t VirtInputState::avail_idx
```

Referenced by vinput_init_device(), and virtio_input_isr().

6.149.1.3 bufs

```
VirtioInputEvent* VirtInputState::bufs
```

Referenced by vinput_init_device(), and virtio_input_isr().

6.149.1.4 common

```
VirtioCommonCfg* VirtInputState::common
```

Referenced by vinput_init_device().

6.149.1.5 desc

```
VirtqDesc* VirtInputState::desc
```

Referenced by vinput_init_device().

6.149.1.6 isr_status

```
volatile uint8_t* VirtInputState::isr_status
```

Referenced by vinput_init_device(), and virtio_input_isr().

6.149.1.7 last_used_idx

```
uint16_t VirtInputState::last_used_idx
```

Referenced by vinput_init_device(), and virtio_input_isr().

6.149.1.8 notify

```
volatile uint16_t* VirtInputState::notify
```

Referenced by vinput_init_device(), and virtio_input_isr().

6.149.1.9 notify_off_mult

```
uint32_t VirtInputState::notify_off_mult
```

Referenced by vinput_init_device(), and virtio_input_isr().

6.149.1.10 queue_notify_off

```
uint16_t VirtInputState::queue_notify_off
```

Referenced by vinput_init_device(), and virtio_input_isr().

6.149.1.11 used

```
VirtqUsed* VirtInputState::used
```

Referenced by vinput_init_device(), and virtio_input_isr().

The documentation for this struct was generated from the following file:

- kernel/src/virtio/virtio_input.c

6.150 VirtioCommonCfg Struct Reference

Data Fields

- volatile uint32_t device_feature_select
- volatile uint32_t device_feature
- volatile uint32_t driver_feature_select
- volatile uint32_t driver_feature
- volatile uint16_t config_msix_vector
- volatile uint16_t num_queues
- volatile uint8_t device_status
- volatile uint8_t config_generation
- volatile uint16_t queue_select
- volatile uint16_t queue_size
- volatile uint16_t queue_msix_vector
- volatile uint16_t queue_enable
- volatile uint16_t queue_notify_off
- volatile uint64_t queue_desc
- volatile uint64_t queue_driver
- volatile uint64_t queue_device
- volatile uint16_t queue_notify_data
- volatile uint16_t queue_reset

6.150.1 Field Documentation

6.150.1.1 config_generation

```
volatile uint8_t VirtioCommonCfg::config_generation
```

6.150.1.2 config_msix_vector

```
volatile uint16_t VirtioCommonCfg::config_msix_vector
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.3 device_feature

```
volatile uint32_t VirtioCommonCfg::device_feature
```

6.150.1.4 device_feature_select

```
volatile uint32_t VirtioCommonCfg::device_feature_select
```

6.150.1.5 device_status

```
volatile uint8_t VirtioCommonCfg::device_status
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.6 driver_feature

```
volatile uint32_t VirtioCommonCfg::driver_feature
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.7 driver_feature_select

```
volatile uint32_t VirtioCommonCfg::driver_feature_select
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.8 num_queues

```
volatile uint16_t VirtioCommonCfg::num_queues
```

Referenced by vinput_init_device().

6.150.1.9 queue_desc

```
volatile uint64_t VirtioCommonCfg::queue_desc
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.10 queue_device

```
volatile uint64_t VirtioCommonCfg::queue_device
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.11 queue_driver

```
volatile uint64_t VirtioCommonCfg::queue_driver
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.12 queue_enable

```
volatile uint16_t VirtioCommonCfg::queue_enable
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.13 queue_msix_vector

```
volatile uint16_t VirtioCommonCfg::queue_msix_vector
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.14 queue_notify_data

```
volatile uint16_t VirtioCommonCfg::queue_notify_data
```

6.150.1.15 queue_notify_off

```
volatile uint16_t VirtioCommonCfg::queue_notify_off
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.16 queue_reset

```
volatile uint16_t VirtioCommonCfg::queue_reset
```

6.150.1.17 queue_select

```
volatile uint16_t VirtioCommonCfg::queue_select
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

6.150.1.18 queue_size

```
volatile uint16_t VirtioCommonCfg::queue_size
```

Referenced by vblk_init_device(), vinput_init_device(), and vrng_init_device().

The documentation for this struct was generated from the following files:

- kernel/src/virtio/virtio_blk.c
- kernel/src/virtio/virtio_input.c
- kernel/src/virtio/virtio_rng.c

6.151 VirtioInputEvent Struct Reference

```
#include <virtio_input.h>
```

Data Fields

- uint16_t type
- uint16_t code
- uint32_t value

6.151.1 Field Documentation

6.151.1.1 code

```
uint16_t VirtioInputEvent::code
```

Referenced by virtio_input_isr().

6.151.1.2 type

```
uint16_t VirtioInputEvent::type
```

Referenced by virtio_input_isr().

6.151.1.3 value

```
uint32_t VirtioInputEvent::value
```

Referenced by virtio_input_isr().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/virtio_input.h

6.152 VirtqAvail Struct Reference

Data Fields

- uint16_t flags
- uint16_t idx
- uint16_t ring [4u]
- uint16_t used_event

6.152.1 Field Documentation

6.152.1.1 flags

```
uint16_t VirtqAvail::flags
```

6.152.1.2 idx

```
uint16_t VirtqAvail::idx
```

Referenced by vblk_io(), vinput_init_device(), virtio_input_isr(), and vrng_request().

6.152.1.3 ring

```
uint16_t VirtqAvail::ring
```

Referenced by vblk_io(), vinput_init_device(), virtio_input_isr(), and vrng_request().

6.152.1.4 used_event

```
uint16_t VirtqAvail::used_event
```

The documentation for this struct was generated from the following files:

- kernel/src/virtio/virtio_blk.c
- kernel/src/virtio/virtio_input.c
- kernel/src/virtio/virtio_rng.c

6.153 VirtqDesc Struct Reference

Data Fields

- uint64_t addr
- uint32_t len
- uint16_t flags
- uint16_t next

6.153.1 Field Documentation

6.153.1.1 addr

```
uint64_t VirtqDesc::addr
```

Referenced by vblk_io(), vinput_init_device(), and vrng_request().

6.153.1.2 flags

```
uint16_t VirtqDesc::flags
```

Referenced by vblk_io(), vinput_init_device(), and vrng_request().

6.153.1.3 len

```
uint32_t VirtqDesc::len
```

Referenced by vblk_io(), vinput_init_device(), and vrng_request().

6.153.1.4 next

```
uint16_t VirtqDesc::next
```

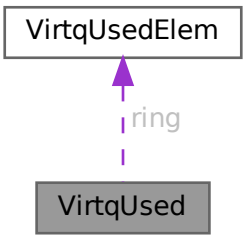
Referenced by vblk_io(), vinput_init_device(), and vrng_request().

The documentation for this struct was generated from the following files:

- kernel/src/virtio/virtio_blk.c
- kernel/src/virtio/virtio_input.c
- kernel/src/virtio/virtio_rng.c

6.154 VirtqUsed Struct Reference

Collaboration diagram for VirtqUsed:



Data Fields

- uint16_t flags
- uint16_t idx
- VirtqUsedElem ring [4u]
- uint16_t avail_event

6.154.1 Field Documentation

6.154.1.1 avail_event

```
uint16_t VirtqUsed::avail_event
```

6.154.1.2 flags

```
uint16_t VirtqUsed::flags
```

6.154.1.3 idx

```
uint16_t VirtqUsed::idx
```

Referenced by vblk_io(), virtio_input_isr(), and vrng_request().

6.154.1.4 ring

```
VirtqUsedElem VirtqUsed::ring
```

Referenced by virtio_input_isr(), and vrng_request().

The documentation for this struct was generated from the following files:

- kernel/src/virtio/virtio_blk.c
- kernel/src/virtio/virtio_input.c
- kernel/src/virtio/virtio_rng.c

6.155 VirtqUsedElem Struct Reference

Data Fields

- uint32_t id
- uint32_t len

6.155.1 Field Documentation

6.155.1.1 id

```
uint32_t VirtqUsedElem::id
```

Referenced by virtio_input_isr().

6.155.1.2 len

```
uint32_t VirtqUsedElem::len
```

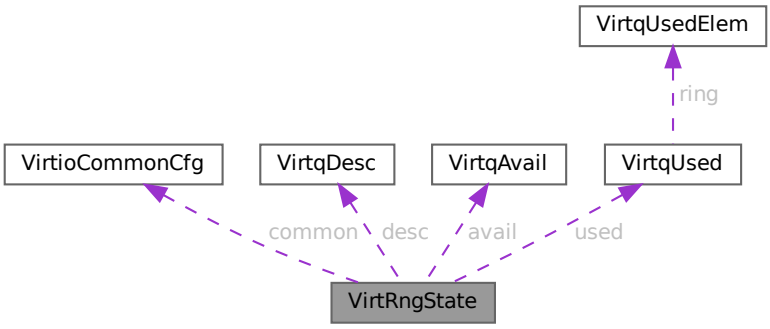
Referenced by vrng_request().

The documentation for this struct was generated from the following files:

- kernel/src/virtio/virtio_blk.c
- kernel/src/virtio/virtio_input.c
- kernel/src/virtio/virtio_rng.c

6.156 VirtRngState Struct Reference

Collaboration diagram for VirtRngState:



Data Fields

- VirtioCommonCfg * common
- volatile uint16_t * notify
- uint32_t notify_off_mult
- uint16_t queue_notify_off
- VirtqDesc * desc
- VirtqAvail * avail
- VirtqUsed * used
- uint16_t avail_idx
- uint16_t last_used_idx
- uint8_t * data_buf

6.156.1 Field Documentation

6.156.1.1 avail

```
VirtqAvail* VirtRngState::avail
```

Referenced by vrng_init_device(), and vrng_request().

6.156.1.2 avail_idx

```
uint16_t VirtRngState::avail_idx
```

Referenced by vrng_init_device(), and vrng_request().

6.156.1.3 common

```
VirtioCommonCfg* VirtRngState::common
```

Referenced by vrng_init_device().

6.156.1.4 data_buf

```
uint8_t* VirtRngState::data_buf
```

Referenced by virtio_rng_get_bytes(), vrng_init_device(), and vrng_request().

6.156.1.5 desc

```
VirtqDesc* VirtRngState::desc
```

Referenced by vrng_init_device(), and vrng_request().

6.156.1.6 last_used_idx

```
uint16_t VirtRngState::last_used_idx
```

Referenced by vrng_init_device(), and vrng_request().

6.156.1.7 notify

```
volatile uint16_t* VirtRngState::notify
```

Referenced by vrng_init_device(), and vrng_request().

6.156.1.8 notify_off_mult

```
uint32_t VirtRngState::notify_off_mult
```

Referenced by vrng_init_device(), and vrng_request().

6.156.1.9 queue_notify_off

```
uint16_t VirtRngState::queue_notify_off
```

Referenced by vrng_init_device(), and vrng_request().

6.156.1.10 used

```
VirtqUsed* VirtRngState::used
```

Referenced by vrng_init_device(), and vrng_request().

The documentation for this struct was generated from the following file:

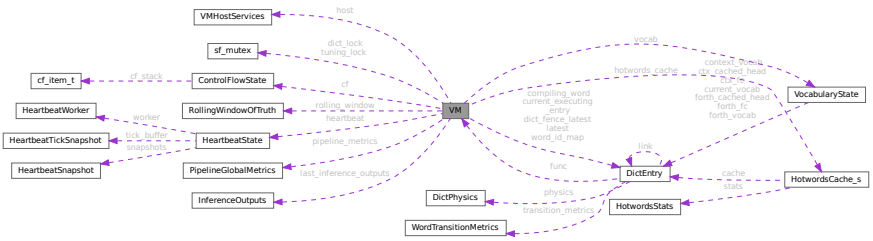
- kernel/src/virtio/virtio_rng.c

6.157 VM Struct Reference

Main virtual machine state container.

```
#include <vm.h>
```

Collaboration diagram for VM:



Data Fields

Stack Management

- cell_t data_stack [1024]
  Parameter stack storage.
- cell_t return_stack [1024]
  Return stack storage.
- int dsp
  Data stack pointer.
- int rsp
  Return stack pointer.
- int exit_colon
  Exit flag for colon definitions.
- int abort_requested
  ABORT flag for immediate termination.
- int ecw_nesting
  Depth of execute_colon_word call stack; >0 means inside a colon word.
- volatile int doe_row_printing
  Non-zero while EMIT-ROW assembles a row; heartbeat output must defer.
- uint8_t emergency_console
  1 = fault handler active; bypasses all ACL checks (C-only write)
- uint8_t zuse_session
  1 = zuse authenticated at console; shows zuse)ok> prompt
- uint8_t acl_skip

1 = skip all ACL hooks (Ananke enforcement VM; prevents recursion)

- uint8_t zuse_cert_installed

  1 = zuse_cert_seed/pubkey hold a real minted cert (one-time fuse)
- uint8_t zuse_cert_seed [32]

  Ed25519 seed, this instance's Zuse identity.
- uint8_t zuse_cert_pubkey [32]

  Ed25519 public key derived from the seed at mint time, cached here so callers don't need to re-derive it.
- uint8_t zuse_root_pubkey_known

  1 = zuse_root_pubkey holds a real value, loaded from the persistent genesis-marker fence (zuse_genesis_marker_t, capsule_zuse_boot_load_root_pubkey()) independently of whether Zuse's own thumbdrive is attached this boot.
- uint8_t zuse_root_pubkey [32]

  Ed25519 public key, loaded from the genesis-marker fence.

Dictionary Management

- const VMHostServices * host

  Host allocator/time services.
- uint8_t * memory

  Unified VM memory buffer.
- uintptr_t here

  Next free memory location (byte offset or absolute pointer if M7)
- DictEntry * latest

  Most recent word.
- DictEntry * word_id_map [1024]

  Stable ID → entry map for speculation.
- uint32_t recycled_word_ids [1024]

  Reusable ID stack for FORGET.
- uint32_t recycled_word_id_count

  Depth of recycled stack.
- uint32_t next_word_id

  Next fresh ID when recycle stack empty.
- DictEntry * dict_fence_latest
- size_t dict_fence_here
- sf_mutex_t dict_lock

  Protects dictionary structural mutations.
- char input_buffer [1025]
- size_t input_length
- size_t input_pos
- vm_mode_t mode
- DictEntry * compiling_word
- int interpreter_enabled
- char current_word_name [31+1]
- cell_t * compile_buffer
- size_t compile_pos
- size_t compile_size
- cell_t state_var
- DictEntry * current_executing_entry
- cell_t * ip
- int error
- int halted
- cell_t base
- unsigned char * tib_buf
- size_t tib_cap
- cell_t * in_var
- cell_t * span_var
- vaddr_t scr_addr
- vaddr_t state_addr
- vaddr_t base_addr
- vaddr_t hold_addr
- int hold_pos
- VocabularyState vocab

- ControlFlowState cf
- uint32_t blk_vm_lbn [4]
- uint8_t * blk_vm_cbuf [4]
- uint8_t blk_vm_dirty [4]
- int blk_vm_next
- uint64_t blk_vm_epoch

Physics Hot-Words Cache

- HotwordsCache * hotwords_cache
  Frequency-driven dictionary acceleration cache.

Rolling Window of Truth (Deterministic Metrics)

- RollingWindowOfTruth rolling_window
  Execution history for reproducible seeding.

Phase 2: Heat-Aware Dictionary Optimization

- cell_t heat_threshold_25th
  25th percentile heat (bucket search priority)
- cell_t heat_threshold_50th
  50th percentile heat
- cell_t heat_threshold_75th
  75th percentile heat
- uint64_t last_bucket_reorg_ns
  When we last reorganized buckets.
- int lookup_strategy
  0=naive, 1=heat-aware, 2=inference-reorg

VM Heartbeat (Loop #3 & #5 Coordination)

- HeartbeatState heartbeat
  Centralized time-driven tuning dispatcher.
- uint32_t heartbeat_decay_cursor_id
  Continuation cursor for background decay sweeps.

Pipelining Global Metrics (Loop #4 & #5 Feedback)

- PipelineGlobalMetrics pipeline_metrics
  Aggregated prefetch accuracy tracking.

Physics Loop #3: Adaptive Heat Decay Tuning

- uint64_t decay_slope_q48
  Current decay rate (Q48.16 fixed-point, starts at 2:1 = 131072)
- uint64_t last_decay_check_ns
  Timestamp of last decay effectiveness measurement.
- uint64_t total_heat_at_last_check
  Total dictionary heat at last check (for slope inference)
- uint64_t hot_word_count_at_check
  Hot words observed at last inference.
- uint64_t stale_word_count_at_check
  Stale words at last check (for trend analysis)
- uint32_t word_count_at_check
  Total word count at last check (for ratio calculation)

- int decay_slope_direction
  -1 = decrease slope, 0 = stable, +1 = increase slope (inference)
- sf_mutex_t tuning_lock
  Guards shared tuning knobs (slope, snapshots)

Unified Inference Engine (Phase 2: Replacing Loops #3 & #5)

- InferenceOutputs * last_inference_outputs
  Cached inference outputs (for ANOVA early-exit and tuning application)

SSM L8: Jacquard Mode Selector (Steady State Machine)

- void * ssm_l8_state
  Opaque pointer to ssm_l8_state_t (avoid circular include)
- void * ssm_config
  Opaque pointer to ssm_config_t (L3/L4 mode bits)

6.157.1 Detailed Description

Main virtual machine state container.

Contains all state information for a StarForth VM instance including stacks, dictionary, memory, and execution state.

6.157.2 Field Documentation

6.157.2.1 abort_requested

```
int VM::abort_requested
```

ABORT flag for immediate termination.

Referenced by block_word_next_block(), block_word_thru(), check_physics_transparent(), defining_runtime_dodoes(), exec_block_with_retry(), execute_colon_word(), restore_vm_state(), sk_repl_run(), sk_repl_step(), system_word_abort(), test_acl_deny_aborts(), vm_init(), vm_init_with_host(), vm_interpret(), and vm_repl().

6.157.2.2 acl_skip

```
uint8_t VM::acl_skip
```

1 = skip all ACL hooks (Ananke enforcement VM; prevents recursion)

6.157.2.3 base

```
cell_t VM::base
```

Referenced by current_base(), dbg_get_base(), vm_get_base(), vm_set_base(), and vm_set_base().

6.157.2.4 base_addr

```
vaddr_t VM::base_addr
```

Referenced by dbg_get_base(), format_word_base(), vm_get_base(), vm_init(), vm_init_with_host(), vm_set_base(), and vm_set_base().

6.157.2.5 blk_vm_cbuf

```
uint8_t* VM::blk_vm_cbuf[4]
```

Referenced by blk_vm_assign(), blk_vm_check_epoch(), blk_vm_evict(), blk_vm_find(), blk_vm_flush_all(), blk_vm_load(), block_word_empty_buffers(), and block_word_update().

6.157.2.6 blk_vm_dirty

```
uint8_t VM::blk_vm_dirty[4]
```

Referenced by blk_vm_assign(), blk_vm_check_epoch(), blk_vm_evict(), blk_vm_flush_all(), blk_vm_load(), block_word_empty_buffers(), and block_word_update().

6.157.2.7 blk_vm_epoch

```
uint64_t VM::blk_vm_epoch
```

Referenced by blk_vm_check_epoch().

6.157.2.8 blk_vm_lbn

```
uint32_t VM::blk_vm_lbn[4]
```

Referenced by blk_vm_assign(), blk_vm_check_epoch(), blk_vm_evict(), blk_vm_find(), blk_vm_flush_all(), blk_vm_is_cross_device_copy(), blk_vm_load(), and block_word_empty_buffers().

6.157.2.9 blk_vm_next

```
int VM::blk_vm_next
```

Referenced by blk_vm_evict().

6.157.2.10 cf

```
ControlFlowState VM::cf
```

Referenced by cf_epoch_sync(), cf_peek_item(), cf_pop_item(), cf_push_item(), control_forth_case(), control_forth_do(), control_forth_endcase(), control_forth_endof(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), vm_init(), and vm_init_with_host().

6.157.2.11 compile_buffer

```
cell_t* VM::compile_buffer
```

6.157.2.12 compile_pos

```
size_t VM::compile_pos
```

6.157.2.13 compile_size

```
size_t VM::compile_size
```

6.157.2.14 compiling_word

```
DictEntry* VM::compiling_word
```

Referenced by restore_vm_state(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_init(), and vm_init_with_host().

6.157.2.15 current_executing_entry

```
DictEntry* VM::current_executing_entry
```

Referenced by acl_recheck(), acl_recheck(), defer_runtime(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_variable(), execute_colon_word(), restore_vm_state(), starforth_word_init(), system_word_execute(), vm_debug_dump_state(), vm_docol_arm64(), vm_docol_asm(), vm_docol_rv64(), vm_init(), vm_init_with_host(), vm_interpret_word(), and vocabulary_select_runtime().

6.157.2.16 current_word_name

```
char VM::current_word_name[31+1]
```

Referenced by vm_enter_compile_mode().

6.157.2.17 data_stack

```
cell_t VM::data_stack[1024]
```

Parameter stack storage.

Referenced by assert_stack_top(), capture_exec_result(), double_word_2dup(), double_word_2over(), double_word_dabs(), format_word_dot_s(), format_word_hash_s(), forth_BENCH_DICT_LOOKUP(), forth_COMPARE_LOOKUPS(), forth_DECAY_RATE_FETCH(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_PERCENTILES(), forth_HEAT_STORE(), forth_LOOKUP_STRATEGY_FETCH(), forth_LOOKUP_STRATEGY_STORE(), forth_PHYSICS_BURN(), forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), io_word_emit(), io_word_key(), io_word_question_terminal(), io_word_spaces(), stack_word_depth(), stack_word_dup(), stack_word_minus_rot(), stack_word_over(), stack_word_pick(), stack_word_question_dup(), stack_word_roll(), stack_word_rot(), stack_word_swap(), vm_pop(), vm_pop_asm(), vm_pop_asm(), vm_pop_asm(), vm_pop_fast(), vm_push(), vm_push_asm(), vm_push_asm(), vm_push_asm(), vm_push_fast(), vm_save_registers(), vm_save_registers_arm64(), vm_save_registers_rv64(), vm_setup_registers(), vm_setup_registers_arm64(), vm_setup_registers_rv64(), and vocabulary_word_paren_find().

6.157.2.18 decay_slope_direction

```
int VM::decay_slope_direction
```

-1 = decrease slope, 0 = stable, +1 = increase slope (inference)

Referenced by vm_init(), vm_init_with_host(), and vm_tick_slope_validator().

6.157.2.19 decay_slope_q48

```
uint64_t VM::decay_slope_q48
```

Current decay rate (Q48.16 fixed-point, starts at 2:1 = 131072)

Referenced by heartbeat_publish_snapshot(), metrics_from_vm(), perturb_physics(), physics_decay_slope_load(), restore_physics(), save_physics(), vm_heartbeat_publish_snapshot(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_slope_validator().

6.157.2.20 dict_fence_here

```
size_t VM::dict_fence_here
```

Referenced by dictionary_word_fence(), dictionary_word_forget(), main(), vm_init(), and vm_init_with_host().

6.157.2.21 dict_fence_latest

```
DictEntry* VM::dict_fence_latest
```

Referenced by dictionary_word_fence(), dictionary_word_forget(), main(), vm_init(), and vm_init_with_host().

6.157.2.22 dict_lock

```
sf_mutex_t VM::dict_lock
```

Protects dictionary structural mutations.

Referenced by dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), dictionary_word_fence(), dictionary_word_forget(), extract_heat_trajectory(), infer_word_run(), metrics_from_vm(), physics_on_lookup(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), vm_cleanup(), vm_create_word(), vm_dict_entry_ok(), vm_find_word(), vm_init(), vm_init_with_host(), vm_interpret_word(), vm_tick_apply_background_decay(), vm_tick_apply_background_decay(), vm_tick_inference_engine(), and vm_tick_slope_validator().

6.157.2.23 doe_row_printing

```
volatile int VM::doe_row_printing
```

Non-zero while EMIT-ROW assembles a row; heartbeat output must defer.

6.157.2.24 dsp

```
int VM::dsp
```

Data stack pointer.

Referenced by arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), assert_stack_depth(), assert_stack_top(), block_word_acl_allow_fetch(), block_word_acl_allow_store(), block_word_acl_ttl_fetch(), block_word_acl_ttl_store(), block_word_blk_attach(), block_word_block(), block_word_buffer(), block_word_confirm_format(), block_word_list(), block_word_load(), block_word_owner_fetch(), block_word_relocate(), block_word_thru(), capture_exec_result(), check_physics_transparent(), control_forth_0branch(), control_forth_runtime_do(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), defining_word_constant(), defining_word_literal(), dictionary_m_word_cfa(), dictionary_m_word_link_from(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), dictionary_m_word_traverse(), dictionary_word_2comma(), dictionary_word_allot(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_sp_fetch(), dictionary_word_sp_store(), double_word_2drop(), double_word_2dup(), double_word_2over(), double_word_2rot(), double_word_2swap(), double_word_2to_r(), double_word_d_equals(), double_word_d_less(), double_word_d_minus(), double_word_d_plus(), double_word_d_two_slash(), double_word_d_two_star(), double_word_d_zero_equals(), double_word_d_zero_less(), double_word_dabs(), double_word_dmax(), double_word_dmin(), double_word_dnegate(), double_word_s_to_d(), editor_word_edit(), editor_word_l(), editor_word_s(), execute_colon_word(), fbw_plot(), format_word_d_dot(), format_word_d_dot_r(), format_word_dot(), format_word_dot_r(), format_word_dot_s(), format_word_dump(), format_word_end_conversion(), format_word_hash(), format_word_hash_s(), format_word_hold(), format_word_question(), format_word_sign(), format_word_u_dot(), format_word_u_dot_r(), forth_acl_allow_store(), forth_acl_inherit(), forth_acl_mode_store(), forth_acl_ttl_store(), forth_BENCH_DICT_LOOKUP(), forth_COMPARE_LOOKUPS(), forth_DECAY_RATE_FETCH(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_PERCENTILES(), forth_HEAT_STORE(), forth_LOOKUP_STRATEGY_FETCH(), forth_LOOKUP_STRATEGY_STORE(), forth_PHYSICS_BURN(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), io_word_emit(), io_word_key(), io_word_question_terminal(), io_word_spaces(), io_word_type(), log_str_emit(), log_word_set_level(), logical_word_and(), logical_word_equals(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), memory_word_2fetch(), memory_word_2store(), memory_word_cells(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), mixed_math_word_m_minus(), mixed_math_word_m_plus(), mixed_math_word_m_slash_mod(), mixed_math_word_m_star(), mixed_math_word_mod(), mixed_math_word_mod(), mixed_math_word_slash_mod(), mixed_math_word_star_slash(), mixed_math_word_star_slash_mod(), pop_xt(), reset_vm_state(), restore_vm_state(), return_stack_word_r_fetch(), return_stack_word_r_from(), return_stack_word_to_r(), runtime_abortq(), save_vm_state(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), stack_word_depth(), stack_word_drop(), stack_word_dup(), stack_word_minus_rot(), stack_word_over(), stack_word_pick(), stack_word_question_dup(), stack_word_roll(), stack_word_rot(), stack_word_swap(), starforth_word_execution_heat_fetch(), starforth_word_execution_heat_stor starforth_word_random(), starforth_word_seed(), starforth_word_top_words(), starforth_word_wait(), starforth_word_zuse_pubkey_fe string_word_blank(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_minus_trailing(), string_word_number(), string_word_scan(), string_word_search(), string_word_skip(), string_word_word(), system_word_abort_quote(), system_word_execute(), system_word_reboot(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_strict_mode(), test_emergency_console_bypass(), vm_debug_dump_state(), vm_init(), vm_init_with_host(), vm_interpret_word(), vm_pop(), vm_pop_asm(), vm_pop_asm(), vm_pop_asm(), vm_pop_fast(), vm_push(), vm_push_asm(), vm_push_asm(), vm_push_asm(), vm_push_fast(), vm_save_registers(), vm_save_registers_arm64(), vm_save_registers_rv64(), vm_setup_registers(), vm_setup_registers_arm64(), vm_setup_registers_rv64(), vocabulary_word_paren_find(), word_divide(), word_divmod(), and word_is().

6.157.2.25 ecw_nesting

```
int VM::ecw_nesting
```

Depth of execute_colon_word call stack; >0 means inside a colon word.

Referenced by double_word_2r_fetch(), double_word_2r_from(), double_word_2to_r(), execute_colon_word(), return_stack_word_r_fetch(), return_stack_word_r_from(), and return_stack_word_to_r().

6.157.2.26 emergency_console

```
uint8_t VM::emergency_console
```

1 = fault handler active; bypasses all ACL checks (C-only write)

Referenced by acl_recheck(), acl_recheck(), execute_colon_word(), run_acl_words_tests(), test_acl_deny_aborts(), test_acl_ttl_hot_path(), test_emergency_console_bypass(), and vm_interpret_word().

6.157.2.27 error

```
int VM::error
```

Referenced by acl_recheck(), acl_recheck(), arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), array_ptr(), assert_vm_error(), block_word_acl_allow_fetch(), block_word_acl_allow_store(), block_word_acl_ttl_fetch(), block_word_acl_ttl_store(), block_word_blk_attach(), block_word_block(), block_word_buffer(), block_word_confirm_format(), block_word_list(), block_word_load(), block_word_next_block(), block_word_owner_fetch(), block_word_relocate(), block_word_thru(), block_word_update(), capsule_vm_alloc_hook(), capture_exec_result(), check_physics_transparent(), control_forth_0branch(), control_forth_again(), control_forth_begin(), control_forth_branch(), control_forth_case(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_EXIT(), control_forth_I(), control_forth_if(), control_forth_J(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_runtime_do(), control_forth_runtime_leave(), control_forth_runtime_loop(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), control_forth_then(), control_forth_UNLOOP(), control_forth_until(), control_forth_while(), defer_runtime(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_does_rt(), defining_runtime_lit(), defining_runtime_variable(), defining_word_colon(), defining_word_compile(), defining_word_constant(), defining_word_create(), defining_word_does(), defining_word_immediate(), defining_word_literal(), defining_word_semicolon(), defining_word_variable(), dictionary_m_word_cfa(), dictionary_m_word_find(), dictionary_m_word_hidden(), dictionary_m_word_link_from(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_smudge(), dictionary_m_word_tick(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), dictionary_m_word_traverse(), dictionary_word_2comma(), dictionary_word_allot(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_forget(), dictionary_word_sp_store(), double_word_2drop(), double_word_2dup(), double_word_2over(), double_word_2r_fetch(), double_word_2r_from(), double_word_2rot(), double_word_2swap(), double_word_2to_r(), double_word_d_equals(), double_word_d_less(), double_word_d_minus(), double_word_d_plus(), double_word_d_two_slash(), double_word_d_two_star(), double_word_d_zero_equals(), double_word_d_zero_less(), double_word_dabs(), double_word_dmax(), double_word_dmin(), double_word_dnegate(), double_word_s_to_d(), editor_word_edit(), editor_word_l(), editor_word_s(), editor_word_show(), emit_cell(), ensure_input(), exec_block_with_retry(), execute_colon_word(), fbw_plot(), format_word_d_dot(), format_word_d_dot_r(), format_word_dot(), format_word_dot_r(), format_word_dump(), format_word_hash(), format_word_hash_s(), format_word_hold(), format_word_question(), format_word_sign(), format_word_u_dot(), format_word_u_dot_r(), forth_acl_allow_store(), forth_acl_inherit(), forth_acl_mode_store(), forth_acl_ttl_store(), forth_ALL_HEATS(), forth_BENCH_DICT_LOOKUP(), forth_COMPARE_LOOKUPS(), forth_DECAY_RATE_FETCH(), forth_FREEZE_CRITICAL(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(),

forth_HEAT_FETCH(), forth_HEAT_PERCENTILES(), forth_HEAT_STORE(), forth_LOOKUP_STRATEGY_FETCH(), forth_LOOKUP_STRATEGY_STORE(), forth_PHYSICS_BAYESIAN_REPORT(), forth_PHYSICS_BUILD_INFO(), forth_PHYSICS_BURN(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_RESET_STATS(), forth_PHYSICS_SHOW_FEEDBACK(), forth_PHYSICS_TOGGLE_CACHE(), forth_PHYSICS_WORD_METRICS(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), forth_REORG_BUCKETS(), forth_SHOW_HEAT(), forth_SHOW_HEAT_OPTIMIZATION(), forth_UNFREEZE_WORD(), infer_word_run(), init_vm_and_subsystem(), init_vocabulary_system(), io_runtime_do_string(), io_word_dot_quote(), io_word_emit(), io_word_key(), io_word_question_terminal(), io_word_spaces(), io_word_type(), kernel_main_deep(), log_do_emit(), log_emit_string(), log_str_emit(), log_word_set_level(), logical_word_and(), logical_word_equals(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), main(), memory_word_2fetch(), memory_word_2store(), memory_word_cells(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), mixed_math_word_m_minus(), mixed_math_word_m_plus(), mixed_math_word_m_slash_mod(), mixed_math_word_m_star(), mixed_math_word_mod(), mixed_math_word_mod(), mixed_math_word_slash_mod(), mixed_math_word_star_slash(), mixed_math_word_star_slash_mod(), pop_xt(), reset_vm_state(), restore_vm_state(), return_stack_word_r_fetch(), return_stack_word_r_from(), return_stack_word_to_r(), run_single_test(), runtime_abortq(), save_vm_state(), sk_repl_run(), sk_repl_step(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), stack_word_depth(), stack_word_drop(), stack_word_dup(), stack_word_minus_rot(), stack_word_over(), stack_word_pick(), stack_word_question_dup(), stack_word_roll(), stack_word_rot(), stack_word_swap(), starforth_word_execution_heat_fetch(), starforth_word_execution_heat_store(), starforth_word_init(), starforth_word_random(), starforth_word_seed(), starforth_word_top_words(), starforth_word_wait(), starforth_word_zuse_pubkey_fetch(), string_runtime_s_quote(), string_word_blank(), string_word_bracket_tick(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_minus_trailing(), string_word_number(), string_word_query(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), string_word_source(), string_word_span(), string_word_tib(), string_word_to_in(), string_word_word(), system_word_abort(), system_word_abort_quote(), system_word_execute(), system_word_quit(), system_word_reboot(), system_word_save_system(), system_word_see(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_blocks_shadow(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_bypass(), vm_align(), vm_allot(), vm_assert_interpreter_enabled(), vm_bootstrap_root_vocabulary(), vm_bootstrap_scr(), vm_compile_call(), vm_compile_exit(), vm_compile_literal(), vm_compile_word(), vm_create_word(), vm_debug_dump_state(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_init(), vm_init_with_host(), vm_input_ensure(), vm_interpret(), vm_interpret_word(), vm_load_cell(), vm_load_u8(), vm_make_immediate(), vm_pin_execution_heat(), vm_pop(), vm_pop_asm(), vm_pop_asm(), vm_pop_asm(), vm_push(), vm_push_asm(), vm_push_asm(), vm_push_asm(), vm_repl(), vm_rpop(), vm_rpop_asm(), vm_rpop_asm(), vm_rpop_asm(), vm_rpush(), vm_rpush_asm(), vm_rpush_asm(), vm_rpush_asm(), vm_smudge_word(), vm_store_cell(), vm_store_u8(), vm_unpin_execution_heat(), vocabulary_create_vocabulary_direct(), vocabulary_word_paren_find(), vocabulary_word_vocabulary(), word_defer(), word_defer_fetch(), word_divide(), word_divmod(), and word_is().

6.157.2.28 exit_colon

```
int VM::exit_colon
```

Exit flag for colon definitions.

Referenced by check_physics_transparent(), control_forth_EXIT(), defining_runtime_dodoes(), execute_colon_word(), restore_vm_state(), vm_init(), and vm_init_with_host().

6.157.2.29 halted

```
int VM::halted
```

Referenced by __attribute__(), capsule_vm_kill_all_nonmama(), kernel_main_deep(), sk_repl_run(), sk_repl_step(), starforth_word_init(), system_word_bye(), system_word_reboot(), vm_init(), vm_init_with_host(), and vm_repl().

6.157.2.30 heartbeat

```
HeartbeatState VM::heartbeat
```

Centralized time-driven tuning dispatcher.

Referenced by block_word_block(), block_word_buffer(), block_word_update(), execute_colon_word(), heartbeat_capture_tick_snapsh heartbeat_publish_snapshot(), heartbeat_thread_main(), metrics_from_vm(), metrics_write_james_law_csv_row(), perturb_physics(), physics_on_lookup(), physics_post_execute(), physics_pre_execute(), restore_physics(), save_physics(), starforth_word_heartbeat_ticks(), vm_cleanup(), vm_find_word(), vm_heartbeat_publish_snapshot(), vm_heartbeat_run_cycle(), vm_heartbeat_start_thread(), vm_heartbeat_stop_thread(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_interpret(), vm_interpret_word(), vm_snapshot_read(), vm_tick(), vm_tick_apply_background_decay(), vm_tick_apply_background_decay(), and vm_tick_inference_engine().

6.157.2.31 heartbeat_decay_cursor_id

```
uint32_t VM::heartbeat_decay_cursor_id
```

Continuation cursor for background decay sweeps.

Referenced by vm_init(), vm_init_with_host(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

6.157.2.32 heat_threshold_25th

```
cell_t VM::heat_threshold_25th
```

25th percentile heat (bucket search priority)

Referenced by dict_update_heat_percentiles(), forth_HEAT_PERCENTILES(), forth_SHOW_HEAT_OPTIMIZATION(), perturb_physics(), restore_physics(), and save_physics().

6.157.2.33 heat_threshold_50th

```
cell_t VM::heat_threshold_50th
```

50th percentile heat

Referenced by dict_update_heat_percentiles(), forth_HEAT_PERCENTILES(), forth_SHOW_HEAT_OPTIMIZATION(), perturb_physics(), restore_physics(), and save_physics().

6.157.2.34 heat_threshold_75th

```
cell_t VM::heat_threshold_75th
```

75th percentile heat

Referenced by dict_update_heat_percentiles(), forth_HEAT_PERCENTILES(), forth_SHOW_HEAT_OPTIMIZATION(), perturb_physics(), restore_physics(), and save_physics().

6.157.2.35 here

```
uintptr_t VM::here
```

Next free memory location (byte offset or absolute pointer if M7)

Referenced by control_forth_again(), control_forth_begin(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_loop(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_then(), control_forth_until(), defining_runtime_does_rt(), defining_word_create(), defining_word_variable(), dictionary_word_2comma(), dictionary_word_allot(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_fence(), dictionary_word_forget(), dictionary_word_here(), dictionary_word_latest(), main(), reset_vm_state(), restore_dict_state(), save_dict_state(), sk_parity_collect(), string_word_s_quote(), system_word_save_system(), vm_align(), vm_allot(), vm_cleanup(), vm_debug_dump_state(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_init(), vm_init_with_host(), and vocabulary_create_vocabulary_direct().

6.157.2.36 hold_addr

```
vaddr_t VM::hold_addr
```

Referenced by format_word_end_conversion(), vm_init(), and vm_init_with_host().

6.157.2.37 hold_pos

```
int VM::hold_pos
```

Referenced by format_word_begin_conversion(), format_word_end_conversion(), format_word_hold(), vm_init(), and vm_init_with_host().

6.157.2.38 host

```
const VMHostServices* VM::host
```

Host allocator/time services.

Referenced by vm_host(), vm_init_with_host(), and vm_reset_hosted_fake_ns().

6.157.2.39 hot_word_count_at_check

```
uint64_t VM::hot_word_count_at_check
```

Hot words observed at last inference.

Referenced by heartbeat_publish_snapshot(), vm_heartbeat_publish_snapshot(), and vm_tick_slope_validator().

6.157.2.40 hotwords_cache

```
HotwordsCache* VM::hotwords_cache
```

Frequency-driven dictionary acceleration cache.

Referenced by bucket_posterior(), cache_posterior(), execute_colon_word(), forth_BENCH_DICT_LOOKUP(), forth_PHYSICS_BAYESIAN_REPORT(), forth_PHYSICS_BUILD_INFO(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_RESET_STATS(), forth_PHYSICS_TOGGLE_CACHE(), metrics_from_vm(), physics_pre_execute(), rolling_window_seed_hotwords_cache(), vm_cleanup(), vm_create_word(), vm_dictionary_untrack_entry(), vm_exit_compile_mode(), vm_find_word(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_hide_word(), vm_init(), vm_init_with_host(), and vm_smudge_word().

6.157.2.41 in_var

```
cell_t* VM::in_var
```

Referenced by vm_input_ensure(), vm_input_in(), and vm_input_load_line().

6.157.2.42 input_buffer

```
char VM::input_buffer[1025]
```

Referenced by forth_backslash(), io_word_dot_quote(), log_emit_string(), starforth_word_paren_dash(), string_word_s_quote(), system_word_abort_quote(), vm_interpret(), and vm_parse_word().

6.157.2.43 input_length

```
size_t VM::input_length
```

Referenced by forth_backslash(), io_word_dot_quote(), log_emit_string(), starforth_word_paren_dash(), string_word_s_quote(), system_word_abort_quote(), vm_init(), vm_init_with_host(), vm_interpret(), and vm_parse_word().

6.157.2.44 input_pos

```
size_t VM::input_pos
```

Referenced by forth_backslash(), io_word_dot_quote(), log_emit_string(), starforth_word_paren_dash(), string_word_s_quote(), system_word_abort_quote(), vm_init(), vm_init_with_host(), vm_interpret(), and vm_parse_word().

6.157.2.45 interpreter_enabled

```
int VM::interpreter_enabled
```

Referenced by vm_assert_interpreter_enabled(), vm_enable_interpreter(), and vm_init_with_host().

6.157.2.46 ip

```
cell_t* VM::ip
```

Referenced by vm_debug_dump_state(), vm_save_registers(), vm_save_registers_arm64(), vm_save_registers_rv64(), vm_setup_registers(), vm_setup_registers_arm64(), and vm_setup_registers_rv64().

6.157.2.47 last_bucket_reorg_ns

```
uint64_t VM::last_bucket_reorg_ns
```

When we last reorganized buckets.

Referenced by dict_adaptive_optimization_pass(), dict_reorganize_buckets_by_heat(), vm_init(), and vm_init_with_host().

6.157.2.48 last_decay_check_ns

```
uint64_t VM::last_decay_check_ns
```

Timestamp of last decay effectiveness measurement.

Referenced by vm_init(), and vm_init_with_host().

6.157.2.49 last_inference_outputs

```
InferenceOutputs* VM::last_inference_outputs
```

Cached inference outputs (for ANOVA early-exit and tuning application)

Referenced by infer_word_decay_fetch(), infer_word_early_exit_fetch(), infer_word_fit_fetch(), infer_word_run(), infer_word_variance_fetch(), infer_word_window_fetch(), metrics_from_vm(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), and vm_tick_inference_engine().

6.157.2.50 latest

```
DictEntry* VM::latest
```

Most recent word.

Referenced by capsule_dict_hash_hook(), defining_runtime_does_rt(), defining_word_immediate(), dict_newer_of(), dict_update_heat_percentiles(), dictionary_m_word_hidden(), dictionary_snapshot_acl_from_parent(), dictionary_word_fence(), dictionary_word_forget(), find_entry_from_name(), forth_acl_init_primitives(), forth_ALL_HEATS(), forth_PHYSICS_BURN(), forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_SHOW_FEEDBACK(), forth_PHYSICS_WORD_METRICS(), forth_PIPELINING_RESET_ALL(), forth_PIPELINING_STATS(), heartbeat_capture_tick_snapshot(), infer_word_run(), init_vocabulary_system(), main(), metrics_from_vm(), restore_dict_state(), save_dict_state(), sf_rebuild_fc_index(), sf_try_fast_append(), sk_dict_canonical_hash(), sk_dict_word_count(), sk_parity_collect(), starforth_word_reset_execution_heat(), starforth_word_top_words(), starforth_word_word_execution_heat(), system_word_vlist(), system_word_words(), vm_cleanup(), vm_create_word(), vm_dict_entry_ok(), vm_dictionary_find_by_func(), vm_find_word(), vm_hide_word(), vm_init(), vm_init_with_host(), vm_make_immediate(), vm_pin_execution_heat(), vm_smudge_word(), vm_tick_apply_background_de vm_tick_apply_background_decay(), vm_tick_inference_engine(), vm_tick_slope_validator(), vm_unpin_execution_heat(), vocabulary_create_vocabulary_direct(), vocabulary_select_runtime(), and vocabulary_word_vocabulary().

6.157.2.51 lookup_strategy

```
int VM::lookup_strategy
```

0=naive, 1=heat-aware, 2=inference-reorg

Referenced by dict_adaptive_optimization_pass(), forth_COMPARE_LOOKUPS(), forth_LOOKUP_STRATEGY_FETCH(), forth_LOOKUP_STRATEGY_STORE(), forth_SHOW_HEAT_OPTIMIZATION(), vm_find_word(), vm_init(), and vm_init_with_host().

6.157.2.52 memory

```
uint8_t* VM::memory
```

Unified VM memory buffer.

Referenced by array_ptr(), blk_vm_assign(), blk_vm_evict(), blk_vm_flush_all(), blk_vm_load(), block_word_update(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_loop(), control_forth_plus_loop(), control_forth_repeat(), control_forth_then(), dbg_get_base(), defining_runtime_does_rt(), emit_cell(), extract_name(), hash_colon_body(), init_vocabulary_system(), io_init(), io_write_block(), log_str_emit(), runtime_abortq(), system_word_abort_quote(), system_word_save_system(), system_word_see(), vaddr_from_ptr(), vm_addr_from_ptr(), vm_addr_ok(), vm_addr_to_block(), vm_allot(), vm_bootstrap_scr(), vm_cleanup(), vm_debug_dump_state(), vm_get_block_addr(), vm_init(), vm_init_with_host(), vm_load_cell(), vm_ptr(), vm_store_cell(), vocab_sync_vm_vars(), vocabulary_create_vocabulary_direct(), and vocabulary_word_vocabulary().

6.157.2.53 mode

```
vm_mode_t VM::mode
```

Referenced by cf_epoch_sync(), control_forth_again(), control_forth_begin(), control_forth_case(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_if(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_then(), control_forth_until(), control_forth_while(), defining_word_colon(), defining_word_does(), defining_word_left_bracket(), defining_word_right_bracket(), defining_word_semicolon(), dictionary_m_word_hidden(), dictionary_m_word_interpret(), dictionary_m_word_smudge(), emit_cell(), exec_block_with_retry(), io_word_dot_quote(), log_emit_string(), reset_vm_state(), restore_vm_state(), save_vm_state(), string_word_bracket_tick(), string_word_s_quote(), system_word_abort_quote(), system_word_quit(), system_word_reboot(), vm_compile_call(), vm_compile_exit(), vm_compile_literal(), vm_compile_word(), vm_debug_dump_state(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_init(), vm_init_with_host(), and vm_interpret_word().

6.157.2.54 next_word_id

```
uint32_t VM::next_word_id
```

Next fresh ID when recycle stack empty.

Referenced by vm_dictionary_acquire_word_id(), vm_init(), and vm_init_with_host().

6.157.2.55 pipeline_metrics

```
PipelineGlobalMetrics VM::pipeline_metrics
```

Aggregated prefetch accuracy tracking.

Referenced by execute_colon_word(), forth_PHYSICS_RESET_STATS(), infer_word_run(), metrics_from_vm(), perturb_physics(), physics_pre_execute(), restore_physics(), save_physics(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_window_tuner().

6.157.2.56 recycled_word_id_count

```
uint32_t VM::recycled_word_id_count
```

Depth of recycled stack.

Referenced by vm_dictionary_acquire_word_id(), vm_dictionary_untrack_entry(), vm_init(), and vm_init_with_host().

6.157.2.57 recycled_word_ids

```
uint32_t VM::recycled_word_ids[1024]
```

Reusable ID stack for FORGET.

Referenced by vm_dictionary_acquire_word_id(), and vm_dictionary_untrack_entry().

6.157.2.58 return_stack

```
cell_t VM::return_stack[1024]
```

Return stack storage.

Referenced by control_forth_0branch(), control_forth_branch(), control_forth_I(), control_forth_J(), control_forth_runtime_do(), control_forth_runtime_leave(), control_forth_runtime_loop(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), control_forth_UNLOOP(), defining_runtime_dodoes(), defining_runtime_does_rt(), defining_runtime_lit(), double_word_2r_fetch(), double_word_2r_from(), double_word_2to_r(), io_runtime_do_string(), log_do_emit(), return_stack_word_r_fetch(), return_stack_word_r_from(), return_stack_word_to_r(), string_runtime_s_quote(), vm_rpop(), vm_rpop_asm(), vm_rpop_asm(), vm_rpop_asm(), vm_rpop_fast(), vm_rpush(), vm_rpush_asm(), vm_rpush_asm(), vm_rpush_asm(), vm_rpush_fast(), vm_save_registers(), vm_save_registers_arm64(), vm_save_registers_rv64(), vm_setup_registers(), vm_setup_registers_arm64(), and vm_setup_registers_rv64().

6.157.2.59 rolling_window

```
RollingWindowOfTruth VM::rolling_window
```

Execution history for reproducible seeding.

Referenced by dict_adaptive_optimization_pass(), execute_colon_word(), heartbeat_capture_tick_snapshot(), heartbeat_publish_snapshot(), infer_word_run(), infer_word_window_diversity(), metrics_from_vm(), perturb_physics(), physics_pre_execute(), restore_physics(), run_doe_experiment(), save_physics(), vm_cleanup(), vm_heartbeat_publish_snapshot(), vm_heartbeat_run_cycle(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_window_tuner().

6.157.2.60 rsp

```
int VM::rsp
```

Return stack pointer.

Referenced by capture_exec_result(), check_physics_transparent(), control_forth_0branch(), control_forth_branch(), control_forth_EXIT(), control_forth_I(), control_forth_J(), control_forth_runtime_do(), control_forth_runtime_leave(), control_forth_runtime_loop(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), control_forth_UNLOOP(), defining_runtime_dodoes(), defining_runtime_does_rt(), defining_runtime_lit(), double_word_2r_fetch(), double_word_2r_from(), double_word_2to_r(), execute_colon_word(), io_runtime_do_string(), log_do_emit(), reset_vm_state(), restore_vm_state(), return_stack_word_r_fetch(), return_stack_word_r_from(), return_stack_word_to_r(), save_vm_state(), string_runtime_s_quote(), system_word_quit(), vm_debug_dump_state(), vm_init(), vm_init_with_host(), vm_interpret_word(), vm_rpop(), vm_rpop_asm(), vm_rpop_asm(), vm_rpop_asm(), vm_rpop_fast(), vm_rpush(), vm_rpush_asm(), vm_rpush_asm(), vm_rpush_asm(), vm_rpush_fast(), vm_save_registers(), vm_save_registers_arm64(), vm_save_registers_rv64(), vm_setup_registers(), vm_setup_registers_arm64(), and vm_setup_registers_rv64().

6.157.2.61 scr_addr

```
vaddr_t VM::scr_addr
```

Referenced by block_word_next_block(), block_word_scr(), block_word_update(), current_scr(), editor_word_edit(), mark_buffer_dirty(), set_scr(), vm_bootstrap_scr(), vm_debug_dump_state(), and vm_init().

6.157.2.62 span_var

```
cell_t* VM::span_var
```

Referenced by vm_input_ensure(), vm_input_load_line(), vm_input_source(), and vm_input_span().

6.157.2.63 ssm_config

```
void* VM::ssm_config
```

Opaque pointer to ssm_config_t (L3/L4 mode bits)

Referenced by infer_word_l8_apply(), infer_word_l8_table_force(), inference_engine_run(), physics_metadata_apply_linear_decay(), physics_pre_execute(), vm_cleanup(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), and vm_init_with_host().

6.157.2.64 ssm_l8_state

```
void* VM::ssm_l8_state
```

Opaque pointer to ssm_l8_state_t (avoid circular include)

Referenced by infer_word_l8_apply(), infer_word_l8_mode(), infer_word_l8_table_force(), infer_word_l8_update(), l8_regime_modulation_q16(), vm_cleanup(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), and vm_init_with_host().

6.157.2.65 stale_word_count_at_check

```
uint64_t VM::stale_word_count_at_check
```

Stale words at last check (for trend analysis)

Referenced by heartbeat_publish_snapshot(), infer_word_run(), vm_heartbeat_publish_snapshot(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_slope_validator().

6.157.2.66 state_addr

```
vaddr_t VM::state_addr
```

Referenced by defining_word_left_bracket(), defining_word_right_bracket(), defining_word_state(), exec_block_with_retry(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_init(), and vm_init_with_host().

6.157.2.67 state_var

```
cell_t VM::state_var
```

Referenced by exec_block_with_retry(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_init(), and vm_init_with_host().

6.157.2.68 tib_buf

```
unsigned char* VM::tib_buf
```

Referenced by vm_input_ensure(), vm_input_load_line(), vm_input_source(), and vm_input_tib().

6.157.2.69 tib_cap

```
size_t VM::tib_cap
```

Referenced by vm_input_ensure(), and vm_input_load_line().

6.157.2.70 total_heat_at_last_check

```
uint64_t VM::total_heat_at_last_check
```

Total dictionary heat at last check (for slope inference)

Referenced by heartbeat_publish_snapshot(), infer_word_run(), vm_heartbeat_publish_snapshot(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_slope_validator().

6.157.2.71 tuning_lock

```
sf_mutex_t VM::tuning_lock
```

Guards shared tuning knobs (slope, snapshots)

Referenced by metrics_from_vm(), vm_cleanup(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_tick_inference_engine(), and vm_tick_slope_validator().

6.157.2.72 vocab

```
VocabularyState VM::vocab
```

Referenced by init_vocabulary_system(), vocab_find_word(), vocab_sync_vm_vars(), vocabulary_select_runtime(), vocabulary_word_context(), vocabulary_word_current(), vocabulary_word_definitions(), vocabulary_word_forth(), and vocabulary_word_order().

6.157.2.73 word_count_at_check

```
uint32_t VM::word_count_at_check
```

Total word count at last check (for ratio calculation)

Referenced by vm_tick_inference_engine(), and vm_tick_slope_validator().

6.157.2.74 word_id_map

```
DictEntry* VM::word_id_map[1024]
```

Stable ID → entry map for speculation.

Referenced by vm_dictionary_lookup_by_word_id(), vm_dictionary_track_entry(), and vm_dictionary_untrack_entry().

6.157.2.75 zuse_cert_installed

```
uint8_t VM::zuse_cert_installed
```

1 = zuse_cert_seed/pubkey hold a real minted cert (one-time fuse)

Referenced by capsule_mint_identity(), starforth_word_zuse_cert_installed_query(), and vm_zuse_cert_install().

6.157.2.76 zuse_cert_pubkey

```
uint8_t VM::zuse_cert_pubkey[32]
```

Ed25519 public key derived from the seed at mint time, cached here so callers don't need to re-derive it.

No FORTH word can write it – deliberately kept out of the dictionary so ACL-PIN's redefinition-only guarantee can't be bypassed via >BODY on a CONSTANT (see FABRIC-2.md). Read-only FORTH access via ZUSE-PUBKEY.

Referenced by starforth_word_zuse_pubkey_fetch(), and vm_zuse_cert_install().

6.157.2.77 zuse_cert_seed

```
uint8_t VM::zuse_cert_seed[32]
```

Ed25519 seed, this instance's Zuse identity.

C-only write via vm_zuse_cert_install(); no FORTH word can read or write it – the seed is the private key, never exposed past this struct.

Referenced by capsule_mint_identity(), and vm_zuse_cert_install().

6.157.2.78 zuse_root_pubkey

```
uint8_t VM::zuse_root_pubkey[32]
```

Ed25519 public key, loaded from the genesis-marker fence.

Same value zuse_cert_pubkey ends up holding once Zuse herself attaches – this is just available earlier and unconditionally.

Referenced by capsule_wirebind_verify_cert(), capsule_zuse_boot_load_root_pubkey(), and install_and_activate().

6.157.2.79 zuse_root_pubkey_known

```
uint8_t VM::zuse_root_pubkey_known
```

1 = zuse_root_pubkey holds a real value, loaded from the persistent genesis-marker fence (zuse_genesis_marker_t, capsule_zuse_boot_load_root_pubkey()) independently of whether Zuse's own thumbdrive is attached this boot.

Deliberately separate from zuse_cert_installed/zuse_cert_seed above – that pair is a one-way fuse gating MINT (needs her live private seed); this pair gates WIREBIND cert *verification* only (needs nothing but her already-public key), so a WIREBIND identity can be verified and attach without requiring Zuse's own live session this boot. Bob, 2026-09-09: "zuse is zuse, [each identity] is [itself]" – one identity's attach must not depend on another's live presence.

Referenced by capsule_wirebind_try_attach(), capsule_wirebind_verify_cert(), capsule_zuse_boot_load_root_pubkey(), and install_and_activate().

6.157.2.80 zuse_session

```
uint8_t VM::zuse_session
```

1 = zuse authenticated at console; shows zuse)ok> prompt

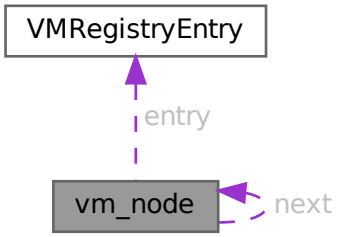
Referenced by capsule_zuse_boot_logout(), capsule_zuse_boot_try_attach(), sk_console_identity_present(), sk_console_user_prefix(), starforth_word_zuse_authenticate(), starforth_word_zuse_session(), vm_exit_compile_mode(), and vm_repl().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.158 vm_node Struct Reference

Collaboration diagram for vm_node:



Data Fields

- VMRegistryEntry entry
- struct vm_node * next

6.158.1 Field Documentation

6.158.1.1 entry

```
VMRegistryEntry vm_node::entry
```

Referenced by capsule_vm_find_by_name(), capsule_vm_find_by_name_nocase(), capsule_vm_kill(), capsule_vm_kill_all_nonmama capsule_vm_registry_get_by_index(), capsule_vm_registry_init(), vm_find_entry_ptr(), vm_registry_alloc(), and vm_registry_live_count().

6.158.1.2 next

```
struct vm_node* vm_node::next
```

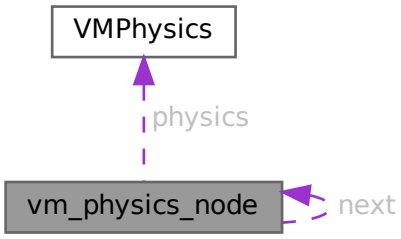
Referenced by capsule_vm_find_by_name(), capsule_vm_find_by_name_nocase(), capsule_vm_kill(), capsule_vm_kill_all_nonmama capsule_vm_registry_get_by_index(), capsule_vm_registry_init(), vm_find_entry_ptr(), vm_registry_alloc(), and vm_registry_live_count().

The documentation for this struct was generated from the following file:

- kernel/src/capsule/capsule_birth.c

6.159 vm_physics_node Struct Reference

Collaboration diagram for vm_physics_node:



Data Fields

- VMUuid vm_id
- VMPhysics physics
- struct vm_physics_node * next

6.159.1 Field Documentation

6.159.1.1 next

```
struct vm_physics_node* vm_physics_node::next
```

Referenced by vm_physics_find(), vm_physics_fleet_heat_sum(), vm_physics_init(), vm_physics_status(), and vm_physics_touch().

6.159.1.2 physics

```
VMPhysics vm_physics_node::physics
```

Referenced by vm_physics_fleet_heat_sum(), vm_physics_heat_of(), vm_physics_init(), vm_physics_retire(), vm_physics_status(), and vm_physics_touch().

6.159.1.3 vm_id

```
VMUuid vm_physics_node::vm_id
```

Referenced by vm_physics_find(), vm_physics_init(), and vm_physics_status().

The documentation for this struct was generated from the following file:

- kernel/src/capsule/capsule_vm_physics.c

6.160 VMFleetTouchSample Struct Reference

Data Fields

- uint64_t elapsed_ticks
- uint64_t amount
- int clamped

6.160.1 Field Documentation

6.160.1.1 amount

```
uint64_t VMFleetTouchSample::amount
```

Referenced by vm_physics_tick(), and vm_physics_touch().

6.160.1.2 clamped

```
int VMFleetTouchSample::clamped
```

Referenced by vm_physics_tick(), and vm_physics_touch().

6.160.1.3 elapsed_ticks

```
uint64_t VMFleetTouchSample::elapsed_ticks
```

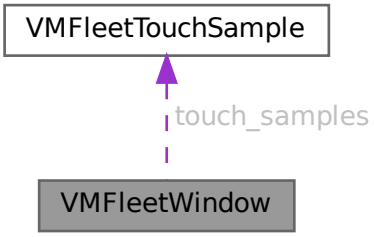
Referenced by vm_physics_tick(), and vm_physics_touch().

The documentation for this struct was generated from the following file:

- kernel/src/capsule/capsule_vm_physics.c

6.161 VMFleetWindow Struct Reference

Collaboration diagram for VMFleetWindow:



Data Fields

- VMFleetTouchSample touch_samples [64]
- uint32_t head
- uint32_t count
- int is_warm

6.161.1 Field Documentation

6.161.1.1 count

```
uint32_t VMFleetWindow::count
```

Referenced by vm_physics_touch().

6.161.1.2 head

```
uint32_t VMFleetWindow::head
```

Referenced by vm_physics_touch().

6.161.1.3 is_warm

```
int VMFleetWindow::is_warm
```

Referenced by vm_physics_status(), vm_physics_tick(), and vm_physics_touch().

6.161.1.4 touch_samples

```
VMFleetTouchSample VMFleetWindow::touch_samples[64]
```

Referenced by vm_physics_tick(), and vm_physics_touch().

The documentation for this struct was generated from the following file:

- kernel/src/capsule/capsule_vm_physics.c

6.162 VMHostServices Struct Reference

VMHostServices - pluggable host operations.

```
#include <vm_host.h>
```

Data Fields

- sk_alloc_fn alloc
- sk_free_fn free
- sk_time_fn monotonic_ns
- sk_mutex_init_fn mutex_init
- sk_mutex_lock_fn mutex_lock
- sk_mutex_unlock_fn mutex_unlock
- sk_mutex_destroy_fn mutex_destroy
- sk_puts_fn puts
- sk_putc_fn putc
- sk_xt_validate_fn is_executable_ptr
- sk_xt_entry_owns_fn owns_xt_entry
- sk_panic_fn panic
- int parity_mode
- int verbose

6.162.1 Detailed Description

VMHostServices - pluggable host operations.

Hosted builds point these at libc functions. Kernel builds point these at kernel implementations.

6.162.2 Field Documentation

6.162.2.1 alloc

```
sk_alloc_fn VMHostServices::alloc
```

Referenced by vm_host_alloc().

6.162.2.2 free

```
sk_free_fn VMHostServices::free
```

Referenced by vm_host_free().

6.162.2.3 is_executable_ptr

```
sk_xt_validate_fn VMHostServices::is_executable_ptr
```

Referenced by vm_bootstrap_root_vocabulary().

6.162.2.4 monotonic_ns

```
sk_time_fn VMHostServices::monotonic_ns
```

Referenced by vm_monotonic_ns().

6.162.2.5 mutex_destroy

sk_mutex_destroy_fn VMHostServices::mutex_destroy

6.162.2.6 mutex_init

sk_mutex_init_fn VMHostServices::mutex_init

6.162.2.7 mutex_lock

sk_mutex_lock_fn VMHostServices::mutex_lock

6.162.2.8 mutex_unlock

sk_mutex_unlock_fn VMHostServices::mutex_unlock

6.162.2.9 owns_xt_entry

sk_xt_entry_owns_fn VMHostServices::owns_xt_entry

6.162.2.10 panic

sk_panic_fn VMHostServices::panic

Referenced by vm_bootstrap_root_vocabulary().

6.162.2.11 parity_mode

int VMHostServices::parity_mode

6.162.2.12 putc

sk_putc_fn VMHostServices::putc

6.162.2.13 puts

sk_puts_fn VMHostServices::puts

6.162.2.14 verbose

```
int VMHostServices::verbose
```

The documentation for this struct was generated from the following file:

- v3/include/vm_host.h

6.163 vmm_page_info_t Struct Reference

```
#include <vmm.h>
```

Data Fields

- int present
- int writable
- int executable

6.163.1 Field Documentation

6.163.1.1 executable

```
int vmm_page_info_t::executable
```

Referenced by vmm_query_page().

6.163.1.2 present

```
int vmm_page_info_t::present
```

Referenced by vmm_query_page().

6.163.1.3 writable

```
int vmm_page_info_t::writable
```

Referenced by vmm_query_page().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/vmm.h

6.164 VMPhysics Struct Reference

Data Fields

- uint64_t execution_heat_q48
- uint64_t last_active_tick
- int touched
- int is_live

6.164.1 Field Documentation

6.164.1.1 execution_heat_q48

```
uint64_t VMPhysics::execution_heat_q48
```

Referenced by vm_physics_fleet_heat_sum(), vm_physics_heat_of(), vm_physics_init(), vm_physics_retire(), vm_physics_status(), vm_physics_touch(), and vm_physics_transfer().

6.164.1.2 is_live

```
int VMPhysics::is_live
```

Referenced by vm_physics_fleet_heat_sum(), vm_physics_heat_of(), vm_physics_init(), vm_physics_retire(), vm_physics_status(), and vm_physics_touch().

6.164.1.3 last_active_tick

```
uint64_t VMPhysics::last_active_tick
```

Referenced by vm_physics_init(), and vm_physics_touch().

6.164.1.4 touched

```
int VMPhysics::touched
```

Referenced by vm_physics_init(), and vm_physics_touch().

The documentation for this struct was generated from the following file:

- kernel/src/capsule/capsule_vm_physics.c

6.165 VMRegistryEntry Struct Reference

```
#include <capsule_run.h>
```

Data Fields

- VMUuid vm_id
- uint32_t state
- uint64_t birth_capsule_id
- uint64_t birth_timestamp_ns
- uint64_t birth_dict_hash
- uint32_t flags
- VMUuid parent_vm_id
- void * vm_ptr
- char name [64]
- size_t stadium_patron_cell
- int pending_reap

6.165.1 Field Documentation

6.165.1.1 birth_capsule_id

```
uint64_t VMRegistryEntry::birth_capsule_id
```

Referenced by capsule_birth_mama(), capsule_vm_registry_init(), and vm_registry_alloc().

6.165.1.2 birth_dict_hash

```
uint64_t VMRegistryEntry::birth_dict_hash
```

Referenced by capsule_birth_mama(), capsule_vm_registry_init(), and vm_registry_alloc().

6.165.1.3 birth_timestamp_ns

```
uint64_t VMRegistryEntry::birth_timestamp_ns
```

Referenced by capsule_vm_registry_init(), and vm_registry_alloc().

6.165.1.4 flags

```
uint32_t VMRegistryEntry::flags
```

Referenced by capsule_vm_registry_init(), and vm_registry_alloc().

6.165.1.5 name

```
char VMRegistryEntry::name[64]
```

Referenced by capsule_vm_find_by_name(), capsule_vm_find_by_name_nocase(), capsule_vm_kill(), capsule_vm_kill_all_nonmama
capsule_vm_registry_init(), doe_log_tick_row(), vm_registry_alloc(), and wirebind_teardown_console().

6.165.1.6 parent_vm_id

```
VMUuid VMRegistryEntry::parent_vm_id
```

Referenced by capsule_vm_registry_init(), vm_physics_find_root_id(), and vm_registry_alloc().

6.165.1.7 pending_reap

```
int VMRegistryEntry::pending_reap
```

6.165.1.8 stadium_patron_cell

```
size_t VMRegistryEntry::stadium_patron_cell
```

Referenced by capsule_vm_kill_all_nonmama(), capsule_vm_registry_init(), and vm_registry_alloc().

6.165.1.9 state

```
uint32_t VMRegistryEntry::state
```

Referenced by capsule_vm_kill_all_nonmama(), capsule_vm_registry_init(), capsule_wirebind_try_attach(), kernel_main_deep(), sk_repl_dispatch_line(), vm_registry_alloc(), vm_registry_live_count(), wirebind_resolve_attached(), and wirebind_teardown_console().

6.165.1.10 vm_id

```
VMUuid VMRegistryEntry::vm_id
```

Referenced by capsule_vm_kill_all_nonmama(), capsule_vm_registry_init(), doe_log_tick_row(), kernel_main_deep(), sk_repl_dispatch_line(), sk_word_blk_attach_ack(), vm_find_entry_ptr(), and vm_registry_alloc().

6.165.1.11 vm_ptr

```
void* VMRegistryEntry::vm_ptr
```

Referenced by capsule_birth_baby(), capsule_vm_kill_all_nonmama(), capsule_vm_registry_init(), kernel_main_deep(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), vm_registry_alloc(), and wirebind_teardown_console().

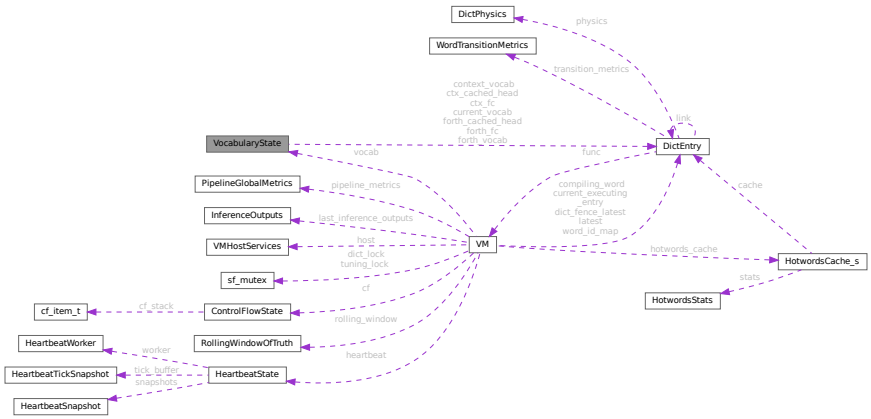
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/capsule_run.h

6.166 VocabularyState Struct Reference

`#include <vm.h>`

Collaboration diagram for VocabularyState:



Data Fields

- int initialized
- DictEntry * forth_vocab
- DictEntry * context_vocab
- DictEntry * current_vocab
- vaddr_t context_var_addr
- vaddr_t current_var_addr
- DictEntry ** ctx_fc [256]
- size_t ctx_n [256]
- DictEntry * ctx_cached_head
- DictEntry ** forth_fc [256]
- size_t forth_n [256]
- DictEntry * forth_cached_head

6.166.1 Field Documentation

6.166.1.1 context_var_addr

`vaddr_t VocabularyState::context_var_addr`

Referenced by init_vocabulary_system(), vocab_sync_vm_vars(), and vocabulary_word_context().

6.166.1.2 context_vocab

`DictEntry* VocabularyState::context_vocab`

Referenced by init_vocabulary_system(), vocab_find_word(), vocab_sync_vm_vars(), vocabulary_select_runtime(), vocabulary_word_definitions(), vocabulary_word_forth(), and vocabulary_word_order().

6.166.1.3 ctx_cached_head

```
DictEntry* VocabularyState::ctx_cached_head
```

Referenced by vocab_find_word().

6.166.1.4 ctx_fc

```
DictEntry** VocabularyState::ctx_fc[256]
```

Referenced by vocab_find_word().

6.166.1.5 ctx_n

```
size_t VocabularyState::ctx_n[256]
```

Referenced by vocab_find_word().

6.166.1.6 current_var_addr

```
vaddr_t VocabularyState::current_var_addr
```

Referenced by init_vocabulary_system(), vocab_sync_vm_vars(), and vocabulary_word_current().

6.166.1.7 current_vocab

```
DictEntry* VocabularyState::current_vocab
```

Referenced by init_vocabulary_system(), vocab_sync_vm_vars(), vocabulary_word_definitions(), and vocabulary_word_order().

6.166.1.8 forth_cached_head

```
DictEntry* VocabularyState::forth_cached_head
```

Referenced by vocab_find_word().

6.166.1.9 forth_fc

```
DictEntry** VocabularyState::forth_fc[256]
```

Referenced by vocab_find_word().

6.166.1.10 forth_n

```
size_t VocabularyState::forth_n[256]
```

Referenced by vocab_find_word().

6.166.1.11 forth_vocab

```
DictEntry* VocabularyState::forth_vocab
```

Referenced by init_vocabulary_system(), vocab_find_word(), vocabulary_word_forth(), and vocabulary_word_order().

6.166.1.12 initialized

```
int VocabularyState::initialized
```

Referenced by init_vocabulary_system().

The documentation for this struct was generated from the following file:

- v3/include/vm.h

6.167 VT100State Struct Reference

Data Fields

- uint32_t cols
- uint32_t rows
- uint32_t cx
- uint32_t cy
- uint32_t saved_cx
- uint32_t saved_cy
- uint32_t fg
- uint32_t bg
- uint32_t def_fg
- uint32_t def_bg
- int bold
- int underline
- int reverse
- ParseState state
- int32_t params [16]
- uint32_t param_count
- int param_has_value
- int initialized

6.167.1 Field Documentation

6.167.1.1 bg

```
uint32_t VT100State::bg
```

Referenced by apply_sgr(), eff_bg(), eff_fg(), erase_line_range(), and reset_attrs().

6.167.1.2 bold

```
int VT100State::bold
```

Referenced by apply_sgr(), draw_cursor_glyph(), and reset_attrs().

6.167.1.3 cols

```
uint32_t VT100State::cols
```

Referenced by advance_cursor(), dispatch_csi(), draw_cursor_glyph(), erase_display(), erase_line(), scroll_up(), scrollback_redraw(), set_cursor(), vt100_cols(), vt100_enable_ttf(), vt100_init(), vt100_putc(), and vt100_reset().

6.167.1.4 cx

```
uint32_t VT100State::cx
```

Referenced by advance_cursor(), dispatch_csi(), draw_cursor_glyph(), erase_display(), erase_line(), set_cursor(), vt100_draw_cursor(), vt100_enable_ttf(), vt100_erase_cursor(), vt100_init(), vt100_putc(), and vt100_reset().

6.167.1.5 cy

```
uint32_t VT100State::cy
```

Referenced by advance_cursor(), dispatch_csi(), draw_cursor_glyph(), erase_display(), erase_line(), set_cursor(), vt100_draw_cursor(), vt100_enable_ttf(), vt100_erase_cursor(), vt100_init(), vt100_putc(), and vt100_reset().

6.167.1.6 def_bg

```
uint32_t VT100State::def_bg
```

Referenced by apply_sgr(), erase_display(), reset_attrs(), scroll_up(), scrollback_redraw(), vt100_enable_ttf(), vt100_init(), vt100_putc(), vt100_reset(), and vt100_toggle_graphics().

6.167.1.7 def_fg

```
uint32_t VT100State::def_fg
```

Referenced by apply_sgr(), reset_attrs(), scrollback_redraw(), and vt100_init().

6.167.1.8 fg

```
uint32_t VT100State::fg
```

Referenced by apply_sgr(), eff_bg(), eff_fg(), and reset_attrs().

6.167.1.9 initialized

```
int VT100State::initialized
```

Referenced by vt100_enable_ttf(), vt100_init(), vt100_putc(), and vt100_reset().

6.167.1.10 param_count

```
uint32_t VT100State::param_count
```

Referenced by apply_sgr(), param(), reset_parser(), and vt100_putc().

6.167.1.11 param_has_value

```
int VT100State::param_has_value
```

Referenced by reset_parser(), and vt100_putc().

6.167.1.12 params

```
int32_t VT100State::params[16]
```

Referenced by param(), reset_parser(), and vt100_putc().

6.167.1.13 reverse

```
int VT100State::reverse
```

Referenced by apply_sgr(), eff_bg(), eff_fg(), and reset_attrs().

6.167.1.14 rows

```
uint32_t VT100State::rows
```

Referenced by advance_cursor(), dispatch_csi(), draw_cursor_glyph(), erase_display(), scroll_up(), scrollback_line_at(), scrollback_redraw(), set_cursor(), vt100_enable_ttf(), vt100_init(), vt100_putc(), vt100_reset(), and vt100_rows().

6.167.1.15 saved_cx

```
uint32_t VT100State::saved_cx
```

Referenced by dispatch_csi(), vt100_init(), vt100_putc(), and vt100_reset().

6.167.1.16 saved_cy

```
uint32_t VT100State::saved_cy
```

Referenced by dispatch_csi(), vt100_init(), vt100_putc(), and vt100_reset().

6.167.1.17 state

```
ParseState VT100State::state
```

Referenced by reset_parser(), and vt100_putc().

6.167.1.18 underline

```
int VT100State::underline
```

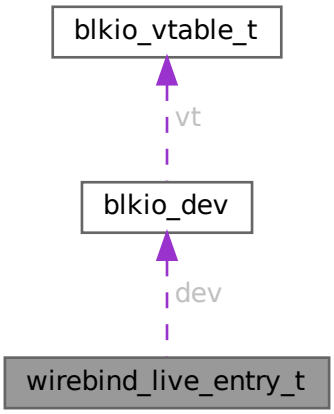
Referenced by apply_sgr(), draw_cursor_glyph(), and reset_attrs().

The documentation for this struct was generated from the following file:

- kernel/src/hal/vt100.c

6.168 wirebind_live_entry_t Struct Reference

Collaboration diagram for wirebind_live_entry_t:



Data Fields

- int in_use
- struct blkio_dev * dev
- VMUuid user_vm_id
- char username [32u]
- int switch_slot

6.168.1 Field Documentation

6.168.1.1 dev

```
struct blkio_dev* wirebind_live_entry_t::dev
```

Referenced by capsule_wirebind_try_attach(), and wirebind_live_remove().

6.168.1.2 in_use

```
int wirebind_live_entry_t::in_use
```

Referenced by capsule_wirebind_try_attach(), and wirebind_live_remove().

6.168.1.3 switch_slot

```
int wirebind_live_entry_t::switch_slot
```

Referenced by capsule_wirebind_try_attach(), and wirebind_live_remove().

6.168.1.4 user_vm_id

```
VMUuid wirebind_live_entry_t::user_vm_id
```

Referenced by capsule_wirebind_try_attach(), capsule_wirebind_unclean_detach(), and wirebind_live_remove().

6.168.1.5 username

```
char wirebind_live_entry_t::username[32u]
```

Referenced by capsule_wirebind_try_attach(), capsule_wirebind_unclean_detach(), and wirebind_live_remove().

The documentation for this struct was generated from the following file:

- kernel/src/capsule/capsule_wirebind.c

6.169 WordContract Struct Reference

```
#include <test_contracts.h>
```

Data Fields

- contract_flags_t flags
- int fuzz_rounds

6.169.1 Field Documentation

6.169.1.1 flags

```
contract_flags_t WordContract::flags
```

Referenced by resolve_contract(), and run_test_suite_m().

6.169.1.2 fuzz_rounds

```
int WordContract::fuzz_rounds
```

Referenced by run_test_suite_m().

The documentation for this struct was generated from the following file:

- v3/src/test_runner/include/test_contracts.h

6.170 WordProfile Struct Reference

Execution statistics for a single word.

```
#include <profiler.h>
```

Data Fields

- const char * word_name
  Name of the profiled word.
- uint64_t total_time
  Total execution time.
- uint64_t call_count
  Number of executions.
- uint64_t avg_time
  Average execution time.
- uint64_t min_time
  Minimum execution time.
- uint64_t max_time
  Maximum execution time.
- double percentage
  Percentage of total runtime.

6.170.1 Detailed Description

Execution statistics for a single word.

6.170.2 Field Documentation

6.170.2.1 avg_time

```
uint64_t WordProfile::avg_time
```

Average execution time.

6.170.2.2 call_count

```
uint64_t WordProfile::call_count
```

Number of executions.

6.170.2.3 max_time

```
uint64_t WordProfile::max_time
```

Maximum execution time.

6.170.2.4 min_time

```
uint64_t WordProfile::min_time
```

Minimum execution time.

6.170.2.5 percentage

```
double WordProfile::percentage
```

Percentage of total runtime.

6.170.2.6 total_time

```
uint64_t WordProfile::total_time
```

Total execution time.

6.170.2.7 word_name

```
const char* WordProfile::word_name
```

Name of the profiled word.

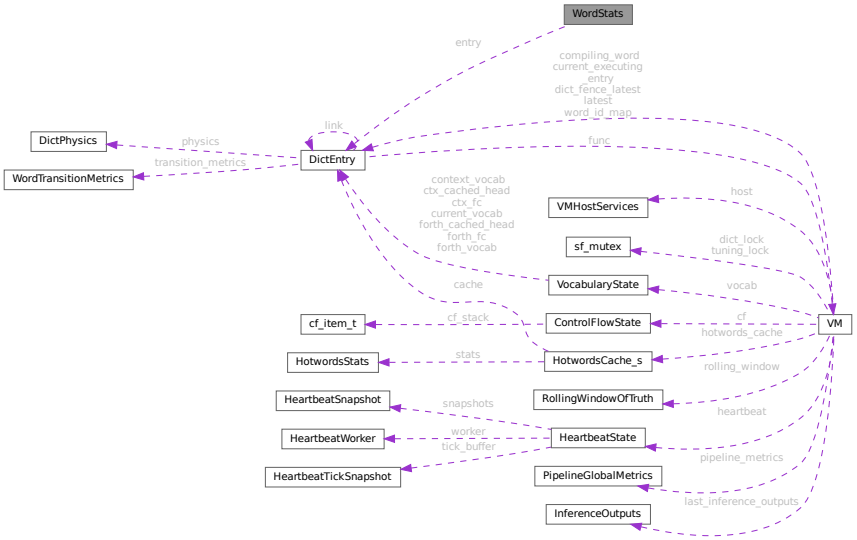
The documentation for this struct was generated from the following file:

- v3/include/profiler.h

6.171 WordStats Struct Reference

Statistics for individual word execution.

Collaboration diagram for WordStats:



Data Fields

- const DictEntry * entry

  Pointer to dictionary entry.
- uint64_t call_count

  Number of times word was called.
- uint64_t total_time_ns

  Total execution time in nanoseconds.
- uint64_t min_time_ns

  Minimum execution time in nanoseconds.
- uint64_t max_time_ns

  Maximum execution time in nanoseconds.

6.171.1 Detailed Description

Statistics for individual word execution.

6.171.2 Field Documentation

6.171.2.1 call_count

```
uint64_t WordStats::call_count
```

Number of times word was called.

Referenced by compare_by_calls(), get_word_stats(), profiler_generate_report(), profiler_print_hotspots(), profiler_word_count(), and profiler_word_exit().

6.171.2.2 entry

```
const DictEntry* WordStats::entry
```

Pointer to dictionary entry.

Referenced by get_word_stats(), profiler_generate_report(), and profiler_print_hotspots().

6.171.2.3 max_time_ns

```
uint64_t WordStats::max_time_ns
```

Maximum execution time in nanoseconds.

Referenced by get_word_stats(), profiler_generate_report(), and profiler_word_exit().

6.171.2.4 min_time_ns

```
uint64_t WordStats::min_time_ns
```

Minimum execution time in nanoseconds.

Referenced by get_word_stats(), and profiler_word_exit().

6.171.2.5 total_time_ns

```
uint64_t WordStats::total_time_ns
```

Total execution time in nanoseconds.

Referenced by compare_by_time(), get_word_stats(), profiler_generate_report(), and profiler_word_exit().

The documentation for this struct was generated from the following file:

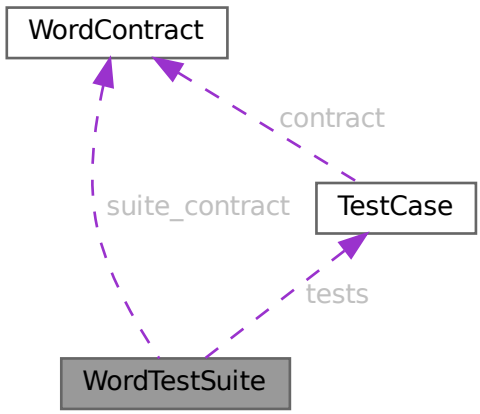
- v3/src/profiler.c

6.172 WordTestSuite Struct Reference

Collection of test cases for a single Forth word.

`#include <test_common.h>`

Collaboration diagram for WordTestSuite:



Data Fields

- const char * word_name
  Name of the Forth word being tested.
- TestCase tests [20]
  Array of test cases.
- int test_count
  Number of tests in the suite.
- WordContract suite_contract
  Suite-level contract; zero = inherit from module.

6.172.1 Detailed Description

Collection of test cases for a single Forth word.

6.172.2 Field Documentation

6.172.2.1 suite_contract

`WordContract WordTestSuite::suite_contract`

Suite-level contract; zero = inherit from module.

Referenced by run_test_suite_m().

6.172.2.2 test_count

```
int WordTestSuite::test_count
```

Number of tests in the suite.

Referenced by run_test_suite(), and run_test_suite_m().

6.172.2.3 tests

```
TestCase WordTestSuite::tests[20]
```

Array of test cases.

Referenced by run_test_suite(), and run_test_suite_m().

6.172.2.4 word_name

```
const char* WordTestSuite::word_name
```

Name of the Forth word being tested.

Referenced by run_arithmetic_words_tests(), run_block_words_tests(), run_control_words_tests(), run_defining_words_tests(), run_dictionary_manipulation_words_tests(), run_dictionary_words_tests(), run_double_words_tests(), run_format_words_tests(), run_inference_words_tests(), run_io_words_tests(), run_logical_words_tests(), run_mama_forth_words_tests(), run_memory_words_tests(), run_mixed_arithmetic_words_tests(), run_physics_freeze_words_tests(), run_q48_words_tests(), run_return_stack_words_tests(), run_stack_words_tests(), run_starforth_words_tests(), run_string_words_tests(), run_system_words_tests(), run_test_suite(), run_test_suite_m(), and run_vocabulary_words_tests().

The documentation for this struct was generated from the following file:

- v3/src/test_runner/include/test_common.h

6.173 WordTransitionMetrics Struct Reference

SPECULATION_THRESHOLD_Q48: Minimum probability (Q48.16) to speculate on a word transition.

```
#include <physics_pipelining_metrics.h>
```

Data Fields

- uint64_t * transition_heat

  transition_heat[i]: Number of times word with index i has been executed immediately after this word.
- uint64_t total_transitions

  Total number of transitions observed from this word.
- uint64_t prefetch_attempts

  Number of speculative prefetch attempts made for this word.
- uint64_t prefetch_hits

  Number of successful prefetch predictions (hits)
- uint64_t prefetch_misses

  Number of failed prefetch predictions (misses)
- int64_t prefetch_latency_saved_q48

  Total latency saved through successful prefetching (Q48.16 nanoseconds).
- int64_t misprediction_cost_q48

  Total latency cost from failed prefetch predictions (Q48.16 nanoseconds).
- int64_t max_transition_probability_q48

  Cached probability of most likely next word (for quick access).
- uint32_t most_likely_next_word_id

  Index (word id) of most likely next word.
- uint32_t * context_window

  Circular buffer of previous word IDs for context tracking.
- uint32_t context_window_pos

  Current position in context_window circular buffer.
- void * context_transitions

  Sparse hash table for context-based transitions.
- uint64_t total_context_transitions

  Total context-based transitions observed.
- uint32_t actual_window_size

  Actual window size for this word's metrics.

6.173.1 Detailed Description

SPECULATION_THRESHOLD_Q48: Minimum probability (Q48.16) to speculate on a word transition.

Range: 0.10 (very aggressive) to 0.95 (very conservative) Default: see starforth_config.h (STARFORTH↩ _CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT) Q48.16 encoding: 0.50 = 0x8000LL (1LL << 15) SPECULATION_DEPTH: How many words ahead to prefetch speculatively. Range: 1 (next word only) to 4 (very aggressive) Default: see starforth_config.h (STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT) MIN_↩ SAMPLES_FOR_SPECULATION: Minimum number of transitions observed before making speculation decisions. Range: 1 (immediate) to 100 (very conservative) Default: see starforth_config.h (STARFORTH_CONFIG_MIN↩ _SAMPLES_FOR_SPECULATION_DEFAULT) MISPREDICTION_COST_Q48: Estimated cost (in nanoseconds, Q48.16 format) of recovering from wrong speculation. This is calibrated to the actual recovery time on target hardware. Range: 0 to 100 ns Default: see starforth_config.h (STARFORTH_CONFIG_MISPREDICTION_COST↩ _Q48_DEFAULT) Q48.16 encoding: 25 ns = 25 << 16 = 0x190000LL MINIMUM_PREFETCH_ROI: Minimum expected Return-On-Investment ratio for speculation to be worthwhile. Speculation succeeds only if: (prefetch↩ _latency_saved / total_attempts) > MINIMUM_PREFETCH_ROI Range: 1.0 (always speculate) to 2.0 (very selective) Default: see starforth_config.h (STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT) Q48.16 encoding: 1.10 = 1.10 * (1 << 16) = 0x1199AL TRANSITION_WINDOW_SIZE: Knob #6 - Execution context depth for prediction

How many previous words to remember when predicting next word. Larger window captures deeper patterns but uses more memory.

Window=1: Only immediate predecessor (A→B predictions) Window=2: Previous 2 words (A,B→C predictions) - good balance Window=4: Previous 4 words (captures multi-level patterns) Window=8: Very deep, expensive – see starforth_config.h for the actual shipped default

Binary Chop Search Space (Phase 2): Iteration 1: Try window=2 (baseline) Iteration 2: Try window=1 and window=4 Iteration 3: Narrow to optimal (1, 2, or 4) Converge: Pick size with highest prediction accuracy

Range: 1 to 8 Default: see starforth_config.h (STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT) Trade-off: Pattern depth vs. memory usage & complexity

Tuning: make TRANSITION_WINDOW_SIZE=1 (or 2, 4, 8)

6.173.2 Field Documentation

6.173.2.1 actual_window_size

```
uint32_t WordTransitionMetrics::actual_window_size
```

Actual window size for this word's metrics.

May differ from TRANSITION_WINDOW_SIZE if memory allocation failed. Phase 2 uses this to validate context data reliability.

Referenced by transition_metrics_context_accuracy_string(), transition_metrics_init(), transition_metrics_reset(), and transition_metrics_update_context_window().

6.173.2.2 context_transitions

```
void* WordTransitionMetrics::context_transitions
```

Sparse hash table for context-based transitions.

Maps: hash(context_window) → counts_array[DICTIONARY_SIZE]

Phase 1: Collect only (simple reference implementation) Phase 2: Analyze prediction accuracy by context depth Phase 3: Use for adaptive prefetching decisions

Opaque pointer - implementation is in transition_metrics.c

Referenced by transition_metrics_cleanup(), transition_metrics_init(), and transition_metrics_reset().

6.173.2.3 context_window

```
uint32_t* WordTransitionMetrics::context_window
```

Circular buffer of previous word IDs for context tracking.

Size = TRANSITION_WINDOW_SIZE (default 2) Used to build multi-word context patterns (e.g., A,B→C)

Example with WINDOW=2: context_window[0] = older word (2 steps back) context_window[1] = recent word (1 step back) Next word completes the pattern: (context_window[0], context_window[1]) → next

Referenced by transition_metrics_cleanup(), transition_metrics_init(), transition_metrics_reset(), and transition_metrics_update_conte

6.173.2.4 context_window_pos

```
uint32_t WordTransitionMetrics::context_window_pos
```

Current position in context_window circular buffer.

Points to slot where next word ID will be stored.

Referenced by transition_metrics_init(), and transition_metrics_reset().

6.173.2.5 max_transition_probability_q48

```
int64_t WordTransitionMetrics::max_transition_probability_q48
```

Cached probability of most likely next word (for quick access).

Updated periodically to avoid recomputation. Q48.16 format (0.0 to 1.0 == 0LL to 1LL<<16)

Referenced by forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), transition_metrics_init(), transition_metrics_reset(), transition_metrics_stats_string(), and transition_metrics_update_cache().

6.173.2.6 misprediction_cost_q48

```
int64_t WordTransitionMetrics::misprediction_cost_q48
```

Total latency cost from failed prefetch predictions (Q48.16 nanoseconds).

Negative value (cost) to be subtracted from savings.

Referenced by forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), transition_metrics_init(), transition_metrics_record_prefetch_miss(), transition_metrics_reset(), and transition_metrics_stats_string().

6.173.2.7 most_likely_next_word_id

```
uint32_t WordTransitionMetrics::most_likely_next_word_id
```

Index (word id) of most likely next word.

Used for fast prefetch without scanning transition_heat array.

Referenced by execute_colon_word(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), physics_pre_execute(), transition_metrics_init(), transition_metrics_reset(), transition_metrics_stats_string(), and transition_metrics_update_cache().

6.173.2.8 prefetch_attempts

```
uint64_t WordTransitionMetrics::prefetch_attempts
```

Number of speculative prefetch attempts made for this word.

Referenced by execute_colon_word(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_STATS(), physics_pre_execute(), transition_metrics_init(), transition_metrics_reset(), and transition_metrics_stats_string().

6.173.2.9 prefetch_hits

```
uint64_t WordTransitionMetrics::prefetch_hits
```

Number of successful prefetch predictions (hits)

Referenced by forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_STATS(), transition_metrics_init(), transition_metrics_record_prefetch_hit(), transition_metrics_reset(), and transition_metrics_stats_string().

6.173.2.10 prefetch_latency_saved_q48

```
int64_t WordTransitionMetrics::prefetch_latency_saved_q48
```

Total latency saved through successful prefetching (Q48.16 nanoseconds).

Positive value indicates net benefit from speculation.

Referenced by forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), transition_metrics_init(), transition_metrics_record_prefetch_hit(), transition_metrics_reset(), and transition_metrics_stats_string().

6.173.2.11 prefetch_misses

```
uint64_t WordTransitionMetrics::prefetch_misses
```

Number of failed prefetch predictions (misses)

Referenced by forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_STATS(), transition_metrics_init(), transition_metrics_record_prefetch_miss(), transition_metrics_reset(), and transition_metrics_stats_string().

6.173.2.12 total_context_transitions

```
uint64_t WordTransitionMetrics::total_context_transitions
```

Total context-based transitions observed.

Counter for Phase 2 analysis (accuracy measurement)

Referenced by transition_metrics_context_accuracy_string(), transition_metrics_init(), transition_metrics_record_context(), and transition_metrics_reset().

6.173.2.13 total_transitions

```
uint64_t WordTransitionMetrics::total_transitions
```

Total number of transitions observed from this word.

Referenced by forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRAN forth_PIPELINING_STATS(), transition_metrics_get_probability_q48(), transition_metrics_init(), transition_metrics_record(), transition_metrics_reset(), transition_metrics_should_speculate(), transition_metrics_stats_string(), and transition_metrics_update_ca

6.173.2.14 transition_heat

```
uint64_t* WordTransitionMetrics::transition_heat
```

transition_heat[i]: Number of times word with index i has been executed immediately after this word.

Dynamically allocated array of size DICTIONARY_SIZE. Initialized to NULL; allocated on first use to save memory for words that are never executed.

Referenced by forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), transition_metrics_cleanup(), transition_metrics_get_probability_q48(), transition_metrics_init(), transition_metrics_record(), transition_metrics_reset(), transition_metrics_should_speculate(), and transition_metrics_update_cache().

The documentation for this struct was generated from the following file:

- v3/include/physics_pipelining_metrics.h

6.174 xhci_cap_regs_t Struct Reference

```
#include <xhci.h>
```

Data Fields

- volatile uint8_t cap_length
- volatile uint8_t reserved0
- volatile uint16_t hci_version
- volatile uint32_t hcs_params1
- volatile uint32_t hcs_params2
- volatile uint32_t hcs_params3
- volatile uint32_t hcc_params1
- volatile uint32_t db_off
- volatile uint32_t rts_off
- volatile uint32_t hcc_params2

6.174.1 Field Documentation

6.174.1.1 cap_length

```
volatile uint8_t xhci_cap_regs_t::cap_length
```

Referenced by xhci_find_and_map().

6.174.1.2 db_off

```
volatile uint32_t xhci_cap_regs_t::db_off
```

Referenced by xhci_find_and_map().

6.174.1.3 hcc_params1

```
volatile uint32_t xhci_cap_regs_t::hcc_params1
```

Referenced by xhci_bringup(), xhci_ctx_bytes(), and xhci_take_from_bios().

6.174.1.4 hcc_params2

```
volatile uint32_t xhci_cap_regs_t::hcc_params2
```

6.174.1.5 hci_version

```
volatile uint16_t xhci_cap_regs_t::hci_version
```

6.174.1.6 hcs_params1

```
volatile uint32_t xhci_cap_regs_t::hcs_params1
```

Referenced by xhci_find_and_map().

6.174.1.7 hcs_params2

```
volatile uint32_t xhci_cap_regs_t::hcs_params2
```

Referenced by xhci_find_and_map().

6.174.1.8 hcs_params3

```
volatile uint32_t xhci_cap_regs_t::hcs_params3
```

6.174.1.9 reserved0

```
volatile uint8_t xhci_cap_regs_t::reserved0
```

6.174.1.10 rts_off

```
volatile uint32_t xhci_cap_regs_t::rts_off
```

Referenced by xhci_find_and_map().

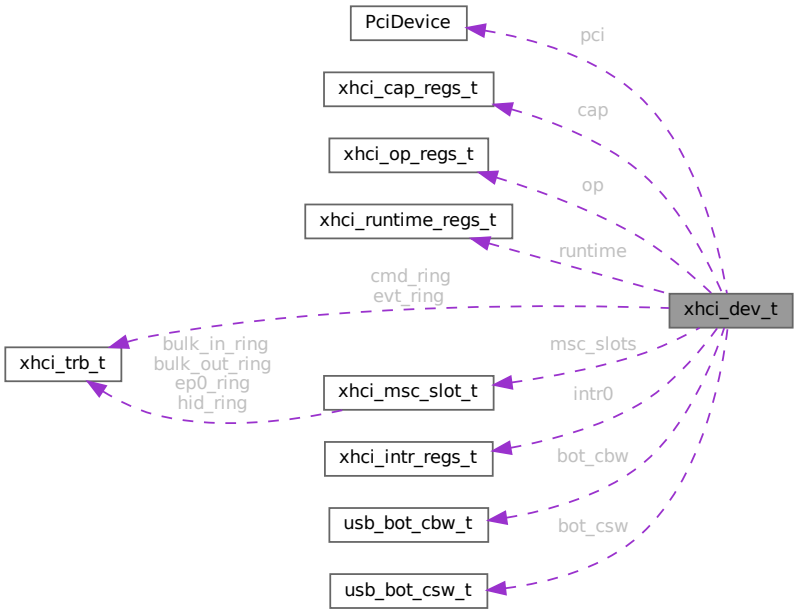
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.175 xhci_dev_t Struct Reference

`#include <xhci_driver.h>`

Collaboration diagram for xhci_dev_t:



Public Types

- enum {
  XHCI_CONN_IDLE = 0 , XHCI_CONN_AWAIT_PORT_RESET , XHCI_CONN_AWAIT_ENABLE_SLOT ,
  XHCI_CONN_AWAIT_ADDRESS_DEVICE ,
  XHCI_CONN_AWAIT_DISABLE_SLOT , XHCI_CONN_AWAIT_CONFIGURE_ENDPOINT , XHCI_CONN_AWAIT_RESET_EN
  , XHCI_CONN_AWAIT_SET_TR_DEQUEUE }
- enum {
  BOT_CMD_NONE = 0 , BOT_CMD_TEST_UNIT_READY , BOT_CMD_READ10 , BOT_CMD_READ_CAPACITY10
  ,
  BOT_CMD_WRITE10 }
- enum { BOT_STATUS_IDLE = 0 , BOT_STATUS_PASS , BOT_STATUS_FAILED , BOT_STATUS_TIMEOUT
  }
- enum { BOT_TUR_CHAIN_NONE = 0 , BOT_TUR_CHAIN_READ10 , BOT_TUR_CHAIN_READ_CAPACITY10
  , BOT_TUR_CHAIN_WRITE10 }

Data Fields

- PciDevice pci
- uint64_t bar0_phys
- xhci_cap_regs_t * cap
- xhci_op_regs_t * op
- xhci_runtime_regs_t * runtime

- xhci_doorbell_t * doorbell
- uint32_t max_slots
- uint32_t max_ports
- uint32_t max_intrs
- uint32_t max_scratchpad_bufs
- void * dcbaa
- void * scratchpad_arr
- xhci_trb_t * cmd_ring
- uint32_t cmd_ring_cycle
- uint32_t cmd_ring_enq
- xhci_trb_t * evt_ring
- void * evt_ring_seg_table
- uint32_t evt_ring_cycle
- uint32_t evt_ring_deq
- xhci_intr_regs_t * intr0
- uint32_t port_slot_id [32u]
- uint32_t pending_connect_port_id
- uint32_t pending_connect_speed
- struct {
  uint8_t valid
  uint8_t is_connect
  uint32_t port_id
  uint32_t portsc
  uint32_t slot_id
  } pending_events [32u]

- enum xhci_dev_t:: { ... } connect_state
- uint32_t pending_connect_slot_id
- uint32_t pending_disable_slot_id
- xhci_msc_slot_t * msc_slots
- usb_bot_cbw_t bot_cbw
- usb_bot_csw_t bot_csw
- uint32_t bot_next_tag
- uint32_t bot_last_tag
- uint8_t bot_data_buf [1024]
- uint32_t bot_expected_data_len
- enum xhci_dev_t:: { ... } bot_cmd_kind
- enum xhci_dev_t:: { ... } bot_last_status
- enum xhci_dev_t:: { ... } bot_tur_chain_target
- uint32_t bot_tur_retries
- uint32_t bot_read10_lba
- uint16_t bot_read10_num_blocks
- uint32_t bot_read10_block_size
- uint32_t bot_write10_lba
- uint16_t bot_write10_num_blocks
- uint32_t bot_write10_block_size
- uint32_t bot_cap_last_lba
- uint32_t bot_cap_block_size
- uint32_t bot_stall_recoveries
- uint32_t stall_dci
- uint8_t stall_ep_addr
- uint8_t bot_reset_clear_remaining
- uint32_t stall_retry_action

6.175.1 Member Enumeration Documentation

6.175.1.1 anonymous enum

anonymous enum

Enumerator

BOT_CMD_NONE	
BOT_CMD_TEST_UNIT_READY	
BOT_CMD_READ10	
BOT_CMD_READ_CAPACITY10	
BOT_CMD_WRITE10	

6.175.1.2 anonymous enum

anonymous enum

Enumerator

BOT_STATUS_IDLE	
BOT_STATUS_PASS	
BOT_STATUS_FAILED	
BOT_STATUS_TIMEOUT	

6.175.1.3 anonymous enum

anonymous enum

Enumerator

BOT_TUR_CHAIN_NONE	
BOT_TUR_CHAIN_READ10	
BOT_TUR_CHAIN_READ_CAPACITY10	
BOT_TUR_CHAIN_WRITE10	

6.175.1.4 anonymous enum

anonymous enum

Enumerator

XHCI_CONN_IDLE	
XHCI_CONN_AWAIT_PORT_RESET	
XHCI_CONN_AWAIT_ENABLE_SLOT	
XHCI_CONN_AWAIT_ADDRESS_DEVICE	

Enumerator

XHCI_CONN_AWAIT_DISABLE_SLOT	
XHCI_CONN_AWAIT_CONFIGURE_ENDPOINT	
XHCI_CONN_AWAIT_RESET_ENDPOINT	
XHCI_CONN_AWAIT_SET_TR_DEQUEUE	

6.175.2 Field Documentation

6.175.2.1 bar0_phys

```
uint64_t xhci_dev_t::bar0_phys
```

Referenced by xhci_find_and_map().

6.175.2.2 bot_cap_block_size

```
uint32_t xhci_dev_t::bot_cap_block_size
```

Referenced by blkio_usb_open_msc(), xhci_bringup(), and xhci_poll_one().

6.175.2.3 bot_cap_last_lba

```
uint32_t xhci_dev_t::bot_cap_last_lba
```

Referenced by blkio_usb_open_msc(), xhci_bringup(), and xhci_poll_one().

6.175.2.4 bot_cbw

```
usb_bot_cbw_t xhci_dev_t::bot_cbw
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), and xhci_bot_send_write10().

6.175.2.5 []

```
enum { ... } xhci_dev_t::bot_cmd_kind
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_wait_for_idle(), xhci_bringup(), xhci_handle_bulk_stall(), xhci_poll_one(), and xhci_stall_fail().

6.175.2.6 bot_csw

```
usb_bot_csw_t xhci_dev_t::bot_csw
```

Referenced by xhci_bot_receive_csw(), and xhci_poll_one().

6.175.2.7 bot_data_buf

```
uint8_t xhci_dev_t::bot_data_buf[1024]
```

Referenced by usb_blk_read(), usb_blk_write(), xhci_bot_read_block(), xhci_bot_read_data_in(), xhci_bot_send_read10(), xhci_bot_send_write10(), xhci_bot_write_block(), xhci_bot_write_data_out(), and xhci_poll_one().

6.175.2.8 bot_expected_data_len

```
uint32_t xhci_dev_t::bot_expected_data_len
```

Referenced by xhci_bot_read_data_in(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready xhci_bot_send_write10(), xhci_bot_write_data_out(), xhci_bringup(), and xhci_poll_one().

6.175.2.9 []

```
enum { ... } xhci_dev_t::bot_last_status
```

Referenced by xhci_bot_wait_for_idle(), xhci_bringup(), xhci_poll_one(), and xhci_stall_fail().

6.175.2.10 bot_last_tag

```
uint32_t xhci_dev_t::bot_last_tag
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bringup(), and xhci_poll_one().

6.175.2.11 bot_next_tag

```
uint32_t xhci_dev_t::bot_next_tag
```

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), and xhci_bringup().

6.175.2.12 bot_read10_block_size

```
uint32_t xhci_dev_t::bot_read10_block_size
```

Referenced by xhci_bot_read_block(), xhci_bringup(), and xhci_poll_one().

6.175.2.13 bot_read10_lba

```
uint32_t xhci_dev_t::bot_read10_lba
```

Referenced by xhci_bot_read_block(), xhci_bringup(), and xhci_poll_one().

6.175.2.14 bot_read10_num_blocks

```
uint16_t xhci_dev_t::bot_read10_num_blocks
```

Referenced by xhci_bot_read_block(), xhci_bringup(), and xhci_poll_one().

6.175.2.15 bot_reset_clear_remaining

```
uint8_t xhci_dev_t::bot_reset_clear_remaining
```

Referenced by xhci_poll_one(), and xhci_stall_fail().

6.175.2.16 bot_stall_recoveries

```
uint32_t xhci_dev_t::bot_stall_recoveries
```

Referenced by xhci_handle_bulk_stall().

6.175.2.17 []

```
enum { ... } xhci_dev_t::bot_tur_chain_target
```

Referenced by xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_write_block(), xhci_bringup(), and xhci_poll_one().

6.175.2.18 bot_tur_retries

```
uint32_t xhci_dev_t::bot_tur_retries
```

Referenced by xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_write_block(), xhci_bringup(), and xhci_poll_one().

6.175.2.19 bot_write10_block_size

```
uint32_t xhci_dev_t::bot_write10_block_size
```

Referenced by xhci_bot_write_block(), and xhci_poll_one().

6.175.2.20 bot_write10_lba

```
uint32_t xhci_dev_t::bot_write10_lba
```

Referenced by xhci_bot_write_block(), and xhci_poll_one().

6.175.2.21 bot_write10_num_blocks

```
uint16_t xhci_dev_t::bot_write10_num_blocks
```

Referenced by xhci_bot_write_block(), and xhci_poll_one().

6.175.2.22 cap

```
xhci_cap_regs_t* xhci_dev_t::cap
```

Referenced by xhci_bringup(), xhci_ctx_bytes(), xhci_find_and_map(), and xhci_take_from_bios().

6.175.2.23 cmd_ring

```
xhci_trb_t* xhci_dev_t::cmd_ring
```

Referenced by xhci_bringup(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), xhci_cmd_configure_hid_endpoint(), xhci_cmd_disable_slot(), xhci_cmd_enable_slot(), xhci_cmd_reset_endpoint(), xhci_cmd_set_tr_dequeue_pointer(), and xhci_submit_command().

6.175.2.24 cmd_ring_cycle

```
uint32_t xhci_dev_t::cmd_ring_cycle
```

Referenced by xhci_bringup(), and xhci_submit_command().

6.175.2.25 cmd_ring_enq

```
uint32_t xhci_dev_t::cmd_ring_enq
```

Referenced by xhci_bringup(), and xhci_submit_command().

6.175.2.26 []

```
enum { ... } xhci_dev_t::connect_state
```

Referenced by xhci_bringup(), xhci_drain_pending_event(), xhci_handle_bulk_stall(), xhci_handle_port_connected(), and xhci_poll_one().

6.175.2.27 dcbaa

```
void* xhci_dev_t::dcbaa
```

Referenced by xhci_bringup(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), xhci_cmd_configure_hid_endpoint(), and xhci_poll_one().

6.175.2.28 doorbell

```
xhci_doorbell_t* xhci_dev_t::doorbell
```

Referenced by xhci_bulk_in_enqueue_and_ring(), xhci_bulk_out_enqueue_and_ring(), xhci_ep0_bot_mass_storage_reset(), xhci_ep0_clear_endpoint_halt(), xhci_ep0_get_config_descriptor(), xhci_ep0_get_device_descriptor(), xhci_ep0_hid_set_leds(), xhci_ep0_hid_set_protocol(), xhci_ep0_set_configuration(), xhci_find_and_map(), xhci_hid_queue_report(), and xhci_submit_command().

6.175.2.29 evt_ring

```
xhci_trb_t* xhci_dev_t::evt_ring
```

Referenced by xhci_bringup(), and xhci_poll_one().

6.175.2.30 evt_ring_cycle

```
uint32_t xhci_dev_t::evt_ring_cycle
```

Referenced by xhci_bringup(), and xhci_poll_one().

6.175.2.31 evt_ring_deq

```
uint32_t xhci_dev_t::evt_ring_deq
```

Referenced by xhci_bringup(), and xhci_poll_one().

6.175.2.32 evt_ring_seg_table

```
void* xhci_dev_t::evt_ring_seg_table
```

Referenced by xhci_bringup().

6.175.2.33 intr0

```
xhci_intr_regs_t* xhci_dev_t::intr0
```

Referenced by xhci_bringup(), and xhci_poll_one().

6.175.2.34 is_connect

```
uint8_t xhci_dev_t::is_connect
```

Referenced by xhci_drain_pending_event(), and xhci_queue_pending_event().

6.175.2.35 max_intrs

```
uint32_t xhci_dev_t::max_intrs
```

Referenced by xhci_find_and_map().

6.175.2.36 max_ports

```
uint32_t xhci_dev_t::max_ports
```

Referenced by xhci_find_and_map(), xhci_port_regs(), and xhci_scan_ports_for_already_connected().

6.175.2.37 max_scratchpad_bufs

```
uint32_t xhci_dev_t::max_scratchpad_bufs
```

Referenced by xhci_bringup(), and xhci_find_and_map().

6.175.2.38 max_slots

```
uint32_t xhci_dev_t::max_slots
```

Referenced by sk_repl_idle(), sk_v4_idle(), usb_blk_state_for(), xhci_bringup(), xhci_find_and_map(), xhci_msc_slot_for(), and xhci_poll_one().

6.175.2.39 msc_slots

```
xhci_msc_slot_t* xhci_dev_t::msc_slots
```

Referenced by xhci_bringup(), and xhci_msc_slot_for().

6.175.2.40 op

```
xhci_op_regs_t* xhci_dev_t::op
```

Referenced by xhci_bringup(), xhci_find_and_map(), and xhci_port_regs().

6.175.2.41 pci

```
PciDevice xhci_dev_t::pci
```

Referenced by xhci_find_and_map().

6.175.2.42 pending_connect_port_id

```
uint32_t xhci_dev_t::pending_connect_port_id
```

Referenced by xhci_bringup(), xhci_handle_port_connected(), and xhci_poll_one().

6.175.2.43 pending_connect_slot_id

```
uint32_t xhci_dev_t::pending_connect_slot_id
```

Referenced by xhci_bringup(), and xhci_poll_one().

6.175.2.44 pending_connect_speed

```
uint32_t xhci_dev_t::pending_connect_speed
```

Referenced by xhci_bringup(), xhci_handle_port_connected(), and xhci_poll_one().

6.175.2.45 pending_disable_slot_id

```
uint32_t xhci_dev_t::pending_disable_slot_id
```

Referenced by xhci_bringup(), xhci_drain_pending_event(), and xhci_poll_one().

6.175.2.46 [struct]

```
struct { ... } xhci_dev_t::pending_events[ 32u ]
```

Referenced by xhci_bringup(), xhci_drain_pending_event(), and xhci_queue_pending_event().

6.175.2.47 port_id

```
uint32_t xhci_dev_t::port_id
```

Referenced by xhci_drain_pending_event(), and xhci_queue_pending_event().

6.175.2.48 port_slot_id

```
uint32_t xhci_dev_t::port_slot_id[32u]
```

Referenced by xhci_bringup(), and xhci_poll_one().

6.175.2.49 portsc

```
uint32_t xhci_dev_t::portsc
```

Referenced by xhci_drain_pending_event(), and xhci_queue_pending_event().

6.175.2.50 runtime

```
xhci_runtime_regs_t* xhci_dev_t::runtime
```

Referenced by xhci_bringup(), and xhci_find_and_map().

6.175.2.51 scratchpad_arr

```
void* xhci_dev_t::scratchpad_arr
```

Referenced by xhci_bringup().

6.175.2.52 slot_id

```
uint32_t xhci_dev_t::slot_id
```

Referenced by xhci_drain_pending_event(), and xhci_queue_pending_event().

6.175.2.53 stall_dci

```
uint32_t xhci_dev_t::stall_dci
```

Referenced by xhci_handle_bulk_stall().

6.175.2.54 stall_ep_addr

```
uint8_t xhci_dev_t::stall_ep_addr
```

Referenced by xhci_handle_bulk_stall(), and xhci_poll_one().

6.175.2.55 stall_retry_action

```
uint32_t xhci_dev_t::stall_retry_action
```

Referenced by xhci_handle_bulk_stall(), xhci_poll_one(), and xhci_stall_fail().

6.175.2.56 valid

```
uint8_t xhci_dev_t::valid
```

Referenced by xhci_bringup(), xhci_drain_pending_event(), and xhci_queue_pending_event().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci_driver.h

6.176 xhci_ep_ctx32_t Struct Reference

`#include <xhci.h>`

Data Fields

- volatile uint32_t dword0
- volatile uint32_t dword1
- volatile uint64_t tr_dequeue_ptr
- volatile uint32_t dword4
- volatile uint32_t reserved [3]

6.176.1 Field Documentation

6.176.1.1 dword0

`volatile uint32_t xhci_ep_ctx32_t::dword0`

Referenced by xhci_cmd_configure_hid_endpoint().

6.176.1.2 dword1

`volatile uint32_t xhci_ep_ctx32_t::dword1`

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

6.176.1.3 dword4

`volatile uint32_t xhci_ep_ctx32_t::dword4`

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

6.176.1.4 reserved

`volatile uint32_t xhci_ep_ctx32_t::reserved[3]`

6.176.1.5 tr_dequeue_ptr

`volatile uint64_t xhci_ep_ctx32_t::tr_dequeue_ptr`

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.177 xhci_input_ctrl_ctx32_t Struct Reference

`#include <xhci.h>`

Data Fields

- volatile uint32_t drop_flags
- volatile uint32_t add_flags
- volatile uint32_t reserved [5]
- volatile uint32_t config_word

6.177.1 Field Documentation

6.177.1.1 add_flags

`volatile uint32_t xhci_input_ctrl_ctx32_t::add_flags`

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

6.177.1.2 config_word

`volatile uint32_t xhci_input_ctrl_ctx32_t::config_word`

6.177.1.3 drop_flags

`volatile uint32_t xhci_input_ctrl_ctx32_t::drop_flags`

6.177.1.4 reserved

`volatile uint32_t xhci_input_ctrl_ctx32_t::reserved[5]`

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.178 xhci_intr_regs_t Struct Reference

`#include <xhci.h>`

Data Fields

- volatile uint32_t iman
- volatile uint32_t imod
- volatile uint32_t erstsz
- volatile uint32_t reserved0
- volatile uint64_t erstba
- volatile uint64_t erdp

6.178.1 Field Documentation

6.178.1.1 erdp

```
volatile uint64_t xhci_intr_regs_t::erdp
```

Referenced by xhci_bringup(), and xhci_poll_one().

6.178.1.2 erstba

```
volatile uint64_t xhci_intr_regs_t::erstba
```

Referenced by xhci_bringup().

6.178.1.3 erstsz

```
volatile uint32_t xhci_intr_regs_t::erstsz
```

Referenced by xhci_bringup().

6.178.1.4 iman

```
volatile uint32_t xhci_intr_regs_t::iman
```

6.178.1.5 imod

```
volatile uint32_t xhci_intr_regs_t::imod
```

6.178.1.6 reserved0

```
volatile uint32_t xhci_intr_regs_t::reserved0
```

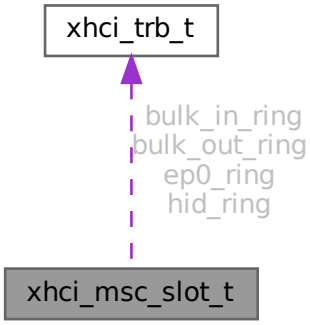
The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.179 xhci_msc_slot_t Struct Reference

`#include <xhci_driver.h>`

Collaboration diagram for xhci_msc_slot_t:



Data Fields

- uint32_t in_use
- void * input_ctx
- void * device_ctx
- xhci_trb_t * ep0_ring
- uint32_t ep0_ring_cycle
- uint32_t ep0_ring_enq
- uint8_t device_descriptor [18]
- uint8_t config_descriptor [128]
- uint16_t config_total_length
- uint8_t bulk_in_ep_addr
- uint16_t bulk_in_max_packet
- uint8_t bulk_out_ep_addr
- uint16_t bulk_out_max_packet
- xhci_trb_t * bulk_in_ring
- uint32_t bulk_in_ring_cycle
- uint32_t bulk_in_ring_enq
- xhci_trb_t * bulk_out_ring
- uint32_t bulk_out_ring_cycle
- uint32_t bulk_out_ring_enq
- uint8_t hid_kbd
- uint8_t hid_running
- uint8_t hid_iface
- uint8_t hid_ep_addr
- uint16_t hid_max_packet
- uint8_t hid_interval
- xhci_trb_t * hid_ring
- uint32_t hid_ring_cycle
- uint32_t hid_ring_enq

- uint8_t * hid_buf
- uint8_t hid_prev [8]
- uint8_t hid_leds_sent
- uint8_t bot_msc_attach_pending
- uint8_t bot_msc_attached
- uint8_t bot_msc_detach_pending
- xhci_transfer_purpose_t transfer_purpose
- xhci_next_action_t next_action
- uint16_t next_action_length
- uint8_t next_action_config_value

6.179.1 Field Documentation

6.179.1.1 bot_msc_attach_pending

```
uint8_t xhci_msc_slot_t::bot_msc_attach_pending
```

Referenced by sk_repl_idle(), sk_v4_idle(), and xhci_poll_one().

6.179.1.2 bot_msc_attached

```
uint8_t xhci_msc_slot_t::bot_msc_attached
```

Referenced by sk_v4_idle(), sk_word_blk_attach_ack(), and xhci_poll_one().

6.179.1.3 bot_msc_detach_pending

```
uint8_t xhci_msc_slot_t::bot_msc_detach_pending
```

Referenced by sk_repl_idle(), sk_v4_idle(), and xhci_poll_one().

6.179.1.4 bulk_in_ep_addr

```
uint8_t xhci_msc_slot_t::bulk_in_ep_addr
```

Referenced by xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_block(), xhci_bulk_in_enqueue_and_ring(), xhci_cmd_configure_endpoint(), xhci_handle_bulk_stall(), and xhci_poll_one().

6.179.1.5 bulk_in_max_packet

```
uint16_t xhci_msc_slot_t::bulk_in_max_packet
```

Referenced by xhci_cmd_configure_endpoint(), and xhci_poll_one().

6.179.1.6 bulk_in_ring

```
xhci_trb_t* xhci_msc_slot_t::bulk_in_ring
```

Referenced by xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_read_data_in(), xhci_bot_receive_csw(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_block(), xhci_bulk_in_enqueue_and_ring(), xhci_cmd_configure_endpoint(), and xhci_poll_one().

6.179.1.7 bulk_in_ring_cycle

```
uint32_t xhci_msc_slot_t::bulk_in_ring_cycle
```

Referenced by xhci_bulk_in_enqueue_and_ring(), xhci_cmd_configure_endpoint(), and xhci_poll_one().

6.179.1.8 bulk_in_ring_enq

```
uint32_t xhci_msc_slot_t::bulk_in_ring_enq
```

Referenced by xhci_bulk_in_enqueue_and_ring(), xhci_cmd_configure_endpoint(), and xhci_poll_one().

6.179.1.9 bulk_out_ep_addr

```
uint8_t xhci_msc_slot_t::bulk_out_ep_addr
```

Referenced by xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_block(), xhci_bulk_out_enqueue_and_ring(), xhci_cmd_configure_endpoint(), xhci_handle_bulk_stall(), and xhci_poll_one().

6.179.1.10 bulk_out_max_packet

```
uint16_t xhci_msc_slot_t::bulk_out_max_packet
```

Referenced by xhci_cmd_configure_endpoint(), and xhci_poll_one().

6.179.1.11 bulk_out_ring

```
xhci_trb_t* xhci_msc_slot_t::bulk_out_ring
```

Referenced by xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_block(), xhci_bot_write_data_out(), xhci_bulk_out_enqueue_and_ring(), xhci_cmd_configure_endpoint(), and xhci_poll_one().

6.179.1.12 bulk_out_ring_cycle

```
uint32_t xhci_msc_slot_t::bulk_out_ring_cycle
```

Referenced by xhci_bulk_out_enqueue_and_ring(), xhci_cmd_configure_endpoint(), and xhci_poll_one().

6.179.1.13 bulk_out_ring_enq

```
uint32_t xhci_msc_slot_t::bulk_out_ring_enq
```

Referenced by xhci_bulk_out_enqueue_and_ring(), xhci_cmd_configure_endpoint(), and xhci_poll_one().

6.179.1.14 config_descriptor

```
uint8_t xhci_msc_slot_t::config_descriptor[128]
```

Referenced by xhci_ep0_get_config_descriptor(), and xhci_poll_one().

6.179.1.15 config_total_length

```
uint16_t xhci_msc_slot_t::config_total_length
```

Referenced by xhci_poll_one().

6.179.1.16 device_ctx

```
void* xhci_msc_slot_t::device_ctx
```

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

6.179.1.17 device_descriptor

```
uint8_t xhci_msc_slot_t::device_descriptor[18]
```

Referenced by xhci_ep0_get_device_descriptor(), and xhci_poll_one().

6.179.1.18 ep0_ring

```
xhci_trb_t* xhci_msc_slot_t::ep0_ring
```

Referenced by xhci_cmd_address_device(), xhci_ep0_bot_mass_storage_reset(), xhci_ep0_clear_endpoint_halt(), xhci_ep0_enqueue_trb(), xhci_ep0_get_config_descriptor(), xhci_ep0_get_device_descriptor(), xhci_ep0_hid_set_leds(), xhci_ep0_hid_set_protocol(), and xhci_ep0_set_configuration().

6.179.1.19 ep0_ring_cycle

```
uint32_t xhci_msc_slot_t::ep0_ring_cycle
```

Referenced by xhci_cmd_address_device(), and xhci_ep0_enqueue_trb().

6.179.1.20 ep0_ring_enq

```
uint32_t xhci_msc_slot_t::ep0_ring_enq
```

Referenced by xhci_cmd_address_device(), and xhci_ep0_enqueue_trb().

6.179.1.21 hid_buf

```
uint8_t* xhci_msc_slot_t::hid_buf
```

Referenced by xhci_cmd_configure_hid_endpoint(), xhci_ep0_hid_set_leds(), xhci_hid_queue_report(), and xhci_poll_one().

6.179.1.22 hid_ep_addr

```
uint8_t xhci_msc_slot_t::hid_ep_addr
```

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_hid_endpoint(), xhci_hid_queue_report(), and xhci_poll_one().

6.179.1.23 hid_iface

```
uint8_t xhci_msc_slot_t::hid_iface
```

Referenced by xhci_ep0_hid_set_leds(), xhci_ep0_hid_set_protocol(), and xhci_poll_one().

6.179.1.24 hid_interval

```
uint8_t xhci_msc_slot_t::hid_interval
```

Referenced by xhci_cmd_configure_hid_endpoint(), and xhci_poll_one().

6.179.1.25 hid_kbd

```
uint8_t xhci_msc_slot_t::hid_kbd
```

Referenced by xhci_cmd_address_device(), and xhci_poll_one().

6.179.1.26 hid_leds_sent

```
uint8_t xhci_msc_slot_t::hid_leds_sent
```

Referenced by xhci_ep0_hid_set_leds(), and xhci_poll_one().

6.179.1.27 hid_max_packet

```
uint16_t xhci_msc_slot_t::hid_max_packet
```

Referenced by xhci_cmd_configure_hid_endpoint(), xhci_hid_queue_report(), and xhci_poll_one().

6.179.1.28 hid_prev

```
uint8_t xhci_msc_slot_t::hid_prev[8]
```

Referenced by xhci_poll_one().

6.179.1.29 hid_ring

```
xhci_trb_t* xhci_msc_slot_t::hid_ring
```

Referenced by xhci_cmd_configure_hid_endpoint(), and xhci_hid_queue_report().

6.179.1.30 hid_ring_cycle

```
uint32_t xhci_msc_slot_t::hid_ring_cycle
```

Referenced by xhci_cmd_configure_hid_endpoint(), and xhci_hid_queue_report().

6.179.1.31 hid_ring_enq

```
uint32_t xhci_msc_slot_t::hid_ring_enq
```

Referenced by xhci_cmd_configure_hid_endpoint(), and xhci_hid_queue_report().

6.179.1.32 hid_running

```
uint8_t xhci_msc_slot_t::hid_running
```

Referenced by xhci_cmd_address_device(), and xhci_poll_one().

6.179.1.33 in_use

```
uint32_t xhci_msc_slot_t::in_use
```

Referenced by xhci_cmd_address_device(), and xhci_poll_one().

6.179.1.34 input_ctx

```
void* xhci_msc_slot_t::input_ctx
```

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

6.179.1.35 next_action

```
xhci_next_action_t xhci_msc_slot_t::next_action
```

Referenced by xhci_handle_bulk_stall(), and xhci_poll_one().

6.179.1.36 next_action_config_value

```
uint8_t xhci_msc_slot_t::next_action_config_value
```

Referenced by xhci_poll_one().

6.179.1.37 next_action_length

```
uint16_t xhci_msc_slot_t::next_action_length
```

Referenced by xhci_poll_one().

6.179.1.38 transfer_purpose

```
xhci_transfer_purpose_t xhci_msc_slot_t::transfer_purpose
```

Referenced by xhci_bot_read_data_in(), xhci_bot_receive_csw(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_data_out(), xhci_ep0_bot_mass_storage_reset(), xhci_ep0_clear_endpoint_halt(), xhci_ep0_get_config_descriptor(), xhci_ep0_get_device_descriptor(), xhci_ep0_hid_set_leds(), xhci_ep0_hid_set_protocol(), xhci_ep0_set_configuration(), and xhci_poll_one().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci_driver.h

6.180 xhci_op_regs_t Struct Reference

```
#include <xhci.h>
```

Data Fields

- volatile uint32_t usb_cmd
- volatile uint32_t usb_sts
- volatile uint32_t page_size
- volatile uint32_t reserved0 [2]
- volatile uint32_t dn_ctrl
- volatile uint64_t crcr
- volatile uint32_t reserved1 [4]
- volatile uint64_t dcbaap
- volatile uint32_t config

6.180.1 Field Documentation

6.180.1.1 config

```
volatile uint32_t xhci_op_regs_t::config
```

Referenced by xhci_bringup().

6.180.1.2 crcr

```
volatile uint64_t xhci_op_regs_t::crcr
```

Referenced by xhci_bringup().

6.180.1.3 dcbaap

```
volatile uint64_t xhci_op_regs_t::dcbaap
```

Referenced by xhci_bringup().

6.180.1.4 dn_ctrl

```
volatile uint32_t xhci_op_regs_t::dn_ctrl
```

6.180.1.5 page_size

```
volatile uint32_t xhci_op_regs_t::page_size
```

Referenced by xhci_bringup().

6.180.1.6 reserved0

```
volatile uint32_t xhci_op_regs_t::reserved0[2]
```

6.180.1.7 reserved1

```
volatile uint32_t xhci_op_regs_t::reserved1[4]
```

6.180.1.8 usb_cmd

```
volatile uint32_t xhci_op_regs_t::usb_cmd
```

Referenced by xhci_bringup().

6.180.1.9 usb_sts

```
volatile uint32_t xhci_op_regs_t::usb_sts
```

Referenced by xhci_bringup().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.181 xhci_port_regs_t Struct Reference

```
#include <xhci.h>
```

Data Fields

- volatile uint32_t portsc
- volatile uint32_t portpmsc
- volatile uint32_t portli
- volatile uint32_t porthlpmc

6.181.1 Field Documentation

6.181.1.1 porthlpmc

```
volatile uint32_t xhci_port_regs_t::porthlpmc
```

6.181.1.2 portli

```
volatile uint32_t xhci_port_regs_t::portli
```

6.181.1.3 portpmsc

```
volatile uint32_t xhci_port_regs_t::portpmsc
```

6.181.1.4 portsc

```
volatile uint32_t xhci_port_regs_t::portsc
```

Referenced by xhci_handle_port_connected(), xhci_poll_one(), and xhci_scan_ports_for_already_connected().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.182 xhci_runtime_regs_t Struct Reference

`#include <xhci.h>`

Data Fields

- volatile uint32_t mf_index
- volatile uint32_t reserved0 [7]

6.182.1 Field Documentation

6.182.1.1 mf_index

`volatile uint32_t xhci_runtime_regs_t::mf_index`

6.182.1.2 reserved0

`volatile uint32_t xhci_runtime_regs_t::reserved0[7]`

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.183 xhci_slot_ctx32_t Struct Reference

`#include <xhci.h>`

Data Fields

- volatile uint32_t dword0
- volatile uint32_t dword1
- volatile uint32_t dword2
- volatile uint32_t dword3
- volatile uint32_t reserved [4]

6.183.1 Field Documentation

6.183.1.1 dword0

`volatile uint32_t xhci_slot_ctx32_t::dword0`

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

6.183.1.2 dword1

```
volatile uint32_t xhci_slot_ctx32_t::dword1
```

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

6.183.1.3 dword2

```
volatile uint32_t xhci_slot_ctx32_t::dword2
```

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

6.183.1.4 dword3

```
volatile uint32_t xhci_slot_ctx32_t::dword3
```

6.183.1.5 reserved

```
volatile uint32_t xhci_slot_ctx32_t::reserved[4]
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.184 xhci_trb_t Struct Reference

```
#include <xhci.h>
```

Data Fields

- volatile uint64_t parameter
- volatile uint32_t status
- volatile uint32_t control

6.184.1 Field Documentation

6.184.1.1 control

```
volatile uint32_t xhci_trb_t::control
```

Referenced by xhci_bringup(), xhci_bulk_in_enqueue_and_ring(), xhci_bulk_out_enqueue_and_ring(), xhci_cmd_address_device(), xhci_ep0_enqueue_trb(), xhci_hid_queue_report(), xhci_poll_one(), and xhci_submit_command().

6.184.1.2 parameter

```
volatile uint64_t xhci_trb_t::parameter
```

Referenced by xhci_bringup(), xhci_bulk_in_enqueue_and_ring(), xhci_bulk_out_enqueue_and_ring(), xhci_cmd_address_device(), xhci_ep0_enqueue_trb(), xhci_hid_queue_report(), xhci_poll_one(), and xhci_submit_command().

6.184.1.3 status

```
volatile uint32_t xhci_trb_t::status
```

Referenced by xhci_bringup(), xhci_bulk_in_enqueue_and_ring(), xhci_bulk_out_enqueue_and_ring(), xhci_cmd_address_device(), xhci_ep0_enqueue_trb(), xhci_hid_queue_report(), xhci_poll_one(), and xhci_submit_command().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xhci.h

6.185 XXHash64State Struct Reference

Streaming state structure.

```
#include <xxhash64.h>
```

Data Fields

- uint64_t total_len
- uint64_t v1
- uint64_t v2
- uint64_t v3
- uint64_t v4
- uint8_t buffer [32]
- uint32_t buffer_size
- uint64_t seed

6.185.1 Detailed Description

Streaming state structure.

6.185.2 Field Documentation

6.185.2.1 buffer

```
uint8_t XXHash64State::buffer[32]
```

Referenced by xxhash64_digest(), and xxhash64_update().

6.185.2.2 buffer_size

```
uint32_t XXHash64State::buffer_size
```

Referenced by xxhash64_digest(), xxhash64_reset(), and xxhash64_update().

6.185.2.3 seed

```
uint64_t XXHash64State::seed
```

Referenced by xxhash64_digest(), and xxhash64_reset().

6.185.2.4 total_len

```
uint64_t XXHash64State::total_len
```

Referenced by xxhash64_digest(), xxhash64_reset(), and xxhash64_update().

6.185.2.5 v1

```
uint64_t XXHash64State::v1
```

Referenced by xxhash64_digest(), xxhash64_reset(), and xxhash64_update().

6.185.2.6 v2

```
uint64_t XXHash64State::v2
```

Referenced by xxhash64_digest(), xxhash64_reset(), and xxhash64_update().

6.185.2.7 v3

```
uint64_t XXHash64State::v3
```

Referenced by xxhash64_digest(), xxhash64_reset(), and xxhash64_update().

6.185.2.8 v4

```
uint64_t XXHash64State::v4
```

Referenced by xxhash64_digest(), xxhash64_reset(), and xxhash64_update().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/xxhash64.h

6.186 zuse_cert_devblock_t Struct Reference

`#include <zuse_cert_devblock.h>`

Data Fields

- uint32_t magic
- uint32_t version
- uint8_t seed [32]
- uint8_t pubkey [32]
- uint64_t crc
- uint8_t _pad [4096 -(4+4+32+32+8)]

6.186.1 Field Documentation

6.186.1.1 _pad

```
uint8_t zuse_cert_devblock_t::_pad[4096 -(4+4+32+32+8)]
```

6.186.1.2 crc

```
uint64_t zuse_cert_devblock_t::crc
```

6.186.1.3 magic

```
uint32_t zuse_cert_devblock_t::magic
```

6.186.1.4 pubkey

```
uint8_t zuse_cert_devblock_t::pubkey[32]
```

6.186.1.5 seed

```
uint8_t zuse_cert_devblock_t::seed[32]
```

6.186.1.6 version

```
uint32_t zuse_cert_devblock_t::version
```

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/zuse_cert_devblock.h

6.187 zuse_genesis_marker_t Struct Reference

`#include <zuse_genesis_marker.h>`

Data Fields

- uint32_t magic
- uint32_t version
- uint8_t zuse_pubkey [32]
- uint64_t crc
- uint8_t _pad [4096 -(4+4+32+8)]

6.187.1 Field Documentation

6.187.1.1 _pad

`uint8_t zuse_genesis_marker_t::_pad[4096 -(4+4+32+8)]`

6.187.1.2 crc

`uint64_t zuse_genesis_marker_t::crc`

Referenced by capsule_zuse_boot_try_attach(), and genesis_marker_read().

6.187.1.3 magic

`uint32_t zuse_genesis_marker_t::magic`

Referenced by capsule_zuse_boot_try_attach(), and genesis_marker_read().

6.187.1.4 version

`uint32_t zuse_genesis_marker_t::version`

Referenced by capsule_zuse_boot_try_attach(), and genesis_marker_read().

6.187.1.5 zuse_pubkey

`uint8_t zuse_genesis_marker_t::zuse_pubkey[32]`

Referenced by capsule_zuse_boot_check(), capsule_zuse_boot_root_pubkey(), and capsule_zuse_boot_try_attach().

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/zuse_genesis_marker.h

6.188 zuse_genesis_marker_v2_t Struct Reference

`#include <zuse_genesis_marker.h>`

Data Fields

- uint32_t magic
- uint32_t version
- uint8_t proxy_pubkey [32]
- uint8_t real_pubkey [32]
- uint32_t has_real
- uint32_t _reserved
- uint64_t crc
- uint8_t _pad [4096 -(4+4+32+32+4+4+8)]

6.188.1 Field Documentation

6.188.1.1 _pad

`uint8_t zuse_genesis_marker_v2_t::_pad[4096 -(4+4+32+32+4+4+8)]`

6.188.1.2 _reserved

`uint32_t zuse_genesis_marker_v2_t::_reserved`

6.188.1.3 crc

`uint64_t zuse_genesis_marker_v2_t::crc`

6.188.1.4 has_real

`uint32_t zuse_genesis_marker_v2_t::has_real`

6.188.1.5 magic

`uint32_t zuse_genesis_marker_v2_t::magic`

6.188.1.6 proxy_pubkey

`uint8_t zuse_genesis_marker_v2_t::proxy_pubkey[32]`

6.188.1.7 real_pubkey

`uint8_t zuse_genesis_marker_v2_t::real_pubkey[32]`

6.188.1.8 version

`uint32_t zuse_genesis_marker_v2_t::version`

The documentation for this struct was generated from the following file:

- kernel/include/starkernel/zuse_genesis_marker.h

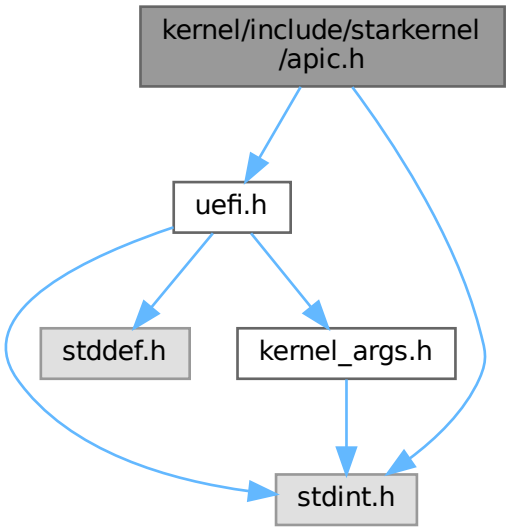
Chapter 7

File Documentation

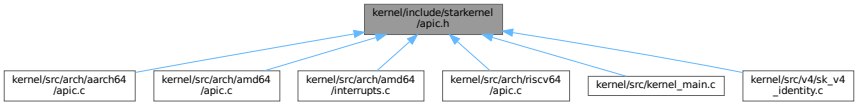
7.1 kernel/include/starkernel/apic.h File Reference

```
#include "uefi.h"
#include <stdint.h>
```

Include dependency graph for apic.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define APIC_TIMER_VECTOR 0x20

  apic.h - Local APIC interface
- #define APIC_SPURIOUS_VECTOR 0xFF

Functions

- int apic_init (BootInfo *boot_info)

  Initialise the GICv2 distributor and CPU interface (M4 milestone).
- uint8_t apic_id (void)

  Return this CPU's xAPIC ID.
- void apic_eoi (void)

  Signal End of Interrupt (common cross-ISA signature — see below).
- int apic_timer_init (uint64_t tsc_hz, uint32_t tick_hz)

  Compute the expected system-counter ticks per heartbeat period.
- void apic_timer_start (void)

  Start the APIC timer (enables periodic interrupts).
- void apic_timer_stop (void)

  Stop the APIC timer (disables periodic interrupts).
- void apic_timer_rearm (void)

  Re-arm the APIC timer at the current adaptive period (item 0.8, ?26).
- uint64_t apic_timer_period_tsc (void)

  Get the configured timer period in TSC ticks.

7.1.1 Macro Definition Documentation

7.1.1.1 APIC_SPURIOUS_VECTOR

```
#define APIC_SPURIOUS_VECTOR 0xFF
```

7.1.1.2 APIC_TIMER_VECTOR

```
#define APIC_TIMER_VECTOR 0x20
```

apic.h - Local APIC interface

7.1.2 Function Documentation

7.1.2.1 apic_eoi()

```
void apic_eoi (
            void  )
```

Signal End of Interrupt (common cross-ISA signature — see below).

The shared apic.h interface has no way to pass which interrupt is being completed, but GICv2's GICC↩ _EOIR needs the exact INTID GICC_IAR returned, not just "an" EOI. aarch64_irq_handler() calls apic_eoi_intid() directly with that value instead; this form exists only to satisfy the common call site and is not used on the interrupt path GIC brings up.

Signal End of Interrupt (common cross-ISA signature — see below).

Writes zero to APIC_REG_EOI (offset 0x0B0). This informs the APIC that the currently serviced interrupt has been handled so that the next interrupt in the vector's priority class can be delivered. Must be called at the end of every interrupt service routine before IRET; omitting the EOI stalls all future interrupt delivery at the same or lower priority.

For the APIC timer heartbeat path this is called inside isr_common_handler() immediately after heartbeat_tick().

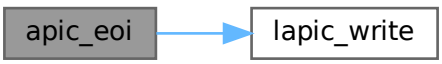
Signal End of Interrupt (common cross-ISA signature — see below).

On x86-64 this writes to the Local APIC EOI register. On RISC-V EOI is handled by reading the PLIC claim/complete register; that path is not yet wired. This stub is a no-op that satisfies the common interface.

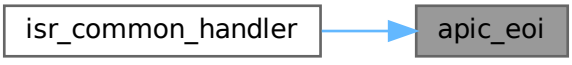
References APIC_REG_EOI, and lapic_write().

Referenced by isr_common_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.2 apic_id()

```
uint8_t apic_id (
            void  )
```

Return this CPU's xAPIC ID.

Reads `APIC_REG_ID` and extracts bits 31:24, the xAPIC ID field. Used by the I/O APIC driver (item 4.3.5) as the destination field when programming a redirection entry — interrupts routed to this ID are delivered to the current CPU's Local APIC.

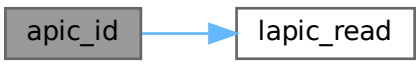
Returns

This CPU's 8-bit xAPIC ID.

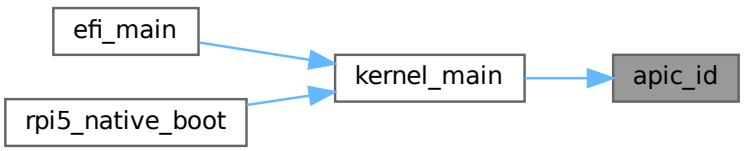
References APIC_REG_ID, and lapic_read().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.3 apic_init()

```
int apic_init (
            BootInfo * boot_info )
```

Initialise the GICv2 distributor and CPU interface (M4 milestone).

Enables exactly one interrupt source: the physical timer PPI for the exception level detected in item 0.4 (30 at EL1, 26 at EL2 — see `TIMER_PPI_EL1` / `TIMER_PPI_EL2` above). Sequence:

0. Try `gic_bases_from_dtb()` against `boot_info->dtb`; on success, `s_gicd_base` / `s_gicc_base` are overridden from the real hardware addresses. On failure (no DTB, e.g. this system's QEMU/UEFI firmware — see file header) they keep the QEMU constant defaults, exactly the behaviour this function had before this step existed.

1. Set this interrupt's priority (`GICD_IPRIORITYR`) below the CPU interface's priority mask, so it is never itself masked out.
2. Enable it in `GICD_ISENABLER0` (PPIs 16–31 live in the first word).
3. Enable the distributor, Group 0 only (`GICD_CTLR`) — this deployment has no GICv2 security-extension split in use.
4. Set the CPU interface priority mask to allow everything through (`GICC_PMR` = 0xFF) and enable the CPU interface (`GICC_CTLR`).

Deliberately not configured: `GICD_ICFGR` (trigger sensitivity). The ARM generic timer's PPI is architecturally level-sensitive; this is not a general GIC driver and item 0.6's own scope does not call for reconfiguring it.

Parameters

boot_info	Kernel `BootInfo`; `dtb` is consulted for real-hardware GIC base discovery as of the 2026-09-04 fix (previously unused — see file header), and, as of the same fix, for `aarch64_psci_conduit_init()`'s PSCI conduit discovery (arch.c) — bundled into this call for the same reason GIC discovery is: this is the one point in boot with `boot_info->dtb` already in hand.

Returns

0 always.

Initialise the GICv2 distributor and CPU interface (M4 milestone).

Called during M4 (interrupt controller init) before the IDT is loaded. Performs the following steps:

1. Reads `IA32_APIC_BASE_MSR` (0x1B) to determine the current APIC mode.
2. If x2APIC mode (bit 10) is active, clears it to switch to xAPIC MMIO mode while keeping the global-enable bit (bit 11) set, as required by Intel SDM Vol.3 ?10.12.5 (never clear both bits simultaneously).
3. If the global-enable bit (bit 11) was clear, sets it.
4. Software-enables the APIC via `APIC_REG_SIVR` (Spurious Interrupt Vector Register) and assigns spurious vector 0xFF.

5. Clears the Task Priority Register (TPR=0) so all interrupt priorities are delivered; a non-zero TPR left by UEFI would silently suppress all APIC interrupts at or below that priority level.
6. Issues a spurious EOI write to clear any stale in-service interrupt that UEFI may have left pending (specifically the APIC timer vector that fires just before `ExitBootServices`).

The `boot_info` parameter is reserved for future use (e.g., reading the ACPI MADT for the physical LAPIC base address); it is currently unused and the default identity-mapped address `LAPIC_DEFAULT_PHYS` = 0xFEE00000 is always used.

Parameters

boot_info	Pointer to kernel `BootInfo` (ACPI/memory-map data); currently unused, reserved for MADT-based LAPIC relocation.

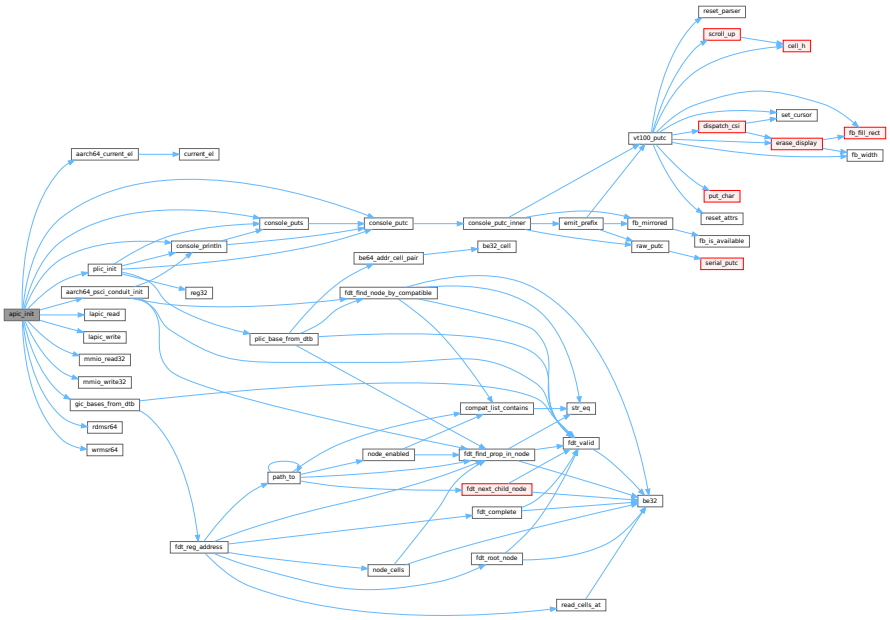
Returns

0 on success (always; the APIC is assumed to be present on x86-64).

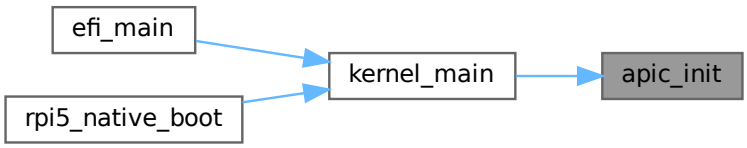
References aarch64_current_el(), aarch64_psci_conduit_init(), APIC_REG_EOI, APIC_REG_ISR_BASE, APIC_REG_SIVR, APIC_REG_TPR, console_println(), console_putc(), console_puts(), BootInfo::dtb, gic_bases_from_dtb(), GICC_CTLR, GICC_CTLR_ENABLE_GRP0, GICC_PMR, GICD_CTLR, GICD_CTLR_ENABLE_GRP0, GICD_IPRIORITYR, GICD_ISENABLER0, IA32_APIC_BASE_GLOBAL_EN, IA32_APIC_BASE_MSR, IA32_APIC_BASE_X2APIC_, lapic_base, LAPIC_DEFAULT_PHYS, lapic_phys_base, lapic_read(), LAPIC_VIRT_BASE, lapic_write(), mmio_read32(), mmio_write32(), plic_init(), PMR_ALLOW_ALL, rdmsr64(), s_gicc_base, s_gicd_base, s_timer_ppi, TIMER_PPI_EL1, TIMER_PPI_EL2, TIMER_PRIORITY, and wrmsr64().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.4 apic_timer_init()

```
int apic_timer_init (
            uint64_t tsc_hz,
            uint32_t tick_hz )
```

Compute the expected system-counter ticks per heartbeat period.

Computes `s_timer_period_tsc` (the expected number of `CNTPCT_EL0` ticks between heartbeats), used both by `heartbeat_init()` to seed the rolling-window `expected_delta` and by `apic_timer_start()` / `apic_timer_rearm()` (item 0.7) as the `TVAL` step written on every arm and re-arm.

If either `tsc_hz` or `tick_hz` is zero, a fallback of 10,000,000 ticks is stored (approximately 160 ms at 62.5 MHz, a safe non-zero sentinel that prevents division-by-zero in the heartbeat path).

Parameters

tsc_hz	System counter frequency from `timer_init()` (Hz).
tick_hz	Desired heartbeat rate (Hz); 0 → uses fallback.

Returns

0 always.

Compute the expected system-counter ticks per heartbeat period.

Calibrates the APIC timer frequency via `calibrate_apic_timer()` then programs the LVT Timer Register and Initial Count Register to deliver `APIC_TIMER_VECTOR` periodically at `tick_hz` interrupts per second. The timer is left **masked** after initialisation; call `apic_timer_start()` after the IDT is loaded and interrupts are enabled to begin delivery.

Derived quantities stored for later use:

- `timer_initial_count` — APIC ticks per heartbeat period (`apic_hz` / `tick_hz`), written to `APIC_REG_TIMER_ICR`.
- `timer_tick_hz` — requested tick rate (100 Hz default).
- `timer_period_tsc_ticks` — expected TSC ticks per heartbeat (`tsc_hz` / `tick_hz`), used by `heartbeat_tick()` to compute inter-tick variance for the TIME-TRUST Q48.16 metric.

If `tick_hz` is 0 it defaults to 100 Hz (10 ms period). If calibration returns 0 (no APIC timer detected or counter stuck) the function prints a diagnostic and returns -1 without programming the timer.

Parameters

tsc_hz	TSC frequency in Hz (from `timer_init()`); used to derive the 10 ms calibration window and `timer_period_tsc_ticks`. Pass 0 if TSC is not yet calibrated.
tick_hz	Desired interrupt rate in Hz; 0 → defaults to 100 Hz.

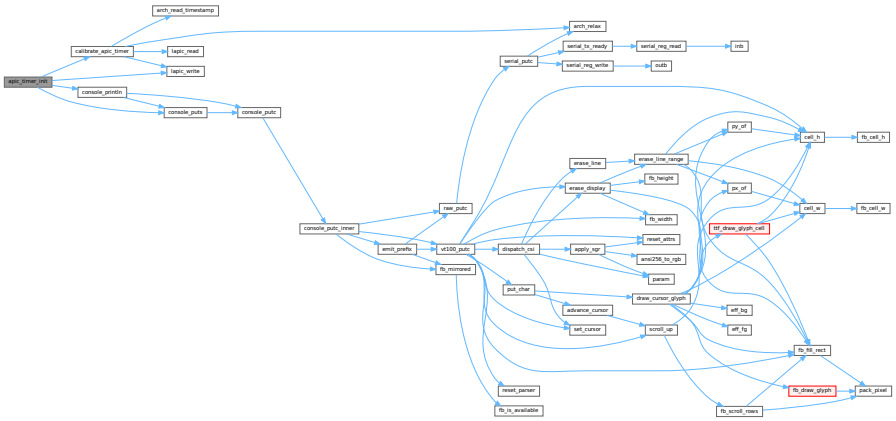
Returns

0 on success, -1 if APIC timer calibration failed.

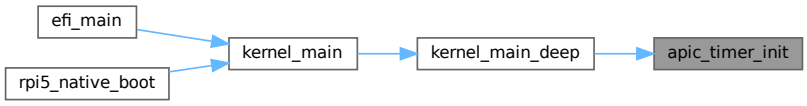
References APIC_REG_LVT_TIMER, APIC_REG_TIMER_DCR, APIC_REG_TIMER_ICR, APIC_TIMER_VECTOR, calibrate_apic_timer(), console_println(), console_puts(), lapic_write(), LVT_MASKED, LVT_MODE_PERIODIC, s_apic_hz, s_counter_hz_apic, s_time_hz, s_timer_period_tsc, TIMER_DIV_1, timer_initial_count, timer_period_tsc_ticks, and timer_tick_hz.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.5 apic_timer_period_tsc()

```
uint64_t apic_timer_period_tsc (
            void  )
```

Get the configured timer period in TSC ticks.

Get the configured timer period in TSC ticks.

Returns `s_timer_period_tsc`, set by `apic_timer_init()` as `tsc_hz / tick_hz`. This value is consumed by `heartbeat_init()` to seed the rolling-window `expected_delta` so that inter-tick deviation measurements are relative to the correct nominal interval.

Returns

Expected CNTPCT_EL0 ticks per heartbeat period; 10,000,000 if frequencies were not available at init time.

Get the configured timer period in TSC ticks.

Returns timer_period_tsc_ticks, computed by apic_timer_init() as tsc_hz / tick_hz. The value represents the ideal number of TSC ticks that should elapse between consecutive APIC_TIMER_VECTOR interrupts when the TSC runs at tsc_hz Hz and the heartbeat runs at tick_hz Hz.

Used by timer.c to seed the heartbeat rolling window with the nominal inter-tick interval so that the first few ticks produce meaningful variance estimates rather than comparing against zero. Returns 0 if tsc_hz was 0 at initialisation time (TSC unknown).

Returns

Expected TSC ticks per heartbeat period, or 0 if unavailable.

Get the configured timer period in TSC ticks.

Returns s_timer_period_tsc, set by apic_timer_init() as tsc_hz / tick_hz. Consumed by heartbeat_init() to seed the rolling-window expected_delta, and by apic_timer_start() / riscv64_timer_rearm() as the SBI deadline step.

Returns

Expected time-counter ticks per heartbeat period; 10,000,000 if frequencies were not available at init time.

References s_timer_period_tsc, and timer_period_tsc_ticks.

7.1.2.6 apic_timer_rearm()

```
void apic_timer_rearm (
            void  )
```

Re-arm the APIC timer at the current adaptive period (item 0.8, ?26).

Recomputes the initial count from heartbeat_next_period_ns() and writes it to the ICR; a periodic-mode ICR write restarts the countdown immediately at the new value. Called once per tick from the ISR, before heartbeat_tick().

Re-arm the APIC timer at the current adaptive period (item 0.8, ?26).

Writes a fresh TVAL — the entirety of a periodic re-arm, since TVAL is relative to "now" rather than an absolute deadline. Unlike riscv64's SBI re-arm (item 0.3), there is no drift-correction bookkeeping: each write means "@c s_timer_period_tsc ticks from this instant," so time spent servicing the interrupt before this call simply lengthens that one period rather than accumulating error across ticks.

This is the one call that must never be skipped. The ARM Generic Timer does not auto-reload: once TVAL crosses zero, CTL.ISTATUS latches and stays asserted — a level condition — until software writes a new TVAL or clears CTL.ENABLE. A return path that skips this leaves the interrupt permanently pending, which the GIC would deliver again the instant it is EOI'd: a real interrupt storm, not a hypothetical one (this is exactly the failure this driver was checked against during item 0.6's own verification, and confirmed absent there only because nothing armed the timer yet).

Called from aarch64_irq_handler() (interrupts.c) before heartbeat_tick() — same ordering discipline as riscv64_timer_rearm() (item 0.3): re-arm first, so a fault in the rest of the handler cannot also cost the next tick.

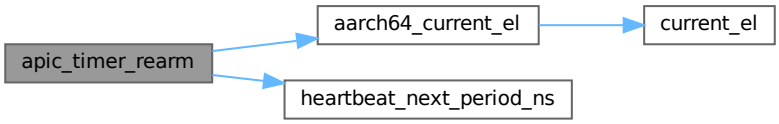
The step written is heartbeat_next_period_ns() (item 0.8, ?26) converted to CNTPCT_EL0 ticks via s_counter_hz_apic, not the fixed s_timer_period_tsc apic_timer_start() used for the very first arm – that value remains apic_timer_period_tsc()'s return for heartbeat_init()'s initial expected_delta, unchanged.

amd64's APIC timer runs in periodic mode: hardware auto-reloads APIC_REG_TIMER_ICR on every expiry with no software intervention, unlike riscv64's one-shot SBI deadline or aarch64's one-shot CNTP_TVAL_EL0 – neither of which needed a rearm function before this item for the same reason this one now exists. To make the period adaptive, this function recomputes timer_initial_count from heartbeat_next_period_ns() (?26) and s_apic_hz, then writes it to APIC_REG_TIMER_ICR. A periodic-mode ICR write takes effect immediately and restarts the countdown at the new value – the same mechanism apic_timer_start() already relies on to force QEMU TCG's emulated APIC to begin counting.

Called from isr_common_handler() on every APIC_TIMER_VECTOR interrupt, before heartbeat_tick() – same ordering discipline as riscv64/aarch64's rearm-before-heartbeat_tick(), so a fault in heartbeat_tick() cannot also cost the next tick.

References aarch64_current_el(), heartbeat_next_period_ns(), s_counter_hz_apic, and s_timer_period_tsc.

Here is the call graph for this function:



7.1.2.7 apic_timer_start()

```
void apic_timer_start (
            void  )
```

Start the APIC timer (enables periodic interrupts).

Call this after IDT and heartbeat handler are set up.

Start the APIC timer (enables periodic interrupts).

Writes CNTP_TVAL_EL0 (or CNTHP_TVAL_EL2 at EL2 — aarch64_current_el(), same EL-aware discipline as items 0.4–0.6) to s_timer_period_tsc, then sets CTL.ENABLE=1, CTL.IMASK=0.

TVAL is a signed 32-bit *relative* countdown — writing it sets "fire @c s_timer_period_tsc ticks from this instant," unlike riscv64's SBI interface (item 0.3), which takes an absolute deadline the caller must compute. No "now" reading is needed here at all.

The ISB after the CTL write is required, not decorative: per the architecture, and confirmed against Linux's own arch_timer_reg_write_cp15() (arch/arm64/include/asm/arch_timer.h), only the *control* register write

needs synchronising before the enable/mask state is guaranteed visible to the interrupt pipeline — `TVAL/CVAL writes` do not carry the same requirement, which is why `apic_timer_rearm()` below omits it.

`s_timer_period_tsc` must already be set by `apic_timer_init()`. If it is still 0, `TVAL=0` fires immediately and continuously until `apic_timer_rearm()` runs — a correctly-functioning if oddly-paced timer, not a hang.

Start the APIC timer (enables periodic interrupts).

Called after `arch_interrupts_init()` and `arch_enable_interrupts()` to start the 100 Hz heartbeat that drives `heartbeat_tick()`. The sequence is:

1. Read the current LVT Timer Register.
2. Clear the `LVT_MASKED` bit (bit 16) to unmask the interrupt.
3. Write the LVT register back.
4. Write `timer_initial_count` to `APIC_REG_TIMER_ICR` to (re-)arm the countdown. QEMU TCG requires an explicit ICR write after unmask — it does not auto-restart a previously masked periodic timer. On real hardware this is harmless and equally correct.
5. Reads back LVT and ICR values and prints them for boot-log verification.

Must be called exactly once per boot after the IDT is loaded. Calling it again restarts the countdown from the initial value without otherwise reconfiguring the timer.

Start the APIC timer (enables periodic interrupts).

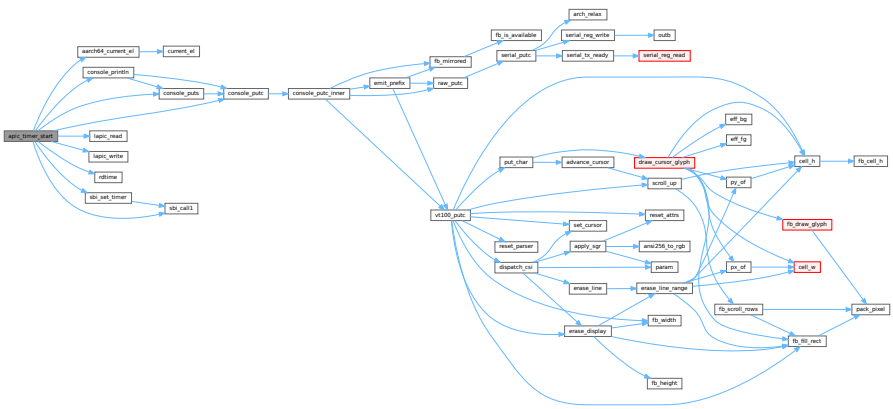
Probes for the TIME extension first. If the SEE does not provide it the timer is **not** armed and the condition is reported loudly rather than papered over with the legacy EID 0x00 call: a heartbeat that silently never ticks is far worse to diagnose than one that says why at boot.

On success: computes the first absolute deadline, arms it, and sets `sie.STIE`. Global delivery is gated separately by `sstatus.SIE`, which `arch_enable_interrupts()` sets.

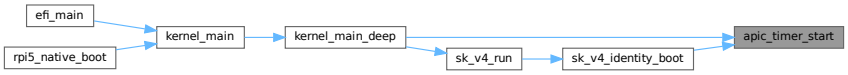
References aarch64_current_el(), APIC_REG_LVT_TIMER, APIC_REG_TIMER_ICR, console_println(), console_putc(), console_puts(), sbiret_t::error, lapic_read(), lapic_write(), rdtime(), s_next_deadline, s_sbi_time_ok, s_timer_period_tsc, SBI_BASE_FID_PROBE_EXT, sbi_call1(), SBI_EXT_BASE, SBI_EXT_TIME, sbi_set_timer(), SBI_SUCCESS, SIE_STIE, timer_initial_count, and sbiret_t::value.

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.8 apic_timer_stop()

```
void apic_timer_stop (
            void  )
```

Stop the APIC timer (disables periodic interrupts).

Stop the APIC timer (disables periodic interrupts).

Per the architecture, `ENABLE=0` unconditionally stops the count and de-asserts the interrupt condition regardless of `IMASK` — a single write is sufficient, no separate mask step needed.

Stop the APIC timer (disables periodic interrupts).

Sets the `LVT_MASKED` bit (bit 16) in the LVT Timer Register by performing a read-modify-write on `APIC_REG_LVT_TIMER`. The existing periodic-mode and vector configuration is preserved; only the mask bit changes. The APIC timer's internal countdown continues running but interrupt delivery is suppressed. Calling `apic_timer_start()` again after this function will unmask and re-arm the timer.

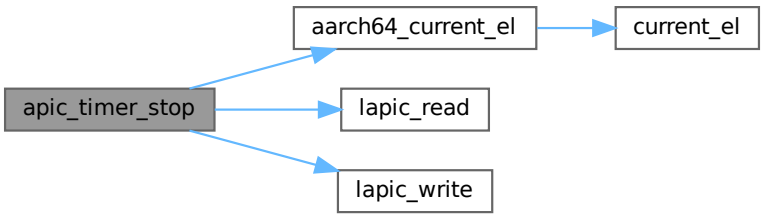
Used in the kernel panic path and during any critical section where heartbeat interrupts would corrupt shared state.

Stop the APIC timer (disables periodic interrupts).

The SBI timer cannot be cancelled outright — masking the enable bit is the supported way to stop delivery. Any deadline already programmed simply goes unserviced.

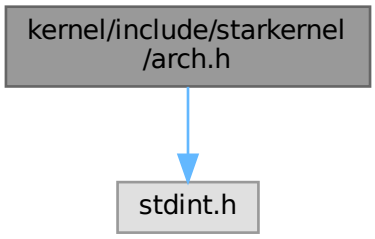
References aarch64_current_el(), APIC_REG_LVT_TIMER, lapic_read(), lapic_write(), LVT_MASKED, s_sbi_time_ok, and SIE_STIE.

Here is the call graph for this function:



7.2 kernel/include/starkernel/arch.h File Reference

`#include <stdint.h>`
Include dependency graph for arch.h:



This graph shows which files directly or indirectly include this file:

Functions

- void arch_early_init (void)

  arch.h - Architecture abstraction layer for StarKernel
- void arch_enable_interrupts (void)

  Enable IRQ delivery on the current core (AArch64 DAIFCLR).
- void arch_disable_interrupts (void)

  Disable IRQ delivery on the current core (AArch64 DAIFSET).
- void arch_interrupts_init (void)

  Install the AArch64 exception vector table (M4 milestone).
- void arch_halt (void)

  Halt the core until the next event (AArch64 WFI).
- void arch_cold_reset (void) __attribute__((noreturn))
- uint64_t arch_read_timestamp (void)

  Read the AArch64 system counter (CNTPCT_EL0).
- void arch_mmu_init (void)

  Initialise the AArch64 MMU (stub — deferred to a later milestone).
- static void arch_relax (void)

7.2.1 Function Documentation

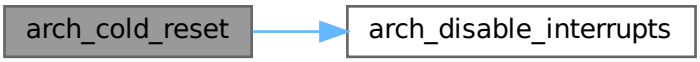
7.2.1.1 arch_cold_reset()

```
void arch_cold_reset (
            void  )
```

References arch_disable_interrupts(), and s_psci_use_smc.

Referenced by sk_v4_serve().

Here is the call graph for this function:



Here is the caller graph for this function:

7.2.1.2 arch_disable_interrupts()

```
void arch_disable_interrupts (
            void  )
```

Disable IRQ delivery on the current core (AArch64 DAIFSET).

Issues `MSR` DAIFSET, #2 which sets the `I` bit (IRQ mask) in the `DAIF` flags, masking IRQ exceptions. FIQ, SError, and Debug exceptions are unaffected. Used around critical sections in the single-threaded kernel where interrupt delivery would corrupt shared state.

The `memory` clobber acts as a compiler-level barrier; for a full hardware memory barrier use `DSB` before calling this function.

Disable IRQ delivery on the current core (AArch64 DAIFSET).

Issues the `CLI` instruction to clear the Interrupt Flag. Used around critical sections in the single-threaded kernel where interrupt delivery would corrupt shared state.

Disable IRQ delivery on the current core (AArch64 DAIFSET).

Issues `CSRCI sstatus, 0x2` which clears bit 1 (`SIE`) in `sstatus`, masking all supervisor-mode interrupts. Machine-mode interrupts are unaffected. Used around critical sections in the single-threaded kernel. The `memory` clobber acts as a compiler barrier; pair with a `FENCE` instruction for a hardware barrier.

Referenced by arch_cold_reset(), and arch_cold_reset().

Here is the caller graph for this function:

7.2.1.3 arch_early_init()

```
void arch_early_init (
            void  )
```

arch.h - Architecture abstraction layer for StarKernel

Provides a minimal interface used across early boot, interrupt control, low-power halting, timestamp reads, and MMU bring-up.

arch.h - Architecture abstraction layer for StarKernel

On x86-64 this function loads the GDT; AArch64 has no equivalent because privilege separation is provided by the exception-level architecture (EL0/EL1/EL2/EL3) rather than segment descriptors. The exception vector table is installed later in `arch_interrupts_init()` by calling `aarch64_install_vectors()` which writes `VBAR_EL1`.

This stub exists to satisfy the common `arch_early_init()` call site in `kernel_main()` so that the same milestone sequence compiles across all three supported ISAs (amd64 / aarch64 / riscv64).

arch.h - Architecture abstraction layer for StarKernel

Called immediately after the UEFI boot stub transfers control to the kernel, before the console or interrupt controller are available. Loads the minimal three-entry GDT via `gdt_init()` so that subsequent code has valid segment selectors. IDT setup is deferred until after the console is live (called separately via `arch_interrupts_init()`).

arch.h - Architecture abstraction layer for StarKernel

RISC-V has no GDT or IDT; privilege separation is handled by the machine/supervisor/user execution modes (M/S/U), and the trap vector table is installed by writing `stvec` in `arch_interrupts_init()` via `riscv64_install_vectors()`. This stub satisfies the common `arch_early_init()` call site in `kernel_main()` without any RISC-V-specific action.

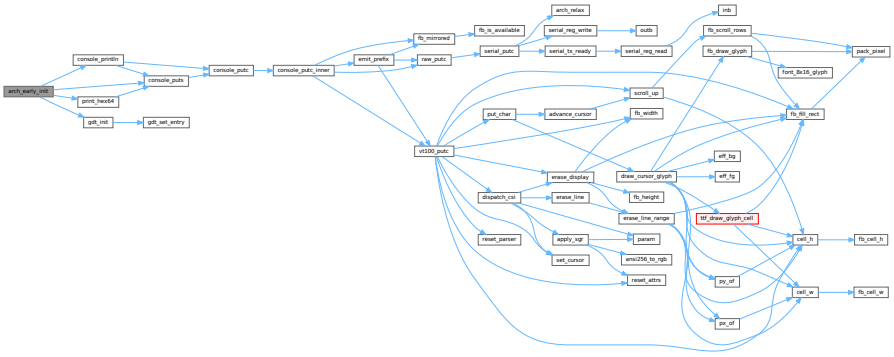
Correction, 2026-09-04 (FABRIC-3.md ?V.3 item 7's code audit). The unconditional Bare-mode switch this function used to perform was justified entirely by behaviour observed under this system's QEMU/EDK2 RISC-V firmware (`satp.MODE=10/Sv57`, kernel identity-mapped within it — see the comment on `print_hex64()` above). That observation cannot apply to the native boot path (U-Boot+OpenSBI, no EDK2 at all, FABRIC-3.md ?V.1) this kernel now also has, and OpenSBI's own S-mode handoff convention leaves `satp` already at 0 (Bare) in the common case — meaning the switch below was very likely running as a harmless no-op there, but on an unverified assumption rather than a checked one. Rather than build a general Sv39/Sv48/Sv57 page-table walker just to *prove* the running address is identity-mapped before switching (out of proportion to this item's real scope), the fix narrows

what actually needs the QEMU-specific safety argument: **skip the switch entirely when `satp` is already 0.** That is the expected case on both a from-Bare native boot and any firmware that already left translation off, and it needs no safety argument at all — there is nothing to switch away from. The unconditional-switch path still exists for the confirmed QEMU/EDK2 case (`mode` != 0), where the identity-mapping argument above remains the actual justification, unchanged.

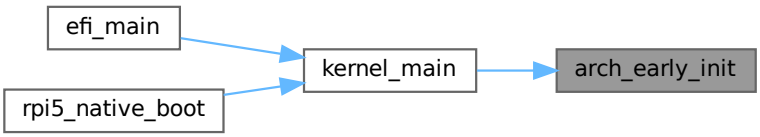
References __kernel_start, console_println(), console_puts(), gdt_init(), and print_hex64().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.1.4 arch_enable_interrupts()

```
void arch_enable_interrupts (
            void  )
```

Enable IRQ delivery on the current core (AArch64 DAIFCLR).

Issues `MSR` DAIFCLR, #2 which clears the `I` bit (IRQ mask) in the `DAIF` (Debug / Abort / IRQ / FIQ) flags of the current exception level. After this instruction the core will take IRQ exceptions from the GIC. The `memory` clobber prevents the compiler from reordering stores across the barrier.

Must be called after `arch_interrupts_init()` has installed the vector table in `VBAR_EL1` and the GIC timer driver has been configured.

Enable IRQ delivery on the current core (AArch64 DAIFCLR).

Issues the `STI` instruction to set the Interrupt Flag. Must be called after the IDT is loaded and the APIC is configured, otherwise the first interrupt will triple-fault.

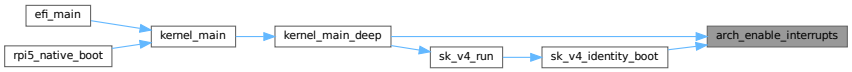
Enable IRQ delivery on the current core (AArch64 DAIFCLR).

Issues `CSRSI sstatus, 0x2` which sets bit 1 (`SIE` — Supervisor Interrupt Enable) in the `sstatus` CSR. While `SIE` = 1, pending supervisor-mode interrupts are taken. The `memory` clobber prevents the compiler from reordering stores across the barrier.

Must be called after `arch_interrupts_init()` has installed the `stvec` trap handler, and after the PLIC driver has been configured to forward the desired interrupt sources.

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the caller graph for this function:



7.2.1.5 arch_halt()

```
void arch_halt (
            void  )
```

Halt the core until the next event (AArch64 WFI).

Issues the `WFI` (Wait For Interrupt) instruction. The core enters a low-power standby state and resumes execution when an IRQ, FIQ, SError, debug event, or reset is detected. In the kernel panic path this is called inside an infinite loop with interrupts disabled (`DAIFSET`) to stop the processor permanently. In the idle path it reduces power consumption between heartbeat ticks.

The `memory` clobber prevents the compiler from sinking stores across the `WFI`.

Halt the core until the next event (AArch64 WFI).

Issues the `HLT` instruction. In the kernel panic path this is called in an infinite loop (interrupts disabled) to stop the processor permanently. In normal use it is called once per idle iteration to reduce power consumption.

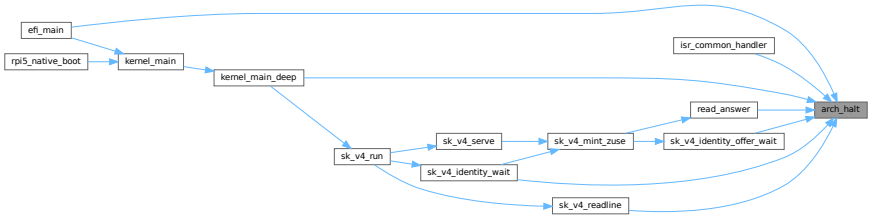
Halt the core until the next event (AArch64 WFI).

Issues the `WFI` (Wait For Interrupt) instruction in supervisor mode. The core enters a low-power state and resumes when a pending interrupt is detected (or after a platform-specific timeout). In the kernel panic path this is called inside an infinite loop with `SIE` cleared to stop the processor permanently. In the idle path it reduces power consumption between heartbeat ticks.

The `memory` clobber prevents store-sinking across the `WFI`.

Referenced by efi_main(), isr_common_handler(), kernel_main_deep(), read_answer(), sk_v4_identity_offer_wait(), sk_v4_identity_wait(), and sk_v4_readline().

Here is the caller graph for this function:



7.2.1.6 arch_interrupts_init()

```
void arch_interrupts_init (
            void  )
```

Install the AArch64 exception vector table (M4 milestone).

First reads and reports the exception level via aarch64_current_el() (punch-list item 0.4) — this is the earliest point in boot with both a working console (up since M1) and a genuine need for the answer (vector installation is the first EL-dependent operation). The read itself is cheap and safe far earlier than this; only the console print is gated on this call site.

Then calls aarch64_install_vectors() (defined in isr.S) which writes the address of the vector table base into VBAR_EL1 or VBAR_EL2 (Vector Base Address Register), whichever matches the exception level aarch64_current_el() just reported. After this instruction all exceptions at that level (synchronous, IRQ, FIQ, SError) dispatch through the 512-byte-aligned vector table assembled in isr.S.

Correction, 2026-09-04 (FABRIC-3.md ?IV.3 item 7's code audit): this doc comment and the runtime console message below previously claimed EL2 was "a known gap... @c VBAR_EL1 is written unconditionally regardless of the detected level," citing punch-list item 0.5 as the still-open work to fix it. Checked against isr.S's own aarch64_install_vectors() (the actual implementation, not this comment) and found that claim stale↩ : the assembly already branches on aarch64_current_el() and writes vbar_el2 at EL2 / vbar_el1 at EL1, threading the same answer through el2_mode_flag for the IRQ trampoline's ELR_ELx/SPSR_ELx selection too (see isr.S's own "EL selection" header comment). There never was a gap here to close — item 0.5's EL2 vector work was already done by the time this comment was written describing it as open; only the comment was wrong, not the code.

On AArch64 there is no IDT and no PIC to disable; the GIC replaces both. The arch_interrupts_init() name is kept identical across all ISAs so that kernel_main() can call the same symbol regardless of the target architecture.

Must be called after the kernel stack is established (so that the correct-EL SP is valid) and before arch_enable_interrupts().

Install the AArch64 exception vector table (M4 milestone).

Calls riscv64_install_vectors() (defined in isr.S) which writes the trap vector base address into stvec (Supervisor Trap Vector Base Address Register, CSR 0x105). The vector mode field (bits [1:0]) is set to 0 (Direct mode) so all traps dispatch to the single base address, or 1 (Vectored mode) so async interrupts dispatch to stvec + 4∗cause.

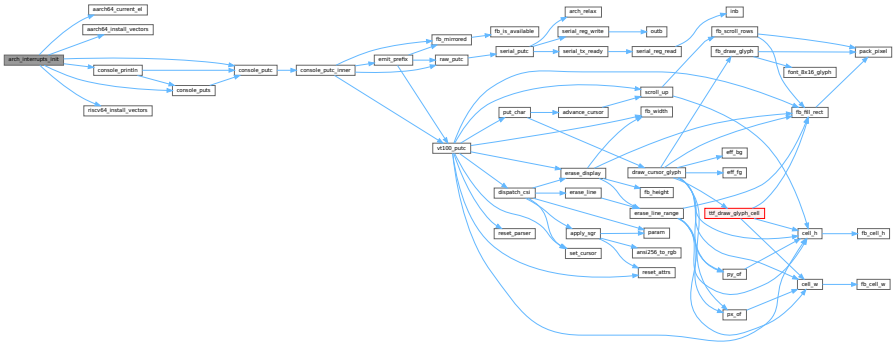
On RISC-V there is no IDT and no legacy PIC; the PLIC replaces both. The arch_interrupts_init() name is shared across all three ISAs so that kernel_main() calls the same symbol regardless of target.

Must be called after the supervisor stack (sp) is established and before arch_enable_interrupts().

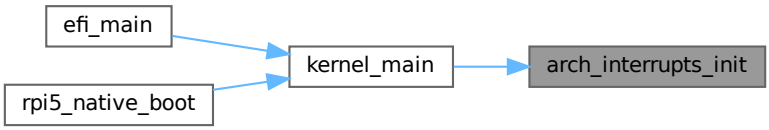
References aarch64_current_el(), aarch64_install_vectors(), console_println(), console_putc(), console_puts(), and riscv64_install_vectors().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.1.7 arch_mmu_init()

```
void arch_mmu_init (
            void  )
```

Initialise the AArch64 MMU (stub — deferred to a later milestone).

Full page-table setup (TTBR0/TTBR1, MAIR, TCR, SCTLR.M) is deferred to a later milestone. This stub exists to satisfy the common `arch_mmu_init()` call site shared across all three ISAs. The VMM subsystem (`vmm.c`) handles page-table management independently of this hook.

Initialise the AArch64 MMU (stub — deferred to a later milestone).

Full four-level page table setup is deferred; the VMM subsystem (`vmm.c`) handles page table management independently. This function exists as an architectural hook for future per-CPU MMU configuration that must run after GDT/IDT setup but before the first memory access through a non-identity-mapped virtual address.

Initialise the AArch64 MMU (stub — deferred to a later milestone).

Full Sv39/Sv48 page-table setup (SATP, PMP, etc.) is deferred to a later milestone. This stub satisfies the common `arch_mmu_init()` call site shared across all three supported ISAs. The VMM subsystem (`vmm.c`) handles page-table management independently of this hook.

7.2.1.8 arch_read_timestamp()

```
uint64_t arch_read_timestamp (
            void  )
```

Read the AArch64 system counter (CNTPCT_EL0).

Issues `MRS %0, cntpct_el0` to read the 64-bit physical system counter. On AArch64 the system counter is the architectural equivalent of the x86-64 TSC: a monotonically incrementing clock whose frequency is reported by `CNTFRQ_EL0` and set by firmware at boot.

Unlike `RDTSC` on x86-64, `CNTPCT_EL0` is fully synchronised across cores (no per-core skew), invariant across power-state changes, and does not require serialisation for most measurement use cases. It is used as the time base for `timer_now_ns()` and `heartbeat_tick()`.

Returns

Current 64-bit system counter value in counter ticks (frequency given by `CNTFRQ_EL0`, typically 10–62.5 MHz).

Read the AArch64 system counter (CNTPCT_EL0).

Executes `RDTSC` and combines `EDX:EAX` into a 64-bit value. Does not serialise instruction execution; use `RDTSCP` or a surrounding `CPUID` where ordering guarantees are required.

Returns

Current TSC value in CPU cycles.

Read the AArch64 system counter (CNTPCT_EL0).

Issues the `RDCYCLE` pseudo-instruction (a `CSRRS` on `cycle`, CSR 0xC00) to read the 64-bit hardware performance counter that counts elapsed clock cycles since reset. On RISC-V the cycle counter is the architectural equivalent of the x86-64 TSC.

Unlike the AArch64 `CNTPCT_EL0`, the RISC-V `cycle` register is a per-hart counter and is not architecturally guaranteed to be synchronised across harts. For single-core LithosAnanke this is not a concern. Frequency is not architecturally specified; `timer_init()` assumes 1 GHz for QEMU `virt` board compatibility.

Returns

Current 64-bit cycle counter value; frequency platform-dependent.

Referenced by calibrate_apic_timer().

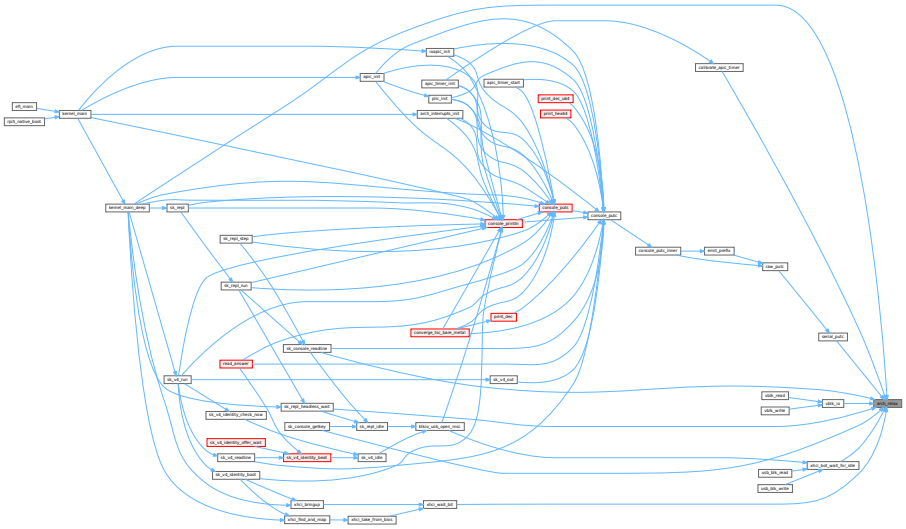
Here is the caller graph for this function:

7.2.1.9 arch_relax()

```
static void arch_relax (
            void  )  [inline], [static]
```

Referenced by calibrate_apic_timer(), kernel_main_deep(), serial_putc(), sk_console_getkey(), sk_console_readline(), sk_repl_headless_wait(), vblk_io(), xhci_bot_wait_for_idle(), and xhci_wait_bit().

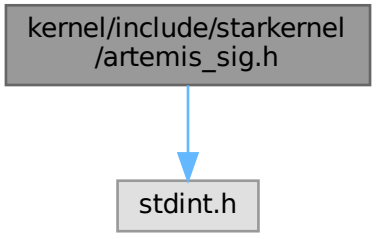
Here is the caller graph for this function:



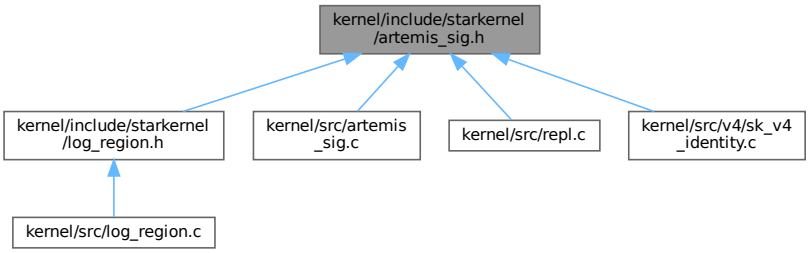
7.3 kernel/include/starkernel/artemis_sig.h File Reference

```
#include <stdint.h>
```

Include dependency graph for artemis_sig.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct artemis_sig_t

Macros

- #define ARTEMIS_SIG_MAGIC 0x4D545241ULL /* 'ARTM' */

  artemis_sig.h - Artemis disk signature format (FABRIC-3.md ?XXVI follow-on, 2026-09-13)
- #define ARTEMIS_SIG_VERSION_0 0
- #define ARTEMIS_SIG_PACK(ver) (ARTEMIS_SIG_MAGIC | ((uint64_t)(ver) << 32))
- #define ARTEMIS_SIG_GET_MAGIC(m) ((uint32_t)((m) & 0xFFFFFFFFULL))
- #define ARTEMIS_SIG_GET_VERSION(m) ((uint8_t)(((m) >> 32) & 0xFF))
- #define ARTEMIS_SIG_DEVBLOCK_FROM_TOP 64u

Typedefs

- typedef char artemis_sig_size_check[(sizeof(artemis_sig_t)==4096) ? 1 :-1]

Enumerations

- enum artemis_sig_result_t {
  ARTEMIS_SIG_OK = 0 , ARTEMIS_SIG_BLANK , ARTEMIS_SIG_BAD_VERSION , ARTEMIS_SIG_BAD_CRC
  ,
  ARTEMIS_SIG_READ_ERROR }

Functions

- artemis_sig_result_t artemis_sig_check (struct blkio_dev *dev, artemis_sig_t *out_sig)
- uint64_t artemis_sig_compute_crc (const artemis_sig_t *sig)

  artemis_sig.c - Artemis disk signature check (FABRIC-3.md ?XXVI follow-on).
- int artemis_sig_genesis_stamp (struct blkio_dev *dev)

7.3.1 Macro Definition Documentation

7.3.1.1 ARTEMIS_SIG_DEVBLOCK_FROM_TOP

```
#define ARTEMIS_SIG_DEVBLOCK_FROM_TOP 64u
```

7.3.1.2 ARTEMIS_SIG_GET_MAGIC

```
#define ARTEMIS_SIG_GET_MAGIC(
            m ) ((uint32_t)((m) & 0xFFFFFFFFULL))
```

7.3.1.3 ARTEMIS_SIG_GET_VERSION

```
#define ARTEMIS_SIG_GET_VERSION(
            m ) ((uint8_t)(((m) >> 32) & 0xFF))
```

7.3.1.4 ARTEMIS_SIG_MAGIC

```
#define ARTEMIS_SIG_MAGIC 0x4D545241ULL /* 'ARTM' */
```

artemis_sig.h - Artemis disk signature format (FABRIC-3.md ?XXVI follow-on, 2026-09-13)

Identifies Artemis's own disk, distinct from an identity thumbdrive's homeblocks_sig_t – needed once Artemis's disk stops being found by a hardcoded PCI virtio-blk vendor/device ID scan (QEMU-only; real hardware has no reason to expose a virtio-blk PCI device at all, since virtio is a paravirtualization standard, not something a physical storage controller speaks) and starts being discovered generically instead, the same way WIREBIND already discovers identity thumbdrives – by content signature, not by which bus happened to present the device. Without a distinct signature, generic discovery on real hardware (where an identity thumbdrive and Artemis's own disk could both be attached as USB-MSC devices simultaneously) would have no way to tell them apart.

Mirrors homeblocks_sig_t's own structural convention (magic + version + CRC, one 4KiB header) – a sibling format, not a field bolted onto homeblocks_sig_t itself: homeblocks_sig_t's own header comment already states it's "deliberately narrow in scope" (identity-drive fields only, no spare room), and Artemis's disk is conceptually a different kind of thing (one dedicated fleet-owned device, not one of many candidate identity drives), not a variant of the same one.

CORRECTION, same day: the first version of this format placed the header at a fixed bottom-of-device forth-block (4, i.e. devblock 1), copying homeblocks_sig_t's own devblock-1 convention. That convention is safe for an identity thumbdrive (raw, dedicated storage – capsule_mint.c writes directly, no block-subsystem format involved), but Artemis's disk is block_subsystem.c's own STFR/v2-formatted volume: devblock 0 holds that format's header (read_header_4k()/write_header_4k(), block_subsystem.c) and devblock 1 is the FIRST DEVBLOCK OF THE LIVE BAM (blk_compute_fresh_geometry(): bam_start = 1). Writing artemis_sig_t there would have overwritten Artemis's own live allocation map on the very first real boot this ran against – caught (via `git status/grep` cross-reference against block_subsystem.c, not against a live disk) before the genesis-stamp call site ever executed against the real image.

The header now lives at a fixed offset from the END of the device instead (ARTEMIS_SIG_DEVBLOCK↩ _FROM_TOP, see below) – the same "top-of-device, outside the user-addressable LBN pool" region block_subsystem.c's own meta_fence_blocks reservation (128 devblocks by default, BLK_META_FENCE_INIT) already carves out for exactly this kind of system metadata, and where Zuse's own genesis marker already lives (blk_meta_zone_read()/write(), devblock_from_top 0; 1-63 reserved, unused, since the eligibility list was removed 2026-10-14). This format deliberately does NOT go through blk_meta_zone_∗, though: that accessor requires

the device to already be blk_subsys_attach_device()'d and format-detected (first_disk_slot()) – exactly the state generic pre-attach discovery (repl.c's idle-loop USB-MSC scan) doesn't have yet, which is the entire reason this format exists. artemis_sig_check()/artemis_sig_genesis_stamp() instead compute the same top-of-device arithmetic independently via blkio_info(), so they work on a raw, not-yet-attached device exactly like homeblocks_sig_check() already does. ARTEMIS_SIG_DEVBLOCK_FROM_TOP is fixed well clear of Zuse's genesis marker (0) and the reserved, unused 1-63.

Reserves offset/size pointers to the growable per-VM log-persistence region (FABRIC-3.md ?XXVI follow-on's own log-record work), the same way homeblocks_sig_t reserves pointers to where the cert and identity source attach – this format doesn't need revisiting when that region's own internal layout is designed.

7.3.1.5 ARTEMIS_SIG_PACK

```
#define ARTEMIS_SIG_PACK(
              ver )  (ARTEMIS_SIG_MAGIC | ((uint64_t)(ver) << 32))
```

7.3.1.6 ARTEMIS_SIG_VERSION_0

```
#define ARTEMIS_SIG_VERSION_0 0
```

7.3.2 Typedef Documentation

7.3.2.1 artemis_sig_size_check

```
typedef char artemis_sig_size_check[(sizeof(artemis_sig_t)==4096) ?  1 :-1]
```

7.3.3 Enumeration Type Documentation

7.3.3.1 artemis_sig_result_t

```
enum artemis_sig_result_t
```

Enumerator

ARTEMIS_SIG_OK	
ARTEMIS_SIG_BLANK	
ARTEMIS_SIG_BAD_VERSION	
ARTEMIS_SIG_BAD_CRC	
ARTEMIS_SIG_READ_ERROR	

7.3.4 Function Documentation

7.3.4.1 artemis_sig_check()

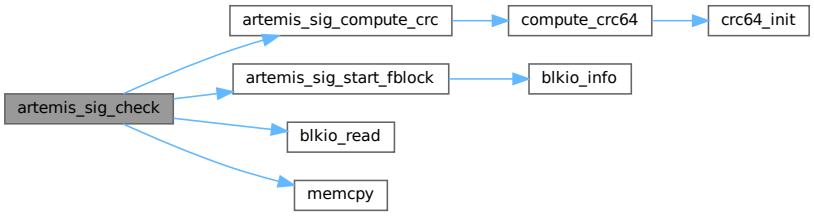
```
artemis_sig_result_t artemis_sig_check (
```

```
            struct blkio_dev * dev,
            artemis_sig_t * out_sig )
```

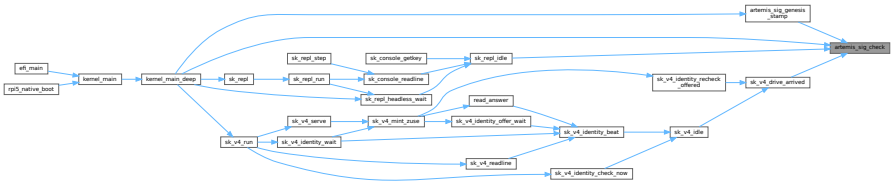
References ARTEMIS_SIG_BAD_CRC, ARTEMIS_SIG_BAD_VERSION, ARTEMIS_SIG_BLANK, artemis_sig_compute_crc(), ARTEMIS_SIG_GET_MAGIC, ARTEMIS_SIG_GET_VERSION, ARTEMIS_SIG_MAGIC, ARTEMIS_SIG_OK, ARTEMIS_SIG_READ_ERROR, artemis_sig_start_fblock(), ARTEMIS_SIG_VERSION_0, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_read(), artemis_sig_t::hdr_crc, artemis_sig_t::magic, and memcpy().

Referenced by artemis_sig_genesis_stamp(), kernel_main_deep(), sk_repl_idle(), and sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.4.2 artemis_sig_compute_crc()

```
uint64_t artemis_sig_compute_crc (
            const artemis_sig_t * sig )
```

artemis_sig.c - Artemis disk signature check (FABRIC-3.md ?XXVI follow-on).

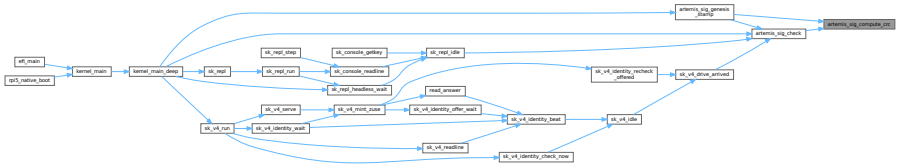
See starkernel/artemis_sig.h for the format and interface design.

References compute_crc64().

Referenced by artemis_sig_check(), and artemis_sig_genesis_stamp().

Here is the call graph for this function:

Here is the caller graph for this function:



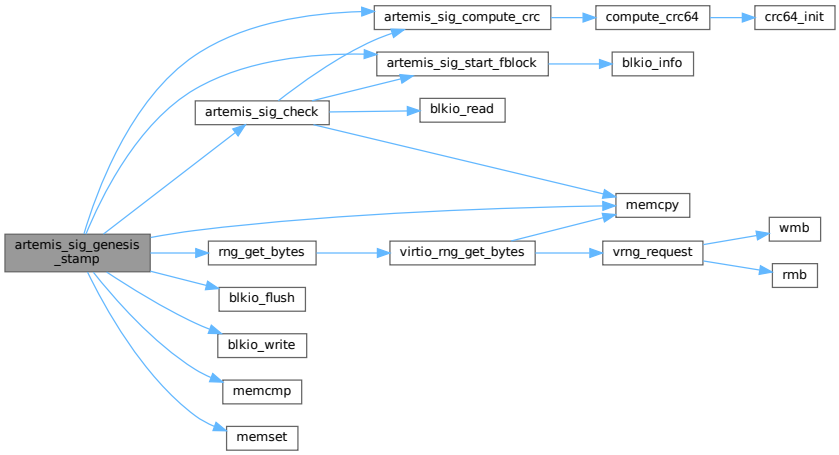
7.3.4.3 artemis_sig_genesis_stamp()

```
int artemis_sig_genesis_stamp (
            struct blkio_dev * dev )
```

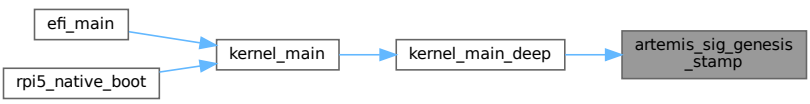
References artemis_sig_check(), artemis_sig_compute_crc(), ARTEMIS_SIG_OK, ARTEMIS_SIG_PACK, artemis_sig_start_fblock(), ARTEMIS_SIG_VERSION_0, blkio_flush(), BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_write(), artemis_sig_t::disk_uuid, artemis_sig_t::genesis_time_ns, artemis_sig_t::hdr_crc, artemis_sig_t::log_region_devblocks, artemis_sig_t::log_region_offset, artemis_sig_t::magic, memcmp(), memcpy(), memset(), artemis_sig_t::metadata_devblocks, and rng_get_bytes().

Referenced by kernel_main_deep().

Here is the call graph for this function:



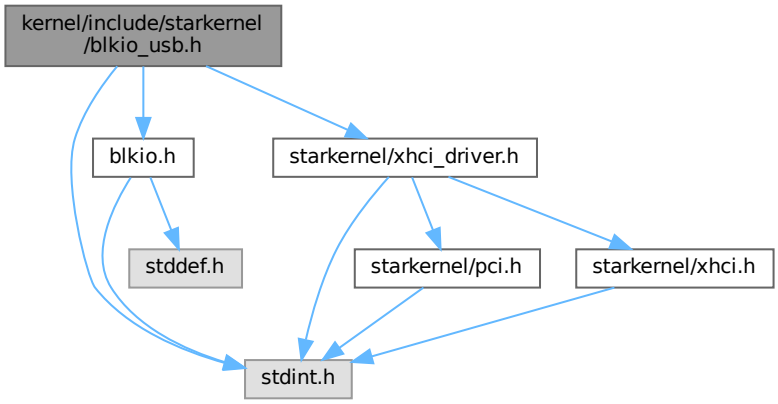
Here is the caller graph for this function:



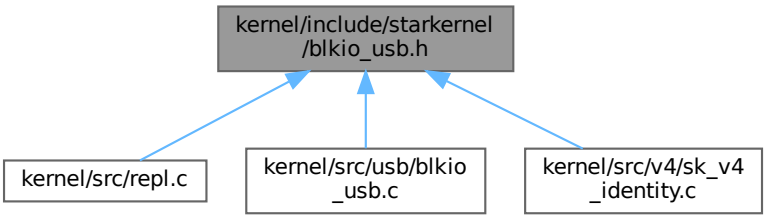
7.4 kernel/include/starkernel/blkio_usb.h File Reference

```
#include <stdint.h>
#include "blkio.h"
#include "starkernel/xhci_driver.h"
```

Include dependency graph for blkio_usb.h:



This graph shows which files directly or indirectly include this file:



Functions

- int blkio_usb_open_msc (blkio_dev_t *dev_out, xhci_dev_t *xdev, uint32_t slot_id)

7.4.1 Function Documentation

7.4.1.1 blkio_usb_open_msc()

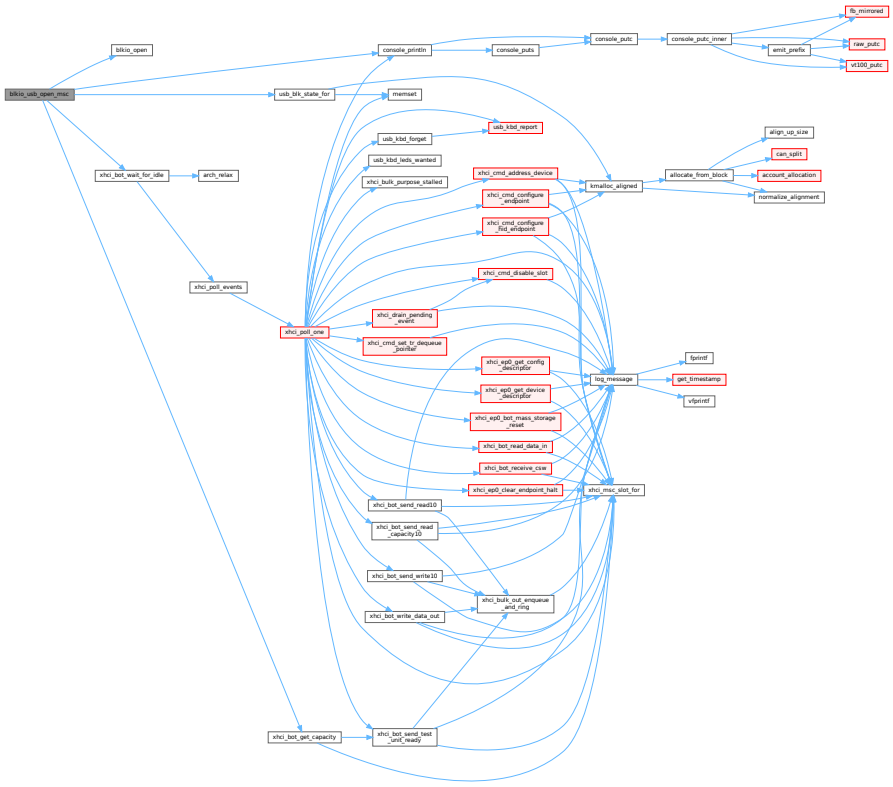
```
int blkio_usb_open_msc (
            blkio_dev_t * dev_out,
```

```
    xhci_dev_t * xdev,
    uint32_t slot_id )
```

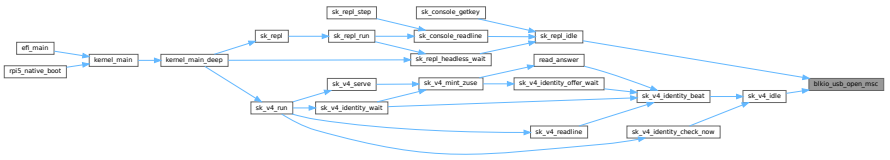
References BLKIO_FORTH_BLOCK_SIZE, blkio_open(), block_size, xhci_dev_t::bot_cap_block_size, xhci_dev_t::bot_cap_last_lba, console_println(), blkio_params_t::forth_block_size, g_usb_blk_vtable, blkio_params_t::opaque, BlkioUsbState::scsi_block_size, BlkioUsbState::scsi_blocks_per_fblock, BlkioUsbState::slot_id, blkio_params_t::total_blocks, BlkioUsbState::total_forth_blocks, usb_blk_state_for(), BlkioUsbState::xdev, xdev, xhci_bot_get_capacity(), and xhci_bot_wait_for_idle().

Referenced by sk_repl_idle(), and sk_v4_idle().

Here is the call graph for this function:

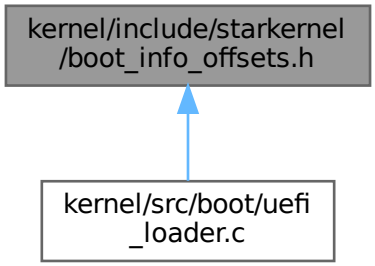


Here is the caller graph for this function:



7.5 kernel/include/starkernel/boot_info_offsets.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define BOOT_INFO_KERNEL_STACK_BASE_OFFSET 88
- #define BOOT_INFO_KERNEL_STACK_SIZE_OFFSET 96

7.5.1 Macro Definition Documentation

7.5.1.1 BOOT_INFO_KERNEL_STACK_BASE_OFFSET

```
#define BOOT_INFO_KERNEL_STACK_BASE_OFFSET 88
```

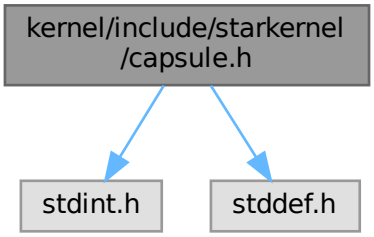
7.5.1.2 BOOT_INFO_KERNEL_STACK_SIZE_OFFSET

```
#define BOOT_INFO_KERNEL_STACK_SIZE_OFFSET 96
```

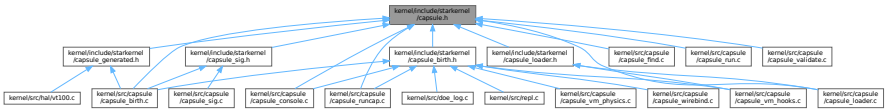
7.6 kernel/include/starkernel/capsule.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for capsule.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct CapsuleNameEntry
- struct CapsuleSigEntry
- struct CapsuleDirHeader

Macros

- #define CAPSULE_DESC_MAGIC 0x53504143ULL /* 'CAPS' little-endian */

  capsule.h - Init Capsule Architecture (M7.1)
- #define CAPSULE_DIR_MAGIC 0x44504143ULL /* 'CAPD' little-endian */
- #define CAPSULE_VERSION_0 0

  Version.
- #define CAPSULE_MAX_COUNT 256

  Limits.
- #define CAPSULE_NAME_MAX 512

  Maximum capsule name length (colon-separated path, null-terminated)
- #define CAPSULE_FLAG_ACTIVE 0x00000001 /* Eligible for use */

  State flags.
- #define CAPSULE_FLAG_REVOKED 0x00000002 /* Birth-blocked forever */
- #define CAPSULE_FLAG_DEPRECATED 0x00000004 /* Eligible but discouraged */
- #define CAPSULE_FLAG_PINNED 0x00000008 /* Immune to GC */
- #define CAPSULE_FLAG_PRODUCTION 0x00000010 /* (p) truth-bearing */

  Mode flags (D2: babies carry both)
- #define CAPSULE_FLAG_EXPERIMENT 0x00000020 /* (e) workload only */
- #define CAPSULE_FLAG_MAMA_INIT 0x00000040 /* (m) Mama's init */

Mama init flag (exactly one capsule must have this)

- #define CAPSULE_FLAG_CONTRIB 0x00000080 /* (c) contributor-submitted */

  Contributor capsule flag (FABRIC-2.md ?I.5, 2026-09-04) – path-match on capsules/contrib/, mirrors FLAG_MAMA↩ _INIT's own exact-match pattern in mkcapsule.c's flags_from_name().
- #define CAPSULE_MODE_VALID(f)

  Validate mode flags.
- #define CAPSULE_IS_MAMA_INIT(f) (((f) & CAPSULE_FLAG_MAMA_INIT) && ((f) & CAPSULE_FLAG_ACTIVE))

  Check if capsule is Mama's init.
- #define CAPSULE_BIRTH_ELIGIBLE(f)

  Birth eligibility: active and not revoked (flag type irrelevant — D2)
- #define CAPSULE_DOE_ELIGIBLE(f)

  DoE eligibility: experiment, active, not revoked.
- #define CAPSULE_MAGIC_PACK(ver, alg) (CAPSULE_DESC_MAGIC | ((uint64_t)(ver) << 32) | ((uint64↩ _t)(alg) << 40))
- #define CAPSULE_MAGIC_GET_SIG(m) ((uint32_t)((m) & 0xFFFFFFFFULL))
- #define CAPSULE_MAGIC_GET_VERSION(m) ((uint8_t)(((m) >> 32) & 0xFF))
- #define CAPSULE_MAGIC_GET_HASHALG(m) ((uint8_t)(((m) >> 40) & 0xFF))

Enumerations

- enum CapsuleHashAlg { CAPSULE_HASH_XXHASH64 = 0 , CAPSULE_HASH_SHA256 = 1 , CAPSULE_HASH_BLAKE3 = 2 }
- enum CapsuleValidateResult {
  CAPSULE_VALID = 0 , CAPSULE_ERR_BAD_MAGIC , CAPSULE_ERR_BAD_VERSION , CAPSULE_ERR_BAD_HASH_AL
  ,
  CAPSULE_ERR_BOUNDS , CAPSULE_ERR_MODE_INVALID , CAPSULE_ERR_REVOKED_ACTIVE ,
  CAPSULE_ERR_HASH_MISMATCH ,
  CAPSULE_ERR_NULL_PTR }

Functions

- struct __attribute__ ((aligned(64)))
- CapsuleValidateResult capsule_validate (const CapsuleDesc *desc, const uint8_t *arena_base, uint64_↩ t arena_size, int verify_hash)

  capsule_validate - Validate a capsule descriptor
- const char * capsule_validate_result_str (CapsuleValidateResult result)

  capsule_validate_result_str - Get string for validation result
- const CapsuleDesc * capsule_find_by_id (const CapsuleDirHeader *dir, const CapsuleDesc *descs, uint64_t id)

  capsule_find_by_id - Find capsule by content hash ID
- const CapsuleDesc * capsule_find_by_name (const CapsuleDirHeader *dir, const CapsuleDesc *descs, const CapsuleNameEntry *names, const char *name)

  capsule_find_by_name - Find capsule by colon-separated name
- const uint8_t * capsule_get_payload (const CapsuleDesc *desc, const uint8_t *arena_base)

  capsule_get_payload - Get pointer to capsule payload bytes
- const CapsuleDesc * capsule_find_mama_init (const CapsuleDirHeader *dir, const CapsuleDesc *descs)

  capsule_find_mama_init - Find the Mama init capsule

Variables

- CapsuleDesc

7.6.1 Macro Definition Documentation

7.6.1.1 CAPSULE_BIRTH_ELIGIBLE

```
#define CAPSULE_BIRTH_ELIGIBLE(
              f )
```

Value:

```
    (((f) & CAPSULE_FLAG_ACTIVE) && \
     !((f) & CAPSULE_FLAG_REVOKED))
```

Birth eligibility: active and not revoked (flag type irrelevant — D2)

7.6.1.2 CAPSULE_DESC_MAGIC

```
#define CAPSULE_DESC_MAGIC 0x53504143ULL /* 'CAPS' little-endian */
```

capsule.h - Init Capsule Architecture (M7.1)

Content-addressed, immutable init capsules for VM birth. See docs/reference/M7.1.md for full specification.

Key invariants:

- Exactly ONE production (p) INIT defines a baby VM
- capsule_id == content_hash (content-addressed)
- No shared/implicit base INITs
- DOMAIN is Mama-only, PERSONALITY is baby-only Magic signatures

7.6.1.3 CAPSULE_DIR_MAGIC

```
#define CAPSULE_DIR_MAGIC 0x44504143ULL /* 'CAPD' little-endian */
```

7.6.1.4 CAPSULE_DOE_ELIGIBLE

```
#define CAPSULE_DOE_ELIGIBLE(
              f )
```

Value:

```
    (((f) & CAPSULE_FLAG_EXPERIMENT) && \
     ((f) & CAPSULE_FLAG_ACTIVE) && \
     !((f) & CAPSULE_FLAG_REVOKED))
```

DoE eligibility: experiment, active, not revoked.

7.6.1.5 CAPSULE_FLAG_ACTIVE

```
#define CAPSULE_FLAG_ACTIVE 0x00000001 /* Eligible for use */
```

State flags.

7.6.1.6 CAPSULE_FLAG_CONTRIB

```
#define CAPSULE_FLAG_CONTRIB 0x00000080 /* (c) contributor-submitted */
```

Contributor capsule flag (FABRIC-2.md ?I.5, 2026-09-04) – path-match on capsules/contrib/, mirrors FLAG_↩ MAMA_INIT's own exact-match pattern in mkcapsule.c's flags_from_name().

Trust-tier enforcement (QEMU-vs- real-hardware, decided in conversation) is a runtime check in capsule_validate()'s callers, not encoded in this bit itself – the bit only marks "this capsule's provenance is a contributor, not this project's own source," same as CAPSULE_FLAG_PRODUCTION/_EXPERIMENT mark mode, not policy.

7.6.1.7 CAPSULE_FLAG_DEPRECATED

```
#define CAPSULE_FLAG_DEPRECATED 0x00000004 /* Eligible but discouraged */
```

7.6.1.8 CAPSULE_FLAG_EXPERIMENT

```
#define CAPSULE_FLAG_EXPERIMENT 0x00000020 /* (e) workload only */
```

7.6.1.9 CAPSULE_FLAG_MAMA_INIT

```
#define CAPSULE_FLAG_MAMA_INIT 0x00000040 /* (m) Mama's init */
```

Mama init flag (exactly one capsule must have this)

7.6.1.10 CAPSULE_FLAG_PINNED

```
#define CAPSULE_FLAG_PINNED 0x00000008 /* Immune to GC */
```

7.6.1.11 CAPSULE_FLAG_PRODUCTION

```
#define CAPSULE_FLAG_PRODUCTION 0x00000010 /* (p) truth-bearing */
```

Mode flags (D2: babies carry both)

7.6.1.12 CAPSULE_FLAG_REVOKED

```
#define CAPSULE_FLAG_REVOKED 0x00000002 /* Birth-blocked forever */
```

7.6.1.13 CAPSULE_IS_MAMA_INIT

```
#define CAPSULE_IS_MAMA_INIT(
              f )  (((f) & CAPSULE_FLAG_MAMA_INIT) && ((f) & CAPSULE_FLAG_ACTIVE))
```

Check if capsule is Mama's init.

7.6.1.14 CAPSULE_MAGIC_GET_HASHALG

```
#define CAPSULE_MAGIC_GET_HASHALG(
            m ) ((uint8_t)(((m) >> 40) & 0xFF))
```

7.6.1.15 CAPSULE_MAGIC_GET_SIG

```
#define CAPSULE_MAGIC_GET_SIG(
            m ) ((uint32_t)((m) & 0xFFFFFFFFULL))
```

7.6.1.16 CAPSULE_MAGIC_GET_VERSION

```
#define CAPSULE_MAGIC_GET_VERSION(
            m ) ((uint8_t)(((m) >> 32) & 0xFF))
```

7.6.1.17 CAPSULE_MAGIC_PACK

```
#define CAPSULE_MAGIC_PACK(
            ver,
            alg )  (CAPSULE_DESC_MAGIC | ((uint64_t)(ver) << 32) | ((uint64_t)(alg) << 40))
```

7.6.1.18 CAPSULE_MAX_COUNT

```
#define CAPSULE_MAX_COUNT 256
```

Limits.

7.6.1.19 CAPSULE_MODE_VALID

```
#define CAPSULE_MODE_VALID(
            f )
```

Value:

```
    ((((f) & CAPSULE_FLAG_MAMA_INIT) != 0) ? \
        (!((f) & (CAPSULE_FLAG_PRODUCTION | CAPSULE_FLAG_EXPERIMENT))) : \
        (((f) & CAPSULE_FLAG_PRODUCTION) || ((f) & CAPSULE_FLAG_EXPERIMENT)))
```

Validate mode flags.

Mama: neither (p) nor (e) may be set. Babies: at least one of (p) or (e) must be set (both is fine — D2).

7.6.1.20 CAPSULE_NAME_MAX

```
#define CAPSULE_NAME_MAX 512
```

Maximum capsule name length (colon-separated path, null-terminated)

7.6.1.21 CAPSULE_VERSION_0

```
#define CAPSULE_VERSION_0 0
```

Version.

7.6.2 Enumeration Type Documentation

7.6.2.1 CapsuleHashAlg

```
enum CapsuleHashAlg
```

Enumerator

CAPSULE_HASH_XXHASH64	
CAPSULE_HASH_SHA256	
CAPSULE_HASH_BLAKE3	

7.6.2.2 CapsuleValidateResult

```
enum CapsuleValidateResult
```

Enumerator

CAPSULE_VALID	
CAPSULE_ERR_BAD_MAGIC	
CAPSULE_ERR_BAD_VERSION	
CAPSULE_ERR_BAD_HASH_ALG	
CAPSULE_ERR_BOUNDS	
CAPSULE_ERR_MODE_INVALID	
CAPSULE_ERR_REVOKED_ACTIVE	
CAPSULE_ERR_HASH_MISMATCH	
CAPSULE_ERR_NULL_PTR	

7.6.3 Function Documentation

7.6.3.1 __attribute__()

```
struct __attribute__ (
            (aligned(64))  )
```

7.6.3.2 capsule_find_by_id()

```
const CapsuleDesc * capsule_find_by_id (
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            uint64_t id )
```

capsule_find_by_id - Find capsule by content hash ID

Parameters

dir	Directory header
descs	Descriptor array
id	Capsule ID (content hash) to find

Returns

Pointer to descriptor, or NULL if not found

References CapsuleDirHeader::desc_count.

7.6.3.3 capsule_find_by_name()

```
const CapsuleDesc * capsule_find_by_name (
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const char * name )
```

capsule_find_by_name - Find capsule by colon-separated name

Parameters

dir	Directory header
descs	Descriptor array
names	Name entry array (parallel to descs)
name	Colon-separated capsule name, e.g. "core:init.4th"

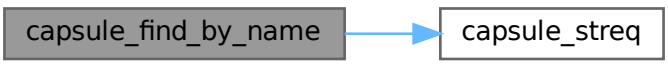
Returns

Pointer to descriptor, or NULL if not found

References capsule_streq(), CapsuleDirHeader::desc_count, and name.

Referenced by capsule_birth_baby(), capsule_exec_init(), and capsule_run_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

7.6.3.4 capsule_find_mama_init()

```
const CapsuleDesc * capsule_find_mama_init (
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs )
```

capsule_find_mama_init - Find the Mama init capsule

Searches the descriptor array for the capsule with CAPSULE_FLAG_MAMA_INIT. There must be exactly one such capsule.

Parameters

dir	Directory header
descs	Descriptor array

Returns

Pointer to Mama's init descriptor, or NULL if not found

References CAPSULE_IS_MAMA_INIT, CapsuleDirHeader::desc_count, and flags.

Referenced by capsule_birth_mama(), and kernel_main_deep().

Here is the caller graph for this function:

7.6.3.5 capsule_get_payload()

```
const uint8_t * capsule_get_payload (
            const CapsuleDesc * desc,
            const uint8_t * arena_base )
```

capsule_get_payload - Get pointer to capsule payload bytes

Parameters

desc	Capsule descriptor
arena_base	Base address of payload arena

Returns

Pointer to payload bytes, or NULL on error

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_exec_init(), and capsule_run_experiment().

Here is the caller graph for this function:

7.6.3.6 capsule_validate()

```
CapsuleValidateResult capsule_validate (
            const CapsuleDesc * desc,
            const uint8_t * arena_base,
            uint64_t arena_size,
            int verify_hash )
```

capsule_validate - Validate a capsule descriptor

Parameters

desc	Capsule descriptor to validate
arena_base	Base address of payload arena
arena_size	Size of payload arena in bytes
verify_hash	If true, recompute and compare content hash

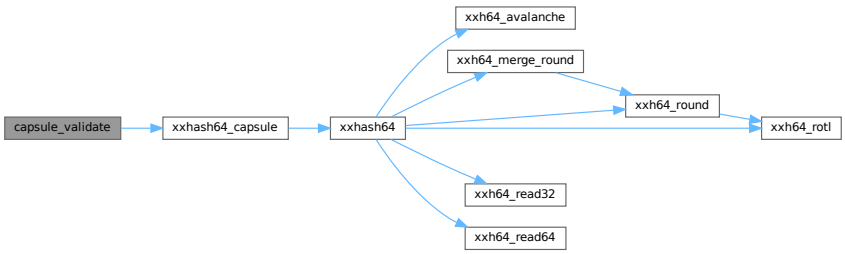
Returns

CAPSULE_VALID on success, error code otherwise

References CAPSULE_DESC_MAGIC, CAPSULE_ERR_BAD_HASH_ALG, CAPSULE_ERR_BAD_MAGIC, CAPSULE_ERR_BAD_VERSION, CAPSULE_ERR_BOUNDS, CAPSULE_ERR_HASH_MISMATCH, CAPSULE_ERR_MODE_INVA CAPSULE_ERR_NULL_PTR, CAPSULE_ERR_REVOKED_ACTIVE, CAPSULE_FLAG_ACTIVE, CAPSULE_FLAG_REVOKED, CAPSULE_HASH_BLAKE3, CAPSULE_HASH_XXHASH64, CAPSULE_MAGIC_GET_HASHALG, CAPSULE_MAGIC_GET_SIG, CAPSULE_MAGIC_GET_VERSION, CAPSULE_MODE_VALID, CAPSULE_VALID, CAPSULE_VERSION_0, and xxhash64_capsule().

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_exec_init(), and capsule_run_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

7.6.3.7 capsule_validate_result_str()

```
const char * capsule_validate_result_str (
            CapsuleValidateResult result )
```

capsule_validate_result_str - Get string for validation result

References validate_result_strings.

7.6.4 Variable Documentation

7.6.4.1 CapsuleDesc

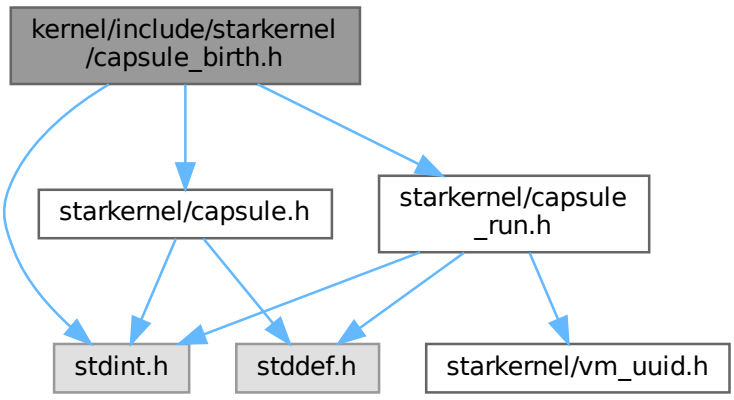
```
CapsuleDesc
```

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_console_birth(), capsule_exec_init(), capsule_run_experiment(), capsule_runcap_birth(), capsule_verify_signature(), find_and_cache_pubkey(), and kernel_main_deep().

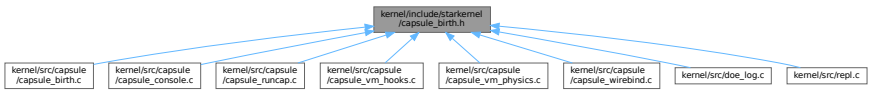
7.7 kernel/include/starkernel/capsule_birth.h File Reference

```
#include <stdint.h>
#include "starkernel/capsule.h"
#include "starkernel/capsule_run.h"
```

Include dependency graph for capsule_birth.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef int(∗ CapsuleExecFn) (void ∗vm_ctx, const char ∗code, uint64_t code_len)

  capsule_birth.h - VM Birth Protocol (M7.1)
- typedef uint64_t(∗ CapsuleDictHashFn) (void ∗vm_ctx)

  Dictionary hash function type.
- typedef void ∗(∗ CapsuleVMAllocFn) (void)

  VM allocation function type (for baby birth)

Functions

- void capsule_birth_set_hooks (CapsuleExecFn exec_fn, CapsuleDictHashFn dict_hash_fn, CapsuleVMAllocFn vm_alloc_fn)

  capsule_birth_set_hooks - Configure VM execution hooks
- CapsuleRunResult capsule_birth_mama (void ∗mama_vm, const CapsuleDirHeader ∗dir, const CapsuleDesc ∗descs, const CapsuleNameEntry ∗names, const uint8_t ∗arena)

  capsule_birth_mama - Execute Mama's init capsule
- CapsuleRunResult capsule_birth_baby (const char ∗capsule_name, const CapsuleDirHeader ∗dir, const CapsuleDesc ∗descs, const CapsuleNameEntry ∗names, const uint8_t ∗arena, VMUuid parent, int skip↩ _pki_sig, VMUuid ∗out_vm_id, void ∗∗out_vm_ctx)

  capsule_birth_baby - Birth a new VM from a named (p) capsule
- CapsuleRunResult capsule_run_experiment (void ∗mama_vm, const char ∗capsule_name, const CapsuleDirHeader ∗dir, const CapsuleDesc ∗descs, const CapsuleNameEntry ∗names, const uint8_t ∗arena, uint64_t ∗out_run_id)

  capsule_run_experiment - Execute a named (e) capsule on Mama
- void capsule_vm_hooks_register (void)

  capsule_vm_hooks_register - Wire concrete VM hooks into the capsule subsystem
- void capsule_vm_registry_init (void ∗mama_vm_ptr)

  capsule_vm_registry_init - Initialize VM registry
- int capsule_vm_kill (const char ∗name)

  capsule_vm_kill - Destroy a named VM and release all its resources
- int capsule_vm_registry_get (VMUuid vm_id, VMRegistryEntry ∗out)

  capsule_vm_registry_get - Get VM registry entry by ID
- uint32_t capsule_vm_registry_count (void)

  capsule_vm_registry_count - Get number of registered VMs
- int capsule_vm_registry_get_by_index (uint32_t index, VMRegistryEntry ∗out)

  capsule_vm_registry_get_by_index - Get registry entry by list position (birth order, stable within a boot session – the registry is append-only).
- int capsule_vm_find_by_name (const char ∗name, VMRegistryEntry ∗out)

  capsule_vm_find_by_name - Find VM registry entry by symbolic name
- int capsule_vm_find_by_name_nocase (const char ∗name, VMRegistryEntry ∗out)

  capsule_vm_find_by_name_nocase - Find VM registry entry by name (case-insensitive)
- void capsule_vm_set_state (VMUuid vm_id, uint32_t state)

capsule_vm_set_state - Update a VM's state in the registry

- void capsule_vm_set_pending_reap (VMUuid vm_id, int pending)

  capsule_vm_set_pending_reap - FABRIC-3.md ?XXVIII Stage 4 (2026-09-14): mark vm_id for deferred teardown once it is no longer worth resuming – see VMRegistryEntry.pending_reap's own doc comment for the full rationale.
- void capsule_vm_force_reap (VMUuid vm_id)

  capsule_vm_force_reap - FABRIC-3.md ?XXVIII Stage 4 (2026-09-14): unconditionally tear down vm_id regardless of VM_STATE_SWITCHED_OUT – the one caller allowed to bypass capsule_vm_kill()'s own refusal there, because this is called by the Stage 3 switcher itself (vm_core.c's checkpoint, on noticing pending_reap set), at the one point that genuinely knows the parked native-stack context will never be resumed.
- void capsule_vm_registry_set_name (VMUuid vm_id, const char *name)

  capsule_vm_registry_set_name - Assign a symbolic name to a registered VM
- void capsule_vm_kill_all_nonmama (void)

  capsule_vm_kill_all_nonmama - Kill every non-Mama VM in the registry.

7.7.1 Typedef Documentation

7.7.1.1 CapsuleDictHashFn

```
typedef uint64_t(* CapsuleDictHashFn) (void *vm_ctx)
```

Dictionary hash function type.

Parameters

vm_ctx	Opaque pointer to VM context

Returns

64-bit hash of dictionary state

7.7.1.2 CapsuleExecFn

```
typedef int(* CapsuleExecFn) (void *vm_ctx, const char *code, uint64_t code_len)
```

capsule_birth.h - VM Birth Protocol (M7.1)

Functions for birthing VMs from capsules:

- Mama init: Execute core/init.4th to establish Mama's PERSONALITY
- Baby birth: Create new VM from (p) capsule
- Experiment run: Execute (e) capsule on Mama VM execution function type

Parameters

vm_ctx	Opaque pointer to VM context
code	FORTH source code to execute
code_len	Length of code in bytes

Returns

0 on success, non-zero on error

7.7.1.3 CapsuleVMAllocFn

```
typedef void *(* CapsuleVMAllocFn) (void)
```

VM allocation function type (for baby birth)

Returns

Opaque pointer to new VM context, or NULL on failure

7.7.2 Function Documentation

7.7.2.1 capsule_birth_baby()

```
CapsuleRunResult capsule_birth_baby (
            const char * capsule_name,
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena,
            VMUuid parent,
            int skip_pki_sig,
            VMUuid * out_vm_id,
            void ** out_vm_ctx )
```

capsule_birth_baby - Birth a new VM from a named (p) capsule

Finds the capsule by colon-separated name, validates it, allocates a new VM, executes the capsule payload as IDENTITY, then (if present) executes unit.4th from the baby's block space as PERSONALITY.

Parameters

capsule_name	Colon-separated capsule name, e.g. "production:myvm.4th"
dir	Capsule directory header
descs	Capsule descriptor array
names	Capsule name entry array (parallel to descs)
arena	Capsule payload arena
parent	Who is birthing this VM (FABRIC-2.md ?H.12 step 7) – the caller's own VMUuid (e.g. vm->stadium_vm_id for a FORTH word handler), recorded on the new VM's Session.parent. Every current call site has one in scope, directly or one level up; traced live rather than assumed (checked all 6 call sites across mama_forth_words.c/capsule_console.c/ capsule_runcap.c/capsule_wirebind.c).
skip_pki_sig	0 for every build-time capsule (the normal case – checked against the compile-time-baked signature array via capsule_get_signatures()). Non-zero only for RUNCAP (FABRIC-2.md ?F.6/F.18): a heap-built, single-entry directory sourced from a user's own thumbdrive has no entry in that array at all – index 0 would silently compare against whatever real capsule happens to occupy slot 0, which is not a security check, just a guaranteed-wrong one. Trust for that content comes from CERTVERIFY (a separate root, the user's own Zuse-signed cert) already having run before RUNCAP is ever called, not from this flag – this only skips a check that was never meaningful for that content in the first place. Deliberately a plain flag, not a
	new entry point, so the policy is one call-site decision, trivially reversible.
out_vm_id	Output: assigned VM ID
out_vm_ctx	Output: new VM context

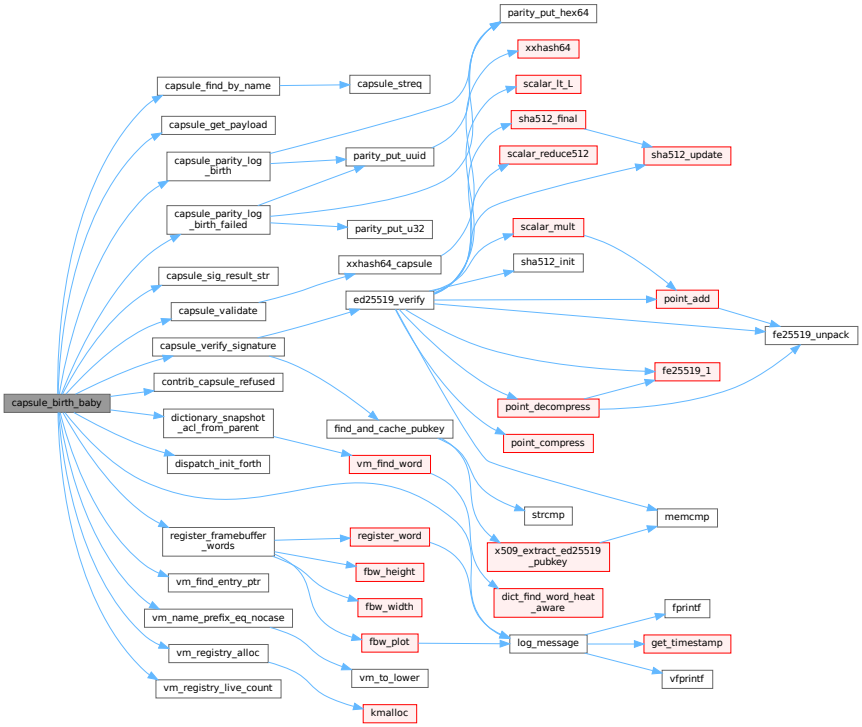
Returns

CAPSULE_RUN_OK on success, error code otherwise

References CapsuleDirHeader::arena_size, CAPSULE_BIRTH_ELIGIBLE, capsule_find_by_name(), capsule_get_payload(), capsule_parity_log_birth(), capsule_parity_log_birth_failed(), CAPSULE_RUN_ERR_EXEC_FAIL, CAPSULE_RUN_ERR_FLEET_FU CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_ERR_NOT_ELIGIBLE, CAPSULE_RUN_ERR_STILLBORN, CAPSULE_RUN_OK, CAPSULE_SIG_INVALID, CAPSULE_SIG_OK, capsule_sig_result_str(), CAPSULE_VALID, capsule_validate(), capsule_verify_signature(), CapsuleDesc, contrib_capsule_refused(), CapsuleDirHeader::desc_count, dictionary_snapshot_acl_from_parent(), dispatch_init_forth(), entry, log_message(), LOG_WARN, name, register_framebuffer_words(), vm_alloc_fn, vm_dict_hash_fn, vm_exec_fn, vm_find_entry_ptr(), vm_name_prefix_eq_nocase(), VMRegistryEntry::vm_ptr, vm_registry_alloc(), vm_registry_live_count(), VM_STATE_EMBRYO, VM_STATE_LIVE, and VM_STATE_STILLBORN.

Referenced by capsule_console_birth(), and capsule_runcap_birth().

Here is the call graph for this function:



Here is the caller graph for this function:

7.7.2.2 capsule_birth_mama()

```
CapsuleRunResult capsule_birth_mama (
            void * mama_vm,
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena )
```

capsule_birth_mama - Execute Mama's init capsule

Finds the MAMA_INIT capsule by flag, validates it, executes it on Mama's VM.

Parameters

mama_vm	Mama's VM context
dir	Capsule directory header
descs	Capsule descriptor array
names	Capsule name entry array (parallel to descs)
arena	Capsule payload arena

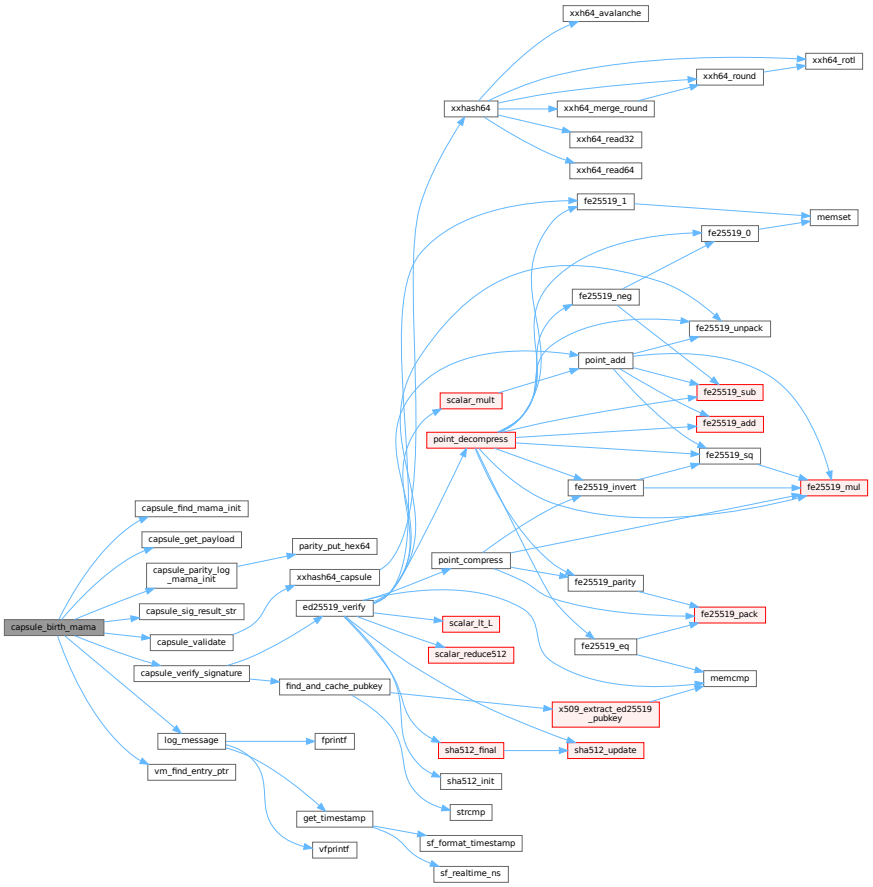
Returns

CAPSULE_RUN_OK on success, error code otherwise

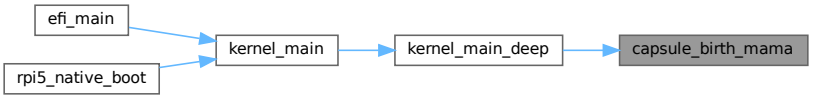
References CapsuleDirHeader::arena_size, VMRegistryEntry::birth_capsule_id, VMRegistryEntry::birth_dict_hash, capsule_find_mama_init(), capsule_get_payload(), capsule_parity_log_mama_init(), CAPSULE_RUN_ERR_EXEC_FAIL, CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_OK, CAPSULE_SIG_INVALID, CAPSULE_SIG_OK, capsule_sig_result_str(), CAPSULE_VALID, capsule_validate(), capsule_verify_signature(), CapsuleDesc, CapsuleDirHeader::desc_count, log_message(), LOG_WARN, name, vm_dict_hash_fn, vm_exec_fn, and vm_find_entry_ptr().

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.2.3 capsule_birth_set_hooks()

```
void capsule_birth_set_hooks (
            CapsuleExecFn exec_fn,
            CapsuleDictHashFn dict_hash_fn,
            CapsuleVMAllocFn vm_alloc_fn )
```

capsule_birth_set_hooks - Configure VM execution hooks

Must be called before any birth operations.

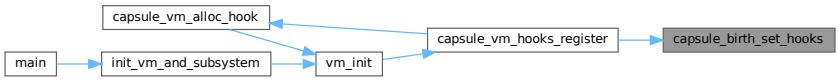
Parameters

exec_fn	Function to execute FORTH code on VM
dict_hash↩ _fn	Function to compute dictionary hash
vm_alloc_fn	Function to allocate new VM (for babies)

References vm_alloc_fn, vm_dict_hash_fn, and vm_exec_fn.

Referenced by capsule_vm_hooks_register().

Here is the caller graph for this function:



7.7.2.4 capsule_run_experiment()

```
CapsuleRunResult capsule_run_experiment (
            void * mama_vm,
            const char * capsule_name,
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena,
            uint64_t * out_run_id )
```

capsule_run_experiment - Execute a named (e) capsule on Mama

Finds the experiment capsule by colon-separated name, validates it, executes it on Mama's VM without creating a new VM.

Parameters

mama_vm	Mama's VM context
capsule_name	Colon-separated capsule name, e.g. "experiments:doe-l8:init-l8-stable.4th"
dir	Capsule directory header
descs	Capsule descriptor array
names	Capsule name entry array (parallel to descs)
arena	Capsule payload arena
out_run_id	Output: assigned run ID

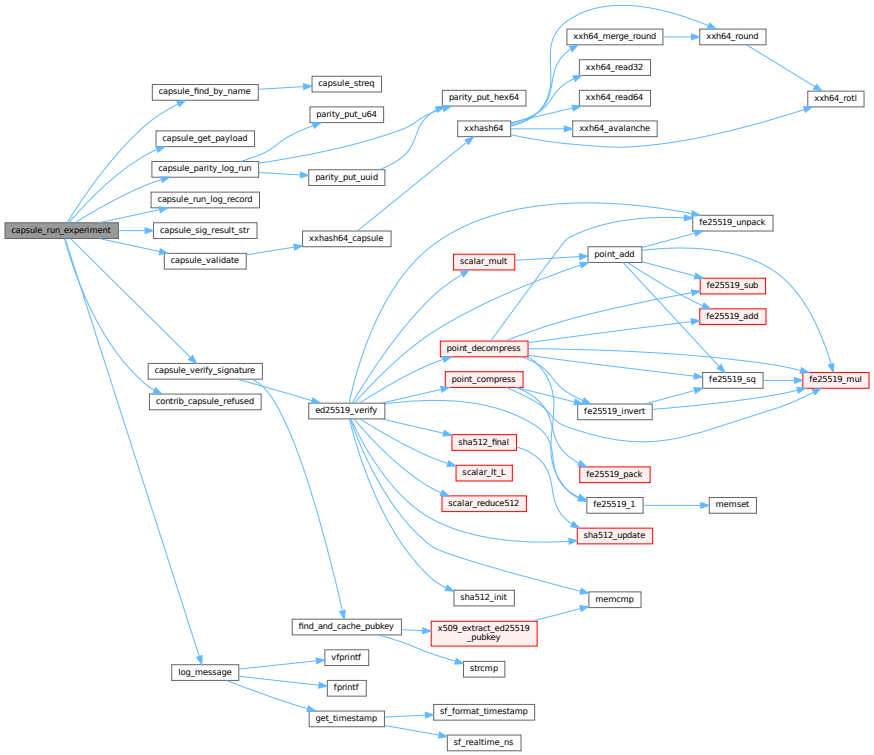
Returns

CAPSULE_RUN_OK on success, error code otherwise

References CapsuleDirHeader::arena_size, CAPSULE_DOE_ELIGIBLE, capsule_find_by_name(), capsule_get_payload(), CapsuleRunRecord::capsule_hash, CapsuleRunRecord::capsule_id, capsule_parity_log_run(), CAPSULE_RUN_ERR_EXEC_FAIL,

CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_ERR_NOT_ELIGIBLE, capsule_run_log_record(), CAPSULE_RUN_OK, CAPSULE_SIG_INVALID, CAPSULE_SIG_OK, capsule_sig_result_str(), CAPSULE_VALID, capsule_validate(), capsule_verify_signature(), CapsuleDesc, contrib_capsule_refused(), CapsuleDirHeader::desc_count, CapsuleRunRecord::ended_ns, CapsuleRunRecord::flags, log_message(), LOG_WARN, name, CapsuleRunRecord::post_dict_hash, CapsuleRunRecord::pre_dict_h, CapsuleRunRecord::reserved, CapsuleRunRecord::result_code, CapsuleRunRecord::run_id, CapsuleRunRecord::started_ns, vm_dict_hash_fn, vm_exec_fn, and CapsuleRunRecord::vm_id.

Here is the call graph for this function:



7.7.2.5 capsule_vm_find_by_name()

```
int capsule_vm_find_by_name (
            const char * name,
            VMRegistryEntry * out )
```

capsule_vm_find_by_name - Find VM registry entry by symbolic name

Case-sensitive. Returns the first match.

Parameters

name	Symbolic VM name, e.g. "Hermes"
out	Output: registry entry copy

Returns

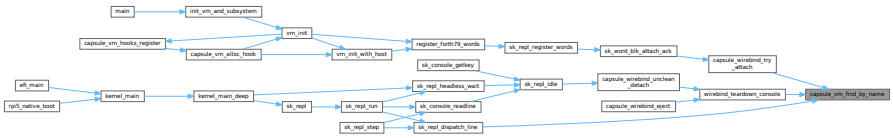
0 if found, -1 if not found

References vm_node::entry, VMRegistryEntry::name, name, vm_node::next, vm_name_eq(), and vm_registry_head.

Referenced by capsule_wirebind_try_attach(), sk_repl_dispatch_line(), and wirebind_teardown_console().

Here is the call graph for this function:

Here is the caller graph for this function:



7.7.2.6 capsule_vm_find_by_name_nocase()

```
int capsule_vm_find_by_name_nocase (
            const char * name,
            VMRegistryEntry * out )
```

capsule_vm_find_by_name_nocase - Find VM registry entry by name (case-insensitive)

Used by BIRTH for idempotency: prevents birthing a second VM with the same name regardless of case differences.

Parameters

name	Symbolic VM name (compared case-insensitively)
out	Output: registry entry copy

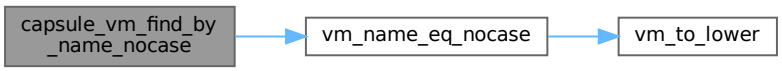
Returns

0 if found, -1 if not found

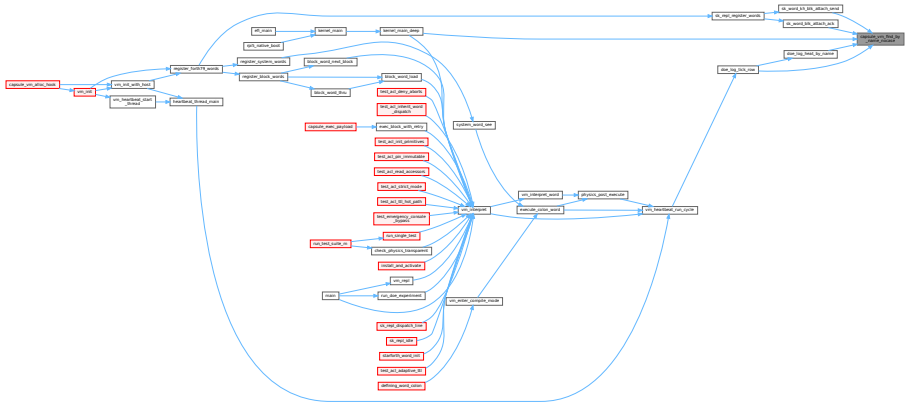
References vm_node::entry, VMRegistryEntry::name, name, vm_node::next, vm_name_eq_nocase(), and vm_registry_head.

Referenced by doe_log_heat_by_name(), doe_log_tick_row(), kernel_main_deep(), sk_word_blk_attach_ack(), and sk_word_kh_blk_attach_send().

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.2.7 capsule_vm_force_reap()

```
void capsule_vm_force_reap (
            VMUuid vm_id )
```

capsule_vm_force_reap - FABRIC-3.md ?XXVIII Stage 4 (2026-09-14): unconditionally tear down vm_id regardless of VM_STATE_SWITCHED_OUT – the one caller allowed to bypass capsule_vm_kill()'s own refusal there, because this is called *by* the Stage 3 switcher itself (vm_core.c's checkpoint, on noticing pending_reap set), at the one point that genuinely knows the parked native-stack context will never be resumed.

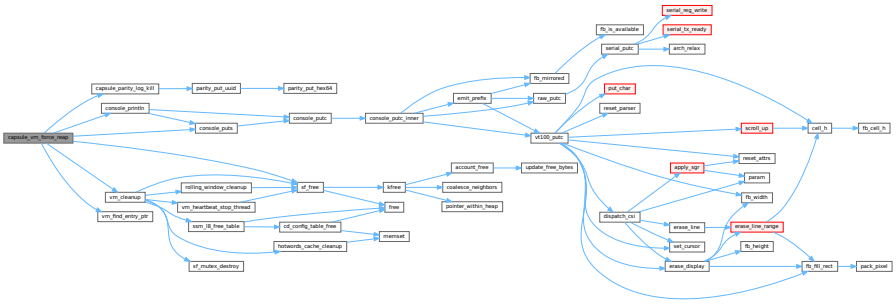
Also releases the VM's own switch-signal slot (sk_vm_switch_signal_unregister()) – generic cleanup, independent of whatever subsystem (WIREBIND today) set pending_reap in the first place. No-op if vm_id isn't registered or is already DEAD.

Parameters

vm_id	VM ID to reap.

References capsule_parity_log_kill(), console_println(), console_puts(), entry, sf_free(), vm_cleanup(), vm_find_entry_ptr(), VM_NAME_MAX, and VM_STATE_DEAD.

Here is the call graph for this function:



7.7.2.8 capsule_vm_hooks_register()

```
void capsule_vm_hooks_register (
            void  )
```

capsule_vm_hooks_register - Wire concrete VM hooks into the capsule subsystem

Registers capsule_exec_hook, capsule_dict_hash_hook, and capsule_vm_alloc_hook. Must be called after vm_init() on Mama's VM and before any birth operations.

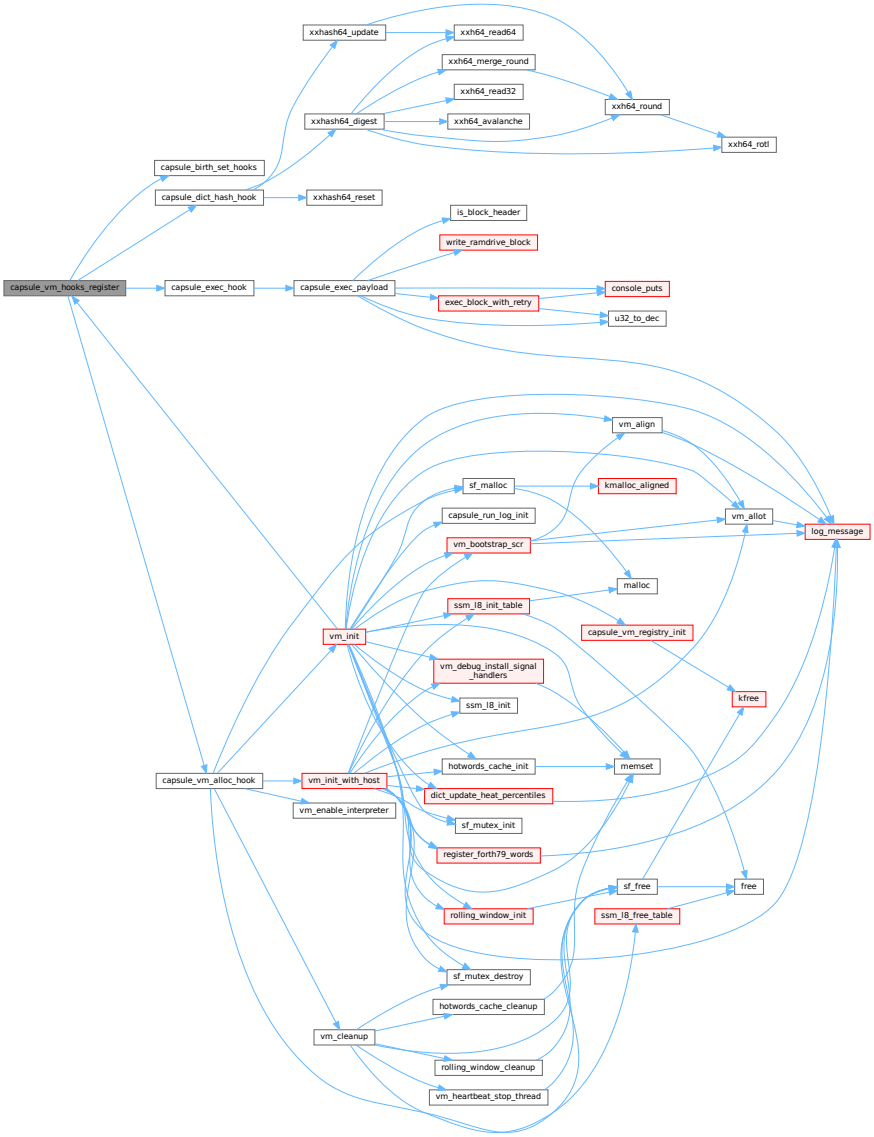
capsule_vm_hooks_register - Wire concrete VM hooks into the capsule subsystem

Must be called after vm_init() completes on Mama's VM, before any capsule birth or experiment operations.

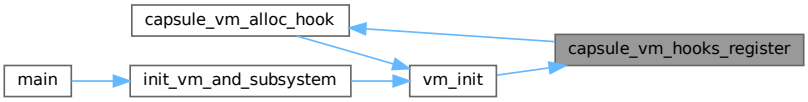
References capsule_birth_set_hooks(), capsule_dict_hash_hook(), capsule_exec_hook(), and capsule_vm_alloc_hook().

Referenced by vm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.2.9 capsule_vm_kill()

```
int capsule_vm_kill (
            const char * name )
```

capsule_vm_kill - Destroy a named VM and release all its resources

Looks up the VM by name (case-insensitive). Hera (VM 0) cannot be killed. If the VM is already DEAD the call is a no-op. On success: vm_cleanup + sf_free, state → VM_STATE_DEAD, name cleared.

Parameters

name	Symbolic name of the VM to kill (case-insensitive)

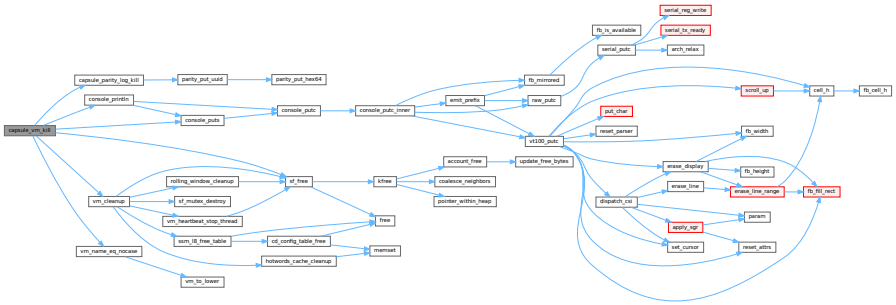
Returns

0 on success (or already dead), -1 if not found or refused

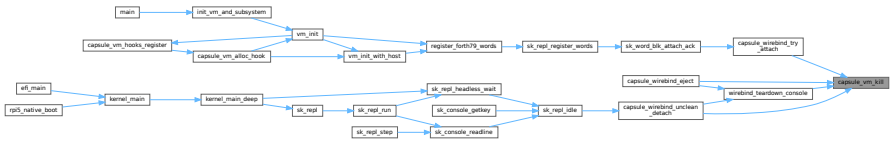
References capsule_parity_log_kill(), console_println(), console_puts(), entry, vm_node::entry, VMRegistryEntry::name, name, vm_node::next, sf_free(), vm_cleanup(), vm_name_eq_nocase(), VM_NAME_MAX, vm_registry_head, VM_STATE_DEAD, and VM_STATE_SWITCHED_OUT.

Referenced by capsule_wirebind_eject(), capsule_wirebind_try_attach(), capsule_wirebind_unclean_detach(), and wirebind_teardown_console().

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.2.10 capsule_vm_kill_all_nonmama()

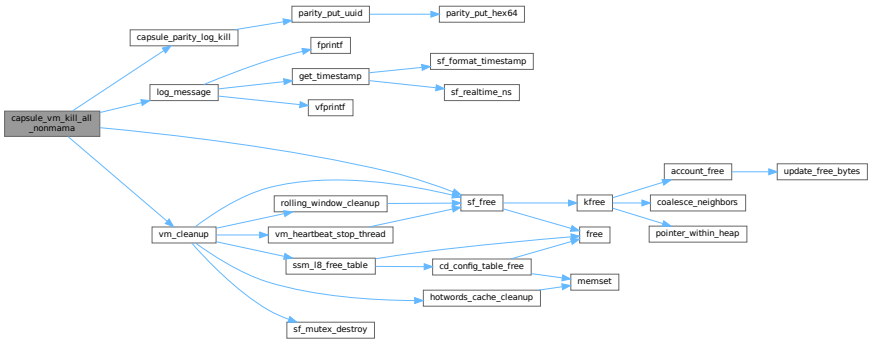
```
void capsule_vm_kill_all_nonmama (
            void  )
```

capsule_vm_kill_all_nonmama - Kill every non-Mama VM in the registry.

Sets halted, calls vm_cleanup + sf_free, marks state DEAD. Used by Hera's BYE immediately before arch_cold_reset() to reap all children.

References capsule_parity_log_kill(), vm_node::entry, VM::halted, log_message(), LOG_WARN, VMRegistryEntry::name, vm_node::next, sf_free(), VMRegistryEntry::stadium_patron_cell, VMRegistryEntry::state, vm_cleanup(), VMRegistryEntry::vm_id, VMRegistryEntry::vm_ptr, vm_registry_head, VM_STATE_DEAD, and VM_STATE_SWITCHED_OUT.

Here is the call graph for this function:



7.7.2.11 capsule_vm_registry_count()

```
uint32_t capsule_vm_registry_count (
            void  )
```

capsule_vm_registry_count - Get number of registered VMs

References vm_registry_count.

7.7.2.12 capsule_vm_registry_get()

```
int capsule_vm_registry_get (
            VMUuid vm_id,
            VMRegistryEntry * out )
```

capsule_vm_registry_get - Get VM registry entry by ID

Parameters

vm↩ _id	VM ID to look up
out	Output: registry entry copy

Returns

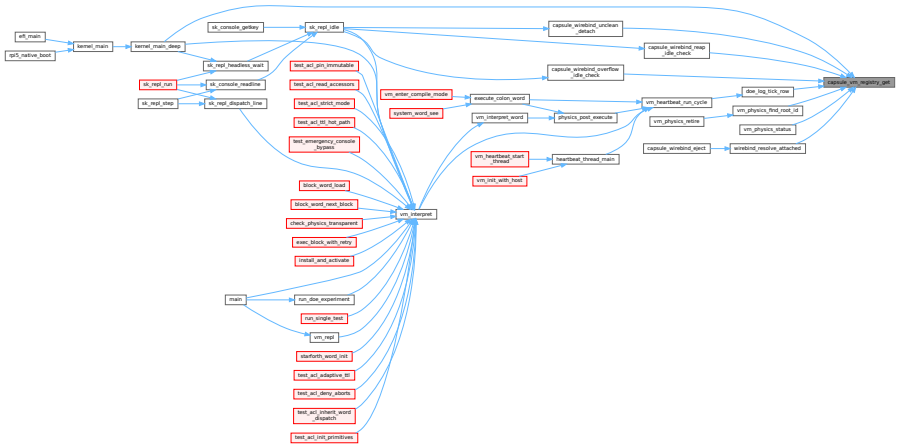
0 on success, -1 if not found

References entry, and vm_find_entry_ptr().

Referenced by capsule_wirebind_overflow_idle_check(), capsule_wirebind_reap_idle_check(), capsule_wirebind_unclean_detach(), doe_log_tick_row(), kernel_main_deep(), vm_physics_find_root_id(), vm_physics_status(), and wirebind_resolve_attached().

Here is the call graph for this function:

Here is the caller graph for this function:



7.7.2.13 capsule_vm_registry_get_by_index()

```
int capsule_vm_registry_get_by_index (
            uint32_t index,
            VMRegistryEntry * out )
```

capsule_vm_registry_get_by_index - Get registry entry by list position (birth order, stable within a boot session – the registry is append-only).

For enumeration (e.g. the idle-loop messaging pump, FABRIC-2.md Phase C, 2026-08-28), where no vm_id is known up front. Index range is [0, capsule_vm_registry_count()).

Parameters

index	Zero-based position in birth order
out	Output: registry entry copy

Returns

0 on success, -1 if index is out of range

References vm_node::entry, vm_node::next, and vm_registry_head.

7.7.2.14 capsule_vm_registry_init()

```
void capsule_vm_registry_init (
            void * mama_vm_ptr )
```

capsule_vm_registry_init - Initialize VM registry

Allocates Mama's registry node (VM 0, name "Hera") via kmalloc and stores mama_vm_ptr so KILL can guard against destroying Mama. Must be called after kmalloc_init().

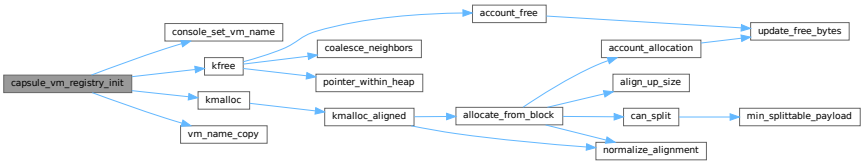
Parameters

mama_vm_ptr	Pointer to Mama's VM object (e.g. &sk_mama_vm)

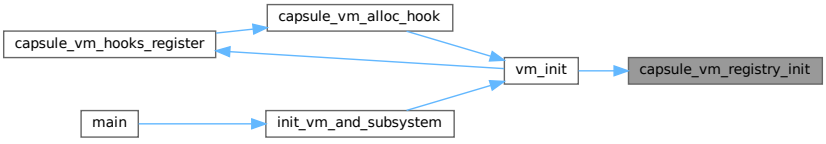
References VMRegistryEntry::birth_capsule_id, VMRegistryEntry::birth_dict_hash, VMRegistryEntry::birth_timestamp_ns, console_set_vm_name(), vm_node::entry, VMRegistryEntry::flags, kfree(), kmalloc(), VMRegistryEntry::name, vm_node::next, VMRegistryEntry::parent_vm_id, VMRegistryEntry::stadium_patron_cell, VMRegistryEntry::state, VMRegistryEntry::vm_id, vm_name_copy(), VM_NAME_MAX, VMRegistryEntry::vm_ptr, vm_registry_count, vm_registry_head, and VM_STATE_LIVE.

Referenced by vm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.2.15 capsule_vm_registry_set_name()

```
void capsule_vm_registry_set_name (
            VMUuid vm_id,
            const char * name )
```

capsule_vm_registry_set_name - Assign a symbolic name to a registered VM

Truncates to VM_NAME_MAX-1 characters. No-op if vm_id not found.

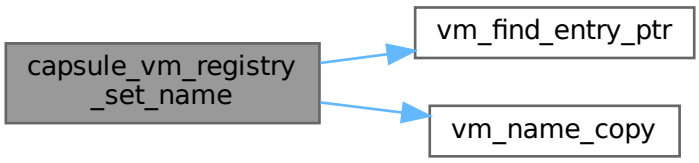
Parameters

vm↩ _id	VM ID to name
name	Symbolic name string

References entry, name, vm_find_entry_ptr(), and vm_name_copy().

Referenced by capsule_console_birth(), and capsule_runcap_birth().

Here is the call graph for this function:



Here is the caller graph for this function:

7.7.2.16 capsule_vm_set_pending_reap()

```
void capsule_vm_set_pending_reap (
            VMUuid vm_id,
            int pending )
```

capsule_vm_set_pending_reap - FABRIC-3.md ?XXVIII Stage 4 (2026-09-14): mark vm_id for deferred teardown once it is no longer worth resuming – see VMRegistryEntry.pending_reap's own doc comment for the full rationale.

No-op if vm_id isn't registered.

Parameters

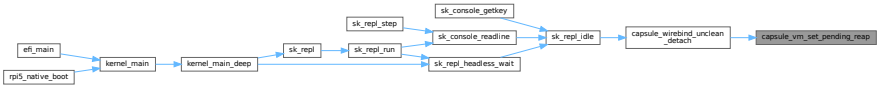
vm_id	VM ID to mark.
pending	1 to mark, 0 to clear (e.g. a re-attach of the same identity before the switcher ever reaped it).

References entry, and vm_find_entry_ptr().

Referenced by capsule_wirebind_unclean_detach().

Here is the call graph for this function:

Here is the caller graph for this function:



7.7.2.17 capsule_vm_set_state()

```
void capsule_vm_set_state (
            VMUuid vm_id,
            uint32_t state )
```

capsule_vm_set_state - Update a VM's state in the registry

Parameters

vm_id	VM ID to update
state	New VMState value

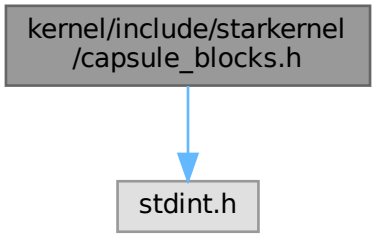
References entry, and vm_find_entry_ptr().

Here is the call graph for this function:

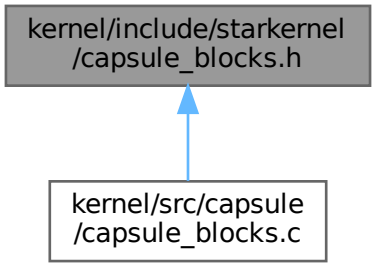
7.8 kernel/include/starkernel/capsule_blocks.h File Reference

```
#include <stdint.h>
```

Include dependency graph for capsule_blocks.h:



This graph shows which files directly or indirectly include this file:



Functions

- int capsule_block_header (const uint8_t *p, const uint8_t *end, uint32_t *out_num, const uint8_t **out_after)

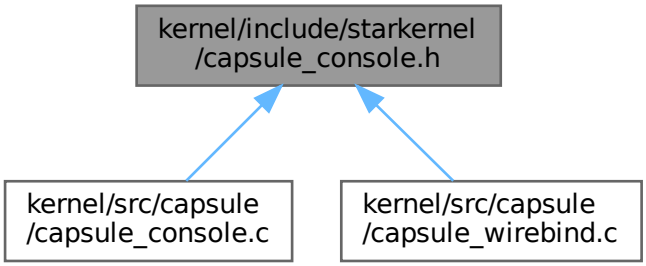
7.8.1 Function Documentation

7.8.1.1 capsule_block_header()

```
int capsule_block_header (
            const uint8_t * p,
            const uint8_t * end,
            uint32_t * out_num,
            const uint8_t ** out_after )
```

7.9 kernel/include/starkernel/capsule_console.h File Reference

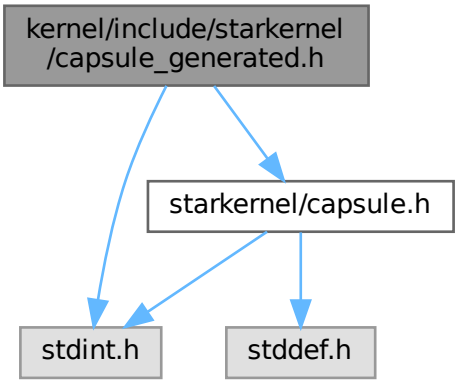
This graph shows which files directly or indirectly include this file:



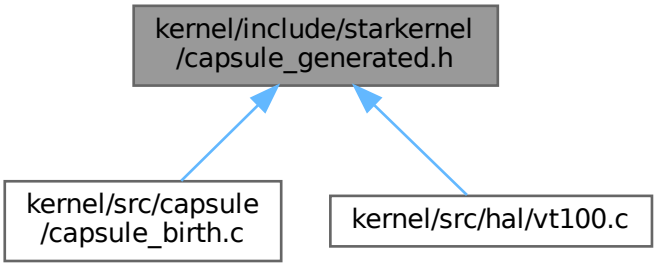
7.10 kernel/include/starkernel/capsule_generated.h File Reference

```
#include <stdint.h>
#include "starkernel/capsule.h"
```

Include dependency graph for capsule_generated.h:



This graph shows which files directly or indirectly include this file:



Functions

- __attribute__ ((visibility("hidden"))) extern const uint8_t capsule_arena[ ]
  capsule_generated.h - Declarations for mkcapsule-generated capsule store.

7.10.1 Function Documentation

7.10.1.1 __attribute__()

```
__attribute__ (
            (visibility("hidden"))  ) const
```

capsule_generated.h - Declarations for mkcapsule-generated capsule store.

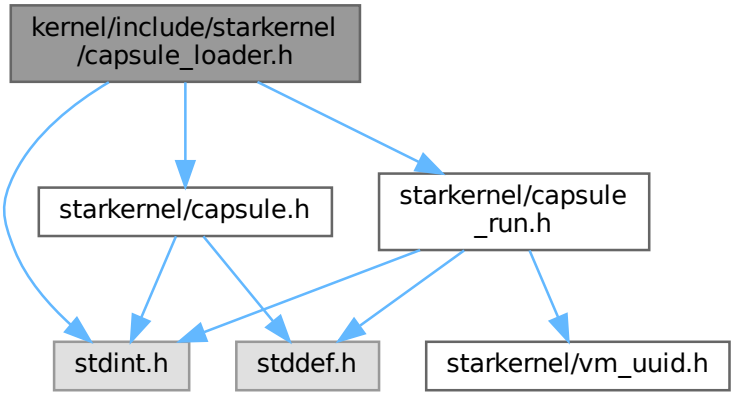
The matching capsule_generated.c is produced at build time by mkcapsule. These three arrays plus the directory header are compiled into .rodata.

7.11 kernel/include/starkernel/capsule_loader.h File Reference

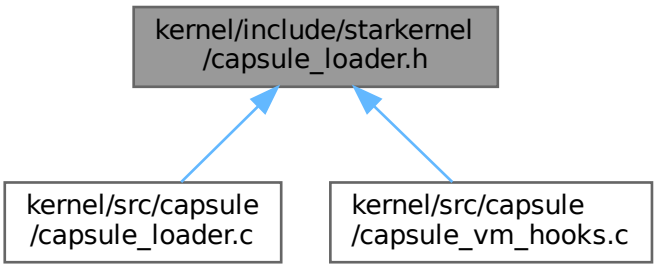
```
#include <stdint.h>
#include "starkernel/capsule.h"
```

```
#include "starkernel/capsule_run.h"
```

Include dependency graph for capsule_loader.h:



This graph shows which files directly or indirectly include this file:



Functions

- int capsule_load_blocks (const uint8_t *payload, uint64_t length, uint32_t *out_entry_block)

  capsule_loader.h - Block Capsule Loader and Executor (M7.1)
- void capsule_clear_blocks (const uint8_t *payload, uint64_t length)

  capsule_clear_blocks - Zero all ramdrive slots referenced in a capsule payload
- int capsule_exec_payload (void *vm_opaque, const uint8_t *payload, uint64_t length)

  capsule_exec_payload - Parse, populate block device, and execute a payload
- CapsuleRunResult capsule_exec_init (void *vm, const char *capsule_name, const CapsuleDirHeader *dir, const CapsuleDesc *descs, const CapsuleNameEntry *names, const uint8_t *arena)

  capsule_exec_init - Load and execute an init capsule

7.11.1 Function Documentation

7.11.1.1 capsule_clear_blocks()

```
void capsule_clear_blocks (
            const uint8_t * payload,
            uint64_t length )
```

capsule_clear_blocks - Zero all ramdrive slots referenced in a capsule payload

Re-parses the payload for "Block <num>" headers and zeros each corresponding 1024-byte ramdrive slot. NOT called by capsule_exec_init() (fixed 2026-09-10 – see its own doc comment): callers that specifically want a capsule's block range freed for reuse (e.g. kernel_main.c, right after Mama's own init.4th birth, to free that range for interactive block-editor use) call this themselves, explicitly, after exec returns.

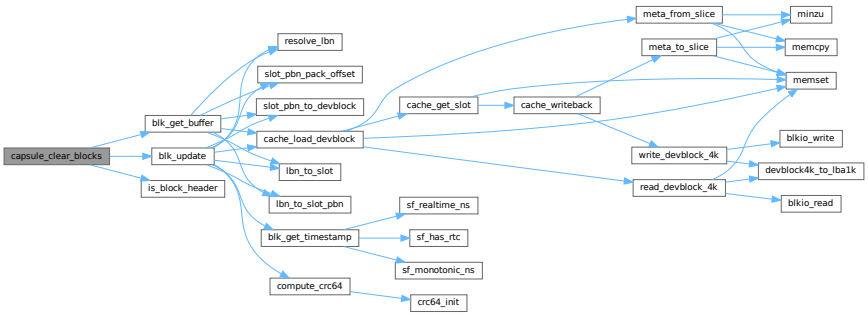
Parameters

payload	Raw capsule payload bytes
length	Payload length in bytes

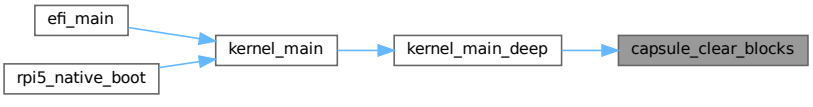
References blk_get_buffer(), blk_update(), is_block_header(), and LOADER_BLOCK_SIZE.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.1.2 capsule_exec_init()

```
CapsuleRunResult capsule_exec_init (
            void * vm,
            const char * capsule_name,
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena )
```

capsule_exec_init - Load and execute an init capsule

Full init sequence:

1. Locate capsule by colon-separated name in the capsule directory
2. Validate content hash
3. capsule_exec_payload: populate block device + execute content

Block content is NOT cleared afterward (fixed 2026-09-10): it stays resident in ramdrive storage so a subsequent Standard BLOCK/LOAD on the same block number reads back what EXEC just wrote, matching FORTH-79 block-persistence semantics. Callers that specifically want the old "free this capsule's block range" behavior (e.g. kernel_main.c freeing init.4th's range for later interactive block-editor use) call capsule_clear_blocks() themselves, explicitly, after this returns.

Parameters

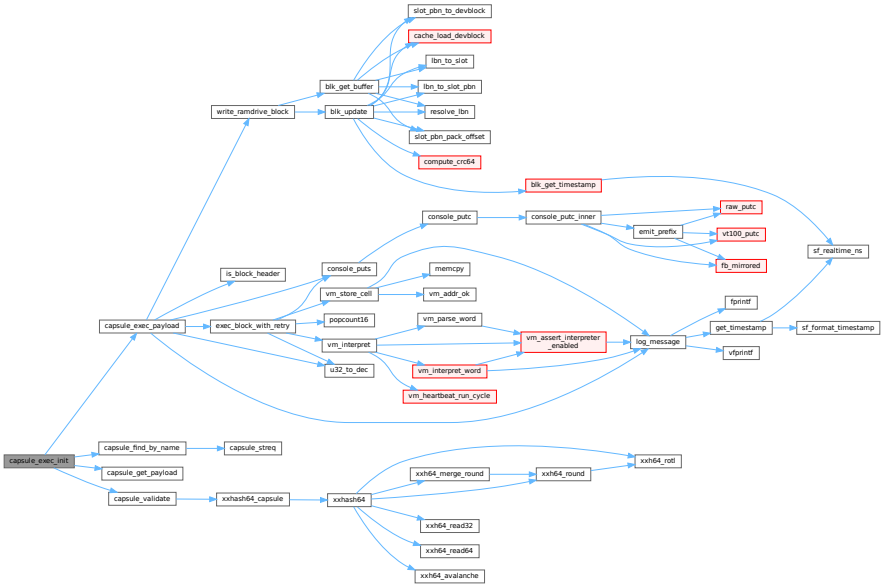
vm	VM to execute on
capsule_name	Colon-separated capsule name, e.g. "init.4th"
dir	Capsule directory header
descs	Capsule descriptor array
names	Capsule name entry array (parallel to descs)
arena	Capsule payload arena

Returns

CAPSULE_RUN_OK on success, error code otherwise

References CapsuleDirHeader::arena_size, capsule_exec_payload(), capsule_find_by_name(), capsule_get_payload(), CAPSULE_RUN_ERR_EXEC_FAIL, CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_OK, CAPSULE_VALID, capsule_validate(), and CapsuleDesc.

Here is the call graph for this function:



7.11.1.3 capsule_exec_payload()

```
int capsule_exec_payload (
            void * vm_opaque,
            const uint8_t * payload,
            uint64_t length )
```

capsule_exec_payload - Parse, populate block device, and execute a payload

For each "Block <num>" section: write content to block device (warn+truncate if > 1KB), then execute line-by-line via vm_interpret. No LOAD is injected.

Parameters

vm_opaque	VM to execute on
payload	Raw payload bytes
length	Payload length in bytes

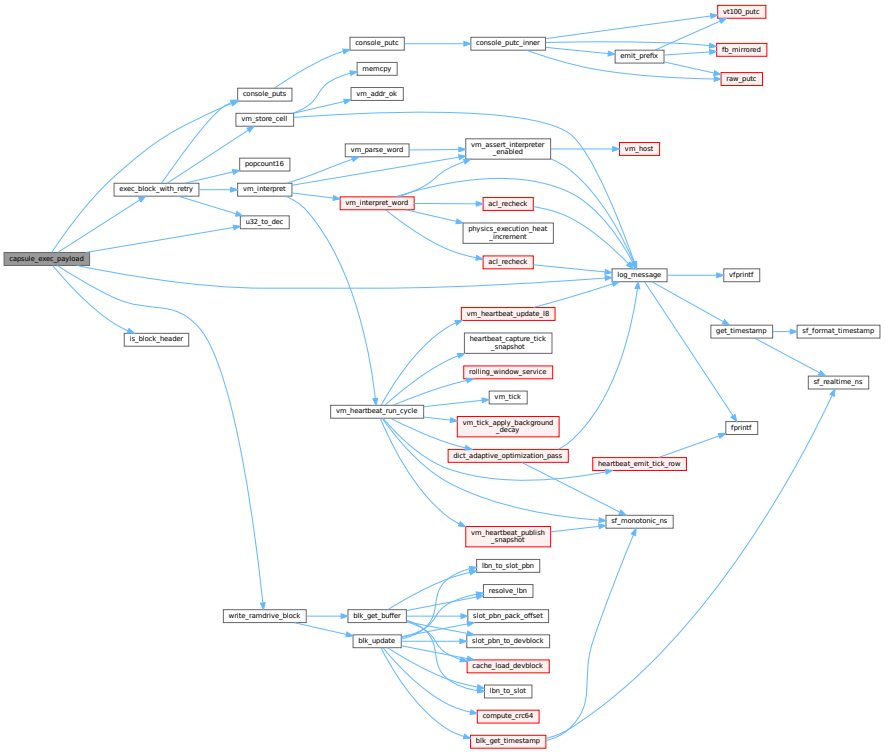
Returns

0 on success, -1 on execution error

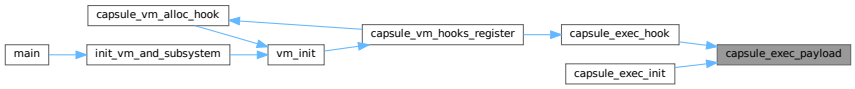
References console_puts(), exec_block_with_retry(), is_block_header(), LOADER_BLOCK_SIZE, LOG_INFO, log_message(), u32_to_dec(), and write_ramdrive_block().

Referenced by capsule_exec_hook(), and capsule_exec_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.1.4 capsule_load_blocks()

```
int capsule_load_blocks (
            const uint8_t * payload,
            uint64_t length,
            uint32_t * out_entry_block )
```

capsule_loader.h - Block Capsule Loader and Executor (M7.1)

Parses a capsule payload for "Block <num>" headers and writes each block's content into the correct ramdrive slot, then executes the entry block and zeros the ramdrive slots afterward.

Format: Block <decimal>
<content: up to 1024 bytes of FORTH source> Block <decimal>
...

Blocks may appear in any order; block numbers are arbitrary. Content exceeding 1024 bytes is truncated with a warning. capsule_load_blocks - Parse a capsule payload and write blocks to the ramdrive

For each "Block <num>" header found in the payload:

1. Zero the 1024-byte ramdrive slot for block <num>
2. Copy content bytes into the slot (excess beyond 1024 silently dropped)
3. Mark the slot dirty via blk_update()

Content between headers may be text or binary. Anything before the first "Block <num>" header is ignored.

Parameters

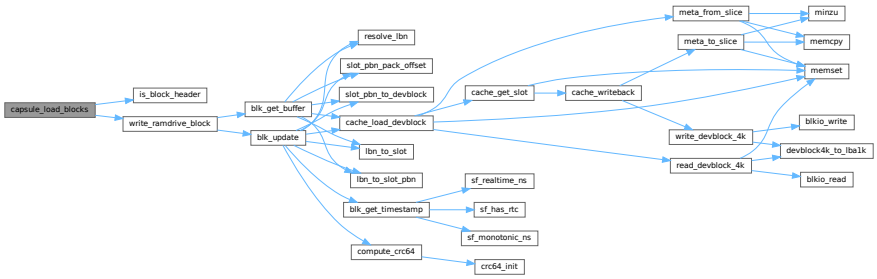
payload	Raw capsule payload bytes (not null-terminated)
length	Payload length in bytes
out_entry_block	If non-NULL, receives the first block number found (the execution entry point); set to 0 if no blocks found

Returns

Number of blocks written, or -1 on invalid input

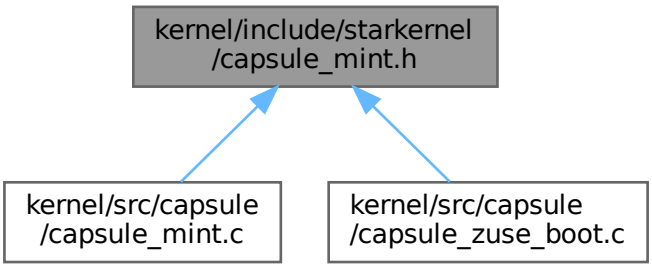
References is_block_header(), and write_ramdrive_block().

Here is the call graph for this function:



7.12 kernel/include/starkernel/capsule_mint.h File Reference

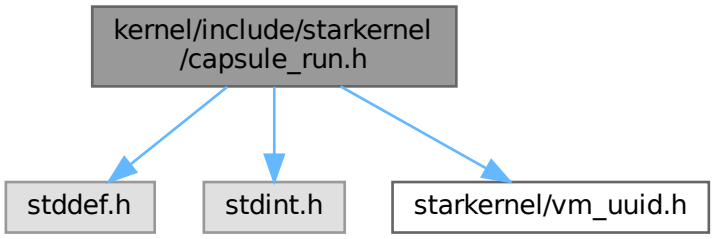
This graph shows which files directly or indirectly include this file:



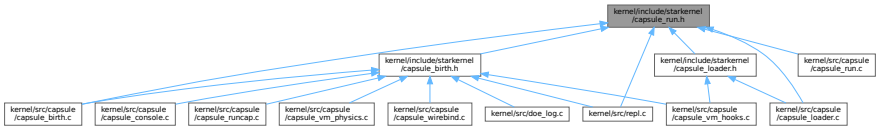
7.13 kernel/include/starkernel/capsule_run.h File Reference

```
#include <stddef.h>
#include <stdint.h>
#include "starkernel/vm_uuid.h"
```

Include dependency graph for capsule_run.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct CapsuleRunRecord
- struct VMRegistryEntry

Macros

- #define CAPSULE_MAX_RUN_RECORDS 1024

  capsule_run.h - DoE Run Logging (M7.1)
- #define VM_NAME_MAX 64

  Maximum length of a VM symbolic name, including null terminator.

Enumerations

- enum CapsuleRunResult {
  CAPSULE_RUN_OK = 0 , CAPSULE_RUN_ERR_INVALID , CAPSULE_RUN_ERR_NOT_ELIGIBLE ,
  CAPSULE_RUN_ERR_EXEC_FAIL ,
  CAPSULE_RUN_ERR_HASH_MISMATCH , CAPSULE_RUN_ERR_STILLBORN , CAPSULE_RUN_ERR_FLEET_FULL
  }
- enum VMState {
  VM_STATE_EMBRYO = 0 , VM_STATE_LIVE , VM_STATE_STOPPED , VM_STATE_STILLBORN ,
  VM_STATE_DEAD , VM_STATE_SWITCHED_OUT }

Functions

- void capsule_run_log_init (void)

  capsule_run_log_init - Initialize run log
- uint64_t capsule_run_log_record (const CapsuleRunRecord *record)

  capsule_run_log_record - Log a run record
- int capsule_run_log_get (uint64_t run_id, CapsuleRunRecord *out)

  capsule_run_log_get - Get a run record by ID
- uint32_t capsule_run_log_count (void)

  capsule_run_log_count - Get number of logged runs
- void capsule_parity_log_birth (VMUuid vm_id, uint64_t capsule_id, uint64_t capsule_hash, uint64_t dict_↩ hash)

  capsule_parity_log_birth - Log VM birth parity record
- void capsule_parity_log_birth_failed (VMUuid vm_id, uint64_t capsule_id, CapsuleRunResult error, uint64_t partial_dict_hash)

  capsule_parity_log_birth_failed - Log failed birth
- void capsule_parity_log_run (VMUuid vm_id, uint64_t run_id, uint64_t capsule_id, uint64_t pre_dict_hash, uint64_t post_dict_hash)

  capsule_parity_log_run - Log DoE run parity record
- void capsule_parity_log_mama_init (uint64_t capsule_id, uint64_t capsule_hash, uint64_t dict_hash)

  capsule_parity_log_mama_init - Log Mama init parity record
- void capsule_parity_log_kill (VMUuid vm_id, const char *name)

  capsule_parity_log_kill - Log VM kill parity record
- void capsule_parity_set_output (void(*putc_fn)(char), void(*puts_fn)(const char *))

  capsule_parity_set_output - Set output hooks for parity logging

7.13.1 Macro Definition Documentation

7.13.1.1 CAPSULE_MAX_RUN_RECORDS

```
#define CAPSULE_MAX_RUN_RECORDS 1024
```

capsule_run.h - DoE Run Logging (M7.1)

Structures for logging capsule execution runs and VM births. Supports provenance tracking and experiment reproducibility. Phase A: Fixed ring buffer size

7.13.1.2 VM_NAME_MAX

```
#define VM_NAME_MAX 64
```

Maximum length of a VM symbolic name, including null terminator.

7.13.2 Enumeration Type Documentation

7.13.2.1 CapsuleRunResult

```
enum CapsuleRunResult
```

Enumerator

CAPSULE_RUN_OK	
CAPSULE_RUN_ERR_INVALID	
CAPSULE_RUN_ERR_NOT_ELIGIBLE	
CAPSULE_RUN_ERR_EXEC_FAIL	
CAPSULE_RUN_ERR_HASH_MISMATCH	
CAPSULE_RUN_ERR_STILLBORN	
CAPSULE_RUN_ERR_FLEET_FULL	

7.13.2.2 VMState

```
enum VMState
```

Enumerator

VM_STATE_EMBRYO	
VM_STATE_LIVE	
VM_STATE_STOPPED	
VM_STATE_STILLBORN	
VM_STATE_DEAD	
VM_STATE_SWITCHED_OUT	

7.13.3 Function Documentation

7.13.3.1 capsule_parity_log_birth()

```
void capsule_parity_log_birth (
            VMUuid vm_id,
            uint64_t capsule_id,
            uint64_t capsule_hash,
            uint64_t dict_hash )
```

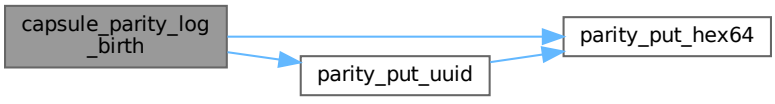
capsule_parity_log_birth - Log VM birth parity record

Output format: PARITY:BIRTH vm_id=N capsule_id=X mode=p capsule_hash=H dict_hash=D

References PARITY_EMIT, PARITY_HAVE_SINK, parity_put_hex64(), and parity_put_uuid().

Referenced by capsule_birth_baby().

Here is the call graph for this function:



Here is the caller graph for this function:

7.13.3.2 capsule_parity_log_birth_failed()

```
void capsule_parity_log_birth_failed (
            VMUuid vm_id,
            uint64_t capsule_id,
            CapsuleRunResult error,
            uint64_t partial_dict_hash )
```

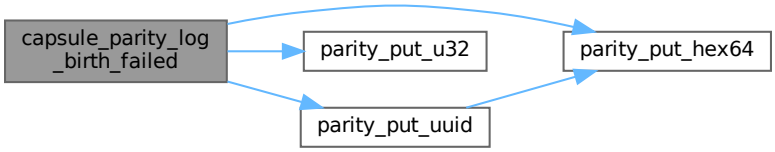
capsule_parity_log_birth_failed - Log failed birth

Output format: PARITY:BIRTH_FAILED vm_id=N capsule_id=X error=E partial_dict_hash=H

References PARITY_EMIT, PARITY_HAVE_SINK, parity_put_hex64(), parity_put_u32(), and parity_put_uuid().

Referenced by capsule_birth_baby().

Here is the call graph for this function:



Here is the caller graph for this function:

7.13.3.3 capsule_parity_log_kill()

```
void capsule_parity_log_kill (
            VMUuid vm_id,
            const char * name )
```

capsule_parity_log_kill - Log VM kill parity record

Output format: PARITY:KILL vm_id=N name=X

References name, PARITY_EMIT, PARITY_HAVE_SINK, and parity_put_uuid().

Referenced by capsule_vm_force_reap(), capsule_vm_kill(), and capsule_vm_kill_all_nonmama().

Here is the call graph for this function:

Here is the caller graph for this function:

7.13.3.4 capsule_parity_log_mama_init()

```
void capsule_parity_log_mama_init (
            uint64_t capsule_id,
            uint64_t capsule_hash,
            uint64_t dict_hash )
```

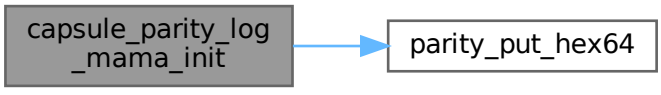
capsule_parity_log_mama_init - Log Mama init parity record

Output format: PARITY:MAMA_INIT capsule_id=X mode=m capsule_hash=H dict_hash=D

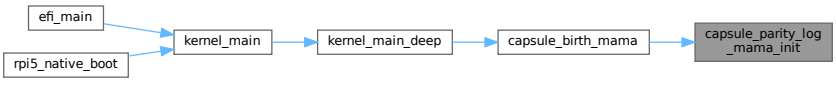
References PARITY_EMIT, PARITY_HAVE_SINK, and parity_put_hex64().

Referenced by capsule_birth_mama().

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.5 capsule_parity_log_run()

```
void capsule_parity_log_run (
            VMUuid vm_id,
            uint64_t run_id,
            uint64_t capsule_id,
            uint64_t pre_dict_hash,
            uint64_t post_dict_hash )
```

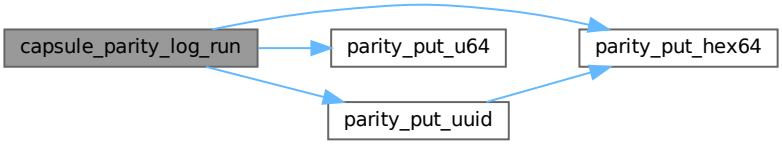
capsule_parity_log_run - Log DoE run parity record

Output format: PARITY:RUN vm_id=N run_id=R capsule_id=X mode=e pre_dict=P post_dict=Q

References PARITY_EMIT, PARITY_HAVE_SINK, parity_put_hex64(), parity_put_u64(), and parity_put_uuid().

Referenced by capsule_run_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

7.13.3.6 capsule_parity_set_output()

```
void capsule_parity_set_output (
            void(*)(char) putc_fn,
            void(*)(const char *) puts_fn )
```

capsule_parity_set_output - Set output hooks for parity logging

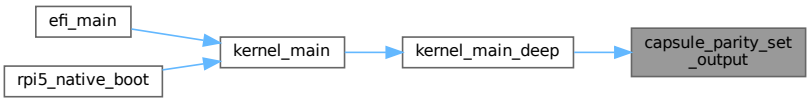
Parameters

putc_fn	Function to output a single character
puts_fn	Function to output a string

References parity_putc, and parity_puts.

Referenced by kernel_main_deep().

Here is the caller graph for this function:



7.13.3.7 capsule_run_log_count()

```
uint32_t capsule_run_log_count (
            void  )
```

capsule_run_log_count - Get number of logged runs

References run_log_count.

7.13.3.8 capsule_run_log_get()

```
int capsule_run_log_get (
            uint64_t run_id,
            CapsuleRunRecord * out )
```

capsule_run_log_get - Get a run record by ID

Parameters

run↩ _id	Run ID to retrieve
out	Output record

Returns

0 on success, -1 if not found

References CAPSULE_MAX_RUN_RECORDS, run_id_counter, and run_log.

7.13.3.9 capsule_run_log_init()

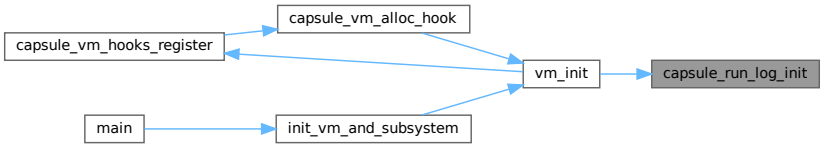
```
void capsule_run_log_init (
            void  )
```

capsule_run_log_init - Initialize run log

References CapsuleRunRecord::capsule_hash, CapsuleRunRecord::capsule_id, CAPSULE_MAX_RUN_RECORDS, CapsuleRunRecord::ended_ns, CapsuleRunRecord::flags, CapsuleRunRecord::post_dict_hash, CapsuleRunRecord::pre_dict_hash, CapsuleRunRecord::reserved, CapsuleRunRecord::result_code, CapsuleRunRecord::run_id, run_id_counter, run_log, run_log_count, run_log_head, CapsuleRunRecord::started_ns, and CapsuleRunRecord::vm_id.

Referenced by vm_init().

Here is the caller graph for this function:



7.13.3.10 capsule_run_log_record()

```
uint64_t capsule_run_log_record (
            const CapsuleRunRecord * record )
```

capsule_run_log_record - Log a run record

Parameters

record	Record to log

Returns

Run ID assigned, or 0 on failure

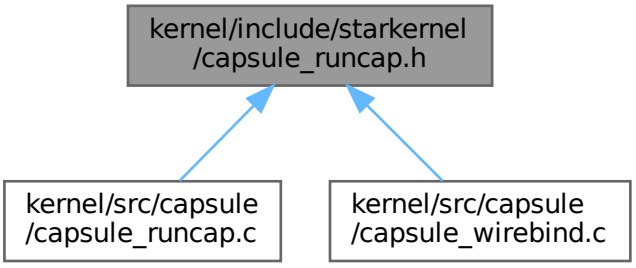
References CAPSULE_MAX_RUN_RECORDS, CapsuleRunRecord::run_id, run_id_counter, run_log, run_log_count, and run_log_head.

Referenced by capsule_run_experiment().

Here is the caller graph for this function:

7.14 kernel/include/starkernel/capsule_runcap.h File Reference

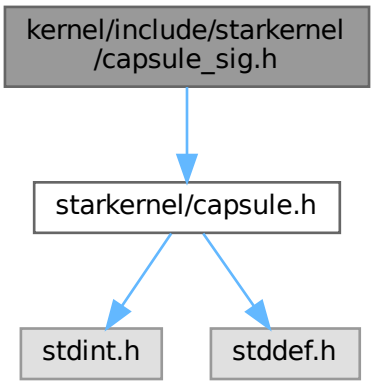
This graph shows which files directly or indirectly include this file:



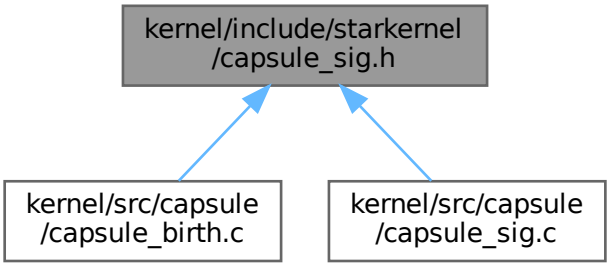
7.15 kernel/include/starkernel/capsule_sig.h File Reference

```
#include "starkernel/capsule.h"
```

Include dependency graph for capsule_sig.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define CAPSULE_SIGNING_CERT_NAME "pki:starshipos-intermediate.der"

Enumerations

- enum CapsuleSigResult { CAPSULE_SIG_OK = 0 , CAPSULE_SIG_MISSING , CAPSULE_SIG_INVALID , CAPSULE_SIG_NO_ROOT_KEY }

Functions

- CapsuleSigResult capsule_verify_signature (const CapsuleDesc *descs, const CapsuleNameEntry *names, const CapsuleSigEntry *sigs, const uint8_t *arena_base, uint32_t desc_count, int index)
- const char * capsule_sig_result_str (CapsuleSigResult result)

7.15.1 Macro Definition Documentation

7.15.1.1 CAPSULE_SIGNING_CERT_NAME

```
#define CAPSULE_SIGNING_CERT_NAME "pki:starshipos-intermediate.der"
```

7.15.2 Enumeration Type Documentation

7.15.2.1 CapsuleSigResult

```
enum CapsuleSigResult
```

Enumerator

CAPSULE_SIG_OK	
CAPSULE_SIG_MISSING	
CAPSULE_SIG_INVALID	
CAPSULE_SIG_NO_ROOT_KEY	

7.15.3 Function Documentation

7.15.3.1 capsule_sig_result_str()

```
const char * capsule_sig_result_str (
            CapsuleSigResult result )
```

References CAPSULE_SIG_INVALID, CAPSULE_SIG_MISSING, CAPSULE_SIG_NO_ROOT_KEY, and CAPSULE_SIG_OK.

Referenced by capsule_birth_baby(), capsule_birth_mama(), and capsule_run_experiment().

Here is the caller graph for this function:

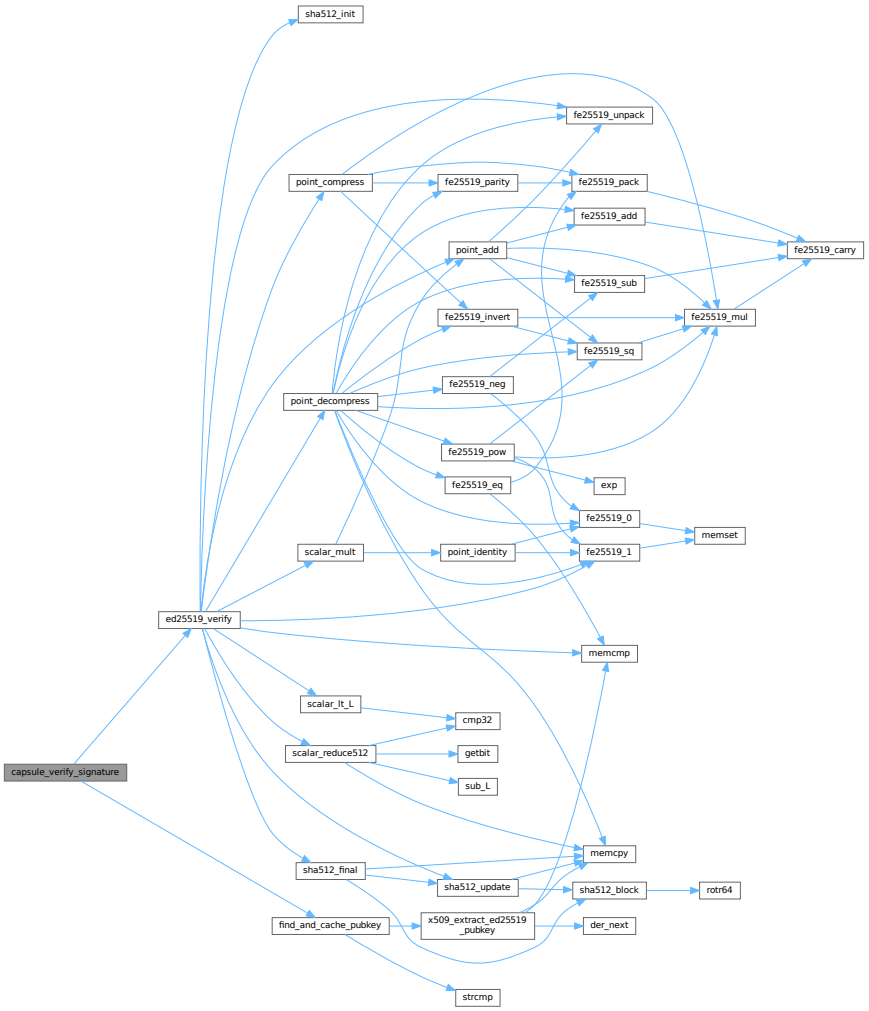
7.15.3.2 capsule_verify_signature()

```
CapsuleSigResult capsule_verify_signature (
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const CapsuleSigEntry * sigs,
            const uint8_t * arena_base,
            uint32_t desc_count,
            int index )
```

References cached_pubkey, CAPSULE_SIG_INVALID, CAPSULE_SIG_MISSING, CAPSULE_SIG_NO_ROOT_KEY, CAPSULE_SIG_OK, CapsuleDesc, ed25519_verify(), find_and_cache_pubkey(), and CapsuleSigEntry::sig.

Referenced by capsule_birth_baby(), capsule_birth_mama(), and capsule_run_experiment().

Here is the call graph for this function:

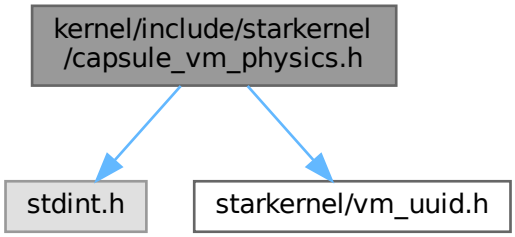


Here is the caller graph for this function:

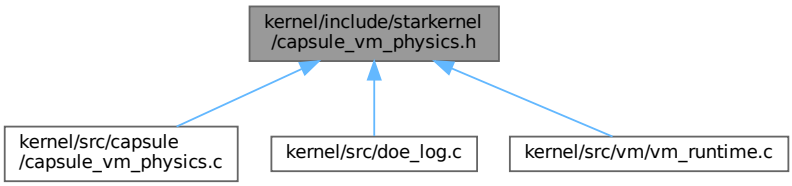
7.16 kernel/include/starkernel/capsule_vm_physics.h File Reference

```
#include <stdint.h>
#include "starkernel/vm_uuid.h"
```

Include dependency graph for capsule_vm_physics.h:



This graph shows which files directly or indirectly include this file:



Functions

- void vm_physics_init (VMUuid vm_id)

  capsule_vm_physics.h - Dynamic VM Fleet Physics
- void vm_physics_retire (VMUuid vm_id)

  vm_physics_retire - Remove a killed VM, returning its heat to Hera
- void vm_physics_touch (VMUuid vm_id)

  vm_physics_touch - Record real dispatch activity against a VM
- void vm_physics_tick (uint64_t now_ns)

  vm_physics_tick - Heartbeat-gated inference pass
- void vm_physics_heartbeat_tick (uint64_t now_ns)

  vm_physics_heartbeat_tick - Fleet-wide heartbeat, gates vm_physics_tick()
- uint64_t vm_physics_fleet_heat_sum (void)

  vm_physics_fleet_heat_sum - Sum of execution_heat_q48 over all LIVE VMs
- uint64_t vm_physics_heat_of (VMUuid vm_id)

  vm_physics_heat_of - Current execution_heat_q48 for one VM
- int vm_physics_conserved (void)

  vm_physics_conserved - Conservation check
- void vm_physics_status (void)

  vm_physics_status - Print a diagnostic status report

7.16.1 Function Documentation

7.16.1.1 vm_physics_conserved()

```
int vm_physics_conserved (
            void  )
```

vm_physics_conserved - Conservation check

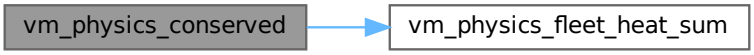
Returns

1 if $|$vm_physics_fleet_heat_sum() - Q.1$| <$ epsilon, else 0

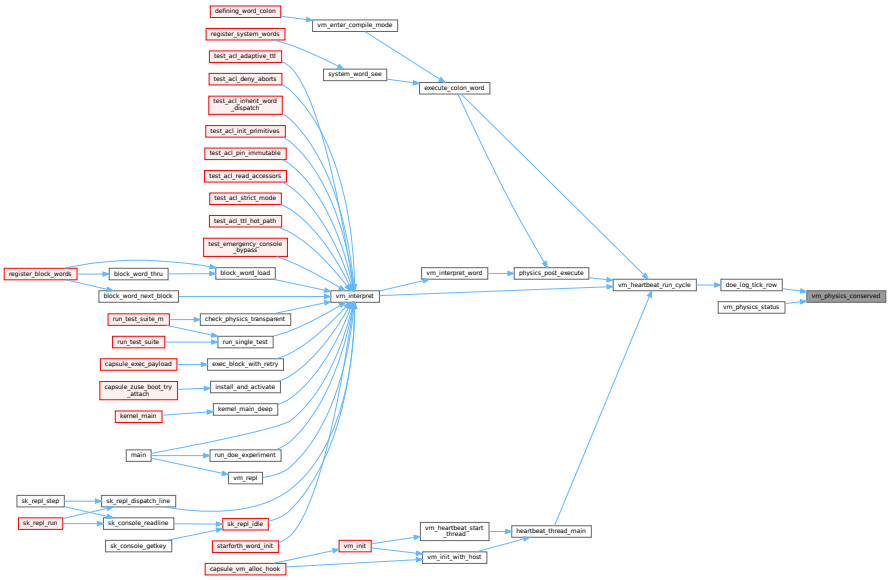
References Q48_ONE, VM_PHYSICS_EPSILON_Q48, and vm_physics_fleet_heat_sum().

Referenced by doe_log_tick_row(), and vm_physics_status().

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.1.2 vm_physics_fleet_heat_sum()

```
uint64_t vm_physics_fleet_heat_sum (
            void  )
```

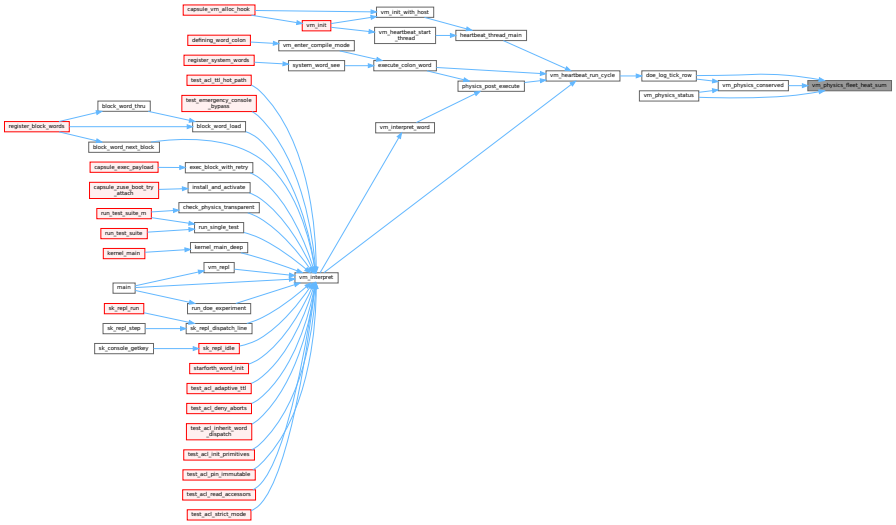
vm_physics_fleet_heat_sum - Sum of execution_heat_q48 over all LIVE VMs

Diagnostic / verification primitive. Should always read Q.1 (65536) if the conservation invariant holds.

References VMPhysics::execution_heat_q48, VMPhysics::is_live, vm_physics_node::next, vm_physics_node::physics, and vm_physics_head.

Referenced by doe_log_tick_row(), vm_physics_conserved(), and vm_physics_status().

Here is the caller graph for this function:



7.16.1.3 vm_physics_heartbeat_tick()

```
void vm_physics_heartbeat_tick (
            uint64_t now_ns )
```

vm_physics_heartbeat_tick - Fleet-wide heartbeat, gates vm_physics_tick()

Call from every VM's own vm_tick(), not just Hera's. Increments a single fleet-wide tick counter (distinct from any VM's own per-VM HeartbeatState.tick_count) and calls vm_physics_tick() once that shared counter has advanced by HEARTBEAT_INFERENCE_FREQUENCY since the last fit.

Fixes a real defect (VM-FLEET-ATTRACTOR-DESIGN-20260705.md rev k): gating this on Hera's own tick_count meant the inference pass almost never fired, because Hera-as-orchestrator mostly blocks on VM-EXEC/ VM-CALL dispatch – work that accrues to the *target* VM's own tick count, not hers. VM-PHYSICS-DYNAMIC-FLEET-DESIGN-20260705.md already specified the fix's direction: fleet_transfer_slope_q48 must be inferred "from aggregate fleet statistics... not one per VM" – the same principle applies to the readiness signal that gates computing it, not just the regression's own input trajectory.

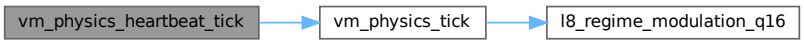
Parameters

now_ns	Current monotonic time in nanoseconds (of whichever VM is calling; unused by vm_physics_tick() today)

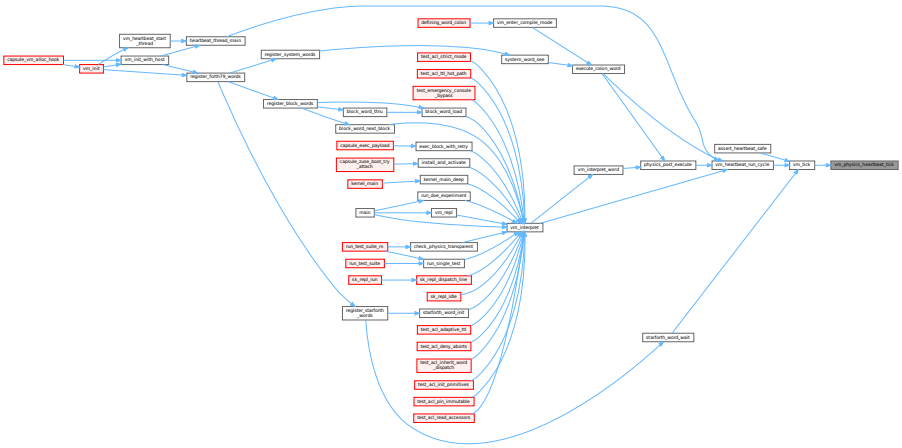
References fleet_heartbeat_tick_count, fleet_last_capacity_tick, STADIUM_CAPACITY_TICK, and vm_physics_tick().

Referenced by vm_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.1.4 vm_physics_heat_of()

```
uint64_t vm_physics_heat_of (
            VMUuid vm_id )
```

vm_physics_heat_of - Current execution_heat_q48 for one VM

Transparency primitive (VM-FLEET-ATTRACTOR-DESIGN-20260705.md): the fleet-wide conservation invariant always reads Q.1 by construction, so it carries no information about how heat is actually distributed among live VMs. This is the per-VM fact that sum alone can't provide.

Parameters

vm↩ _id	Registry VM ID

Returns

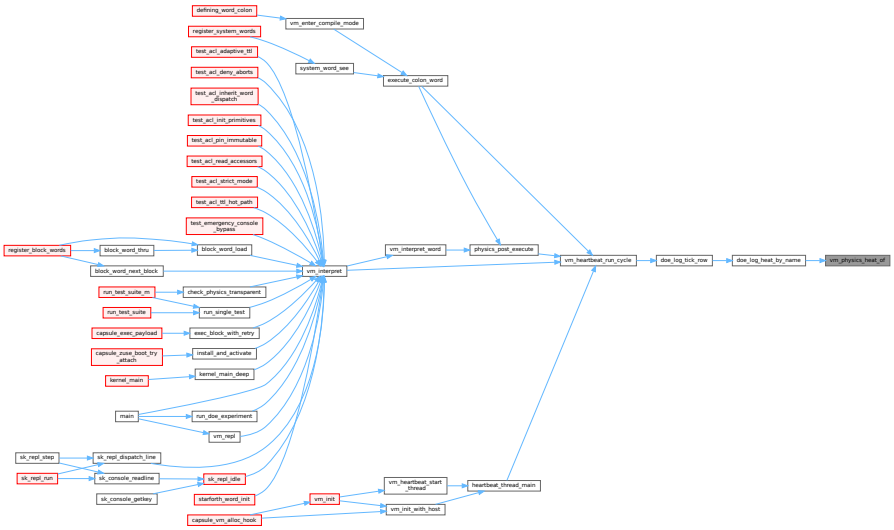
execution_heat_q48, or 0 if vm_id is unknown or not live

References VMPhysics::execution_heat_q48, VMPhysics::is_live, vm_physics_node::physics, and vm_physics_find().

Referenced by doe_log_heat_by_name().

Here is the call graph for this function:

Here is the caller graph for this function:



7.16.1.5 vm_physics_init()

```
void vm_physics_init (
            VMUuid vm_id )
```

capsule_vm_physics.h - Dynamic VM Fleet Physics

Structural parallel to the word-level physics engine (execution heat, decay, rolling-window history, regression-inferred slope) applied to VMs instead of dictionary words. Kernel-only: no hosted-build equivalent, since the hosted build has no multi-VM fleet.

sum(execution_heat_q48 for all LIVE VMs) == Q.1 is a conservation invariant, held by construction: every state change is a balanced transfer (see design doc VM-PHYSICS-DYNAMIC-FLEET-DESIGN-20260705.md).

This is a passive observer of VM activity, never a driver of it — there is no scheduler here. Recording happens only at existing dispatch points (BIRTH, KILL, VM-EXEC, VM-CALL, VM-STEP). vm_physics_init - Register a newly-born VM at zero heat

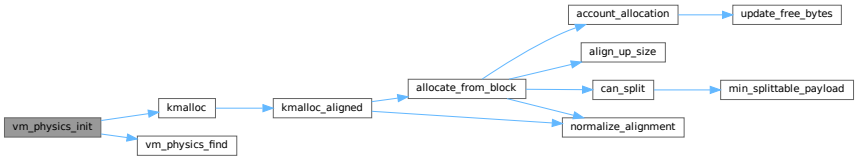
Called from mama_word_birth once a VM reaches VM_STATE_LIVE. Starts at execution_heat_q48 = 0, so conservation holds trivially — no fan-out from existing VMs is needed.

Parameters

vm↩_id	Registry VM ID assigned at birth

References VMPhysics::execution_heat_q48, VMPhysics::is_live, kmalloc(), VMPhysics::last_active_tick, vm_physics_node::next, vm_physics_node::physics, Q48_ONE, VMPhysics::touched, vm_physics_node::vm_id, vm_physics_find(), and vm_physics_head.

Here is the call graph for this function:



7.16.1.6 vm_physics_retire()

```
void vm_physics_retire (
            VMUuid vm_id )
```

vm_physics_retire - Remove a killed VM, returning its heat to Hera

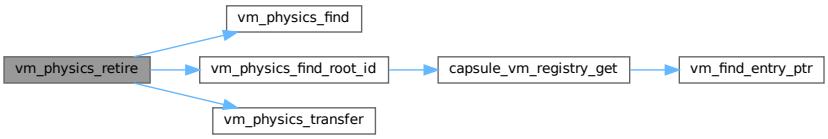
Called from mama_word_kill before the registry entry is torn down. The dying VM's entire remaining heat is transferred up its parent_vm_id chain (VMRegistryEntry, capsule_run.h) to the fleet's single structural root, Hera – no division, no per-survivor weighting. The chain walk tolerates already-dead intermediate parents (a parent's own parent_vm_id was set once at its birth and never rewritten, so the walk continues through it).

Parameters

vm↩_id	Registry VM ID of the VM being killed

References VMPhysics::execution_heat_q48, VMPhysics::is_live, vm_physics_node::physics, vm_physics_find(), vm_physics_find_root_id(), and vm_physics_transfer().

Here is the call graph for this function:



7.16.1.7 vm_physics_status()

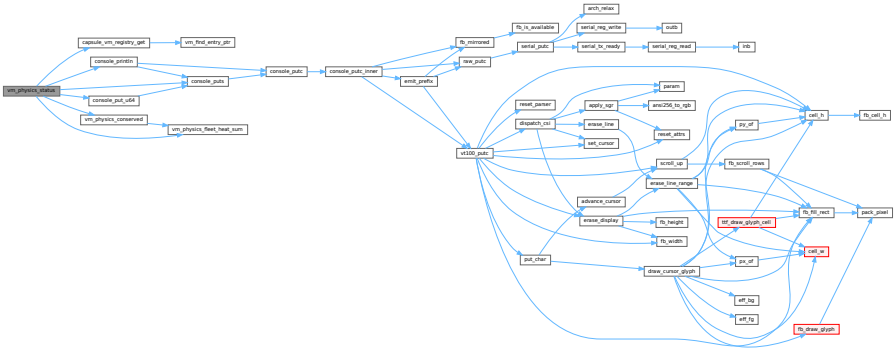
```
void vm_physics_status (
            void  )
```

vm_physics_status - Print a diagnostic status report

Replaces K-STATUS/VM-STATUS. Reports fleet heat sum, conservation verdict, and the current fleet-wide inferred slope / fit quality / warm-up state – the fleet-level analog of K-STATUS's per-VM report, adapted to a mechanism with no fixed VM count to enumerate.

References capsule_vm_registry_get(), console_println(), console_put_u64(), console_puts(), entry, VMPhysics::execution_heat_q48, fleet_transfer_slope_q48, fleet_window, VMPhysics::is_live, VMFleetWindow::is_warm, vm_physics_node::next, vm_physics_node::physics, slope_fit_quality_q48, vm_physics_node::vm_id, vm_physics_conserved(), vm_physics_fleet_heat_sum(), and vm_physics_head.

Here is the call graph for this function:



7.16.1.8 vm_physics_tick()

```
void vm_physics_tick (
            uint64_t now_ns )
```

vm_physics_tick - Heartbeat-gated inference pass

Mirrors vm_tick_inference_engine: when the fleet touch-history window is warm, replays it to re-fit fleet_transfer_↩ slope_q48 via the same closed-form log-linear OLS regression used for word-level decay. Skips the update (does not substitute a default) when unwarmed or when fit quality is not usable.

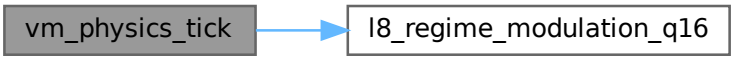
Parameters

now_ns	Current monotonic time in nanoseconds

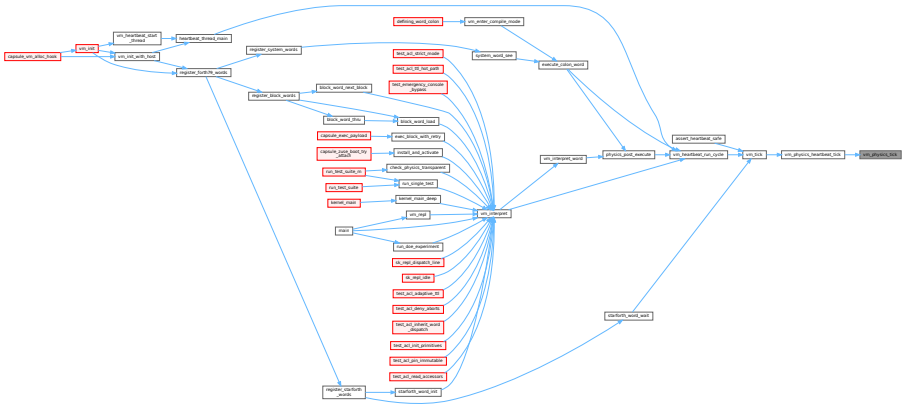
References VMFleetTouchSample::amount, VMFleetTouchSample::clamped, VMFleetTouchSample::elapsed_ticks, fleet_transfer_slope_q48, fleet_window, VMFleetWindow::is_warm, l8_regime_modulation_q16(), slope_fit_quality_q48, VMFleetWindow::touch_samples, VM_FLEET_WINDOW_DEPTH, and VM_PHYSICS_MIN_INFORMATIVE_SAMPLES.

Referenced by vm_physics_heartbeat_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.1.9 vm_physics_touch()

```
void vm_physics_touch (
            VMUuid vm_id )
```

vm_physics_touch - Record real dispatch activity against a VM

Called from the three existing dispatch primitives (VM-EXEC, VM-CALL, VM-STEP) at the point they've already confirmed the target VM is live. Pulls heat from the rest of the fleet toward the touched VM (amount = elapsed_ticks * fleet_transfer_slope_q48 >> 16) and appends vm_id to the fleet's rolling touch-history window.

Restated on the virtual tick (FABRIC-0.md item 2.1, 2026-08-04): no longer takes a wall-clock timestamp. Reads fleet_heartbeat_tick_count internally, which is execution-paced (advanced once per vm_tick() call, see vm_↩ physics_heartbeat_tick), so the transfer this produces is a deterministic function of the execution stream, not of wall time.

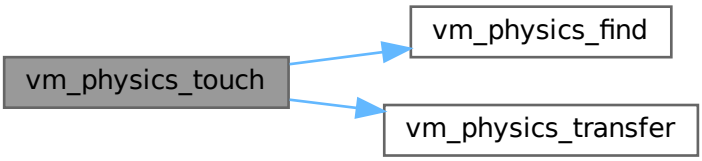
Parameters

vm↩ _id	Registry VM ID being dispatched to

References VMFleetTouchSample::amount, VMFleetTouchSample::clamped, VMFleetWindow::count, VMFleetTouchSample::elapsed, VMPhysics::execution_heat_q48, fleet_heartbeat_tick_count, fleet_transfer_slope_q48, fleet_window, VMFleetWindow::head, VMPhysics::is_live, VMFleetWindow::is_warm, VMPhysics::last_active_tick, vm_physics_node::next, vm_physics_node::physics,

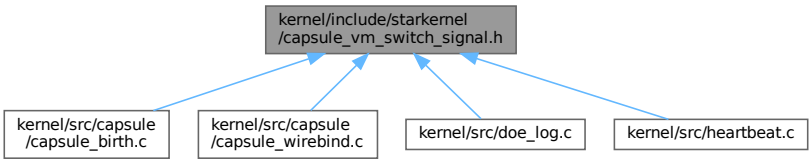
VMFleetWindow::touch_samples, VMPhysics::touched, VM_FLEET_WINDOW_DEPTH, vm_physics_find(), vm_physics_head, and vm_physics_transfer().

Here is the call graph for this function:



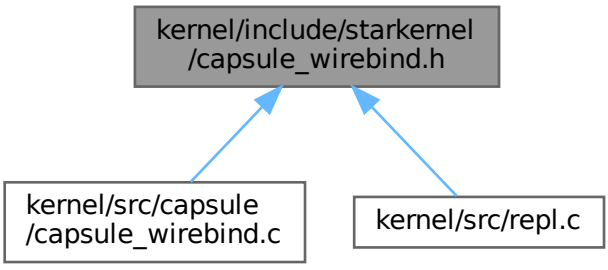
7.17 kernel/include/starkernel/capsule_vm_switch_signal.h File Reference

This graph shows which files directly or indirectly include this file:



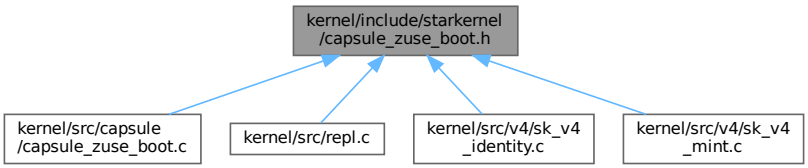
7.18 kernel/include/starkernel/capsule_wirebind.h File Reference

This graph shows which files directly or indirectly include this file:



7.19 kernel/include/starkernel/capsule_zuse_boot.h File Reference

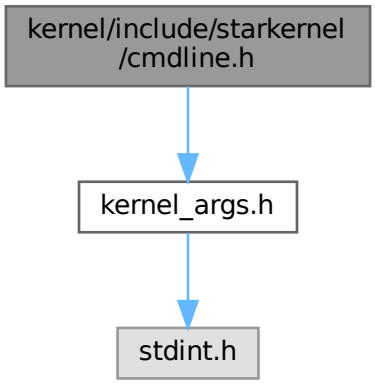
This graph shows which files directly or indirectly include this file:



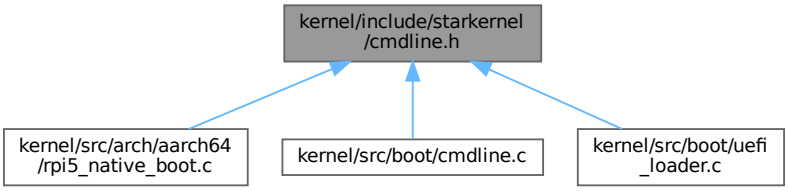
7.20 kernel/include/starkernel/cmdline.h File Reference

```
#include "kernel_args.h"
```

Include dependency graph for cmdline.h:



This graph shows which files directly or indirectly include this file:



Functions

- void cmdline_parse_ascii (const char *cmdline, KernelArgs *args)

  Parse a NUL-terminated ASCII command-line string into KernelArgs.
- uint64_t cmdline_parse_size (const char *s)

  Parse a decimal integer with optional K/M/G size suffix.

7.20.1 Function Documentation

7.20.1.1 cmdline_parse_ascii()

```
void cmdline_parse_ascii (
            const char * cmdline,
            KernelArgs * args )
```

Parse a NUL-terminated ASCII command-line string into KernelArgs.

Populates args from space-separated tokens in cmdline. Always initialises all fields to their defaults first (so partial or empty command lines produce safe values):

- stack_size = 0 (use BSS static stack)
- heap_size = 0 (use built-in kmalloc arena)
- run_doe = KARGS_DEFAULT_RUN_DOE
- log_level = KARGS_DEFAULT_LOG_LEVEL
- raw[0] = '\0'

Recognised flags:

- --doe — sets args->run_doe = 1.
- --log-level=<level> — sets args->log_level via parse_log_level().
- --stack=<size> — sets args->stack_size via cmdline_parse_size().
- --heap=<size> — sets args->heap_size via cmdline_parse_size().

Unknown flags are silently ignored for forward-compatibility with flags introduced in future kernel versions. The raw command-line string is copied verbatim into args->raw (truncated to KERNEL_ARGS_CMDLINE_MAX - 1) for diagnostic logging.

Called from efi_main() (in uefi_loader.c) after assembling the ASCII command line from the three priority sources: NVRAM variable, starforth.cfg on the ESP, and EFI LoadOptions.

Parameters

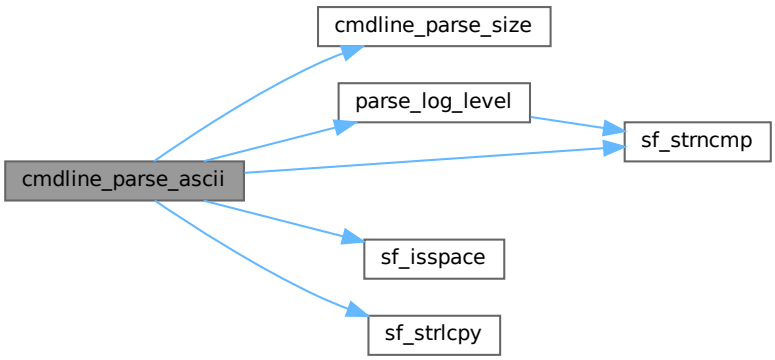
cmdline	NUL-terminated ASCII command line; may be NULL or empty.
args	Output KernelArgs structure; must not be NULL.

References cmdline_parse_size(), KernelArgs::heap_size, KARGS_DEFAULT_LOG_LEVEL, KARGS_DEFAULT_RUN_DOE,

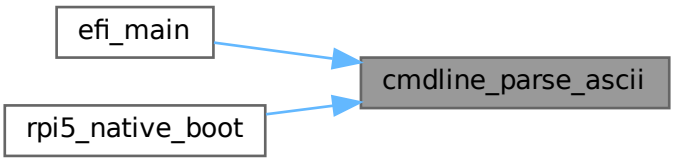
KARGS_SYSTEM_DEFAULT, KARGS_SYSTEM_V3, KARGS_SYSTEM_V4, KERNEL_ARGS_CMDLINE_MAX, KernelArgs::log_level, parse_log_level(), KernelArgs::raw, KernelArgs::run_doe, KernelArgs::screen_boot, sf_isspace(), sf_strlcpy(), sf_strncmp(), KernelArgs::stack_size, and KernelArgs::system.

Referenced by efi_main(), and rpi5_native_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.20.1.2 cmdline_parse_size()

```
uint64_t cmdline_parse_size (
            const char * s )
```

Parse a decimal integer with optional K/M/G size suffix.

Reads a non-negative decimal integer from `s` and multiplies it by:

- 1024 for 'K' or 'k' (kibibytes)
- 1024^2 for 'M' or 'm' (mebibytes)

- 1024^3 for 'G' or 'g' (gibibytes)
- 1 for any other or absent suffix

Returns 0 if `s` is NULL, empty, or does not begin with a digit. Overflow is not checked; caller must validate the resulting value is sensible for its intended use (stack size, heap size, etc.).

Used by `cmdline_parse_ascii()` to decode the `--stack=<size>` and `--heap=<size>` arguments from the boot command line.

Parameters

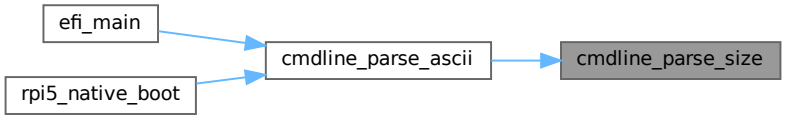
s	NUL-terminated string beginning with a decimal integer and optionally followed by a size suffix character.

Returns

Parsed byte count (0 on parse failure or zero input).

Referenced by cmdline_parse_ascii().

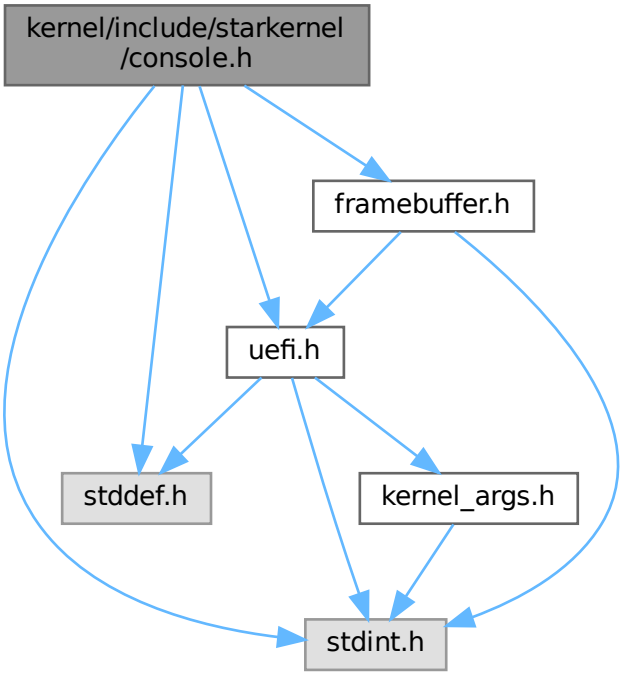
Here is the caller graph for this function:



7.21 kernel/include/starkernel/console.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "uefi.h"
#include "framebuffer.h"
```

Include dependency graph for console.h:



This graph shows which files directly or indirectly include this file:

Typedefs

- typedef const char ∗(∗ console_user_prefix_fn) (void)

  FABRIC-3.md SXXV follow-up (2026-09-13): the line prefix used to be a bare "[VMName] ", ambiguous between a console-proxy VM and the actual identity VM behind it (e.g.

Functions

- void console_attach_serial (const void ∗dtb)

  console.h - Serial console + framebuffer VT100 interface for StarKernel
- void console_init (void)

  Initialize serial console (UART).
- void console_describe_serial (void)

  Print one line saying which serial line the console found, or that there is none.
- void console_fb_init (const FramebufferInfo ∗info, FbPixelFormat fmt)

  Initialize the framebuffer VT100 terminal.
- void console_fb_enable_ttf (void)

FABRIC-0.md item 4.4j: switch the framebuffer console's glyph backend from font_8x16.c to TTF-TEXT's rasterizer.

- void console_fb_mirror (int on)
- void console_fb_clear (void)
- void console_fb_scroll_back (uint32_t n)

  FABRIC-0.md item 4.4q: thin wrappers over vt100_scroll_back()/ vt100_scroll_fwd() – see those functions' doc comments for the full contract.
- void console_fb_scroll_fwd (uint32_t n)
- void console_fb_toggle_graphics (void)

  FABRIC-0.md item 4.4y-revised: thin wrapper over vt100_toggle_graphics() – see that function's doc comment for the full contract (the Alt+TAB graphics/text state machine).
- void console_fb_draw_cursor (void)

  Thin wrapper over vt100_draw_cursor() – see that function's doc comment for the full contract (a static block cursor at the terminal's current position).
- void console_fb_erase_cursor (void)

  Thin wrapper over vt100_erase_cursor().
- void console_putc (char c)

  Write a single character to serial console.
- void console_puts (const char *s)

  Write a null-terminated string to serial console.
- void console_println (const char *s)

  Write a string with newline to serial console.
- void console_ensure_line_start (void)

  console_ensure_line_start - Make sure the next real output starts at the beginning of a fresh output line, closing off whatever is currently mid-line (e.g.
- void console_cancel_deferred_line_start (void)

  console_cancel_deferred_line_start - Discard a pending deferred newline from console_ensure_line_start() without ever emitting it.
- uint64_t console_tx_count (void)

  console_tx_count - Monotonic count of console_putc() calls delivered to either output (serial and/or framebuffer).
- int console_getc (void)

  Read a single character from serial console (non-blocking) Returns -1 if no character available.
- int console_poll (void)

  Check if character is available for reading.
- void console_set_vm_name (const char *name)

  Set the active VM name shown as [Name] prefix on each output line.
- const char * console_get_vm_name (void)
- void console_save_vm_name (char *out, size_t cap)

  console_save_vm_name - Copy the current active-VM name into the caller's own buffer, for a later console_set_vm_name() restore.
- void console_set_user_prefix_provider (console_user_prefix_fn fn)

Variables

- volatile const char * g_sk_fault_word

  interrupts.c (aarch64) - Exception vector setup and common handler

7.21.1 Typedef Documentation

7.21.1.1 console_user_prefix_fn

```
typedef const char *(* console_user_prefix_fn) (void)
```

FABRIC-3.md SXXV follow-up (2026-09-13): the line prefix used to be a bare "[VMName] ", ambiguous between a console-proxy VM and the actual identity VM behind it (e.g.

"rajames" the console vs. "rajames∼user" the real WIREBIND-restricted identity) – confirmed live to cause real confusion during testing. Unified to "[user@VMName] " when a user context is available, still "[VMName] " when not (unauthenticated/ pre-login, unchanged from today).

console.c is a clean HAL module with no dependency on capsule/WIREBIND logic (zuse_session, capsule_wirebind_attached_userna – pulling either in directly here would be a real layering violation, not just a style preference. This callback lets repl.c (which already computes exactly this for sk_print_prompt()) supply the "user" half without console.c knowing anything about VMs, sessions, or WIREBIND. Returns NULL/empty for "no user context" (falls back to the bare "[VMName] " form); the returned pointer must remain valid until the next call (matches console_get_vm_name()'s own single-buffer convention).

7.21.2 Function Documentation

7.21.2.1 console_attach_serial()

```
void console_attach_serial (
            const void * dtb )
```

console.h - Serial console + framebuffer VT100 interface for StarKernel

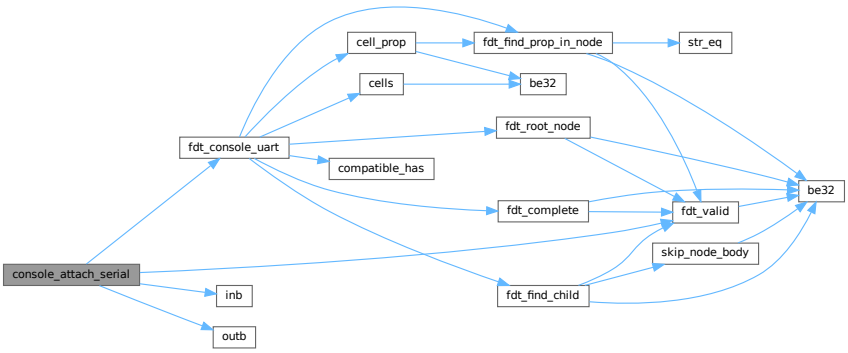
Output policy:

- Serial UART is always active (initialized by console_init).
- When console_fb_init() has been called and the framebuffer is ready, every character is also rendered through the VT100 terminal on screen.
- Both outputs are always live simultaneously; serial cannot be disabled. Find the board's serial console line. Must be called before anything is printed: until it is, and on a board with no such line, the console writes to no serial port and reads from none. On a PC it is COM1, if a UART answers there; elsewhere it is the UART the devicetree names as the console (/chosen/stdout-path); with no devicetree at all it is the UART of QEMU's virt machine. `dtb` may be NULL.

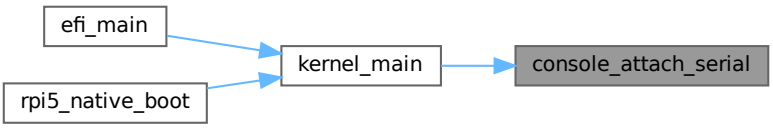
References fdt_uart_t::base, fdt_console_uart(), FDT_UART_PL011, fdt_valid(), g_serial_base, g_serial_from_dt, g_serial_kind, g_serial_shift, g_serial_width, inb(), fdt_uart_t::io_width, fdt_uart_t::kind, outb(), fdt_uart_t::reg_shift, SERIAL_MMIO16550, SERIAL_NONE, SERIAL_PL011, and SERIAL_PORT16550.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.2 console_cancel_deferred_line_start()

```
void console_cancel_deferred_line_start (
            void  )
```

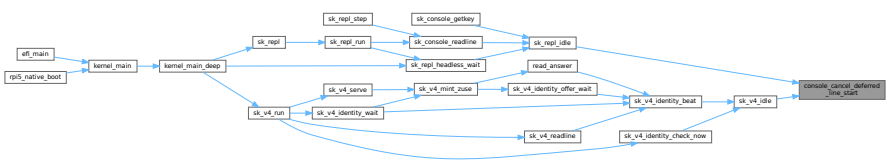
console_cancel_deferred_line_start - Discard a pending deferred newline from console_ensure_line_start() without ever emitting it.

No-op if no newline is currently deferred. Callers that speculatively deferred a line break before checking whether they actually have anything to print (sk_repl_idle()'s idle-beat check being the motivating case, FABRIC-2.md ?I.9) call this when the check comes back negative, so the console is left exactly as it was – no stray newline, no phantom blank line.

References g_pending_line_close.

Referenced by sk_repl_idle(), and sk_v4_idle().

Here is the caller graph for this function:



7.21.2.3 console_describe_serial()

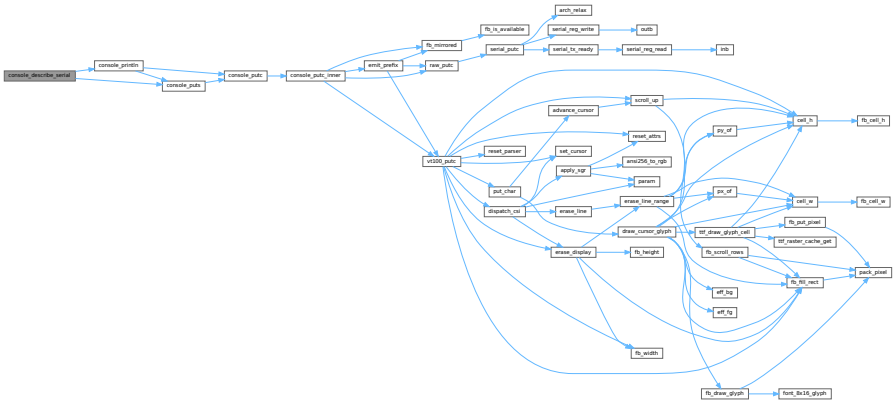
```
void console_describe_serial (
            void  )
```

Print one line saying which serial line the console found, or that there is none.

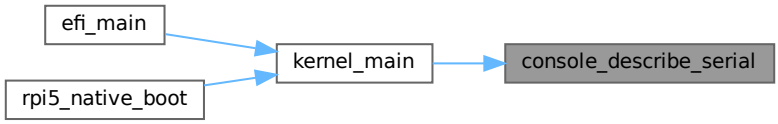
References console_println(), console_puts(), g_serial_base, g_serial_from_dt, g_serial_kind, SERIAL_NONE, SERIAL_PL011, and SERIAL_PORT16550.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.4 console_ensure_line_start()

```
void console_ensure_line_start (
            void  )
```

console_ensure_line_start - Make sure the *next real output* starts at the beginning of a fresh output line, closing off whatever is currently mid-line (e.g.

a dangling prompt). No-op if already at line start.

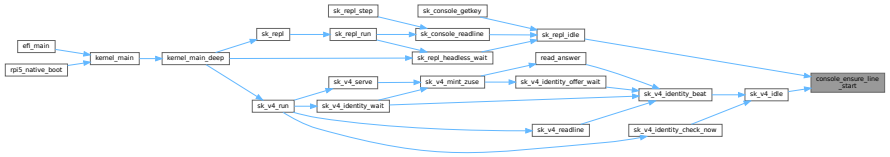
FABRIC-2.md ?I.9 fix, 2026-09-05: the newline is DEFERRED, not emitted immediately – it only actually reaches the console on the next real console_putc() call, and is silently dropped (never emitted at all) if

console_cancel_deferred_line_start() is called first instead. Before this fix, an immediate, unconditional newline here meant any caller invoking this function "just in case" (sk_repl_idle() being the one real caller) would visibly snap a bare, unfinished prompt line to a fresh blank line even when nothing was actually about to be printed – indistinguishable from Enter having already been pressed at that prompt. Deferring means a caller that turns out to have nothing to print can cancel cleanly, with zero visible effect, while a caller that does print gets the correct "close the old line first" behavior for free, still counted by console_tx_count() as real output (unlike the old always-silent inner form) since it is realized through the normal console_putc() path.

References g_line_start, and g_pending_line_close.

Referenced by sk_repl_idle(), and sk_v4_idle().

Here is the caller graph for this function:



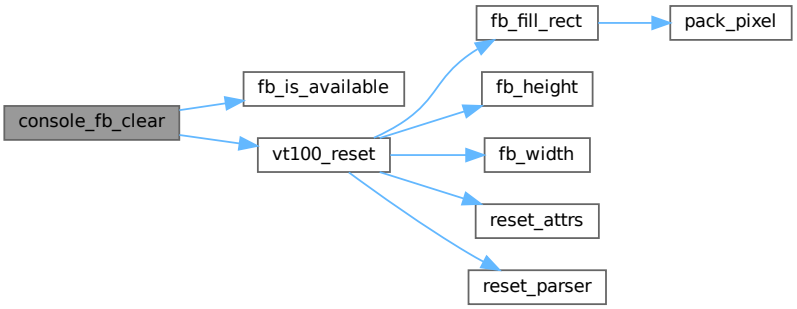
7.21.2.5 console_fb_clear()

```
void console_fb_clear (
            void  )
```

References fb_is_available(), and vt100_reset().

Referenced by sk_v4_drive_left(), sk_v4_fixed_screen(), and sk_v4_session_end().

Here is the call graph for this function:



Here is the caller graph for this function:

7.21.2.6 console_fb_draw_cursor()

```
void console_fb_draw_cursor (
            void  )
```

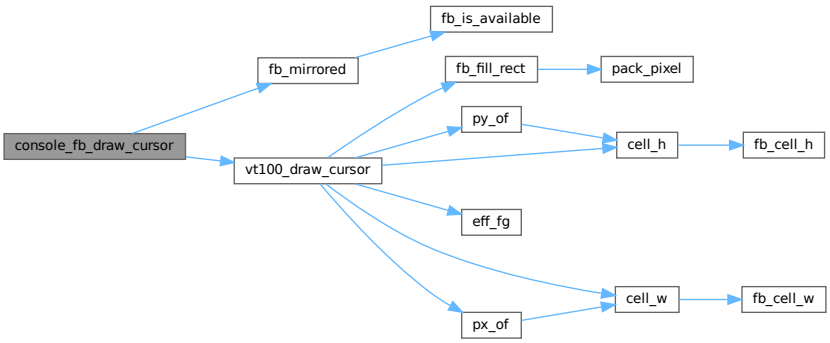
Thin wrapper over vt100_draw_cursor() – see that function's doc comment for the full contract (a static block cursor at the terminal's current position).

No-op if the framebuffer console was never initialized.

References fb_mirrored(), and vt100_draw_cursor().

Referenced by sk_console_readline(), sk_cursor_show(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:

7.21.2.7 console_fb_enable_ttf()

```
void console_fb_enable_ttf (
            void  )
```

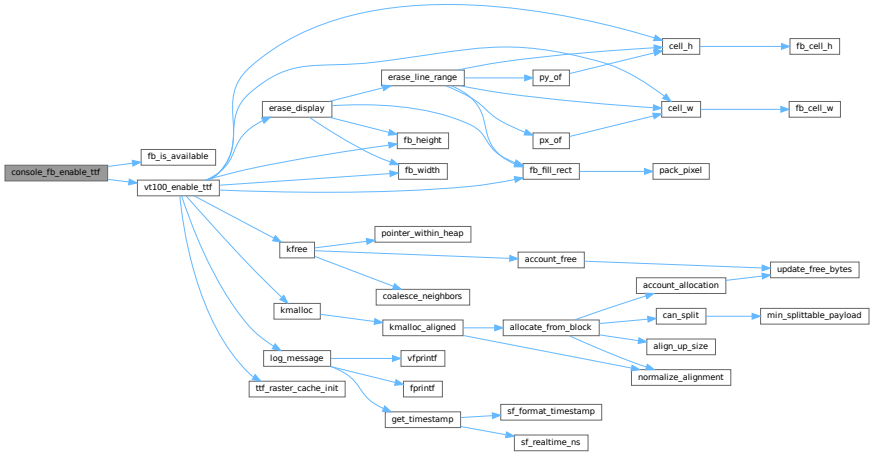
FABRIC-0.md item 4.4j: switch the framebuffer console's glyph backend from font_8x16.c to TTF-TEXT's rasterizer.

Thin wrapper over vt100_enable_ttf() – see that function's doc comment for the full contract (lazy font load, cell-geometry/cols/rows recompute, screen clear, one-shot). No-op if the framebuffer console was never initialized (console_fb_init() not called, or it no-op'd on a NULL framebuffer).

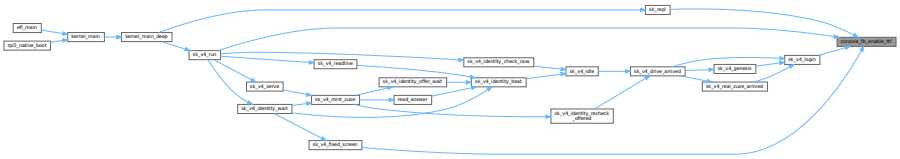
References fb_is_available(), and vt100_enable_ttf().

Referenced by sk_repl(), sk_v4_fixed_screen(), sk_v4_login(), and sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.8 console_fb_erase_cursor()

```
void console_fb_erase_cursor (
            void  )
```

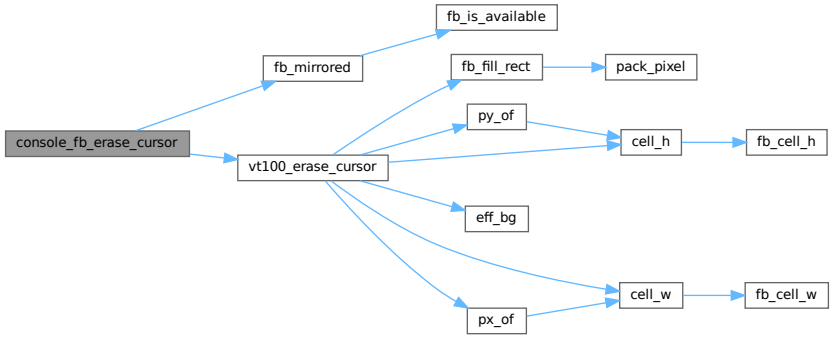
Thin wrapper over vt100_erase_cursor().

No-op if the framebuffer console was never initialized.

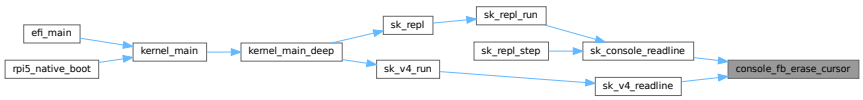
References fb_mirrored(), and vt100_erase_cursor().

Referenced by sk_console_readline(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.9 console_fb_init()

```
void console_fb_init (
            const FramebufferInfo * info,
            FbPixelFormat fmt )
```

Initialize the framebuffer VT100 terminal.

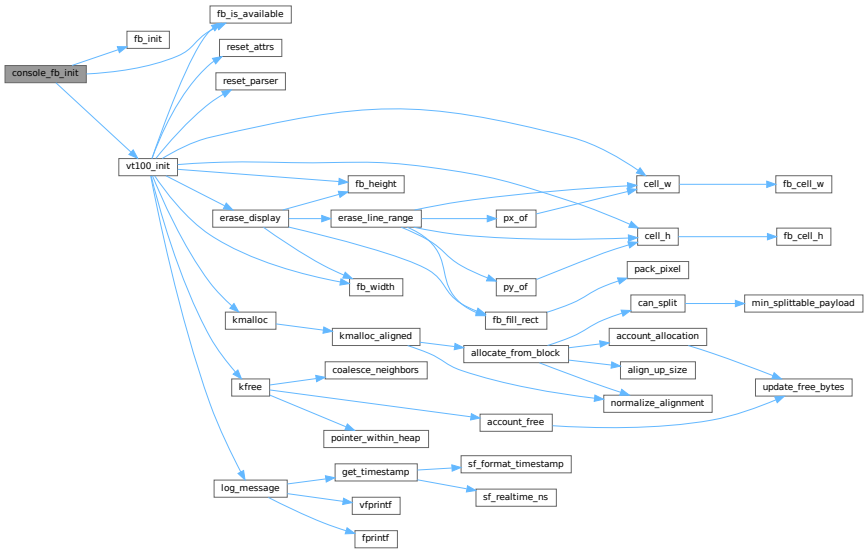
Call after UEFI boot services have been exited and the GOP framebuffer address is known (from BootInfo). Safe to call with info==NULL (no-op). fmt: FB_PIXEL_BGRX32 is correct for most QEMU / real hardware GOP.

Initialize the framebuffer VT100 terminal.

Safe to call with info == NULL or when the framebuffer base is zero (no-op). After this call every console_putc / console_puts output is mirrored to the screen; the serial UART continues to function concurrently.

References FramebufferInfo::base, fb_init(), fb_is_available(), and vt100_init().

Here is the call graph for this function:



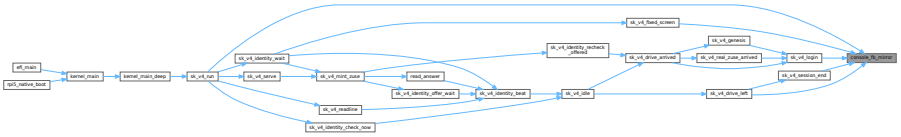
7.21.2.10 console_fb_mirror()

```
void console_fb_mirror (
            int on )
```

References g_fb_mirror.

Referenced by sk_v4_drive_left(), sk_v4_fixed_screen(), sk_v4_login(), sk_v4_run(), and sk_v4_session_end().

Here is the caller graph for this function:



7.21.2.11 console_fb_scroll_back()

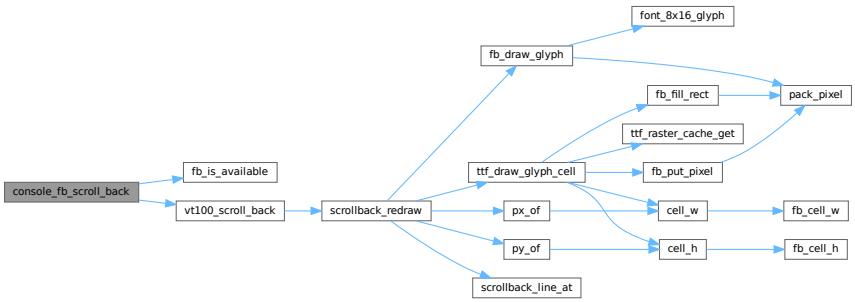
```
void console_fb_scroll_back (
            uint32_t n )
```

FABRIC-0.md item 4.4q: thin wrappers over vt100_scroll_back()/ vt100_scroll_fwd() – see those functions' doc comments for the full contract.

No-op if the framebuffer console was never initialized.

References fb_is_available(), and vt100_scroll_back().

Here is the call graph for this function:

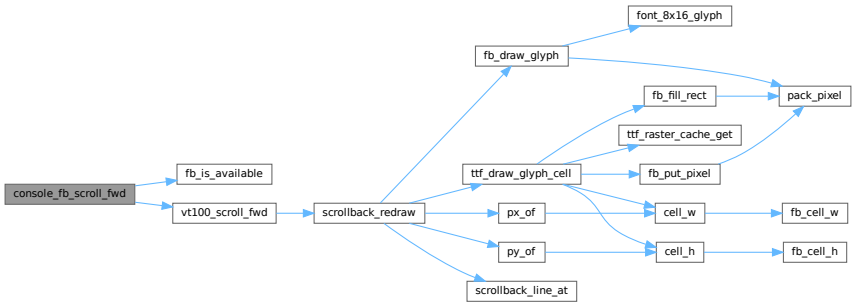


7.21.2.12 console_fb_scroll_fwd()

```
void console_fb_scroll_fwd (
            uint32_t n )
```

References fb_is_available(), and vt100_scroll_fwd().

Here is the call graph for this function:



7.21.2.13 console_fb_toggle_graphics()

```
void console_fb_toggle_graphics (
            void  )
```

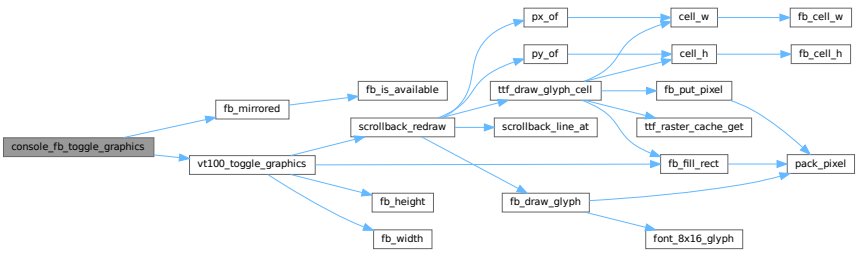
FABRIC-0.md item 4.4y-revised: thin wrapper over vt100_toggle_graphics() – see that function's doc comment for the full contract (the Alt+TAB graphics/text state machine).

No-op if the framebuffer console was never initialized.

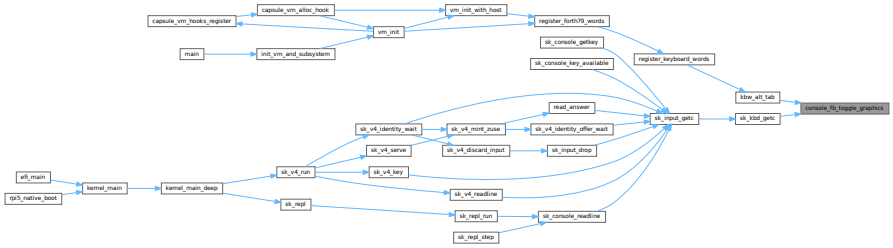
References fb_mirrored(), and vt100_toggle_graphics().

Referenced by kbw_alt_tab(), and sk_kbd_getc().

Here is the call graph for this function:



Here is the caller graph for this function:



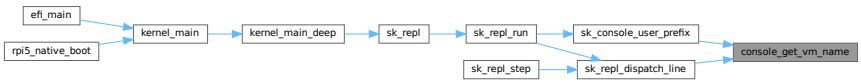
7.21.2.14 console_get_vm_name()

```
const char * console_get_vm_name (
            void  )
```

References g_active_vm_name.

Referenced by sk_console_user_prefix(), and sk_repl_dispatch_line().

Here is the caller graph for this function:



7.21.2.15 console_getc()

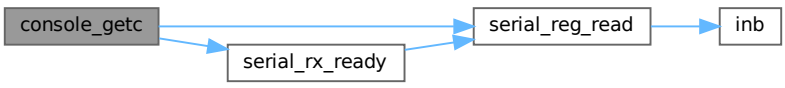
```
int console_getc (
            void  )
```

Read a single character from serial console (non-blocking) Returns -1 if no character available.

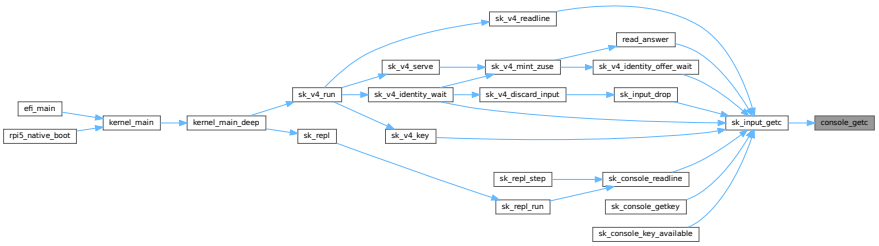
References serial_reg_read(), and serial_rx_ready().

Referenced by sk_input_getc().

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.16 console_init()

```
void console_init (
            void  )
```

Initialize serial console (UART).

Must be called once during early kernel boot, after console_attach_serial().

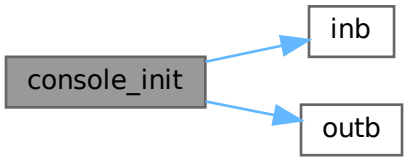
Initialize serial console (UART).

aarch64, riscv64: the line was set up by the firmware; nothing to do.

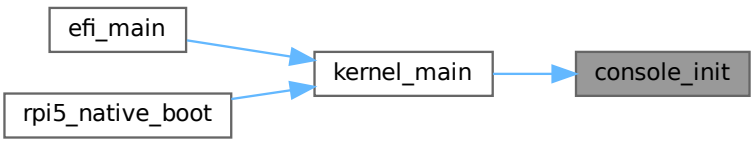
References g_serial_kind, inb(), outb(), SERIAL_NONE, and SERIAL_PORT16550.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.17 console_poll()

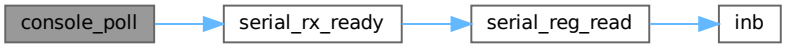
```
int console_poll (
            void  )
```

Check if character is available for reading.

Check if character is available for reading.

References serial_rx_ready().

Here is the call graph for this function:



7.21.2.18 console_println()

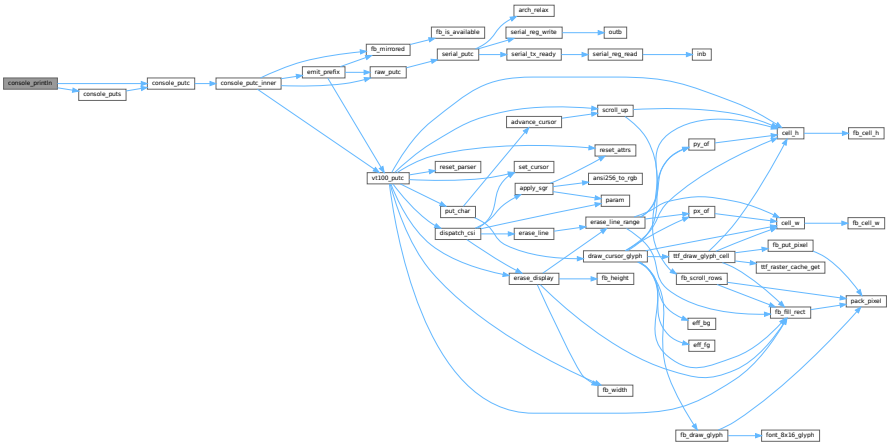
```
void console_println (
            const char * s )
```

Write a string with newline to serial console.

References console_putc(), and console_puts().

Referenced by aarch64_exception_handler(), aarch64_psci_conduit_init(), apic_init(), apic_timer_init(), apic_timer_start(), arch_early_init(), arch_interrupts_init(), blkio_usb_open_msc(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_wirebind_eject(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), capsule_wirebind_unclean_detac capsule_zuse_boot_logout(), capsule_zuse_boot_try_attach(), console_describe_serial(), converge_tsc_bare_metal(), decode_pf_error(), enable_interrupt_route(), failed(), gpt_read(), i8042_init(), init_timer_vm_mode(), ioapic_init(), isr_common_handler(), kernel_main(), kernel_main_deep(), pci_init(), plic_init(), print_banner(), print_boot_info(), print_heap_stats(), print_nl(), print_pmm_stats(), print_uint(), riscv64_exception_handler(), rng_init(), say_size(), sk_repl(), sk_repl_dispatch_line(), sk_repl_run(), sk_repl_step(), sk_v4_drive_arrived(), sk_v4_fixed_screen(), sk_v4_genesis(), sk_v4_grow_stick(), sk_v4_identity_boot(), sk_v4_login(), sk_v4_mint_zuse(), sk_v4_real_zuse_arrived(), sk_v4_run(), sk_v4_serve(), sk_v4_session_end(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), timer_fatal(), timer_init(), vblk_init_device(), vinput_init_device(), virtio_blk_find_artemis(), virtio_input_find_keyboard(), virtio_rng_init(), vm_bootstrap_root_vocabulary(), vm_init_with_host(), vm_physics_status(), vmm_init(), vmm_map_page(), vocab_sync_vm_vars(), vocabulary_create_vocabulary_direct(), vrng_init_device(), and xhci_poll_one().

Here is the call graph for this function:



7.21.2.19 console_putc()

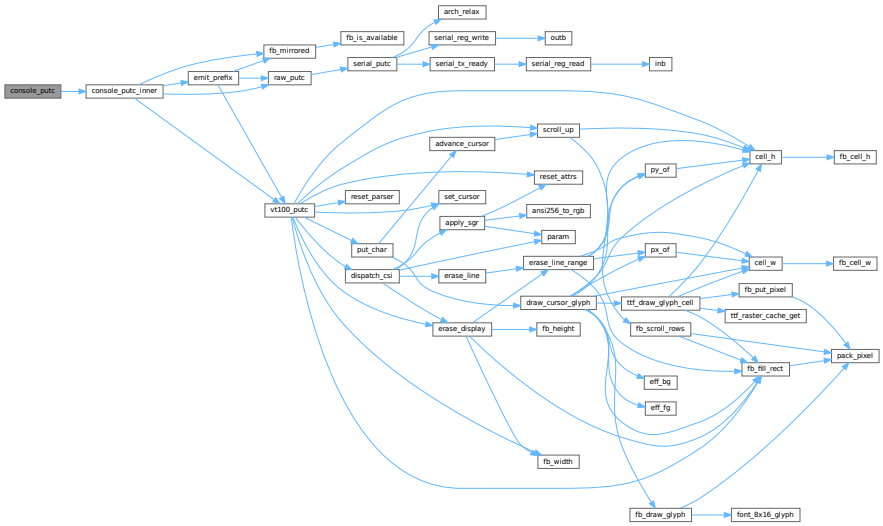
```
void console_putc (
            char c )
```

Write a single character to serial console.

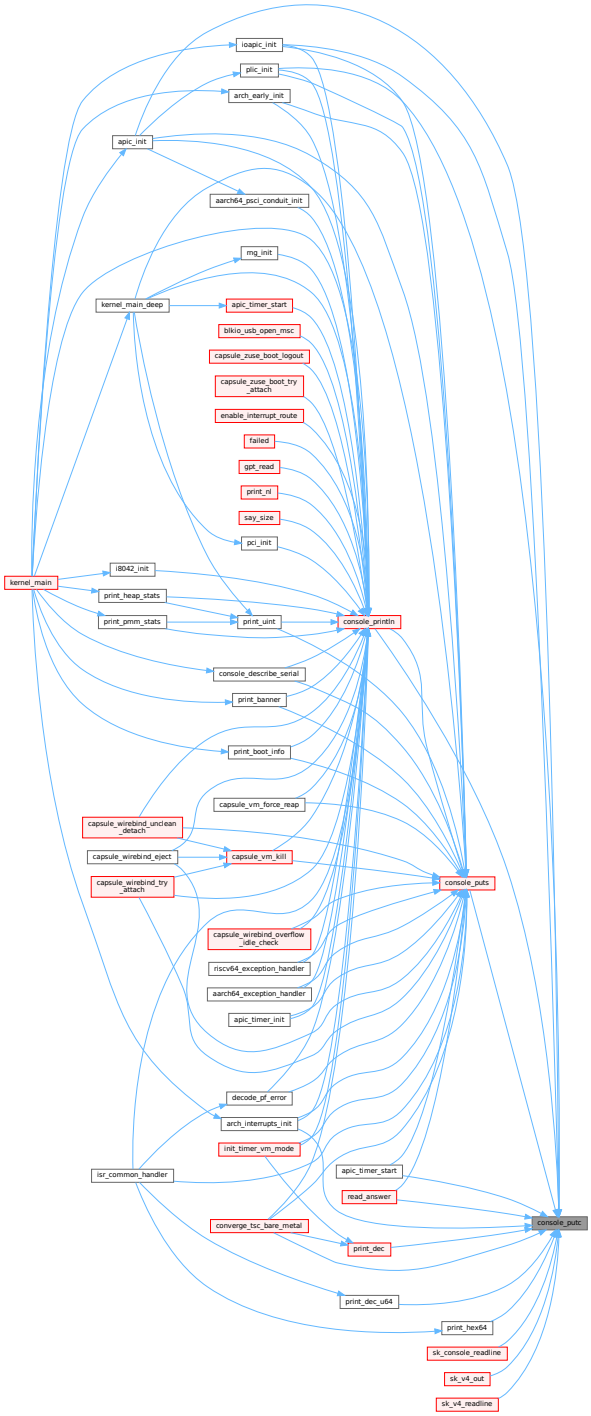
References console_putc_inner(), g_console_tx_count, and g_pending_line_close.

Referenced by apic_init(), apic_timer_start(), arch_interrupts_init(), console_println(), console_puts(), converge_tsc_bare_metal(), ioapic_init(), plic_init(), print_dec(), print_dec_u64(), print_hex64(), read_answer(), sk_console_readline(), sk_v4_out(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.20 console_puts()

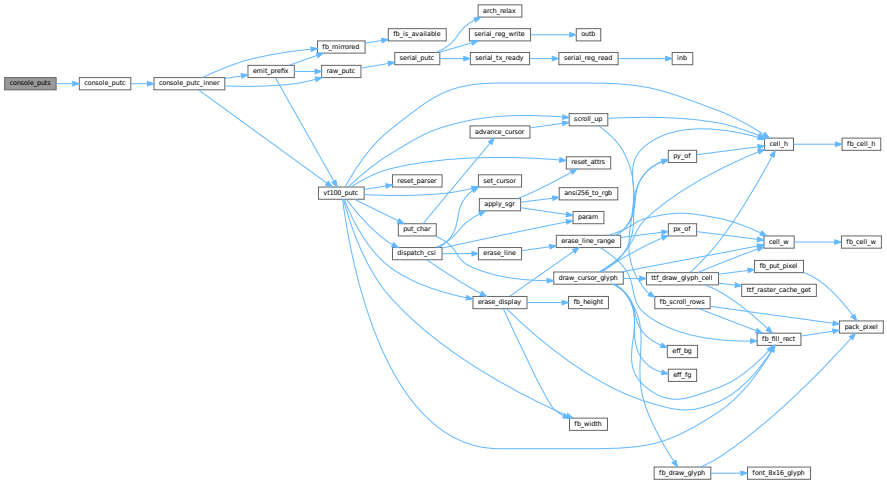
```
void console_puts (
            const char * s )
```

Write a null-terminated string to serial console.

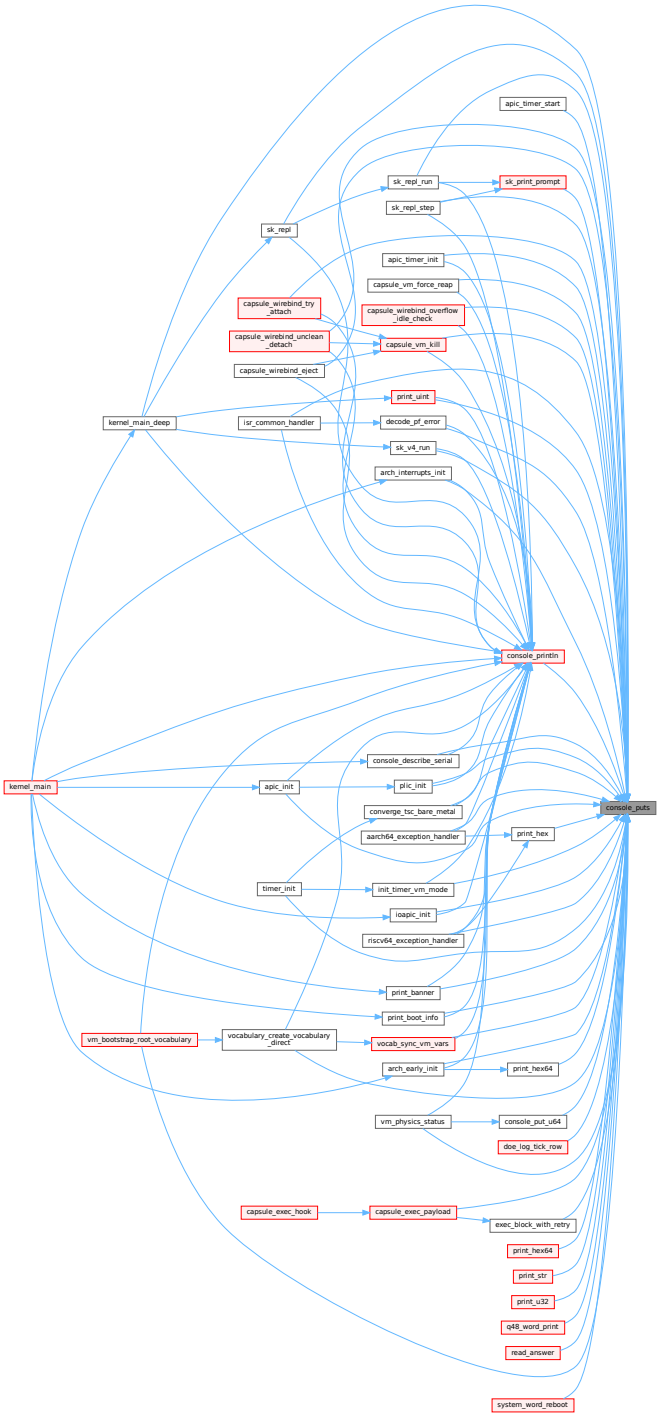
References console_putc().

Referenced by aarch64_exception_handler(), apic_init(), apic_timer_init(), apic_timer_start(), arch_early_init(), arch_interrupts_init(), capsule_exec_payload(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_wirebind_eject(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), capsule_wirebind_unclean_detach(), console_describe_serial(), console_println(), console_put_u64(), converge_tsc_bare_metal(), decode_pf_error(), doe_log_tick_row(), exec_block_with_retry(), init_timer_vm_mode(), ioapic_init(), isr_common_handler(), kernel_main_deep(), plic_init(), print_banner(), print_boot_info(), print_hex(), print_hex64(), print_hex64(), print_str(), print_u32(), print_uint(), q48_word_print(), read_answer(), riscv64_exception_handler(), sk_print_prompt(), sk_repl(), sk_repl_run(), sk_repl_step(), sk_v4_run(), system_word_reboot(), timer_init(), vm_bootstrap_root_vocabulary(), vm_physics_status(), vocab_sync_vm_vars(), and vocabulary_create_vocabulary_direct().

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.21 console_save_vm_name()

```
void console_save_vm_name (
            char * out,
            size_t cap )
```

console_save_vm_name - Copy the current active-VM name into the caller's own buffer, for a later console_set_vm_name() restore.

console_get_vm_name() alone is NOT safe for save-then-restore: it returns a pointer into the single internal buffer console_set_vm_name() copies into, so an intervening console_set_vm_name() call (the normal "switch, do work, switch back" pattern every BIRTH/VM-EXEC/CONNECT-∗ call site uses) overwrites the very bytes the saved pointer points at before the restore ever runs – found live 2026-08-28, the restore silently no-ops. Copies at most cap-1 bytes plus a NUL terminator; writes "" if there was no active name (NULL) to save.

Parameters

out	Caller-owned buffer.
cap	Its size in bytes.

References g_active_vm_name.

7.21.2.22 console_set_user_prefix_provider()

```
void console_set_user_prefix_provider (
            console_user_prefix_fn fn )
```

References g_user_prefix_fn.

Referenced by sk_repl_run(), and sk_v4_login().

Here is the caller graph for this function:

7.21.2.23 console_set_vm_name()

```
void console_set_vm_name (
            const char * name )
```

Set the active VM name shown as [Name] prefix on each output line.

Pass NULL to suppress the prefix (kernel-only output before any VM). Copies into internal storage (FABRIC-2.md Phase F, 2026-08-28) – the caller's own pointer does not need to remain valid afterward.

References CONSOLE_VM_NAME_BUF, g_active_vm_name, g_active_vm_name_buf, and name.

Referenced by capsule_vm_registry_init(), and sk_v4_login().

Here is the caller graph for this function:

7.21.2.24 console_tx_count()

```
uint64_t console_tx_count (
            void  )
```

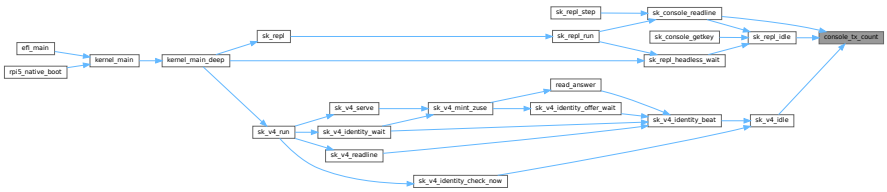
console_tx_count - Monotonic count of console_putc() calls delivered to either output (serial and/or framebuffer).

In use by the REPL to detect that an idle bottom half (heartbeat, USB attach/detach) wrote to the console while the top-level prompt was showing, so it can re-anchor the prompt afterward. Never decreases.

References g_console_tx_count.

Referenced by sk_console_readline(), sk_repl_idle(), and sk_v4_idle().

Here is the caller graph for this function:



7.21.3 Variable Documentation

7.21.3.1 g_sk_fault_word

```
volatile const char* g_sk_fault_word  [extern]
```

interrupts.c (aarch64) - Exception vector setup and common handler

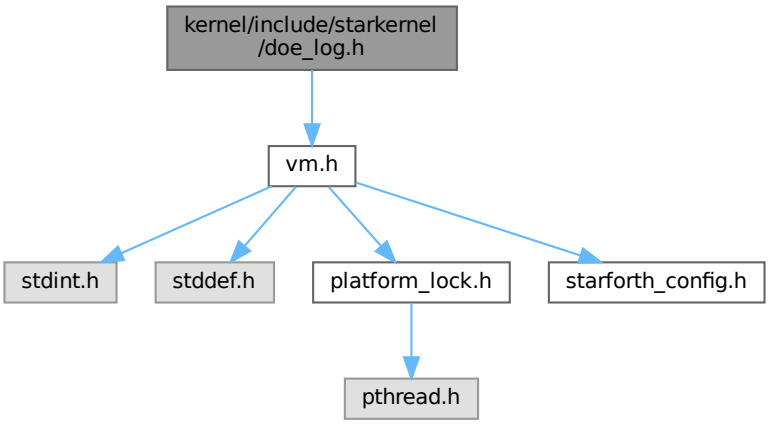
interrupts.c (aarch64) - Exception vector setup and common handler

Referenced by execute_colon_word(), isr_common_handler(), and vm_interpret_word().

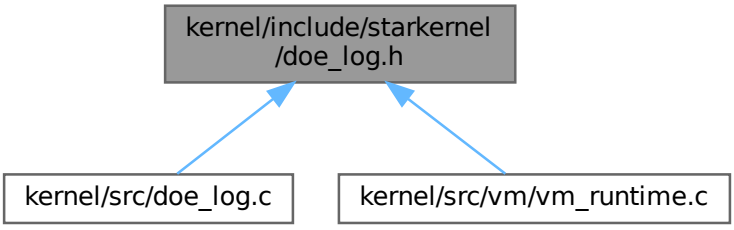
7.22 kernel/include/starkernel/doe_log.h File Reference

```
#include "vm.h"
```

Include dependency graph for doe_log.h:



This graph shows which files directly or indirectly include this file:



Functions

- void doe_log_tick_row (VM *vm, const HeartbeatTickSnapshot *snap)

  Emit one CSV row for the current heartbeat tick, unless g_doe_log_enabled is 0 (a no-op then).
- void register_doe_log_words (VM *vm)

  Registers HB-ON (–) and HB-OFF (–), which set g_doe_log_enabled to 1 and 0 respectively.

Variables

- int g_doe_log_enabled

  starkernel/doe_log.h — DoE (Design of Experiments) CSV logger

7.22.1 Function Documentation

7.22.1.1 doe_log_tick_row()

```
void doe_log_tick_row (
            VM * vm,
            const HeartbeatTickSnapshot * snap )
```

Emit one CSV row for the current heartbeat tick, unless g_doe_log_enabled is 0 (a no-op then).

Pulls APIC TimeTrustState via heartbeat_state(). Prints the column header once, before the first row ever emitted – not re-printed on every HB-ON, so a session toggled off and back on stays one continuous CSV rather than getting a second header embedded partway through.

Parameters

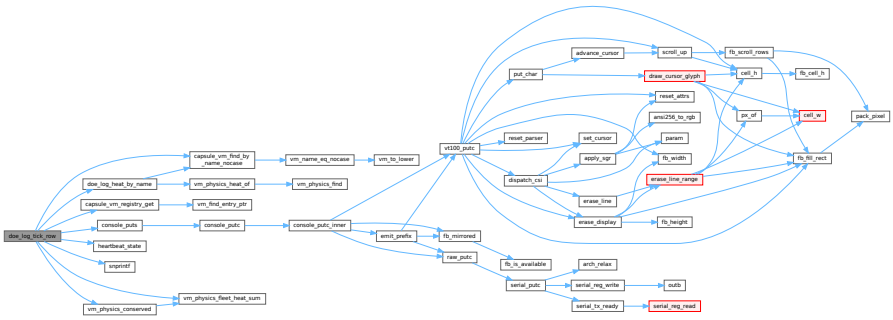
vm	VM instance (used for snapshot data)
snap	Populated HeartbeatTickSnapshot from heartbeat_capture_tick_snapshot()

References HeartbeatTickSnapshot::actual_window_size, HeartbeatTickSnapshot::avg_word_heat, HeartbeatTickSnapshot::bucket_
HeartbeatTickSnapshot::cache_hits_delta, capsule_vm_find_by_name_nocase(), capsule_vm_registry_get(),

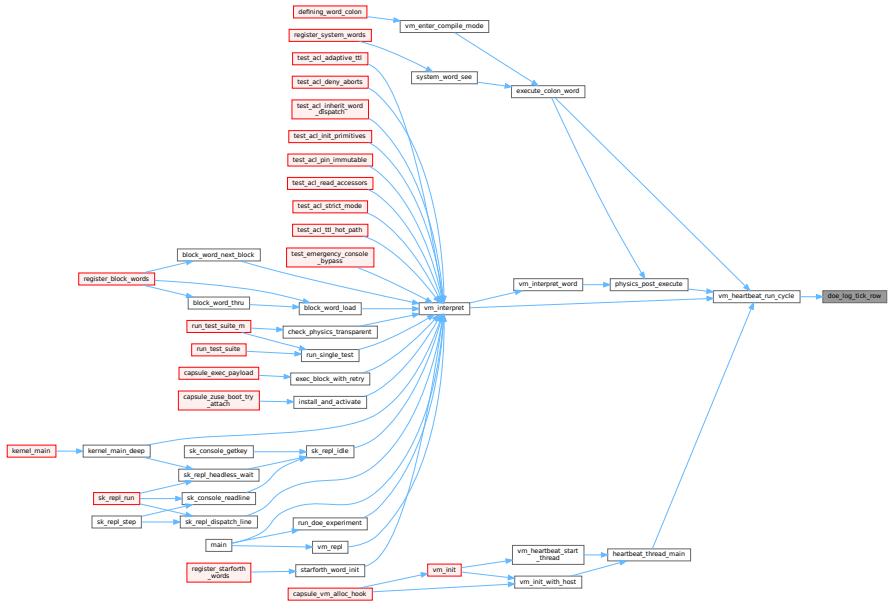
console_puts(), DOE_BUF_SIZE, doe_header, doe_log_heat_by_name(), DOE_PREFIX, HeartbeatTickSnapshot::elapsed_ns, HeartbeatTickSnapshot::estimated_jitter_ns, g_doe_log_enabled, heartbeat_state(), HeartbeatTickSnapshot::hot_word_count, VMRegistryEntry::name, HeartbeatTickSnapshot::predicted_label_hits, snprintf(), HeartbeatTickSnapshot::tick_interval_ns, HeartbeatTickSnapshot::tick_number, time_trust_state::ticks, time_trust_state::trust, time_trust_state::variance, VMRegistryEntry::vm_id, vm_physics_conserved(), vm_physics_fleet_heat_sum(), HeartbeatTickSnapshot::window_width, and HeartbeatTickSnapshot::word_executions_delta.

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.22.1.2 register_doe_log_words()

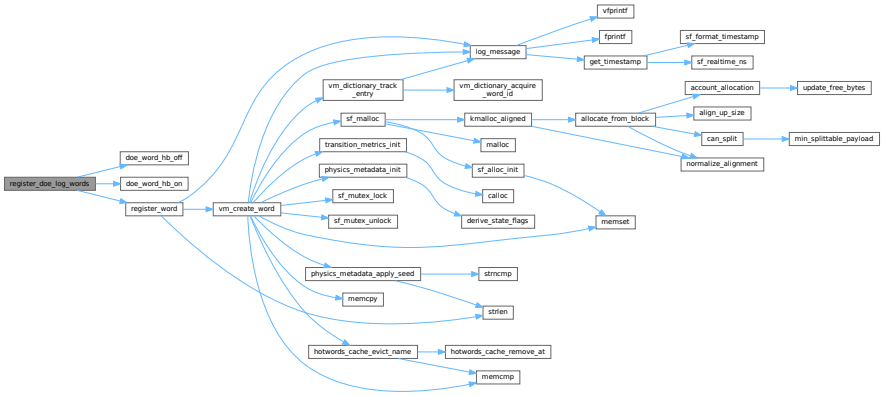
```
void register_doe_log_words (
            VM * vm )
```

Registers HB-ON (–) and HB-OFF (–), which set g_doe_log_enabled to 1 and 0 respectively.

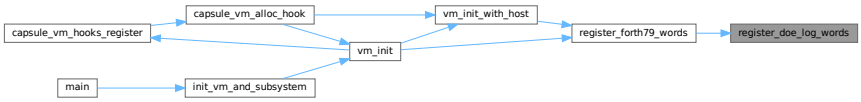
References doe_word_hb_off(), doe_word_hb_on(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.22.2 Variable Documentation

7.22.2.1 g_doe_log_enabled

```
int g_doe_log_enabled  [extern]
```

starkernel/doe_log.h — DoE (Design of Experiments) CSV logger

Emits per-heartbeat-tick CSV rows to the serial log, tagged with [HADES][DOE] so they can be grepped cleanly from the QEMU log.

Each row contains the 12 standard heartbeat snapshot fields plus 3 APIC timer fields from TimeTrustState: apic↩ _ticks, time_trust_q48, variance_q48

A header row is printed automatically before the first data row.

Always compiled in – gating moved from a build-time flag (HEARTBEAT_DOE_LOG) to a runtime one (g_doe_log↩ _enabled below) so the same build can run with or without per-tick instrumentation, toggled live via the HB-ON/HB-OFF FORTH words (register_doe_log_words()), no rebuild required. Runtime enable flag for doe_log_tick_row() below. Defaults to 1 (matches the old HEARTBEAT_DOE_LOG=1 default – instrumentation on unless explicitly turned off). Toggled by HB-ON/HB-OFF; read, not meant to be written directly outside those two words.

starkernel/doe_log.h — DoE (Design of Experiments) CSV logger

Emits CSV rows to the serial log once per heartbeat tick. Each row is prefixed with a cyan [HADES][DOE] tag so it can be extracted cleanly:

grep -aP '[HADES][DOE]' qemu-amd64-∗.log

Columns (in order): 0a. vm_name string — the emitting VM's own registry name (reverse lookup on vm->stadium↩ _vm_id, "?" if unregistered at this tick). Added 2026-09-16 for the multiuser/ multitasking DoE (FABRIC-3.md ?↩ XXXIII): every prior column already described *a* VM's state each row, but nothing said *which* VM emitted it, making concurrent-VM rows indistinguishable by source. 0b. vm_id_hex string — the emitting VM's own stadium_vm_↩ id as "hi-lo" hex, same format PARITY:BIRTH/PARITY:KILL already print. Names get reused across a trial's own KILL+rebirth cycles; this is the one column unique to a single birth instance.

1. tick_number uint32 — monotonic heartbeat counter
2. elapsed_ns uint64 — ns since run start
3. tick_interval_ns uint64 — actual interval from prior tick
4. cache_hits_delta uint32 — hot-words cache hits this tick (TODO: 0)
5. bucket_hits_delta uint32 — bucket hits this tick (TODO: 0)
6. word_executions_delta uint32 — words executed this tick (TODO: 0)
7. hot_word_count uint64 — words with heat >= threshold
8. avg_word_heat_q48 uint64 — mean execution heat as raw Q48.16 integer
9. window_width uint32 — rolling window effective size
10. actual_window_size uint32 — true analysis width: min(total_executions, effective_window_size)
11. predicted_label_hits uint32 — ANOVA early-exit confirmations per tick (L8 validation signal)
12. jitter_bits uint64 — estimated_jitter_ns IEEE 754 raw bits (union-punned)
13. apic_ticks uint64 — APIC timer monotonic tick count
14. time_trust_q48 uint64 — TIME-TRUST in Q48.16 format
15. variance_q48 uint64 — timing variance in Q48.16 format
16. hera_heat_q48 uint64 — VM fleet physics: Hera's execution_heat_q48
17. hermes_heat_q48 uint64 — VM fleet physics: Hermes's execution_heat_q48
18. artemis_heat_q48 uint64 — VM fleet physics: Artemis's execution_heat_q48
19. fleet_k_q48 uint64 — "K": vm_physics_fleet_heat_sum() over ALL live VMs (not just the 3 named Tripod members above – this is the genuine conservation invariant, which should read Q48_ONE (65536) whenever conservation holds). Requested 2026-09-12 alongside the 3x9 factorial analysis in experiments/std79-doe/ – columns 16-18 alone cannot reconstruct this since they omit every identity VM's own heat.
20. fleet_conserved uint32 — vm_physics_conserved() as 0/1: whether $|$fleet_k_q48 - Q48_ONE$|$ < VM_↩ PHYSICS_EPSILON_Q48 at the moment this row was captured.
21. switch_count_cumulative uint64 — Stage 3 preemptive-switch count, cumulative since boot (capsule_vm_switch_signal.c).
22. switch_current_slot uint32 — which registered switch-signal participant is executing at this tick (0=Hera,1=Hermes, 2=Artemis by current registration order; 0xFFFFFFFF = none/unregistered at this moment).
23. switch_hera_readiness uint32 — Hera's raw readiness accumulator (slot 0).
24. switch_hermes_readiness uint32 — Hermes's raw readiness accumulator (slot 1).

25. switch_artemis_readiness uint32 — Artemis's raw readiness accumulator (slot 2).
26. switch_ticks_since uint32 — ticks elapsed since the last real switch (capsule_vm_switch_signal.c); empirically verifies the fixed 50-tick threshold under real DoE load rather than assuming it.

Columns 21-26 (FABRIC-3.md ?XXVIII Stage 3 follow-on, 2026-09-13): added alongside Stage 3's timer-driven preemptive context switching – read-only exposure of state the switch-signal module already tracked internally for its own decision-making; no new measurement logic, just made observable. Slot readiness columns are hardcoded to the 3 known Tripod slots (same convention as columns 16-18) rather than a dynamic per-slot column set, for the same reason: this is a logging schema for the currently-known fleet, not the switch-signal mechanism itself (which stays slot-count-agnostic, headroom for up to 8 participants – see capsule_vm_switch_signal.h).

Note: avg_word_heat and estimated_jitter_ns are stored as double in the snapshot but the freestanding snprintf has no %f support. avg_word_heat is emitted as raw Q48.16 (multiply by 65536); jitter is union-punned to its IEEE 754 uint64 bit pattern to avoid UB from out-of-range cast.

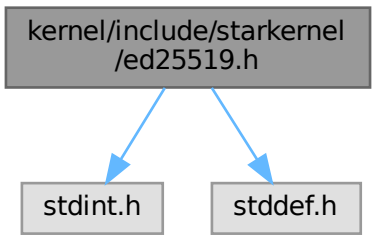
Columns 16-18 (VM-FLEET-ATTRACTOR-DESIGN-20260705.md, Phase 3): looked up by name each tick rather than by a fixed vm_id, since kill/rebirth (e.g. TRIPOD-TEST's K-soak) assigns a fresh vm_id on every rebirth. Fixed to the three known Tripod VMs, not a dynamic per-VM column set – this is a logging schema for a specific known fleet, not the physics mechanism itself (which stays genuinely VM-count-agnostic). Reads 0 for any VM not currently found in the registry (e.g. Hermes momentarily during its kill/rebirth window), which is a legitimate observation, not an error.

Referenced by doe_log_tick_row(), doe_word_hb_off(), and doe_word_hb_on().

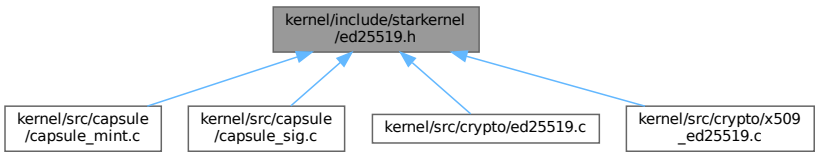
7.23 kernel/include/starkernel/ed25519.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for ed25519.h:



This graph shows which files directly or indirectly include this file:



Functions

- int ed25519_verify (const uint8_t pubkey[32], const uint8_t *msg, size_t msg_len, const uint8_t sig[64])
- void ed25519_keygen (const uint8_t seed[32], uint8_t pubkey_out[32])
- void ed25519_sign (const uint8_t seed[32], const uint8_t *msg, size_t msg_len, uint8_t sig_out[64])

7.23.1 Function Documentation

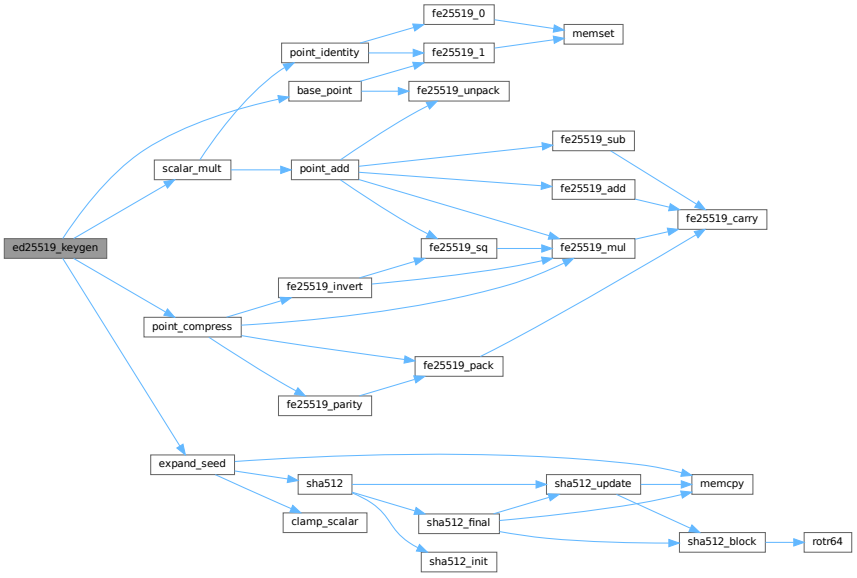
7.23.1.1 ed25519_keygen()

```
void ed25519_keygen (
            const uint8_t seed[32],
            uint8_t pubkey_out[32] )
```

References base_point(), expand_seed(), point_compress(), and scalar_mult().

Referenced by capsule_mint_identity().

Here is the call graph for this function:



Here is the caller graph for this function:

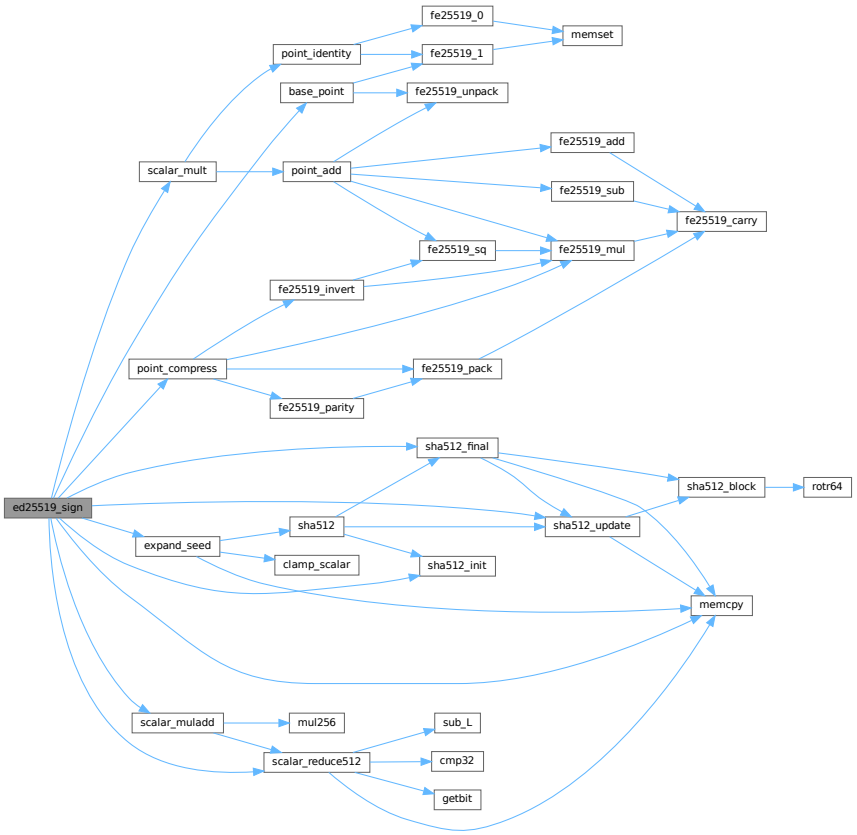
7.23.1.2 ed25519_sign()

```
void ed25519_sign (
            const uint8_t seed[32],
            const uint8_t * msg,
            size_t msg_len,
            uint8_t sig_out[64] )
```

References base_point(), expand_seed(), memcpy(), point_compress(), scalar_muladd(), scalar_mult(), scalar_reduce512(), sha512_final(), sha512_init(), and sha512_update().

Referenced by x509_build_user_cert().

Here is the call graph for this function:



Here is the caller graph for this function:

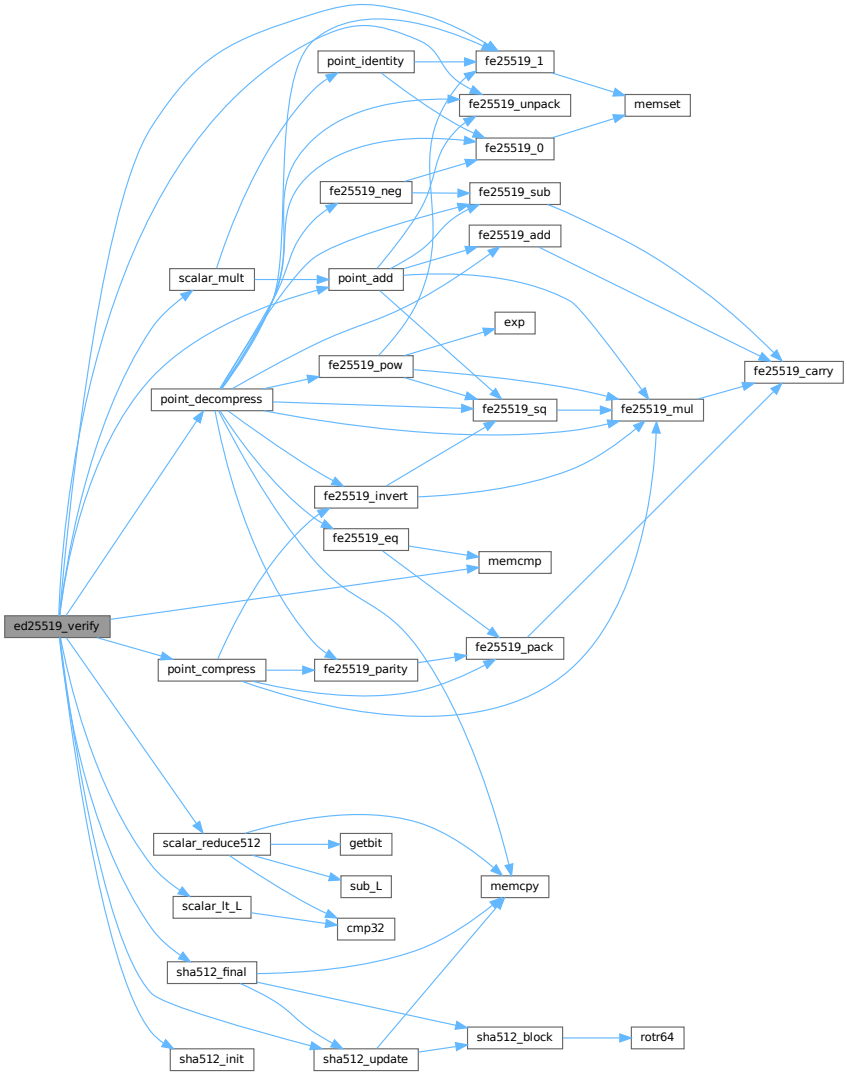
7.23.1.3 ed25519_verify()

```
int ed25519_verify (
            const uint8_t pubkey[32],
            const uint8_t * msg,
            size_t msg_len,
            const uint8_t sig[64] )
```

References BX_BYTES, BY_BYTES, fe25519_1(), fe25519_unpack(), memcmp(), point_add(), point_compress(), point_decompress(), scalar_lt_L(), scalar_mult(), scalar_reduce512(), sha512_final(), sha512_init(), sha512_update(), ed_point_t::x, ed_point_t::y, and ed_point_t::z.

Referenced by capsule_verify_signature(), and x509_verify_signature().

Here is the call graph for this function:

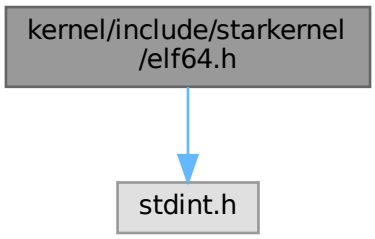


Here is the caller graph for this function:

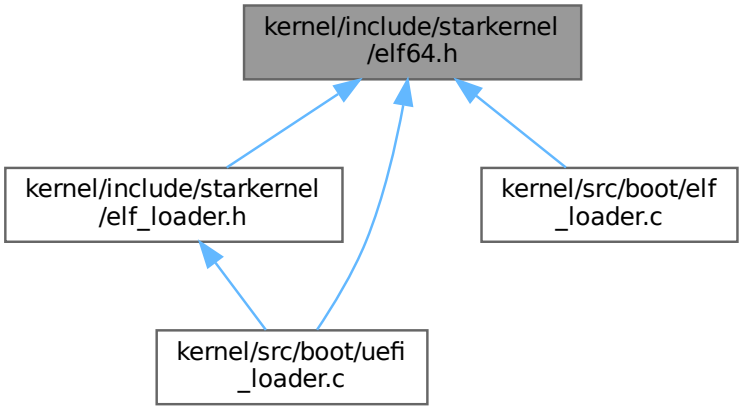
7.24 kernel/include/starkernel/elf64.h File Reference

```
#include <stdint.h>
```

Include dependency graph for elf64.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Elf64_Ehdr

- struct Elf64_Phdr
- struct Elf64_Shdr
- struct Elf64_Rela
- struct Elf64_Sym
- struct Elf64_Rel

Macros

- #define EI_MAG0 0

  elf64.h - ELF64 structures for kernel loading Minimal ELF64 definitions for the UEFI loader
- #define EI_MAG1 1
- #define EI_MAG2 2
- #define EI_MAG3 3
- #define EI_CLASS 4
- #define EI_DATA 5
- #define EI_VERSION 6
- #define EI_OSABI 7
- #define EI_ABIVERSION 8
- #define EI_PAD 9
- #define EI_NIDENT 16
- #define ELFMAG0 0x7F
- #define ELFMAG1 'E'
- #define ELFMAG2 'L'
- #define ELFMAG3 'F'
- #define ELFCLASS32 1
- #define ELFCLASS64 2
- #define ELFDATA2LSB 1 /* Little-endian */
- #define ELFDATA2MSB 2 /* Big-endian */
- #define EV_CURRENT 1
- #define ELFOSABI_NONE 0 /* UNIX System V ABI */
- #define ET_NONE 0
- #define ET_REL 1
- #define ET_EXEC 2
- #define ET_DYN 3
- #define ET_CORE 4
- #define EM_X86_64 62 /* AMD x86-64 */
- #define EM_AARCH64 183 /* ARM AARCH64 */
- #define EM_RISCV 243 /* RISC-V */
- #define PT_NULL 0
- #define PT_LOAD 1
- #define PT_DYNAMIC 2
- #define PT_INTERP 3
- #define PT_NOTE 4
- #define PT_SHLIB 5
- #define PT_PHDR 6
- #define PT_TLS 7
- #define PF_X 0x1 /* Execute */
- #define PF_W 0x2 /* Write */
- #define PF_R 0x4 /* Read */
- #define SHT_NULL 0
- #define SHT_PROGBITS 1
- #define SHT_SYMTAB 2
- #define SHT_STRTAB 3
- #define SHT_RELA 4

- #define SHT_HASH 5
- #define SHT_DYNAMIC 6
- #define SHT_NOTE 7
- #define SHT_NOBITS 8
- #define SHT_REL 9
- #define SHT_SHLIB 10
- #define SHT_DYNSYM 11
- #define R_X86_64_NONE 0
- #define R_X86_64_64 1
- #define R_X86_64_PC32 2
- #define R_X86_64_RELATIVE 8
- #define R_X86_64_32 10
- #define R_X86_64_32S 11
- #define R_X86_64_PLT32 4
- #define R_AARCH64_NONE 0
- #define R_AARCH64_ABS64 257
- #define R_AARCH64_RELATIVE 1027
- #define R_RISCV_NONE 0
- #define R_RISCV_64 2
- #define R_RISCV_RELATIVE 3
- #define ELF64_R_SYM(i) ((i) >> 32)
- #define ELF64_R_TYPE(i) ((i) & 0xffffffffL)
- #define ELF64_R_INFO(s, t) (((Elf64_Xword)(s) << 32) + ((Elf64_Xword)(t) & 0xffffffffL))

Typedefs

- typedef uint64_t Elf64_Addr
- typedef uint64_t Elf64_Off
- typedef uint16_t Elf64_Half
- typedef uint32_t Elf64_Word
- typedef int32_t Elf64_Sword
- typedef uint64_t Elf64_Xword
- typedef int64_t Elf64_Sxword

7.24.1 Macro Definition Documentation

7.24.1.1 EI_ABIVERSION

```
#define EI_ABIVERSION 8
```

7.24.1.2 EI_CLASS

```
#define EI_CLASS 4
```

7.24.1.3 EI_DATA

```
#define EI_DATA 5
```

7.24.1.4 EI_MAG0

```
#define EI_MAG0 0
```

elf64.h - ELF64 structures for kernel loading Minimal ELF64 definitions for the UEFI loader

7.24.1.5 EI_MAG1

```
#define EI_MAG1 1
```

7.24.1.6 EI_MAG2

```
#define EI_MAG2 2
```

7.24.1.7 EI_MAG3

```
#define EI_MAG3 3
```

7.24.1.8 EI_NIDENT

```
#define EI_NIDENT 16
```

7.24.1.9 EI_OSABI

```
#define EI_OSABI 7
```

7.24.1.10 EI_PAD

```
#define EI_PAD 9
```

7.24.1.11 EI_VERSION

```
#define EI_VERSION 6
```

7.24.1.12 ELF64_R_INFO

```
#define ELF64_R_INFO(
              s,
              t ) (((Elf64_Xword)(s) << 32) + ((Elf64_Xword)(t) & 0xffffffffL))
```

7.24.1.13 ELF64_R_SYM

```
#define ELF64_R_SYM(
             i ) ((i) >> 32)
```

7.24.1.14 ELF64_R_TYPE

```
#define ELF64_R_TYPE(
             i ) ((i) & 0xffffffffL)
```

7.24.1.15 ELFCLASS32

```
#define ELFCLASS32 1
```

7.24.1.16 ELFCLASS64

```
#define ELFCLASS64 2
```

7.24.1.17 ELFDATA2LSB

```
#define ELFDATA2LSB 1 /* Little-endian */
```

7.24.1.18 ELFDATA2MSB

```
#define ELFDATA2MSB 2 /* Big-endian */
```

7.24.1.19 ELFMAG0

```
#define ELFMAG0 0x7F
```

7.24.1.20 ELFMAG1

```
#define ELFMAG1 'E'
```

7.24.1.21 ELFMAG2

```
#define ELFMAG2 'L'
```

7.24.1.22 ELFMAG3

```
#define ELFMAG3 'F'
```

7.24.1.23 ELFOSABI_NONE

```
#define ELFOSABI_NONE 0 /* UNIX System V ABI */
```

7.24.1.24 EM_AARCH64

```
#define EM_AARCH64 183 /* ARM AARCH64 */
```

7.24.1.25 EM_RISCV

```
#define EM_RISCV 243 /* RISC-V */
```

7.24.1.26 EM_X86_64

```
#define EM_X86_64 62 /* AMD x86-64 */
```

7.24.1.27 ET_CORE

```
#define ET_CORE 4
```

7.24.1.28 ET_DYN

```
#define ET_DYN 3
```

7.24.1.29 ET_EXEC

```
#define ET_EXEC 2
```

7.24.1.30 ET_NONE

```
#define ET_NONE 0
```

7.24.1.31 ET_REL

```
#define ET_REL 1
```

7.24.1.32 EV_CURRENT

```
#define EV_CURRENT 1
```

7.24.1.33 PF_R

```
#define PF_R 0x4 /* Read */
```

7.24.1.34 PF_W

```
#define PF_W 0x2 /* Write */
```

7.24.1.35 PF_X

```
#define PF_X 0x1 /* Execute */
```

7.24.1.36 PT_DYNAMIC

```
#define PT_DYNAMIC 2
```

7.24.1.37 PT_INTERP

```
#define PT_INTERP 3
```

7.24.1.38 PT_LOAD

```
#define PT_LOAD 1
```

7.24.1.39 PT_NOTE

```
#define PT_NOTE 4
```

7.24.1.40 PT_NULL

```
#define PT_NULL 0
```

7.24.1.41 PT_PHDR

```
#define PT_PHDR 6
```

7.24.1.42 PT_SHLIB

```
#define PT_SHLIB 5
```

7.24.1.43 PT_TLS

```
#define PT_TLS 7
```

7.24.1.44 R_AARCH64_ABS64

```
#define R_AARCH64_ABS64 257
```

7.24.1.45 R_AARCH64_NONE

```
#define R_AARCH64_NONE 0
```

7.24.1.46 R_AARCH64_RELATIVE

```
#define R_AARCH64_RELATIVE 1027
```

7.24.1.47 R_RISCV_64

```
#define R_RISCV_64 2
```

7.24.1.48 R_RISCV_NONE

```
#define R_RISCV_NONE 0
```

7.24.1.49 R_RISCV_RELATIVE

```
#define R_RISCV_RELATIVE 3
```

7.24.1.50 R_X86_64_32

```
#define R_X86_64_32 10
```

7.24.1.51 R_X86_64_32S

```
#define R_X86_64_32S 11
```

7.24.1.52 R_X86_64_64

```
#define R_X86_64_64 1
```

7.24.1.53 R_X86_64_NONE

```
#define R_X86_64_NONE 0
```

7.24.1.54 R_X86_64_PC32

```
#define R_X86_64_PC32 2
```

7.24.1.55 R_X86_64_PLT32

```
#define R_X86_64_PLT32 4
```

7.24.1.56 R_X86_64_RELATIVE

```
#define R_X86_64_RELATIVE 8
```

7.24.1.57 SHT_DYNAMIC

```
#define SHT_DYNAMIC 6
```

7.24.1.58 SHT_DYNSYM

```
#define SHT_DYNSYM 11
```

7.24.1.59 SHT_HASH

```
#define SHT_HASH 5
```

7.24.1.60 SHT_NOBITS

```
#define SHT_NOBITS 8
```

7.24.1.61 SHT_NOTE

```
#define SHT_NOTE 7
```

7.24.1.62 SHT_NULL

```
#define SHT_NULL 0
```

7.24.1.63 SHT_PROGBITS

```
#define SHT_PROGBITS 1
```

7.24.1.64 SHT_REL

```
#define SHT_REL 9
```

7.24.1.65 SHT_RELA

```
#define SHT_RELA 4
```

7.24.1.66 SHT_SHLIB

```
#define SHT_SHLIB 10
```

7.24.1.67 SHT_STRTAB

```
#define SHT_STRTAB 3
```

7.24.1.68 SHT_SYMTAB

```
#define SHT_SYMTAB 2
```

7.24.2 Typedef Documentation

7.24.2.1 Elf64_Addr

```
typedef uint64_t Elf64_Addr
```

7.24.2.2 Elf64_Half

```
typedef uint16_t Elf64_Half
```

7.24.2.3 Elf64_Off

```
typedef uint64_t Elf64_Off
```

7.24.2.4 Elf64_Sword

```
typedef int32_t Elf64_Sword
```

7.24.2.5 Elf64_Sxword

```
typedef int64_t Elf64_Sxword
```

7.24.2.6 Elf64_Word

```
typedef uint32_t Elf64_Word
```

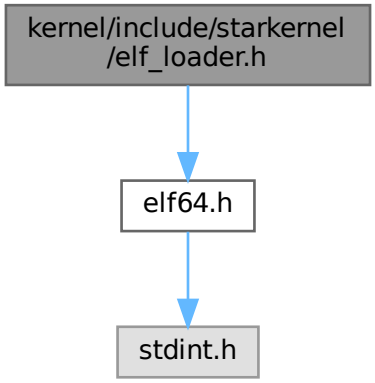
7.24.2.7 Elf64_Xword

```
typedef uint64_t Elf64_Xword
```

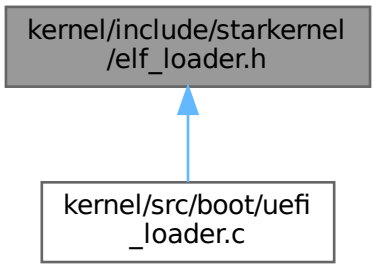
7.25 kernel/include/starkernel/elf_loader.h File Reference

```
#include "elf64.h"
```

Include dependency graph for elf_loader.h:



This graph shows which files directly or indirectly include this file:



Functions

- int elf_load_kernel (const uint8_t *elf_data, uint64_t elf_size, Elf64_Addr *entry_out)
  elf_loader.h - ELF64 kernel loader interface

7.25.1 Function Documentation

7.25.1.1 elf_load_kernel()

```
int elf_load_kernel (
            const uint8_t * elf_data,
            uint64_t elf_size,
            Elf64_Addr * entry_out )
```

elf_loader.h - ELF64 kernel loader interface

elf_loader.h - ELF64 kernel loader interface

Top-level ELF loader called by `efi_main()` (in the split-build path) after `ExitBootServices()`. Performs three operations in sequence:

1. **Header validation** via `elf_validate_header()` — checks magic, class, data encoding, version, type, and machine-specific architecture. Returns 0 immediately on failure.
2. **Segment loading** via `elf_load_segments()` — maps all `PT_LOAD` segments to their target addresses, initialises BSS to zero. The `load_base` is:
   - 0 for `ET_EXEC` (absolute-address kernel; typical for StarKernel).
   - 0x400000 (4 MB) for `ET_DYN` (position-independent kernel).
3. **Relocation processing** via `elf_apply_relocations()` — applies `SHT_RELA` sections to patch in the correct runtime addresses. Returns 0 on any relocation error.

On success, stores `load_base` + `e_entry` in `*entry_out` — the virtual address of the kernel's C entry function `kernel_main()` — and returns 1. The caller performs an indirect call to this address.

`elf_size` is currently unused (reserved for future bounds-checking of program-header and section-header offsets against the file boundary).

Parameters

elf_data	Pointer to the ELF64 file image in UEFI loader memory; must remain valid until after the call returns.
elf_size	Size of the ELF image in bytes (currently unchecked).
entry_out	Output: receives the kernel entry point address on success.

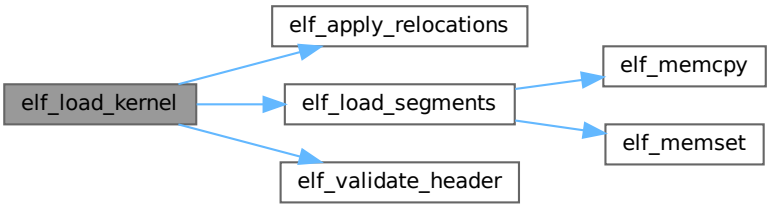
Returns

1 on success, 0 if validation, segment loading, or relocation fails.

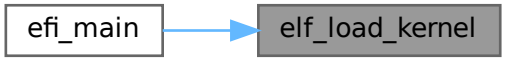
References Elf64_Ehdr::e_entry, Elf64_Ehdr::e_type, elf_apply_relocations(), elf_load_segments(), elf_validate_header(), ET_DYN, and ET_EXEC.

Referenced by efi_main().

Here is the call graph for this function:

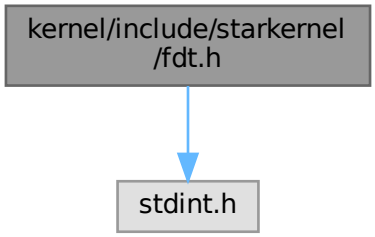


Here is the caller graph for this function:



7.26 kernel/include/starkernel/fdt.h File Reference

`#include <stdint.h>`
Include dependency graph for fdt.h:



This graph shows which files directly or indirectly include this file:

Functions

- int fdt_valid (const void *fdt)

  fdt.h - Minimal flattened-devicetree reader
- const void * fdt_find_prop (const void *fdt, const char *name, uint32_t *len_out)

  Find the first property with `name` anywhere in the tree.
- int fdt_prop_u32 (const void *fdt, const char *name, uint32_t *out)

  Read a single-cell (32-bit) property by name.
- const void * fdt_find_node_by_compatible (const void *fdt, const char *compatible)

  Find the first node whose "compatible" property matches `compatible`.
- const void * fdt_find_node_by_device_type (const void *fdt, const char *type)

  Find the first node whose "device_type" property equals `type`.
- const void * fdt_find_node_by_name (const void *fdt, const char *name)

  Find the first node whose own name matches `name`.
- const void * fdt_next_child_node (const void *fdt, const void *parent, const void *prev_child)

  Iterate the direct children of one node.
- const void * fdt_find_prop_in_node (const void *fdt, const void *node, const char *name, uint32_t *len_out)

  Find a property by name, scoped to one node.
- int fdt_complete (const void *fdt)

  Is the structure block whole? `fdt_valid()` checks the header; this checks also that the structure block ends with its end token, as a blob that has not been cut short does.
- const void * fdt_root_node (const void *fdt)

  The root node.
- const void * fdt_find_child (const void *fdt, const void *parent, const char *name, uint32_t name_len)

  Find a direct child of a node by its whole name.
- int fdt_reg_address (const void *fdt, const char *compatible, uint32_t index, uint64_t *out)

  The CPU's address for entry `index` of the `reg` of the first enabled node compatible with `compatible` (2026-10-14).

7.26.1 Function Documentation

7.26.1.1 fdt_complete()

```
int fdt_complete (
            const void * fdt )
```

Is the structure block whole? `fdt_valid()` checks the header; this checks also that the structure block ends with its end token, as a blob that has not been cut short does.

A reader that takes a property's absence to mean its default (a UART's "reg-shift") must not be handed half a tree.

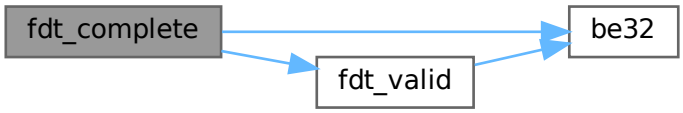
Returns

1 if `fdt` is valid and its structure block ends with FDT_END.

References be32(), FDT_END, fdt_valid(), fdt_header_t::off_dt_struct, and fdt_header_t::size_dt_struct.

Referenced by fdt_console_uart(), and fdt_reg_address().

Here is the call graph for this function:



Here is the caller graph for this function:

7.26.1.2 fdt_find_child()

```
const void * fdt_find_child (
            const void * fdt,
            const void * parent,
            const char * name,
            uint32_t name_len )
```

Find a direct child of a node by its whole name.

The name is matched exactly, unit address included ("serial@10000000" is not "serial"), and is given with its length so that one component of a path can be passed without copying it. Only `parent's` own children are looked at, never their descendants.

Parameters

fdt	Blob; checked with `fdt_valid()`.
parent	Node handle; NULL is safe and returns NULL.
name	The child's name; need not be NUL-terminated.
name_len	Its length in bytes.

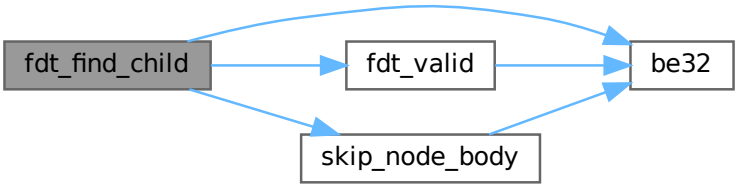
Returns

Handle to the child, or NULL if `parent` has none of that name.

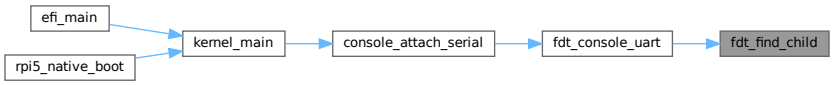
References be32(), FDT_BEGIN_NODE, FDT_NOP, FDT_PROP, fdt_valid(), name, name_len, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and skip_node_body().

Referenced by fdt_console_uart().

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.1.3 fdt_find_node_by_compatible()

```
const void * fdt_find_node_by_compatible (
            const void * fdt,
            const char * compatible )
```

Find the first node whose "compatible" property matches `compatible`.

"compatible" is a NUL-separated list of strings (DT spec ?2.3.1) — matches if `compatible` equals any one entry in the list, not just the whole property verbatim. Scans the whole tree in document order; the first matching node wins if more than one exists.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
compatible	Compatible string to match, NUL-terminated.

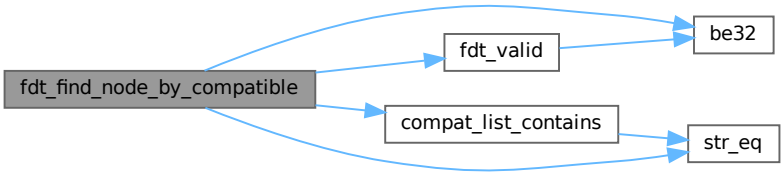
Returns

An opaque handle to the matched node, for use with `fdt_find_prop_in_node()` only (not a raw offset or a pointer to anything else meaningful) — or NULL if no node matches.

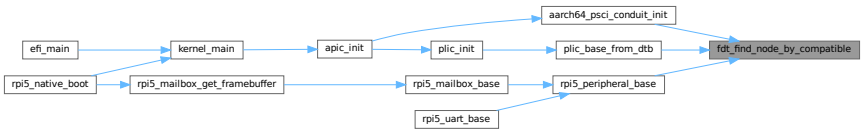
References base, be32(), compat_list_contains(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, FDT_PROP, fdt_valid(), fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and str_eq().

Referenced by aarch64_psci_conduit_init(), plic_base_from_dtb(), and rpi5_peripheral_base().

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.1.4 fdt_find_node_by_device_type()

```
const void * fdt_find_node_by_device_type (
            const void * fdt,
            const char * type )
```

Find the first node whose "device_type" property equals `type`.

Some standard nodes (`/memory` per DT spec ?3.4) are identified by `device_type`, not `compatible` — unlike `compatible`, `device_type` is a single NUL-terminated string, not a list, so this matches the whole property value rather than scanning entries within it. Scans the whole tree in document order; the first matching node wins if more than one exists.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
type	device_type value to match, NUL-terminated.

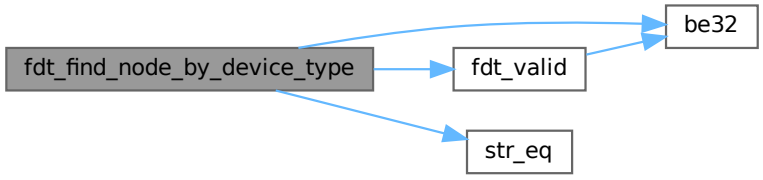
Returns

An opaque handle to the matched node, for use with `fdt_find_prop_in_node()` only — or NULL if no node matches.

References base, be32(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, FDT_PROP, fdt_valid(), fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and str_eq().

Referenced by collect_ram_regions(), and rpi5_mmu_map_ram().

Here is the call graph for this function:



Here is the caller graph for this function:

7.26.1.5 fdt_find_node_by_name()

```
const void * fdt_find_node_by_name (
            const void * fdt,
            const char * name )
```

Find the first node whose own name matches `name`.

Node names follow the DT spec ?2.2.1 `name[@unit-address]` convention — matches if `name` equals the node's name up to (not including) an `@` suffix, or the whole name if there is none. For a singleton node with no unit address (`/reserved-memory` being the concrete case this was added for), this is an exact match. Scans the whole tree in document order; the first matching node wins if more than one exists.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
name	Node name to match, NUL-terminated, no `@` suffix.

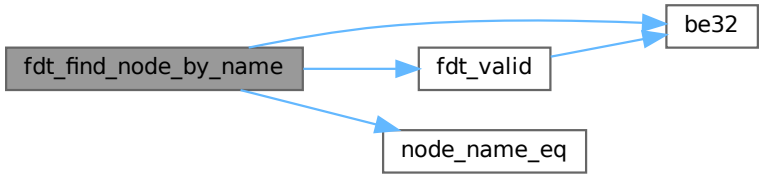
Returns

An opaque handle to the matched node, for use with `fdt_find_prop_in_node()` / `fdt_next_child_node()` only — or NULL if no node matches.

References base, be32(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, FDT_PROP, fdt_valid(), name, node_name_eq(), fdt_header_t::off_dt_struct, and fdt_header_t::size_dt_struct.

Referenced by collect_reserved_ranges().

Here is the call graph for this function:



Here is the caller graph for this function:

7.26.1.6 fdt_find_prop()

```
const void * fdt_find_prop (
             const void * fdt,
             const char * name,
             uint32_t * len_out )
```

Find the first property with `name` anywhere in the tree.

Scans the structure block in document order and returns the first match regardless of which node it belongs to. That is sufficient for properties which are uniform across a machine (timebase-frequency being the case this was written for) and is *not* sufficient for anything node-scoped.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
name	Property name, NUL-terminated.
len_out	Receives the property length in bytes; may be NULL.

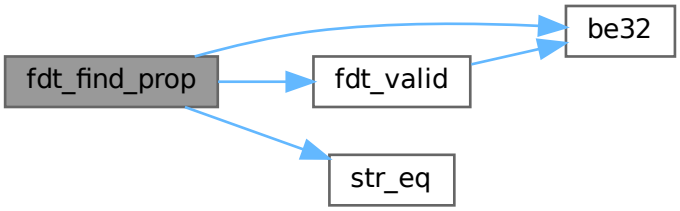
Returns

Pointer to the property value inside `fdt`, or NULL if not found. The value is big-endian as stored in the blob.

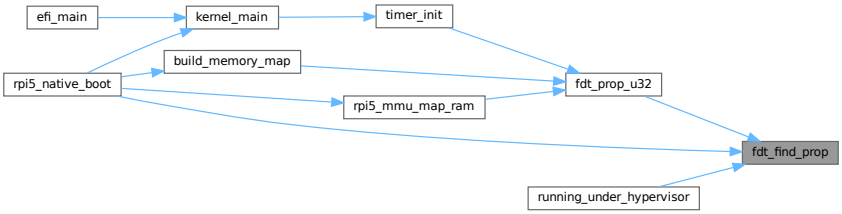
References base, be32(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, FDT_PROP, fdt_valid(), name, fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and str_eq().

Referenced by fdt_prop_u32(), rpi5_native_boot(), and running_under_hypervisor().

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.1.7 fdt_find_prop_in_node()

```
const void * fdt_find_prop_in_node (
            const void * fdt,
            const void * node,
            const char * name,
            uint32_t * len_out )
```

Find a property by name, scoped to one node.

Like `fdt_find_prop()`, but scans only `node`'s own direct properties (as returned by `fdt_find_node_by_compatible()` — stops at the first child node or the end of `node`'s property list, never descends into children, never continues into a sibling. This is the difference that matters for a property name like "reg", which is not unique across the tree the way "timebase-frequency" (the whole reason `fdt_find_prop()` was originally sufficient) happens to be.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
node	Handle from `fdt_find_node_by_compatible()`; NULL is safe and returns NULL (propagates a failed node lookup without a separate caller-side check).
name	Property name, NUL-terminated.
len_out	Receives the property length in bytes; may be NULL.

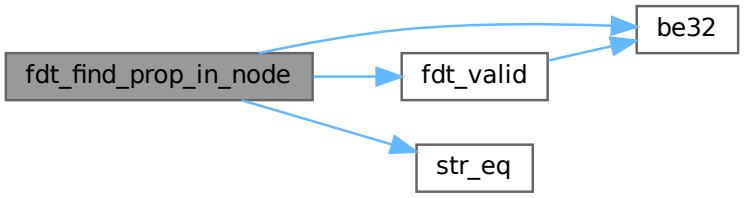
Returns

Pointer to the property value inside `fdt`, or NULL if not found (or if `node` is NULL). The value is big-endian as stored in the blob.

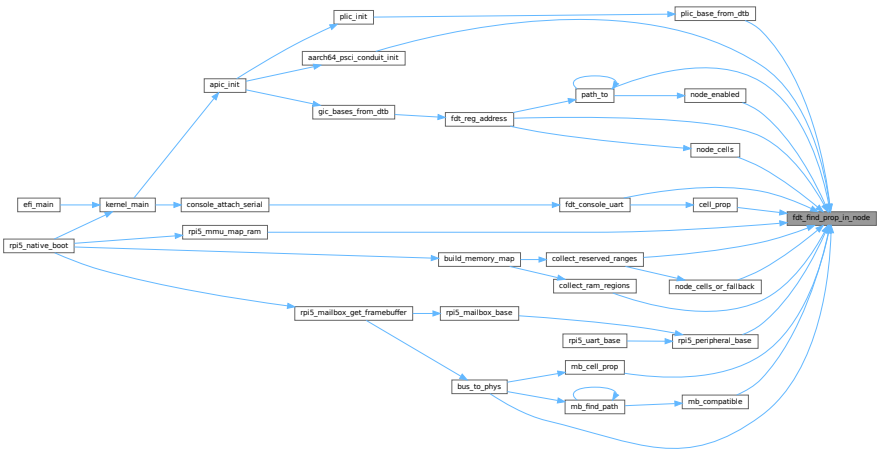
References be32(), FDT_NOP, FDT_PROP, fdt_valid(), name, fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and str_eq().

Referenced by aarch64_psci_conduit_init(), bus_to_phys(), cell_prop(), collect_ram_regions(), collect_reserved_ranges(), fdt_console_uart(), fdt_reg_address(), mb_cell_prop(), mb_compatible(), node_cells(), node_cells_or_fallback(), node_enabled(), path_to(), plic_base_from_dtb(), rpi5_mmu_map_ram(), and rpi5_peripheral_base().

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.1.8 fdt_next_child_node()

```
const void * fdt_next_child_node (
            const void * fdt,
            const void * parent,
            const void * prev_child )
```

Iterate the direct children of one node.

Pass `prev_child` as NULL to get the first child; pass a previous result back in to get the next one. Stops (returns NULL) once there are no more children. Skips over each child's own descendants correctly (so a child with grandchildren doesn't confuse the scan), but does not itself descend into them — only direct children of `parent` are ever returned.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
parent	Handle from one of the `fdt_find_node_by_*()` functions.
prev_child	NULL for the first child, or a handle previously returned by this function for `parent` to continue from.

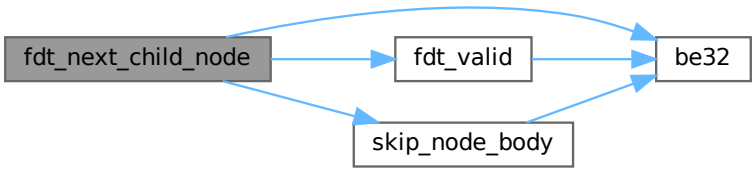
Returns

Handle to the next direct child, for use with `fdt_find_prop_in_node()` / `fdt_next_child_node()` only — or NULL once `parent`'s children are exhausted.

References be32(), FDT_BEGIN_NODE, FDT_NOP, FDT_PROP, fdt_valid(), fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and skip_node_body().

Referenced by collect_reserved_ranges(), mb_find_path(), and path_to().

Here is the call graph for this function:



Here is the caller graph for this function:

7.26.1.9 fdt_prop_u32()

```
int fdt_prop_u32 (
            const void * fdt,
            const char * name,
            uint32_t * out )
```

Read a single-cell (32-bit) property by name.

Convenience over `fdt_find_prop()` that also handles the big-endian conversion. Fails if the property is absent or not exactly 4 bytes.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
name	Property name, NUL-terminated.
out	Receives the host-order value on success; untouched on failure.

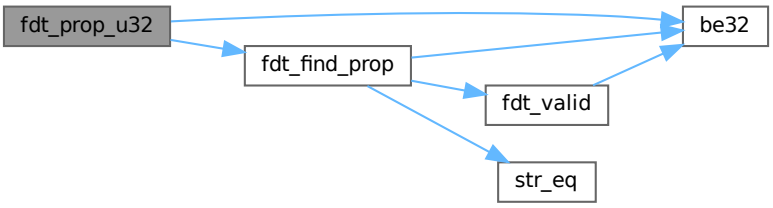
Returns

1 on success, 0 on failure.

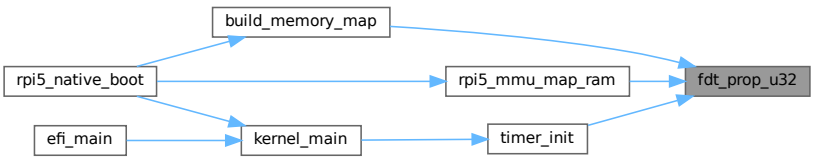
References be32(), fdt_find_prop(), and name.

Referenced by build_memory_map(), rpi5_mmu_map_ram(), and timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.1.10 fdt_reg_address()

```
int fdt_reg_address (
            const void * fdt,
            const char * compatible,
            uint32_t index,
            uint64_t * out )
```

The CPU's address for entry `index` of the `reg` of the first enabled node compatible with `compatible` (2026-10-14).

The entry is read in its parent's #address-cells/#size-cells and carried up through every bus's `ranges` to the root – an empty `ranges` keeps the address, a bus with no `ranges` has no way up. The Pi 5's GIC, for one, sits at 0x7fff9000 on /soc, which is 0x107fff9000 to the CPU.

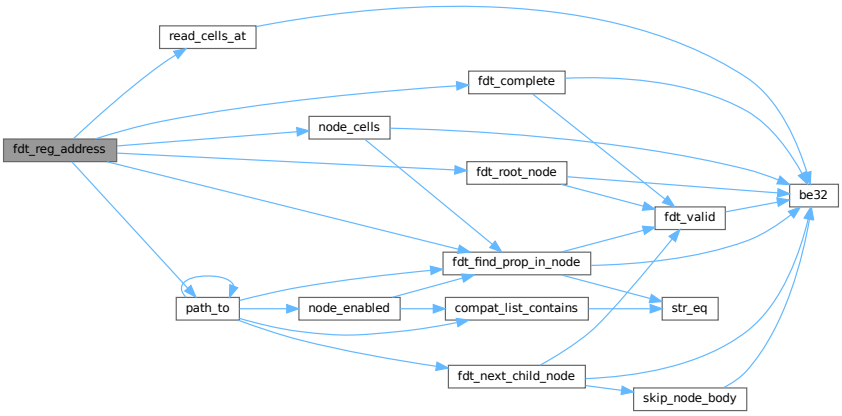
Returns

1 and the address in `out`, or 0 if there is no such node, entry or way up.

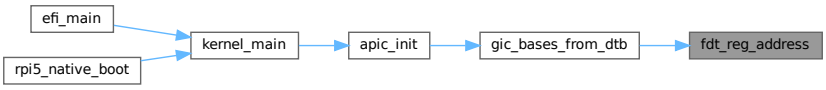
References entry, fdt_complete(), fdt_find_prop_in_node(), FDT_PATH_MAX, fdt_root_node(), node_cells(), path_to(), and read_cells_at().

Referenced by gic_bases_from_dtb().

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.1.11 fdt_root_node()

```
const void * fdt_root_node (
            const void * fdt )
```

The root node.

Parameters

fdt	Blob; checked with `fdt_valid()`.

Returns

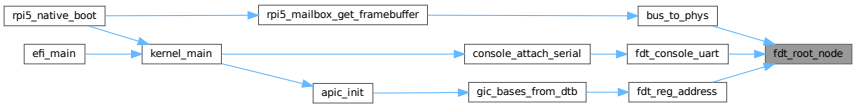
Handle to the root node, for `fdt_find_child()` and `fdt_find_prop_in_node()`, or NULL.

References be32(), FDT_BEGIN_NODE, FDT_NOP, fdt_valid(), fdt_header_t::off_dt_struct, and fdt_header_t::size_dt_struct.

Referenced by bus_to_phys(), fdt_console_uart(), and fdt_reg_address().

Here is the call graph for this function:

Here is the caller graph for this function:



7.26.1.12 fdt_valid()

```
int fdt_valid (
            const void * fdt )
```

fdt.h - Minimal flattened-devicetree reader

Just enough of the Devicetree Specification v0.4 ?5 to pull values out of the blob the UEFI firmware publishes under EFI_DTB_TABLE_GUID, or that a native (non-UEFI) boot entry passes directly. Read-only, no allocation, no tree construction — it walks the structure block each call, which is fine for the handful of boot-time lookups the kernel needs.

Deliberately not a general devicetree library. Added for punch-list item 0.3 (riscv64 timebase-frequency); extended (FABRIC-3.md ?IV.3/?V.3, 2026-09-04) with node-scoped lookup, for exactly the case this header originally flagged as a future need (item 0.6's aarch64 GIC) plus its real, concrete consumers as of this pass: the Raspberry Pi 5's UART/ mailbox register addresses (native boot, no ACPI) and the Milk-V Mars's real PLIC base address (currently hardcoded to QEMU-virt's own value, `arch/riscv64/plic.c`'s own doc comment already warned this isn't assumed stable across configurations).

Test whether `fdt` points at a valid flattened devicetree.

Checks the 0xd00dfeed magic and that the structure and strings blocks lie inside totalsize. Does not validate the token stream.

Parameters

fdt	Candidate blob; NULL is safe and returns 0.

Returns

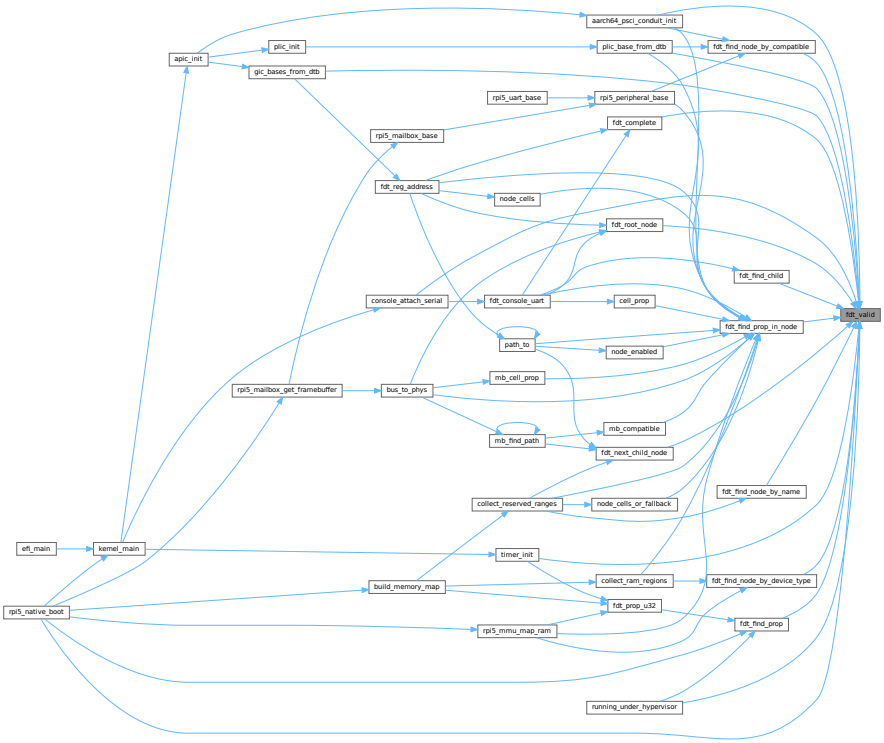
1 if the header is usable, 0 otherwise.

References be32(), FDT_MAGIC, fdt_header_t::magic, fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_strings, fdt_header_t::size_dt_struct, and fdt_header_t::totalsize.

Referenced by aarch64_psci_conduit_init(), console_attach_serial(), fdt_complete(), fdt_find_child(), fdt_find_node_by_compatible(), fdt_find_node_by_device_type(), fdt_find_node_by_name(), fdt_find_prop(), fdt_find_prop_in_node(), fdt_next_child_node(), fdt_root_node(), gic_bases_from_dtb(), plic_base_from_dtb(), rpi5_native_boot(), running_under_hypervisor(), and timer_init().

Here is the call graph for this function:

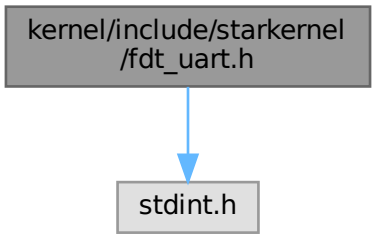
Here is the caller graph for this function:



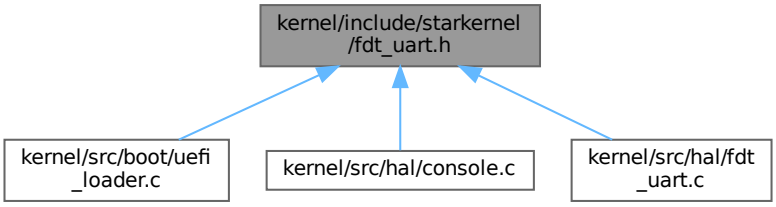
7.27 kernel/include/starkernel/fdt_uart.h File Reference

```
#include <stdint.h>
```

Include dependency graph for fdt_uart.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct fdt_uart_t

Enumerations

- enum { FDT_UART_NONE = 0 , FDT_UART_PL011 , FDT_UART_16550 }

  fdt_uart.h - the console's serial line, as a board's devicetree names it.

Functions

- int fdt_console_uart (const void *fdt, fdt_uart_t *out)

  Find the console's serial line in a devicetree.

7.27.1 Enumeration Type Documentation

7.27.1.1 anonymous enum

```
anonymous enum
```

fdt_uart.h - the console's serial line, as a board's devicetree names it.

A board has one serial line that is its console, and its devicetree says which: /chosen/stdout-path, either the path of a node or an alias for one (/aliases), with line settings after a colon that are not used here. The firmware that handed the devicetree over has already set that line up. This finds what kind of UART it is, where it is as the CPU addresses it, and how its registers are laid out.

Nothing is guessed. If the devicetree does not name a console, or names one this cannot place or has no driver for, there is no line.

Verified off the boards by tools/fdt_uart_test.c.

Enumerator

FDT_UART_NONE	no line
FDT_UART_PL011	ARM PL011: 32-bit registers, four bytes apart.
FDT_UART_16550	16550 and its kin (ns16550a, snps,dw-apb-uart), by memory

7.27.2 Function Documentation

7.27.2.1 fdt_console_uart()

```
int fdt_console_uart (
            const void * fdt,
            fdt_uart_t * out )
```

Find the console's serial line in a devicetree.

Parameters

fdt	Blob; NULL and what is not a devicetree are safe.
out	Receives the line; its kind is FDT_UART_NONE when 0 is returned.

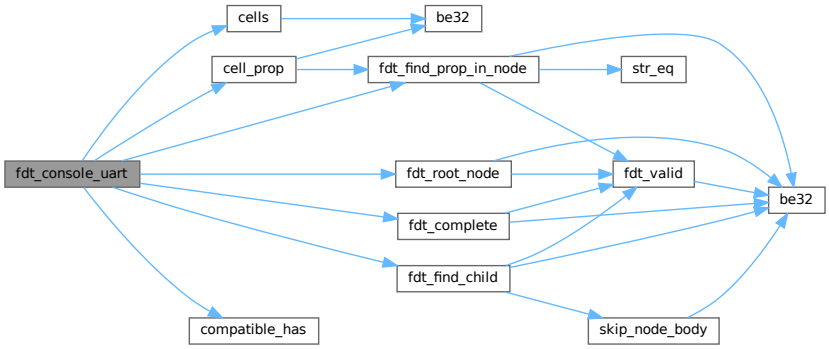
Returns

1 if the devicetree names a console UART that was found, placed and is of a kind listed above; 0 otherwise.

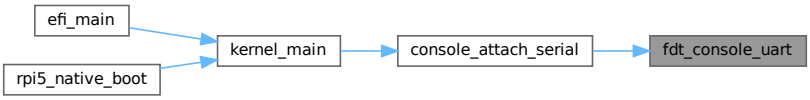
References fdt_uart_t::base, cell_prop(), cells(), compatible_has(), entry, fdt_complete(), fdt_find_child(), fdt_find_prop_in_node(), fdt_root_node(), FDT_UART_16550, FDT_UART_NONE, FDT_UART_PL011, fdt_uart_t::io_width, fdt_uart_t::kind, PATH_DEPTH, and fdt_uart_t::reg_shift.

Referenced by console_attach_serial().

Here is the call graph for this function:

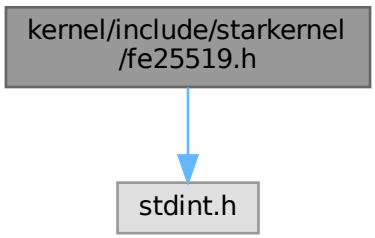


Here is the caller graph for this function:

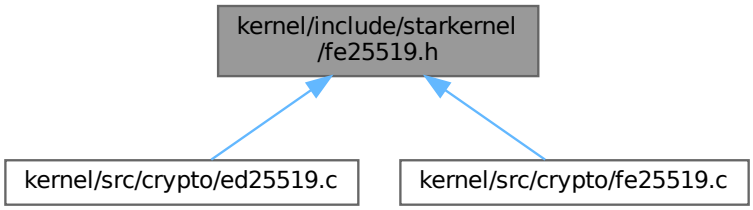


7.28 kernel/include/starkernel/fe25519.h File Reference

`#include <stdint.h>`
Include dependency graph for fe25519.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct fe25519

Functions

- void fe25519_0 (fe25519 *r)
- void fe25519_1 (fe25519 *r)
- void fe25519_copy (fe25519 *r, const fe25519 *a)
- void fe25519_add (fe25519 *r, const fe25519 *a, const fe25519 *b)
- void fe25519_sub (fe25519 *r, const fe25519 *a, const fe25519 *b)
- void fe25519_neg (fe25519 *r, const fe25519 *a)
- void fe25519_mul (fe25519 *r, const fe25519 *a, const fe25519 *b)
- void fe25519_sq (fe25519 *r, const fe25519 *a)
- void fe25519_invert (fe25519 *r, const fe25519 *a)
- void fe25519_mul_small (fe25519 *r, const fe25519 *a, uint32_t c)
- void fe25519_pack (uint8_t out[32], const fe25519 *a)
- void fe25519_unpack (fe25519 *r, const uint8_t in[32])
- int fe25519_eq (const fe25519 *a, const fe25519 *b)
- int fe25519_parity (const fe25519 *a)

7.28.1 Function Documentation

7.28.1.1 fe25519_0()

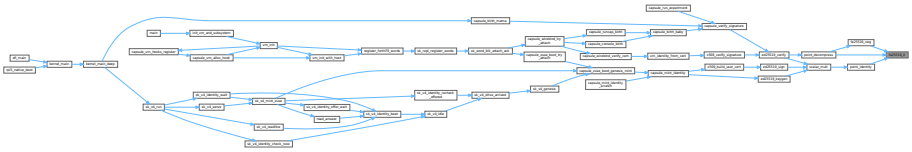
```
void fe25519_0 (
            fe25519 * r )
```

References memset(), and fe25519::v.

Referenced by fe25519_neg(), point_decompress(), and point_identity().

Here is the call graph for this function:

Here is the caller graph for this function:



7.28.1.2 fe25519_1()

```
void fe25519_1 (
            fe25519 * r )
```

References memset(), and fe25519::v.

Referenced by base_point(), ed25519_verify(), fe25519_pow(), point_decompress(), and point_identity().

Here is the call graph for this function:

Here is the caller graph for this function:

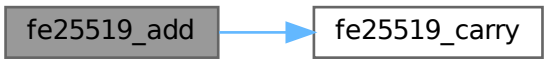
7.28.1.3 fe25519_add()

```
void fe25519_add (
              fe25519 * r,
              const fe25519 * a,
              const fe25519 * b )
```

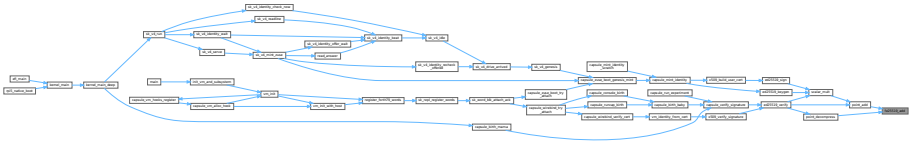
References fe25519_carry(), and fe25519::v.

Referenced by point_add(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:



7.28.1.4 fe25519_copy()

```
void fe25519_copy (
              fe25519 * r,
              const fe25519 * a )
```

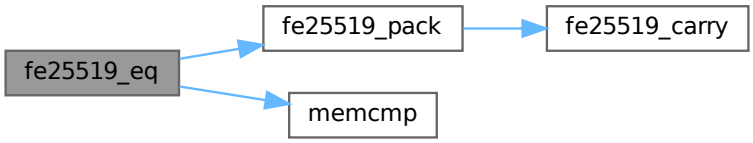
7.28.1.5 fe25519_eq()

```
int fe25519_eq (
              const fe25519 * a,
              const fe25519 * b )
```

References fe25519_pack(), and memcmp().

Referenced by point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

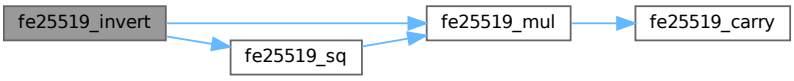
7.28.1.6 fe25519_invert()

```
void fe25519_invert (
            fe25519 * r,
            const fe25519 * a )
```

References fe25519_mul(), and fe25519_sq().

Referenced by point_compress(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

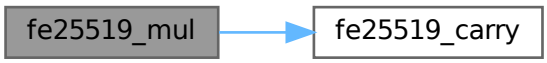
7.28.1.7 fe25519_mul()

```
void fe25519_mul (
             fe25519 * r,
             const fe25519 * a,
             const fe25519 * b )
```

References fe25519_carry(), MASK51, fe25519::v, and WBITS.

Referenced by fe25519_invert(), fe25519_pow(), fe25519_sq(), point_add(), point_compress(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

7.28.1.8 fe25519_mul_small()

```
void fe25519_mul_small (
             fe25519 * r,
             const fe25519 * a,
             uint32_t c )
```

References fe25519_carry(), MASK51, fe25519::v, and WBITS.

Here is the call graph for this function:

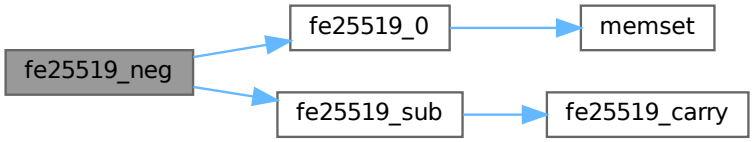
7.28.1.9 fe25519_neg()

```
void fe25519_neg (
            fe25519 * r,
            const fe25519 * a )
```

References fe25519_0(), fe25519_sub(), and zero.

Referenced by point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

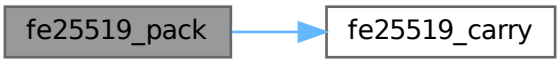
7.28.1.10 fe25519_pack()

```
void fe25519_pack (
            uint8_t out[32],
            const fe25519 * a )
```

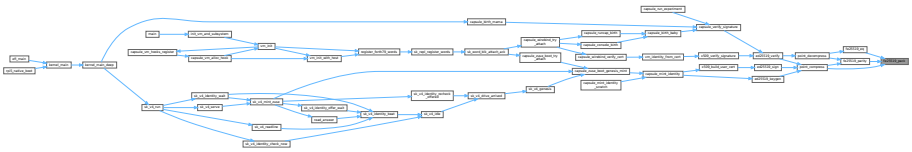
References fe25519_carry(), MASK51, fe25519::v, and WBITS.

Referenced by fe25519_eq(), fe25519_parity(), and point_compress().

Here is the call graph for this function:



Here is the caller graph for this function:



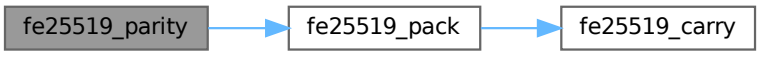
7.28.1.11 fe25519_parity()

```
int fe25519_parity (
            const fe25519 * a )
```

References fe25519_pack().

Referenced by point_compress(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

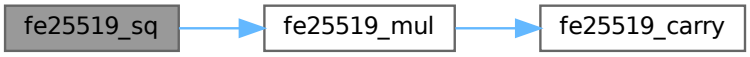
7.28.1.12 fe25519_sq()

```
void fe25519_sq (
            fe25519 * r,
            const fe25519 * a )
```

References fe25519_mul().

Referenced by fe25519_invert(), fe25519_pow(), point_add(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

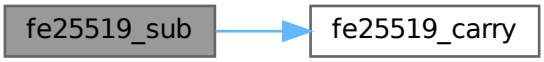
7.28.1.13 fe25519_sub()

```
void fe25519_sub (
            fe25519 * r,
            const fe25519 * a,
            const fe25519 * b )
```

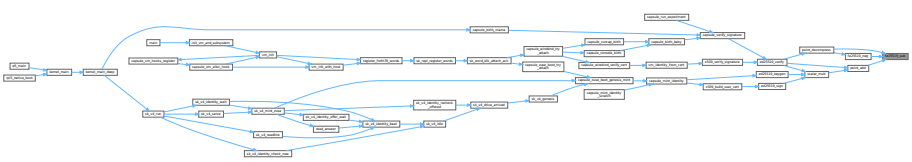
References fe25519_carry(), and fe25519::v.

Referenced by fe25519_neg(), point_add(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:



7.28.1.14 fe25519_unpack()

```
void fe25519_unpack (
            fe25519 * r,
            const uint8_t in[32] )
```

References MASK51, fe25519::v, and WBITS.

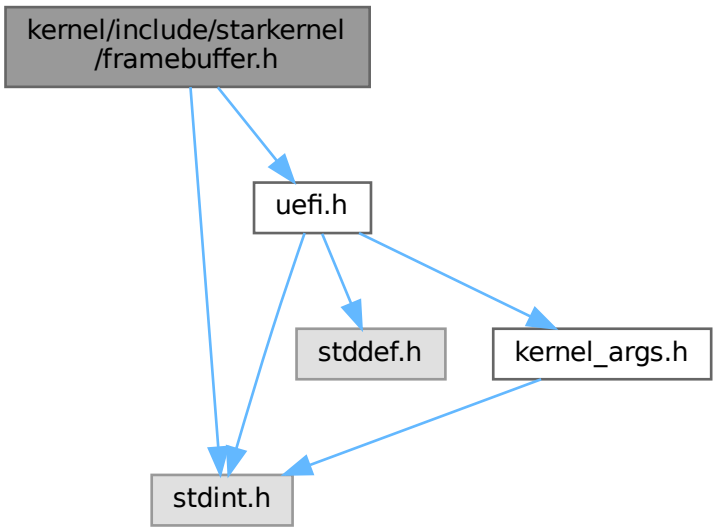
Referenced by base_point(), ed25519_verify(), point_add(), and point_decompress().

Here is the caller graph for this function:

7.29 kernel/include/starkernel/framebuffer.h File Reference

```
#include <stdint.h>
#include "uefi.h"
```

Include dependency graph for framebuffer.h:



This graph shows which files directly or indirectly include this file:

Macros

- #define FB_RGB(r, g, b) (((uint32_t)(r) << 16) | ((uint32_t)(g) << 8) | (uint32_t)(b))
- #define FB_R(c) (((uint32_t)(c) >> 16) & 0xFFu)
- #define FB_G(c) (((uint32_t)(c) >> 8) & 0xFFu)
- #define FB_B(c) ((uint32_t)(c) & 0xFFu)

Enumerations

- enum FbPixelFormat { FB_PIXEL_BGRX32 = 0 , FB_PIXEL_RGBX32 = 1 }

  framebuffer.h — UEFI GOP framebuffer driver interface

Functions

- const uint32_t FB_ANSI_PALETTE[16] __attribute__ ((visibility("hidden")))
- void fb_init (const FramebufferInfo *info, FbPixelFormat fmt)

  Initialize the framebuffer from UEFI bootloader GOP data.
- int fb_is_available (void)

  Returns 1 if the framebuffer has been successfully initialized.
- uint32_t fb_width (void)
- uint32_t fb_height (void)
- uint32_t fb_cell_w (void)

  Effective character cell width/height in pixels (8/16 × scale factor).
- uint32_t fb_cell_h (void)
- void fb_put_pixel (uint32_t x, uint32_t y, uint32_t rgb)

  Write a single pixel.
- void fb_fill_rect (uint32_t x, uint32_t y, uint32_t w, uint32_t h, uint32_t rgb)

  Fill a rectangle with a solid color.
- void fb_draw_glyph (uint32_t px, uint32_t py, uint8_t ch, uint32_t fg, uint32_t bg)

  Draw one 8×16 character glyph at pixel position (px, py).
- void fb_draw_orientation_test (void)

  One-time boot diagnostic (FABRIC-0.md item 4.3.1): fills each raster corner with a distinct solid color so a screendump reveals orientation.
- void fb_scroll_rows (uint32_t pixel_rows, uint32_t bg)

  Scroll the whole framebuffer up by `pixel_rows` pixel rows.
- void fb_scroll_rect (uint32_t x, uint32_t y, uint32_t w, uint32_t h, uint32_t pixel_rows, uint32_t bg)

  Scroll a sub-rectangle of the framebuffer up by `pixel_rows` pixel rows (FABRIC-0.md item 4.4t: box-confined REPL scrolling).

7.29.1 Macro Definition Documentation

7.29.1.1 FB_B

```
#define FB_B(
             c ) ( (uint32_t)(c) & 0xFFu)
```

7.29.1.2 FB_G

```
#define FB_G(
             c ) (((uint32_t)(c) >> 8) & 0xFFu)
```

7.29.1.3 FB_R

```
#define FB_R(
              c ) (((uint32_t)(c) >> 16) & 0xFFu)
```

7.29.1.4 FB_RGB

```
#define FB_RGB(
              r,
              g,
              b )  (((uint32_t)(r) << 16) | ((uint32_t)(g) << 8) | (uint32_t)(b))
```

7.29.2 Enumeration Type Documentation

7.29.2.1 FbPixelFormat

```
enum FbPixelFormat
```

framebuffer.h — UEFI GOP framebuffer driver interface

Pixel format: BGRX32 (PixelBlueGreenRedReserved8BitPerColor), the dominant format for QEMU virt GOP. RGBX32 is also supported.

Color values throughout this API are packed as 0x00RRGGBB.

Enumerator

FB_PIXEL_BGRX32	
FB_PIXEL_RGBX32	

7.29.3 Function Documentation

7.29.3.1 __attribute__()

```
const uint32_t FB_ANSI_PALETTE[16] __attribute__ (
             (visibility("hidden"))  )  [extern]
```

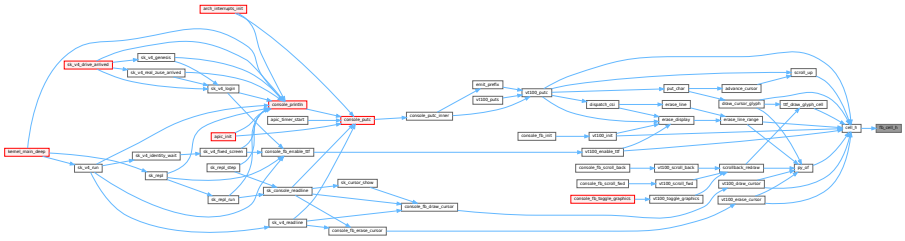
7.29.3.2 fb_cell_h()

```
uint32_t fb_cell_h (
             void  )
```

References g_fb, and FbState::scale.

Referenced by cell_h().

Here is the caller graph for this function:



7.29.3.3 fb_cell_w()

```
uint32_t fb_cell_w (
            void  )
```

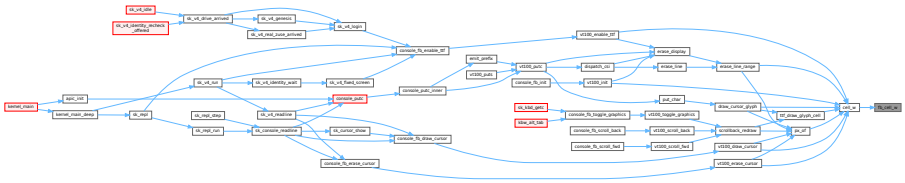
Effective character cell width/height in pixels (8/16 × scale factor).

Effective character cell width/height in pixels (8/16 × scale factor).

References g_fb, and FbState::scale.

Referenced by cell_w().

Here is the caller graph for this function:



7.29.3.4 fb_draw_glyph()

```
void fb_draw_glyph (
            uint32_t px,
            uint32_t py,
            uint8_t ch,
            uint32_t fg,
            uint32_t bg )
```

Draw one 8×16 character glyph at pixel position (px, py).

fg / bg are packed 0x00RRGGBB colors.

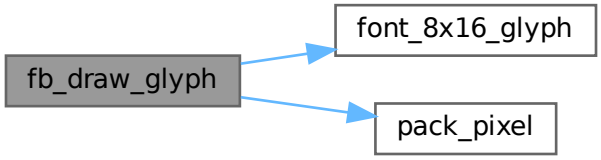
Draw one 8×16 character glyph at pixel position (px, py).

Scales by g_fb.scale so the same font works at any resolution.

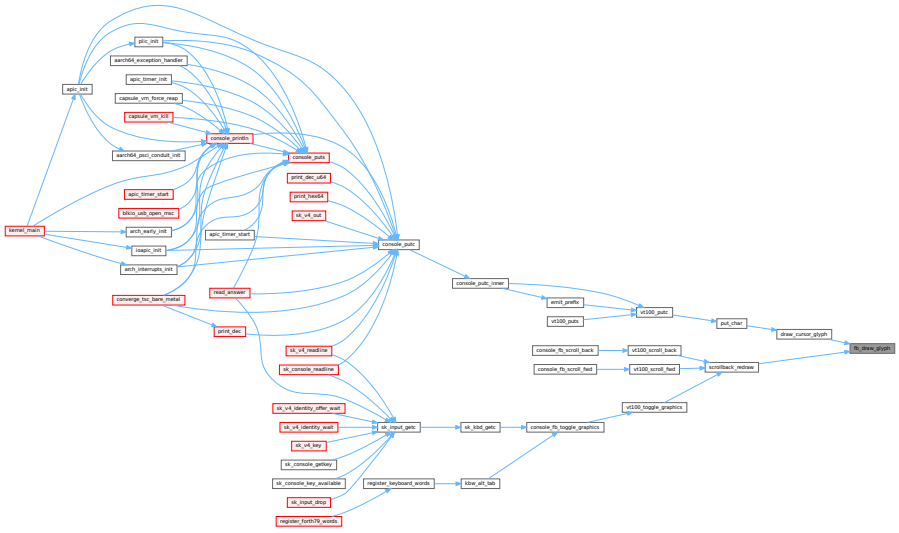
References FbState::base, font_8x16_glyph(), g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::scale, FbState::stride, and FbState::width.

Referenced by draw_cursor_glyph(), and scrollback_redraw().

Here is the call graph for this function:



Here is the caller graph for this function:



7.29.3.5 fb_draw_orientation_test()

```
void fb_draw_orientation_test (
            void  )
```

One-time boot diagnostic (FABRIC-0.md item 4.3.1): fills each raster corner with a distinct solid color so a screen-dump reveals orientation.

Not part of the Console drawing fabric – diagnostic-only.

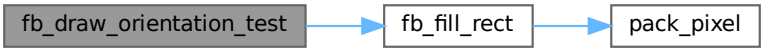
One-time boot diagnostic (FABRIC-0.md item 4.3.1): fills each raster corner with a distinct solid color so a screen-dump reveals orientation.

Fills each raster corner with a distinct solid color so a screendump immediately reveals whether the image is flipped or rotated. Not part of the Console drawing fabric itself – diagnostic-only, analogous to item 4.2's self-test scaffolding.

Corner colors (raster/top-left-origin coordinates, pre-Cartesian-flip): top-left RED top-right GREEN bottom-left BLUE bottom-right YELLOW

References fb_fill_rect(), FB_RGB, g_fb, FbState::height, FbState::ready, and FbState::width.

Here is the call graph for this function:



7.29.3.6 fb_fill_rect()

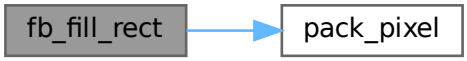
```
void fb_fill_rect (
            uint32_t x,
            uint32_t y,
            uint32_t w,
            uint32_t h,
            uint32_t rgb )
```

Fill a rectangle with a solid color.

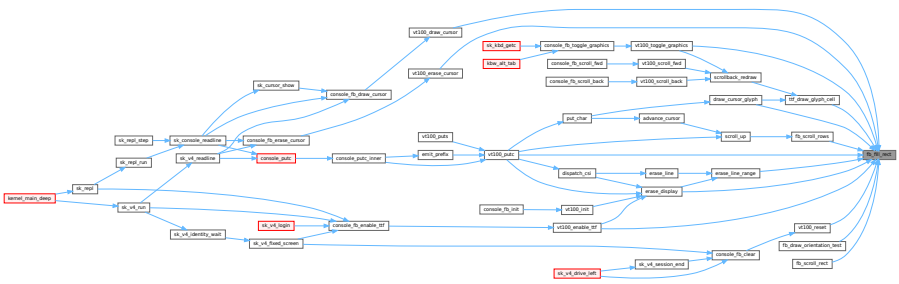
References FbState::base, g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::stride, and FbState::width.

Referenced by draw_cursor_glyph(), erase_display(), erase_line_range(), fb_draw_orientation_test(), fb_scroll_rect(), fb_scroll_rows(), ttf_draw_glyph_cell(), vt100_draw_cursor(), vt100_enable_ttf(), vt100_erase_cursor(), vt100_putc(), vt100_reset(), and vt100_toggle_graphics().

Here is the call graph for this function:



Here is the caller graph for this function:



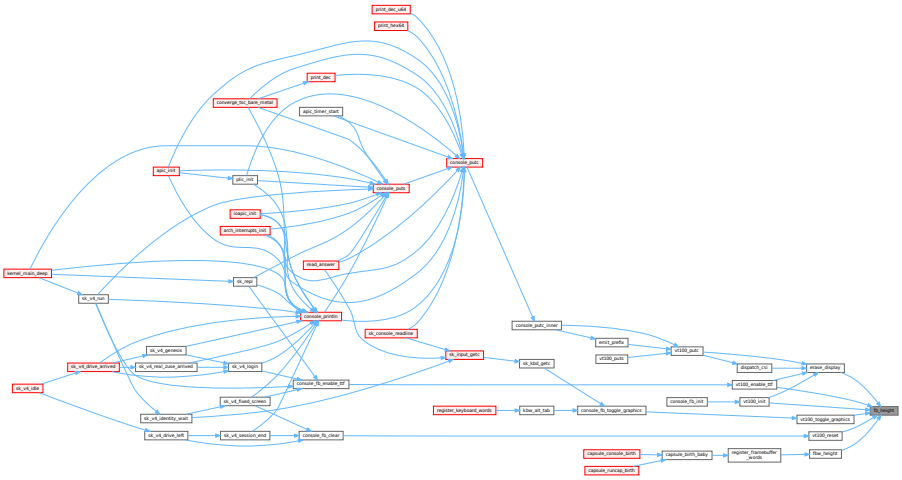
7.29.3.7 fb_height()

```
uint32_t fb_height (
            void  )
```

References g_fb, and FbState::height.

Referenced by erase_display(), fbw_height(), vt100_enable_ttf(), vt100_init(), vt100_reset(), and vt100_toggle_graphics().

Here is the caller graph for this function:



7.29.3.8 fb_init()

```
void fb_init (
            const FramebufferInfo * info,
            FbPixelFormat fmt )
```

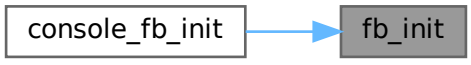
Initialize the framebuffer from UEFI bootloader GOP data.

Must be called before any other fb_∗ function. fmt: pixel layout reported by GOP (default FB_PIXEL_BGRX32).

References FramebufferInfo::base, FbState::base, FbState::fmt, g_fb, FramebufferInfo::height, FbState::height, FramebufferInfo::pixels_per_scanline, FbState::ready, FbState::scale, FbState::stride, FramebufferInfo::width, and FbState::width.

Referenced by console_fb_init().

Here is the caller graph for this function:



7.29.3.9 fb_is_available()

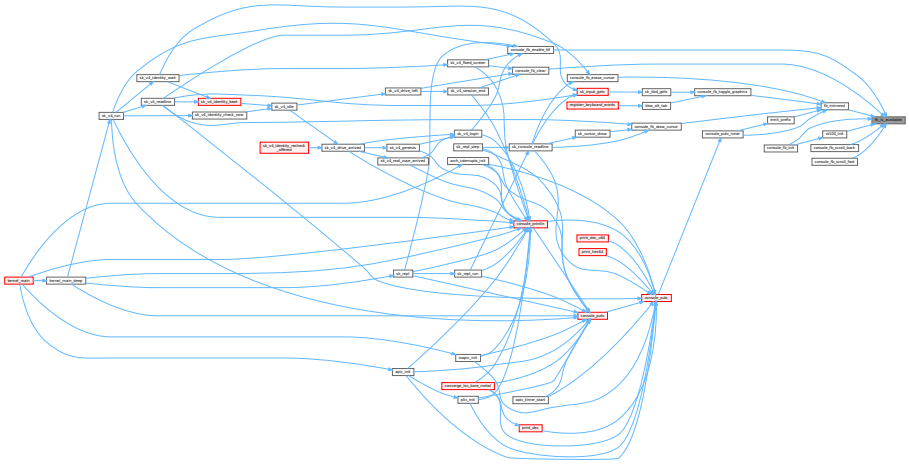
```
int fb_is_available (
            void  )
```

Returns 1 if the framebuffer has been successfully initialized.

References g_fb, and FbState::ready.

Referenced by console_fb_clear(), console_fb_enable_ttf(), console_fb_init(), console_fb_scroll_back(), console_fb_scroll_fwd(), fb_mirrored(), and vt100_init().

Here is the caller graph for this function:



7.29.3.10 fb_put_pixel()

```
void fb_put_pixel (
            uint32_t x,
            uint32_t y,
            uint32_t rgb )
```

Write a single pixel.

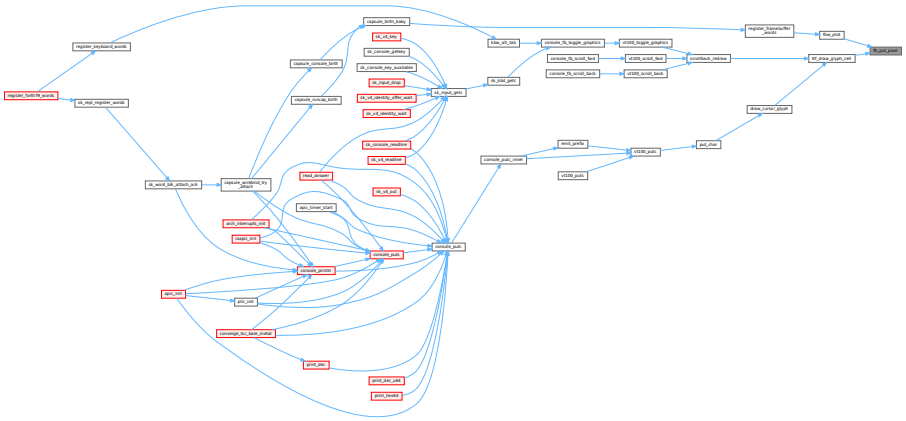
Out-of-bounds writes are silently ignored.

References FbState::base, g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::stride, and FbState::width.

Referenced by fbw_plot(), and ttf_draw_glyph_cell().

Here is the call graph for this function:

Here is the caller graph for this function:



7.29.3.11 fb_scroll_rect()

```
void fb_scroll_rect (
            uint32_t x,
            uint32_t y,
            uint32_t w,
            uint32_t h,
            uint32_t pixel_rows,
            uint32_t bg )
```

Scroll a sub-rectangle of the framebuffer up by `pixel_rows` pixel rows (FABRIC-0.md item 4.4t: box-confined REPL scrolling).

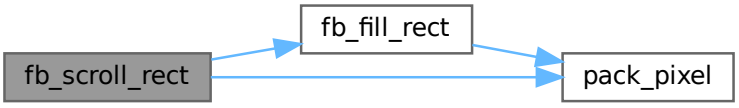
Unlike fb_scroll_rows() (whole-framebuffer), this is bounded to [x, x+w) x [y, y+h). Both take an explicit pixel-row count so callers with a non-8x16 cell height (e.g. TTF mode) pass their own cell height directly. Pixels outside the rect are untouched. The vacated rows at the bottom of the rect are filled with bg.

Scroll a sub-rectangle of the framebuffer up by `pixel_rows` pixel rows (FABRIC-0.md item 4.4t: box-confined REPL scrolling).

Same non-volatile bulk-copy rationale as fb_scroll_rows() above, but bounded to a caller-supplied rectangle instead of the whole framebuffer – needed once REPL text is confined to the 4.4o CANVAS box, since a whole-framebuffer scroll would drag pixels from outside the box (e.g. a future 4.4r border) into the text area. Takes pixel_rows directly rather than char_rows ∗ a hardcoded cell height, since TTF mode's cell height (24px) differs from bitmap mode's (16px ∗ scale).

References FbState::base, fb_fill_rect(), g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::stride, and FbState::width.

Here is the call graph for this function:



7.29.3.12 fb_scroll_rows()

```
void fb_scroll_rows (
            uint32_t pixel_rows,
            uint32_t bg )
```

Scroll the whole framebuffer up by `pixel_rows` pixel rows.

The vacated rows at the bottom are filled with bg. Takes an explicit pixel-row count (not a hardcoded 8x16-cell assumption) so callers with a non-8x16 cell height (e.g. TTF mode, 24px) pass their own cell height directly – same convention fb_scroll_rect() below already uses, for the same reason (a caller-computed char_rows $*$ fixed-16px assumption drifts out of sync with the text model's own row height in TTF mode, and that drift compounds with every scroll).

Scroll the whole framebuffer up by `pixel_rows` pixel rows.

The vacated rows at the bottom are filled with bg.

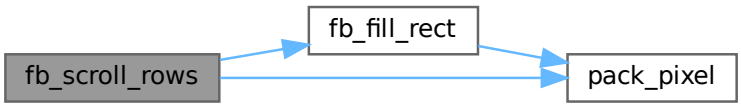
Takes a pixel-row count directly rather than a character-row count scaled by a hardcoded 16px cell height: the caller (vt100.c's scroll_up()) knows the actual active glyph cell height (16 $\times$ scale for bitmap mode, 24 for TTF mode), and a mismatch between the height this function scrolls by and the height the text model actually draws rows at compounds by the gap on every single scroll – a few scrolls barely show it, but enough scrolls (or scrolling quickly) accumulates visible pixel overlap between rows.

Copies through non-volatile pointers (FABRIC-0.md item 4.4g performance fix, 2026-08-11): the GOP framebuffer is mapped write-back, not cache-disabled MMIO (vmm.c:350-363 – "QEMU's VGA emulation is coherent and UC-mapping causes #GP"), so it behaves as ordinary RAM and does not need per-word volatile access. `g_fb.base` stays `volatile` for single-pixel writes elsewhere in this file (draw calls that must not be reordered relative to other framebuffer state), but that qualifier forced this copy into one un-batchable 4-byte access per element with no compiler vectorization – catastrophic once boot output exceeds one screen ($\sim$10^10 volatile accesses for an 18,000-line boot transcript under TCG). Casting away volatile here for the bulk copy/fill lets the compiler batch these loops normally.

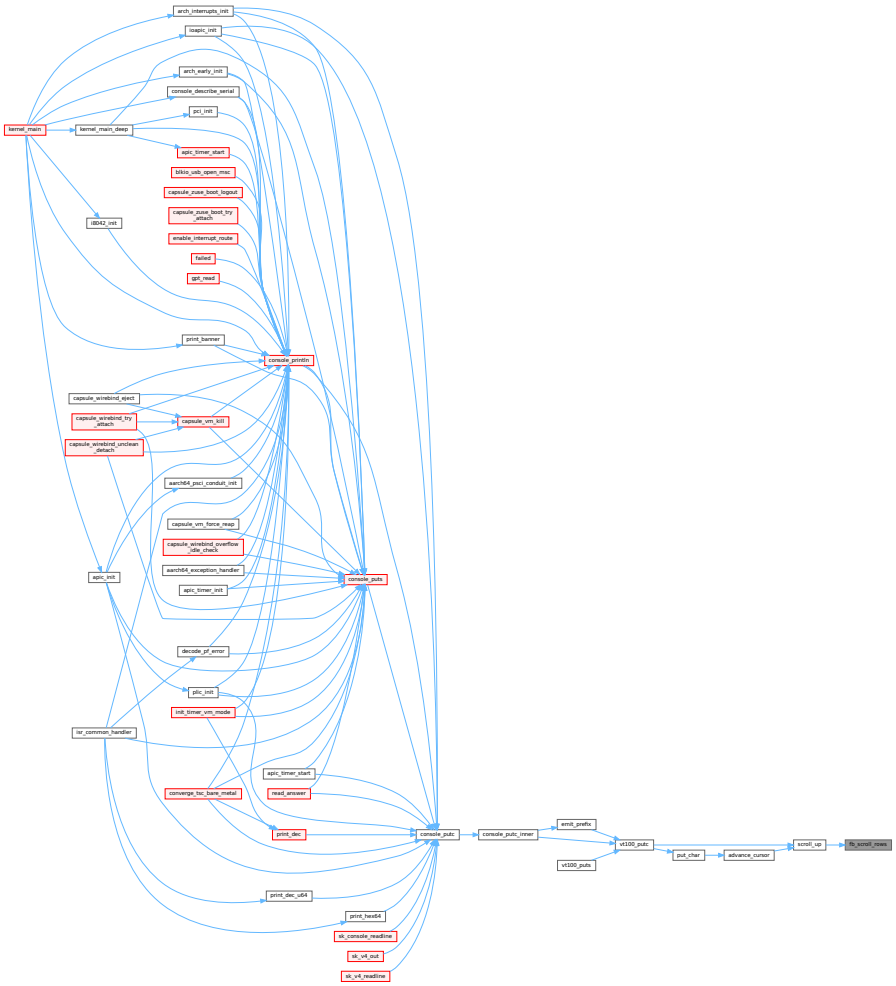
References FbState::base, fb_fill_rect(), g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::stride, and FbState::width.

Referenced by scroll_up().

Here is the call graph for this function:



Here is the caller graph for this function:



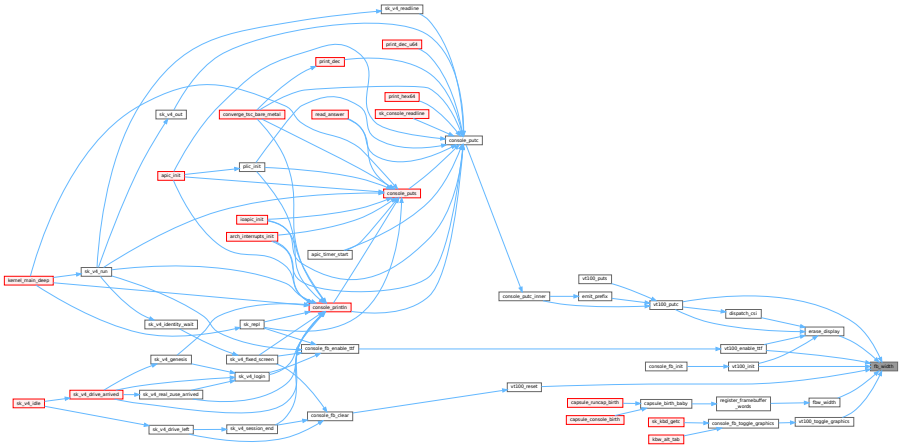
7.29.3.13 fb_width()

```
uint32_t fb_width (
            void  )
```

References g_fb, and FbState::width.

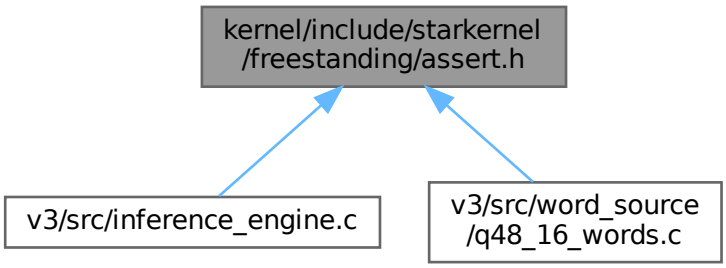
Referenced by erase_display(), fbw_width(), vt100_enable_ttf(), vt100_init(), vt100_putc(), vt100_reset(), and vt100_toggle_graphics().

Here is the caller graph for this function:



7.30 kernel/include/starkernel/freestanding/assert.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define assert(expr) ((void)(expr))
- #define static_assert _Static_assert

7.30.1 Macro Definition Documentation

7.30.1.1 assert

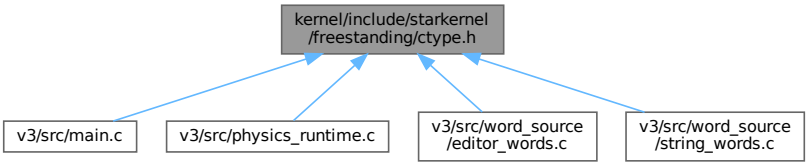
```
#define assert(
              expr ) ((void)(expr))
```

7.30.1.2 static_assert

```
#define static_assert _Static_assert
```

7.31 kernel/include/starkernel/freestanding/ctype.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- static int isdigit (int c)
- static int isspace (int c)
- static int isalpha (int c)
- static int isalnum (int c)
- static int isupper (int c)
- static int islower (int c)
- static int isprint (int c)
- static int isxdigit (int c)
- static int toupper (int c)
- static int tolower (int c)
- static int ispunct (int c)
- static int iscntrl (int c)

7.31.1 Function Documentation

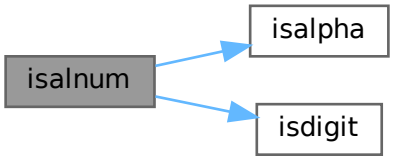
7.31.1.1 isalnum()

```
static int isalnum (
            int c )  [inline], [static]
```

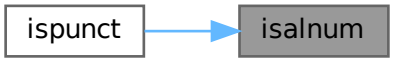
References isalpha(), and isdigit().

Referenced by ispunct().

Here is the call graph for this function:



Here is the caller graph for this function:

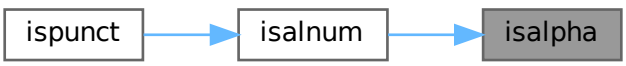


7.31.1.2 isalpha()

```
static int isalpha (
            int c )  [inline], [static]
```

Referenced by isalnum().

Here is the caller graph for this function:



7.31.1.3 iscntrl()

```
static int iscntrl (
            int c )  [inline], [static]
```

7.31.1.4 isdigit()

```
static int isdigit (
            int c )  [inline], [static]
```

Referenced by editor_word_edit(), isalnum(), and isxdigit().

Here is the caller graph for this function:

7.31.1.5 islower()

```
static int islower (
            int c )  [inline], [static]
```

Referenced by toupper().

Here is the caller graph for this function:

7.31.1.6 isprint()

```
static int isprint (
            int c )  [inline], [static]
```

Referenced by ispunct().

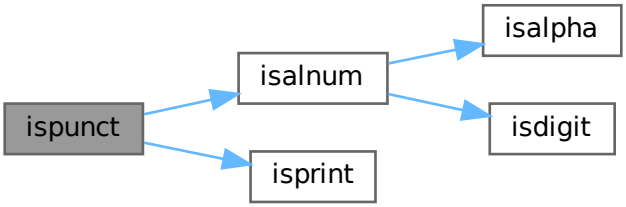
Here is the caller graph for this function:

7.31.1.7 ispunct()

```
static int ispunct (
            int c )  [inline], [static]
```

References isalnum(), and isprint().

Here is the call graph for this function:



7.31.1.8 isspace()

```
static int isspace (
            int c )  [inline], [static]
```

Referenced by editor_word_edit().

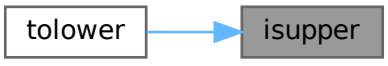
Here is the caller graph for this function:

7.31.1.9 isupper()

```
static int isupper (
            int c )  [inline], [static]
```

Referenced by tolower().

Here is the caller graph for this function:



7.31.1.10 isxdigit()

```
static int isxdigit (
            int c )  [inline], [static]
```

References isdigit().

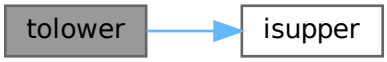
Here is the call graph for this function:

7.31.1.11 tolower()

```
static int tolower (
            int c )  [inline], [static]
```

References isupper().

Here is the call graph for this function:



7.31.1.12 toupper()

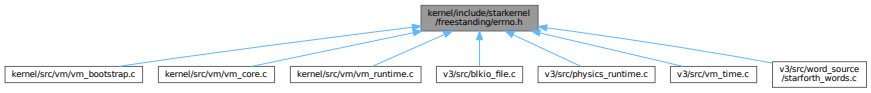
```
static int toupper (
            int c )  [inline], [static]
```

References islower().

Here is the call graph for this function:

7.32 kernel/include/starkernel/freestanding/errno.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define errno _freestanding_errno
- #define EPERM 1
- #define ENOENT 2
- #define EINVAL 22
- #define ENOMEM 12

Variables

- static int _freestanding_errno = 0

7.32.1 Macro Definition Documentation

7.32.1.1 EINVAL

```
#define EINVAL 22
```

7.32.1.2 ENOENT

```
#define ENOENT 2
```

7.32.1.3 ENOMEM

```
#define ENOMEM 12
```

7.32.1.4 EPERM

```
#define EPERM 1
```

7.32.1.5 errno

```
#define errno _freestanding_errno
```

7.32.2 Variable Documentation

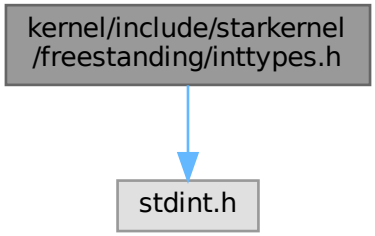
7.32.2.1 _freestanding_errno

```
int _freestanding_errno = 0  [static]
```

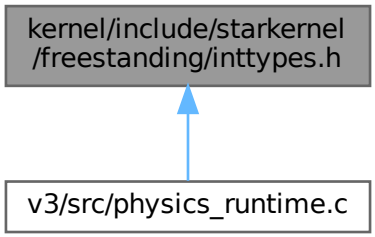
7.33 kernel/include/starkernel/freestanding/inttypes.h File Reference

```
#include <stdint.h>
```

Include dependency graph for inttypes.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define PRId8 "d"
- #define PRId16 "d"
- #define PRId32 "d"
- #define PRId64 "lld"
- #define PRIu8 "u"

- #define PRIu16 "u"
- #define PRIu32 "u"
- #define PRIu64 "llu"
- #define PRIx8 "x"
- #define PRIx16 "x"
- #define PRIx32 "x"
- #define PRIx64 "llx"
- #define PRIX8 "X"
- #define PRIX16 "X"
- #define PRIX32 "X"
- #define PRIX64 "llX"
- #define PRIi8 "d"
- #define PRIi16 "d"
- #define PRIi32 "d"
- #define PRIi64 "lld"

7.33.1 Macro Definition Documentation

7.33.1.1 PRId16

```
#define PRId16 "d"
```

7.33.1.2 PRId32

```
#define PRId32 "d"
```

7.33.1.3 PRId64

```
#define PRId64 "lld"
```

7.33.1.4 PRId8

```
#define PRId8 "d"
```

7.33.1.5 PRIi16

```
#define PRIi16 "d"
```

7.33.1.6 PRIi32

```
#define PRIi32 "d"
```

7.33.1.7 PRIi64

```
#define PRIi64 "lld"
```

7.33.1.8 PRIi8

```
#define PRIi8 "d"
```

7.33.1.9 PRIu16

```
#define PRIu16 "u"
```

7.33.1.10 PRIu32

```
#define PRIu32 "u"
```

7.33.1.11 PRIu64

```
#define PRIu64 "llu"
```

7.33.1.12 PRIu8

```
#define PRIu8 "u"
```

7.33.1.13 PRIx16

```
#define PRIx16 "x"
```

7.33.1.14 PRIX16

```
#define PRIX16 "X"
```

7.33.1.15 PRIx32

```
#define PRIx32 "x"
```

7.33.1.16 PRIX32

```
#define PRIX32 "X"
```

7.33.1.17 PRIx64

```
#define PRIx64 "llx"
```

7.33.1.18 PRIX64

```
#define PRIX64 "llX"
```

7.33.1.19 PRIx8

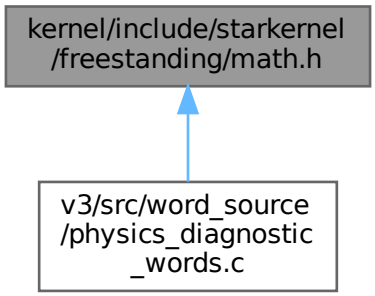
```
#define PRIx8 "x"
```

7.33.1.20 PRIX8

```
#define PRIX8 "X"
```

7.34 kernel/include/starkernel/freestanding/math.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define HUGE_VAL __builtin_huge_val()
- #define NAN __builtin_nanf("")
- #define INFINITY __builtin_inff()

Functions

- double sqrt (double x)
- double pow (double base, double exp)
- double fabs (double x)
- double ceil (double x)
- double floor (double x)
- double log (double x)
- double exp (double x)
- double sin (double x)
- double cos (double x)
- double atan2 (double y, double x)

7.34.1 Macro Definition Documentation

7.34.1.1 HUGE_VAL

```
#define HUGE_VAL __builtin_huge_val()
```

7.34.1.2 INFINITY

```
#define INFINITY __builtin_inff()
```

7.34.1.3 NAN

```
#define NAN __builtin_nanf("")
```

7.34.2 Function Documentation

7.34.2.1 atan2()

```
double atan2 (
            double y,
            double x )
```

7.34.2.2 ceil()

```
double ceil (
            double x )
```

7.34.2.3 cos()

```
double cos (
            double x )
```

7.34.2.4 exp()

```
double exp (
            double x )
```

Referenced by fe25519_pow().

Here is the caller graph for this function:

7.34.2.5 fabs()

```
double fabs (
            double x )
```

7.34.2.6 floor()

```
double floor (
            double x )
```

7.34.2.7 log()

```
double log (
            double x )
```

7.34.2.8 pow()

```
double pow (
            double base,
            double exp )
```

7.34.2.9 sin()

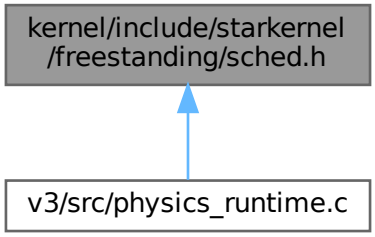
```
double sin (
            double x )
```

7.34.2.10 sqrt()

```
double sqrt (
            double x )
```

7.35 kernel/include/starkernel/freestanding/sched.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- static int sched_yield (void)

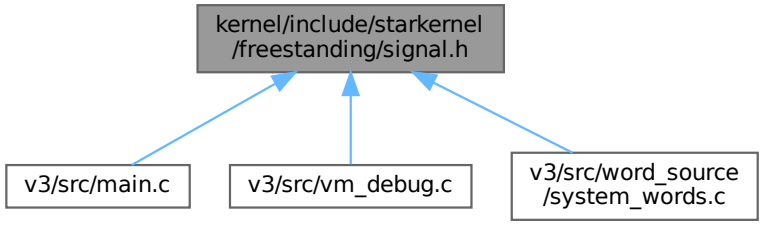
7.35.1 Function Documentation

7.35.1.1 sched_yield()

```
static int sched_yield (
            void  )  [inline], [static]
```

7.36 kernel/include/starkernel/freestanding/signal.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct sigaction

Macros

- #define SIGINT 2
- #define SIGABRT 6
- #define SIGSEGV 11
- #define SIGTERM 15
- #define SIG_DFL ((sighandler_t)0)
- #define SIG_IGN ((sighandler_t)1)
- #define SA_RESETHAND 0

Typedefs

- typedef void(∗ sighandler_t) (int)

Functions

- static sighandler_t signal (int sig, sighandler_t handler)
- static int sigaction (int sig, const struct sigaction *act, struct sigaction *old)

7.36.1 Macro Definition Documentation

7.36.1.1 SA_RESETHAND

```
#define SA_RESETHAND 0
```

7.36.1.2 SIG_DFL

```
#define SIG_DFL ((sighandler_t)0)
```

7.36.1.3 SIG_IGN

```
#define SIG_IGN ((sighandler_t)1)
```

7.36.1.4 SIGABRT

```
#define SIGABRT 6
```

7.36.1.5 SIGINT

```
#define SIGINT 2
```

7.36.1.6 SIGSEGV

```
#define SIGSEGV 11
```

7.36.1.7 SIGTERM

```
#define SIGTERM 15
```

7.36.2 Typedef Documentation

7.36.2.1 sighandler_t

```
typedef void(* sighandler_t) (int)
```

7.36.3 Function Documentation

7.36.3.1 sigaction()

```
static int sigaction (
            int sig,
            const struct sigaction * act,
            struct sigaction * old )  [inline], [static]
```

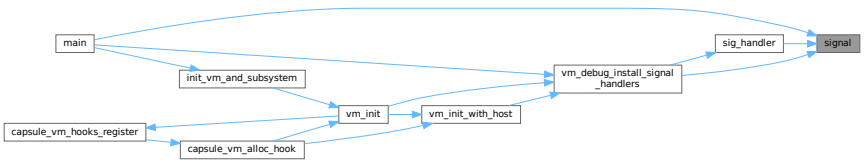
7.36.3.2 signal()

```
static sighandler_t signal (
            int sig,
            sighandler_t handler )  [inline], [static]
```

References SIG_DFL.

Referenced by main(), sig_handler(), and vm_debug_install_signal_handlers().

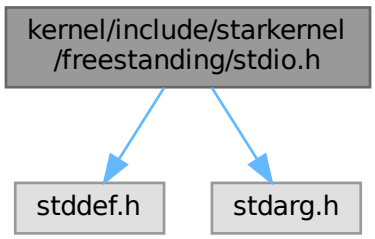
Here is the caller graph for this function:



7.37 kernel/include/starkernel/freestanding/stdio.h File Reference

```
#include <stddef.h>
#include <stdarg.h>
```

Include dependency graph for stdio.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct FILE

Macros

- #define stderr ((FILE *)0)
- #define stdout ((FILE *)1)
- #define stdin ((FILE *)2)
- #define SEEK_SET 0
- #define SEEK_CUR 1
- #define SEEK_END 2

Functions

- int printf (const char *fmt,...)
- int fprintf (FILE *stream, const char *fmt,...)
- int vfprintf (FILE *stream, const char *fmt, va_list args)
- int snprintf (char *buf, size_t n, const char *fmt,...)
- int vsnprintf (char *buf, size_t n, const char *fmt, va_list args)
- int puts (const char *s)
- int fputs (const char *s, FILE *stream)
- int fflush (FILE *stream)
- int putchar (int c)
- int fseek (FILE *stream, long offset, int whence)
- long ftell (FILE *stream)
- int sscanf (const char *str, const char *fmt,...)
- int getchar (void)
- int fgetc (FILE *stream)
- int fputc (int c, FILE *stream)
- char * fgets (char *s, int n, FILE *stream)
- FILE * fopen (const char *path, const char *mode)
- int fclose (FILE *stream)
- size_t fread (void *ptr, size_t size, size_t nmemb, FILE *stream)
- size_t fwrite (const void *ptr, size_t size, size_t nmemb, FILE *stream)
- int fscanf (FILE *stream, const char *fmt,...)
- void rewind (FILE *stream)

7.37.1 Macro Definition Documentation

7.37.1.1 SEEK_CUR

```
#define SEEK_CUR 1
```

7.37.1.2 SEEK_END

```
#define SEEK_END 2
```

7.37.1.3 SEEK_SET

```
#define SEEK_SET 0
```

7.37.1.4 stderr

```
#define stderr ((FILE *)0)
```

7.37.1.5 stdin

```
#define stdin ((FILE *)2)
```

7.37.1.6 stdout

```
#define stdout ((FILE *)1)
```

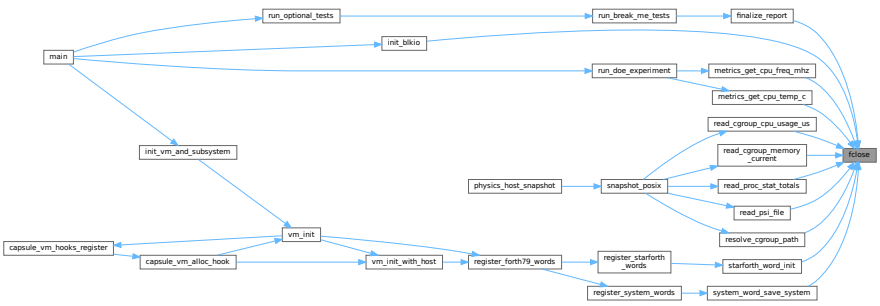
7.37.2 Function Documentation

7.37.2.1 fclose()

```
int fclose (
            FILE * stream )
```

Referenced by finalize_report(), init_blkio(), metrics_get_cpu_freq_mhz(), metrics_get_cpu_temp_c(), read_cgroup_cpu_usage_us(), read_cgroup_memory_current(), read_proc_stat_totals(), read_psi_file(), resolve_cgroup_path(), starforth_word_init(), and system_word_save_system().

Here is the caller graph for this function:

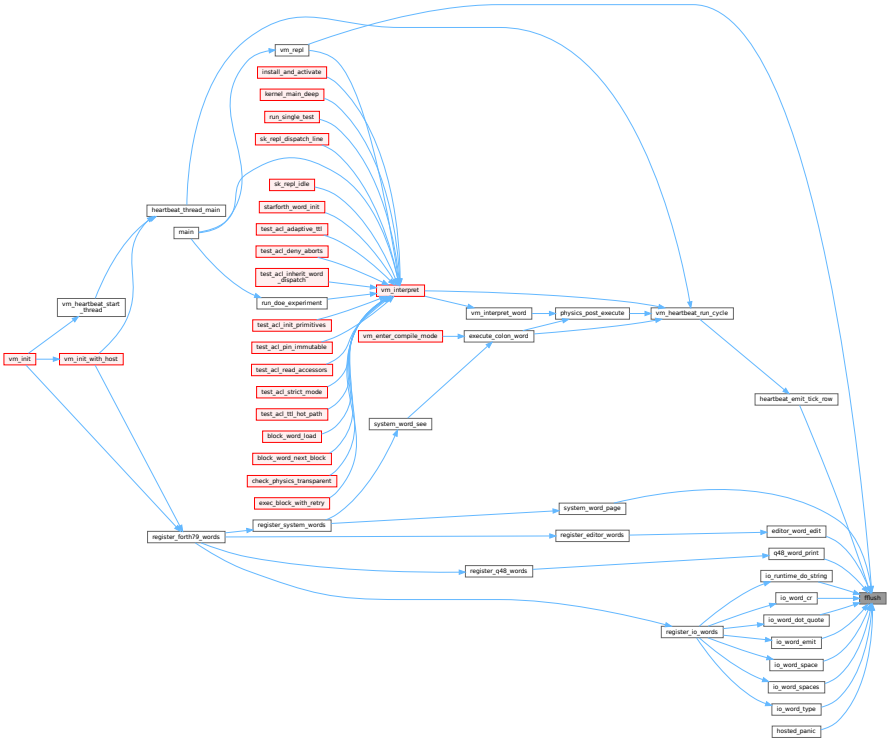


7.37.2.2 fflush()

```
int fflush (
            FILE * stream )
```

Referenced by editor_word_edit(), heartbeat_emit_tick_row(), hosted_panic(), io_runtime_do_string(), io_word_cr(), io_word_dot_quote(), io_word_emit(), io_word_space(), io_word_spaces(), io_word_type(), q48_word_print(), system_word_page(), and vm_repl().

Here is the caller graph for this function:



7.37.2.3 fgetc()

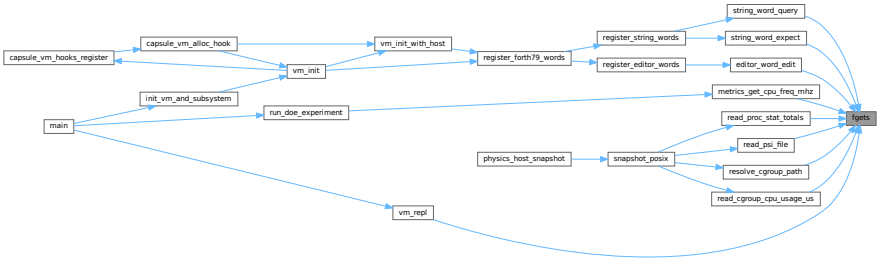
```
int fgetc (
            FILE * stream )
```

7.37.2.4 fgets()

```
char * fgets (
            char * s,
            int n,
            FILE * stream )
```

Referenced by editor_word_edit(), metrics_get_cpu_freq_mhz(), read_cgroup_cpu_usage_us(), read_proc_stat_totals(), read_psi_file(), resolve_cgroup_path(), string_word_expect(), string_word_query(), and vm_repl().

Here is the caller graph for this function:

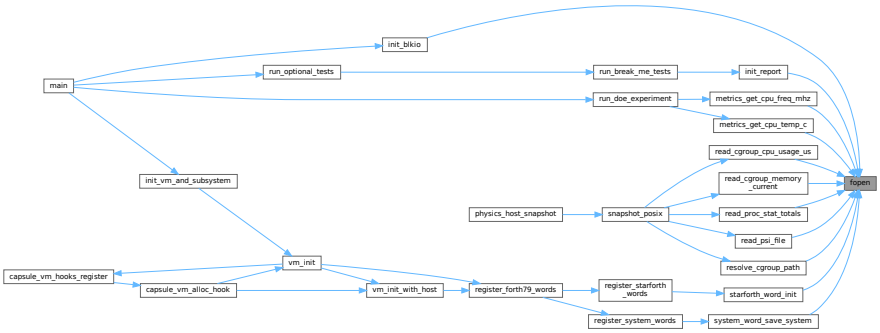


7.37.2.5 fopen()

```
FILE * fopen (
            const char * path,
            const char * mode )
```

Referenced by init_blkio(), init_report(), metrics_get_cpu_freq_mhz(), metrics_get_cpu_temp_c(), read_cgroup_cpu_usage_us(), read_cgroup_memory_current(), read_proc_stat_totals(), read_psi_file(), resolve_cgroup_path(), starforth_word_init(), and system_word_save_system().

Here is the caller graph for this function:

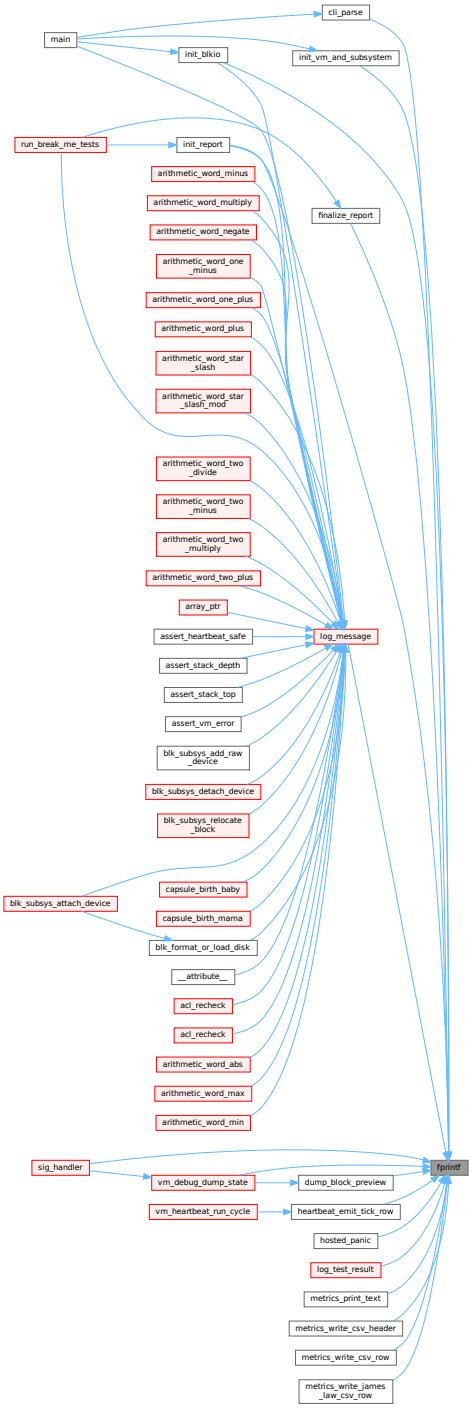


7.37.2.6 fprintf()

```
int fprintf (
            FILE * stream,
            const char * fmt,
             ...  )
```

Referenced by cli_parse(), dump_block_preview(), finalize_report(), heartbeat_emit_tick_row(), hosted_panic(), init_blkio(), init_report(), init_vm_and_subsystem(), log_message(), log_test_result(), metrics_print_text(), metrics_write_csv_header(), metrics_write_csv_row(), metrics_write_james_law_csv_row(), sig_handler(), and vm_debug_dump_state().

Here is the caller graph for this function:



7.37.2.7 fputc()

```
int fputc (
            int c,
            FILE * stream )
```

Referenced by hosted_putc(), runtime_abortq(), system_word_abort_quote(), and system_word_reboot().

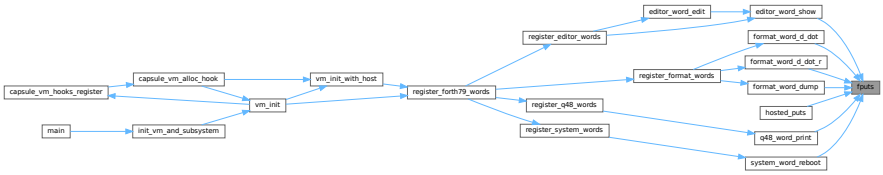
Here is the caller graph for this function:

7.37.2.8 fputs()

```
int fputs (
            const char * s,
            FILE * stream )
```

Referenced by editor_word_show(), format_word_d_dot(), format_word_d_dot_r(), format_word_dump(), hosted_puts(), q48_word_print(), and system_word_reboot().

Here is the caller graph for this function:

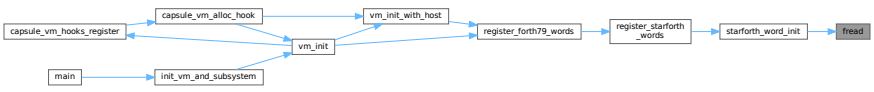


7.37.2.9 fread()

```
size_t fread (
            void * ptr,
            size_t size,
            size_t nmemb,
            FILE * stream )
```

Referenced by starforth_word_init().

Here is the caller graph for this function:

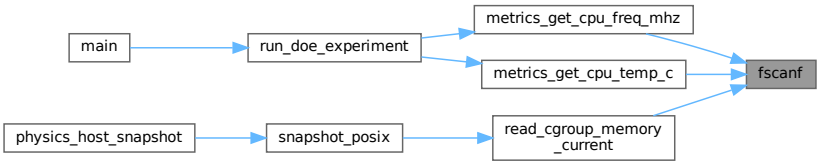


7.37.2.10 fscanf()

```
int fscanf (
            FILE * stream,
            const char * fmt,
             ...  )
```

Referenced by metrics_get_cpu_freq_mhz(), metrics_get_cpu_temp_c(), and read_cgroup_memory_current().

Here is the caller graph for this function:



7.37.2.11 fseek()

```
int fseek (
            FILE * stream,
            long offset,
            int whence )
```

Referenced by starforth_word_init().

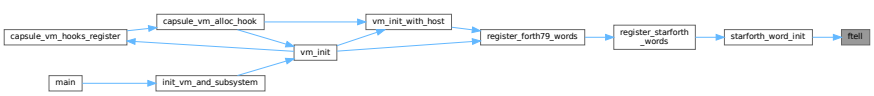
Here is the caller graph for this function:

7.37.2.12 ftell()

```
long ftell (
            FILE * stream )
```

Referenced by starforth_word_init().

Here is the caller graph for this function:

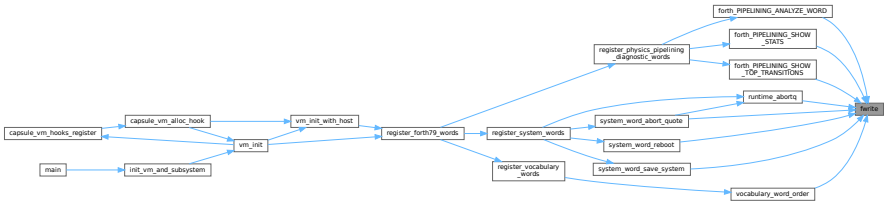


7.37.2.13 fwrite()

```
size_t fwrite (
            const void * ptr,
            size_t size,
            size_t nmemb,
            FILE * stream )
```

Referenced by forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRAN runtime_abortq(), system_word_abort_quote(), system_word_reboot(), system_word_save_system(), and vocabulary_word_order().

Here is the caller graph for this function:

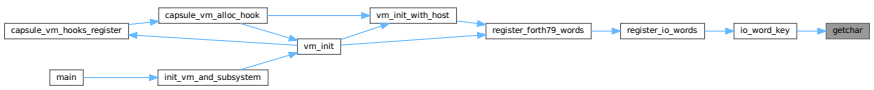


7.37.2.14 getchar()

```
int getchar (
            void  )
```

Referenced by io_word_key().

Here is the caller graph for this function:



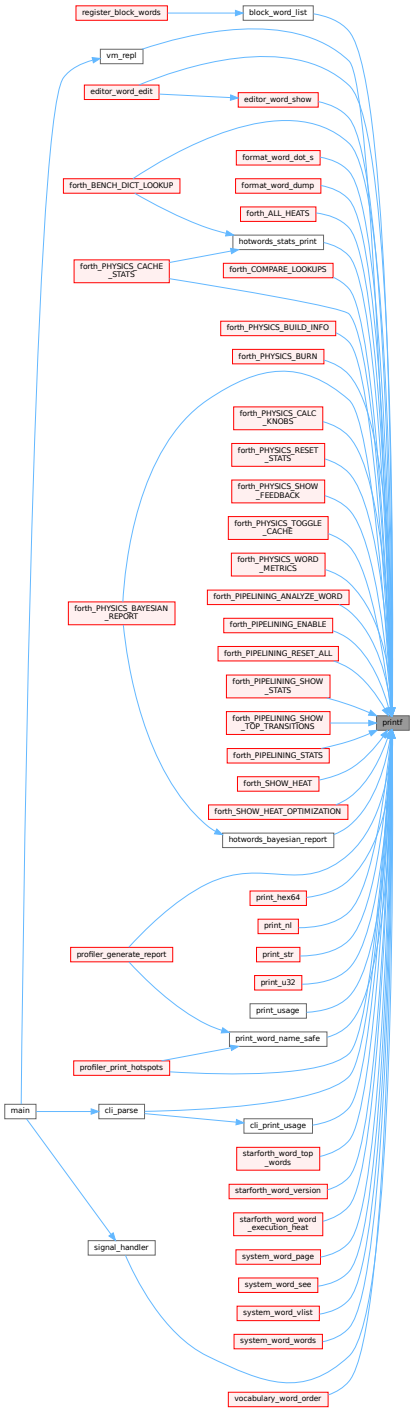
7.37.2.15 printf()

```
int printf (
            const char * fmt,
             ...  )
```

Referenced by block_word_list(), cli_parse(), cli_print_usage(), editor_word_edit(), editor_word_show(), format_word_dot_s(), format_word_dump(), forth_ALL_HEATS(), forth_BENCH_DICT_LOOKUP(), forth_COMPARE_LOOKUPS(), forth_PHYSICS_BAYESIAN_REPORT(), forth_PHYSICS_BUILD_INFO(), forth_PHYSICS_BURN(), forth_PHYSICS_CACHE_STATS forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_RESET_STATS(), forth_PHYSICS_SHOW_FEEDBACK(), forth_PHYSICS_TOGGLE_CACHE(), forth_PHYSICS_WORD_METRICS(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_ENABLE(), forth_PIPELINING_RESET_ALL(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_ forth_PIPELINING_STATS(), forth_SHOW_HEAT(), forth_SHOW_HEAT_OPTIMIZATION(), hotwords_bayesian_report(),

hotwords_stats_print(), print_hex64(), print_nl(), print_str(), print_u32(), print_usage(), print_word_name_safe(), profiler_generate_report(), profiler_print_hotspots(), signal_handler(), starforth_word_top_words(), starforth_word_version(), starforth_word_word_execution_heat(), system_word_page(), system_word_see(), system_word_vlist(), system_word_words(), vm_repl(), and vocabulary_word_order().

Here is the caller graph for this function:



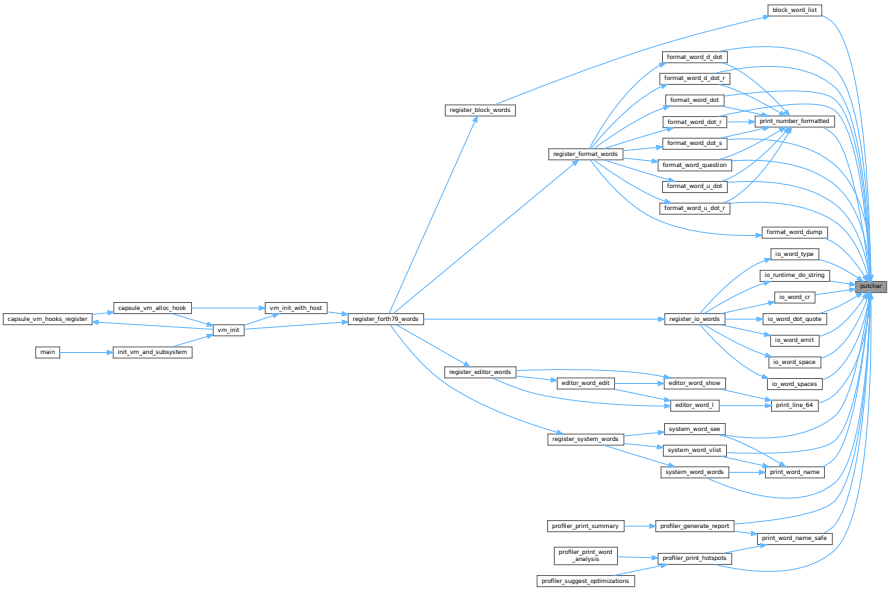
7.37.2.16 putchar()

```
int putchar (
```

```
            int c )
```

Referenced by block_word_list(), format_word_d_dot(), format_word_d_dot_r(), format_word_dot(), format_word_dot_r(), format_word_dot_s(), format_word_dump(), format_word_question(), format_word_u_dot(), format_word_u_dot_r(), io_runtime_do_string(), io_word_cr(), io_word_dot_quote(), io_word_emit(), io_word_space(), io_word_spaces(), io_word_type(), print_line_64(), print_number_formatted(), print_word_name(), print_word_name_safe(), profiler_generate_report(), profiler_print_hotspots(), system_word_see(), system_word_vlist(), and system_word_words().

Here is the caller graph for this function:

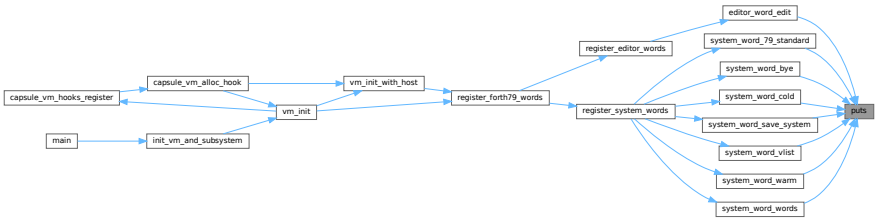


7.37.2.17 puts()

```
int puts (
            const char * s )
```

Referenced by editor_word_edit(), system_word_79_standard(), system_word_bye(), system_word_cold(), system_word_save_system(), system_word_vlist(), system_word_warm(), and system_word_words().

Here is the caller graph for this function:



7.37.2.18 rewind()

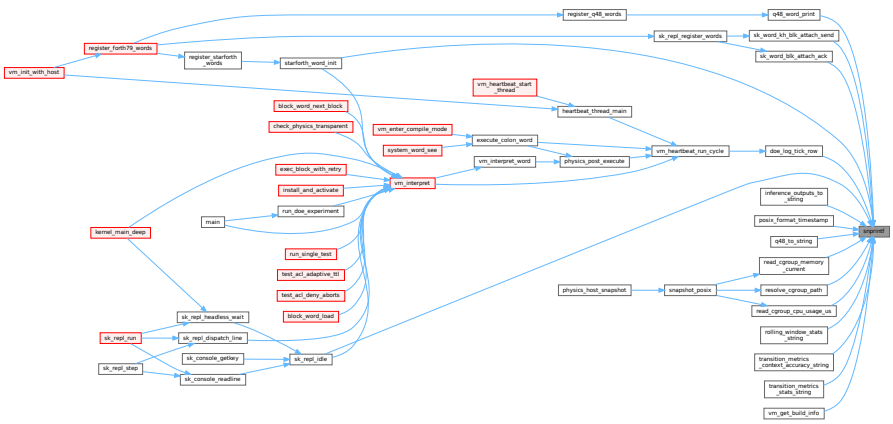
```
void rewind (
            FILE * stream )
```

7.37.2.19 snprintf()

```
int snprintf (
            char * buf,
            size_t n,
            const char * fmt,
             ...  )
```

Referenced by doe_log_tick_row(), inference_outputs_to_string(), posix_format_timestamp(), q48_to_string(), q48_word_print(), read_cgroup_cpu_usage_us(), read_cgroup_memory_current(), resolve_cgroup_path(), rolling_window_stats_string(), sk_repl_idle(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), starforth_word_init(), transition_metrics_context_accuracy_string(), transition_metrics_stats_string(), and vm_get_build_info().

Here is the caller graph for this function:

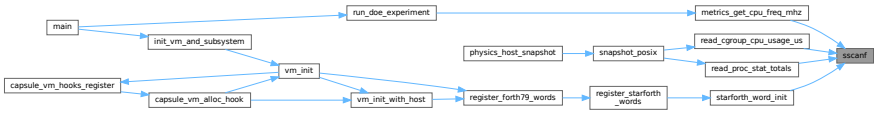


7.37.2.20 sscanf()

```
int sscanf (
            const char * str,
            const char * fmt,
             ...  )
```

Referenced by metrics_get_cpu_freq_mhz(), read_cgroup_cpu_usage_us(), read_proc_stat_totals(), and starforth_word_init().

Here is the caller graph for this function:

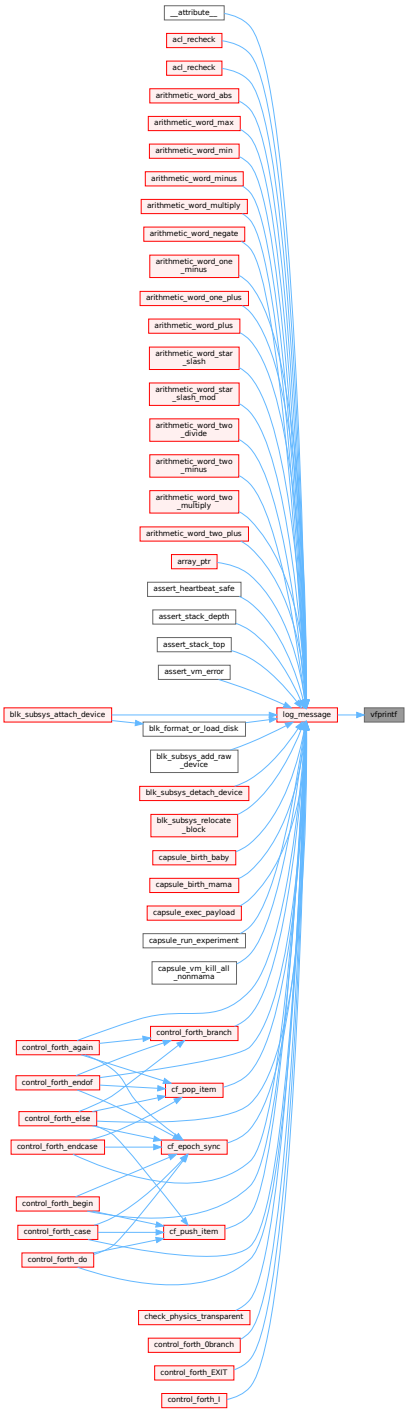


7.37.2.21 vfprintf()

```
int vfprintf (
            FILE * stream,
            const char * fmt,
            va_list args )
```

Referenced by log_message().

Here is the caller graph for this function:

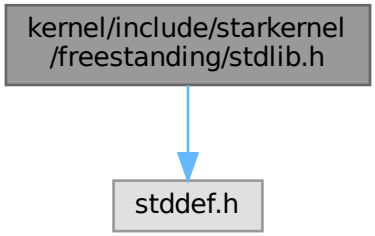


7.37.2.22 vsnprintf()

```
int vsnprintf (
              char * buf,
              size_t n,
              const char * fmt,
              va_list args )
```

7.38 kernel/include/starkernel/freestanding/stdlib.h File Reference

`#include <stddef.h>`
Include dependency graph for stdlib.h:



This graph shows which files directly or indirectly include this file:

Functions

- void ∗ malloc (size_t size)
- void free (void ∗ptr)
- void ∗ calloc (size_t n, size_t size)
- void ∗ realloc (void ∗ptr, size_t size)
- void abort (void)
- void qsort (void ∗base, size_t nmemb, size_t size, int(∗compar)(const void ∗, const void ∗))
- long strtol (const char ∗str, char ∗∗endptr, int base)
- unsigned long strtoul (const char ∗str, char ∗∗endptr, int base)
- long long strtoll (const char ∗str, char ∗∗endptr, int base)
- int atoi (const char ∗str)
- long atol (const char ∗str)

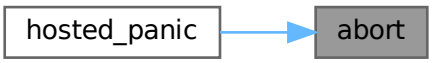
7.38.1 Function Documentation

7.38.1.1 abort()

```
void abort (
            void  )
```

Referenced by hosted_panic().

Here is the caller graph for this function:

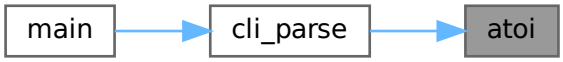


7.38.1.2 atoi()

```
int atoi (
            const char * str )
```

Referenced by cli_parse().

Here is the caller graph for this function:



7.38.1.3 atol()

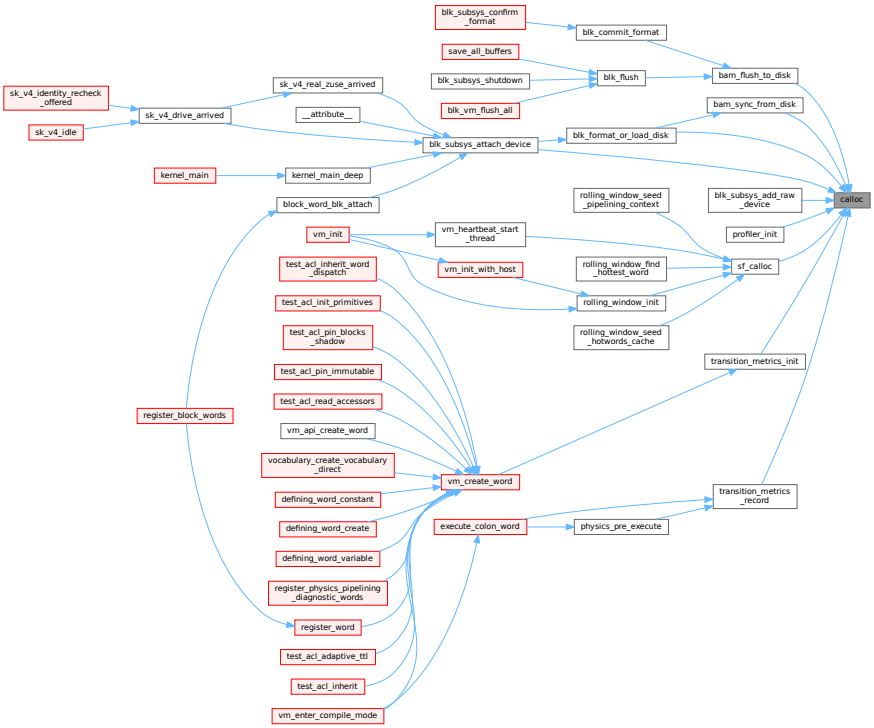
```
long atol (
            const char * str )
```

7.38.1.4 calloc()

```
void * calloc (
            size_t n,
            size_t size )
```

Referenced by bam_flush_to_disk(), bam_sync_from_disk(), blk_format_or_load_disk(), blk_subsys_add_raw_device(), blk_subsys_attach_device(), profiler_init(), sf_calloc(), transition_metrics_init(), and transition_metrics_record().

Here is the caller graph for this function:

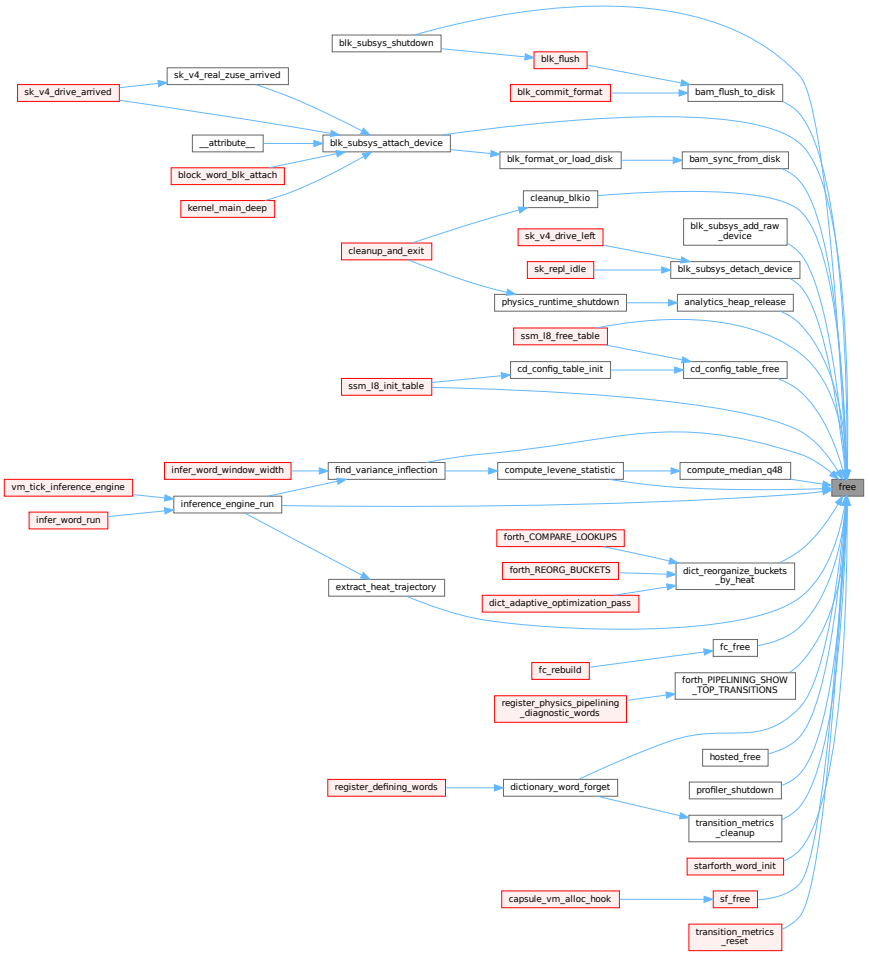


7.38.1.5 free()

```
void free (
            void * ptr )
```

Referenced by analytics_heap_release(), bam_flush_to_disk(), bam_sync_from_disk(), blk_subsys_add_raw_device(), blk_subsys_attach_device(), blk_subsys_detach_device(), blk_subsys_shutdown(), cd_config_table_free(), cleanup_blkio(), compute_levene_statistic(), compute_median_q48(), dict_reorganize_buckets_by_heat(), dictionary_word_forget(), extract_heat_trajectory(), fc_free(), find_variance_inflection(), forth_PIPELINING_SHOW_TOP_TRANSITIC hosted_free(), inference_engine_run(), profiler_shutdown(), sf_free(), ssm_l8_free_table(), ssm_l8_init_table(), starforth_word_init(), transition_metrics_cleanup(), and transition_metrics_reset().

Here is the caller graph for this function:

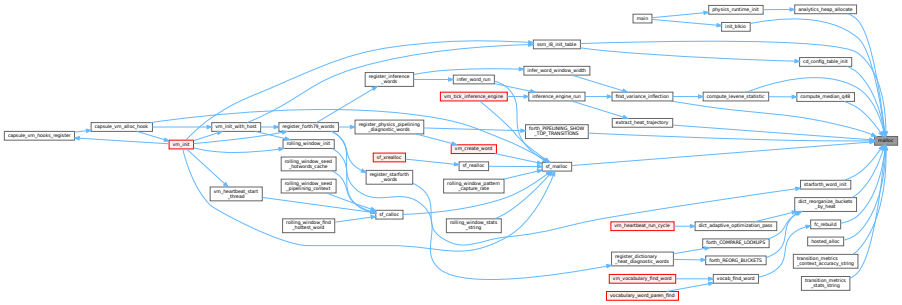


7.38.1.6 malloc()

```
void * malloc (
            size_t size )
```

Referenced by analytics_heap_allocate(), cd_config_table_init(), compute_levene_statistic(), compute_median_q48(), dict_reorganize_buckets_by_heat(), extract_heat_trajectory(), fc_rebuild(), find_variance_inflection(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), hosted_alloc(), init_blkio(), sf_malloc(), ssm_l8_init_table(), starforth_word_init(), transition_metrics_context_accuracy_string(), and transition_metrics_stats_string().

Here is the caller graph for this function:

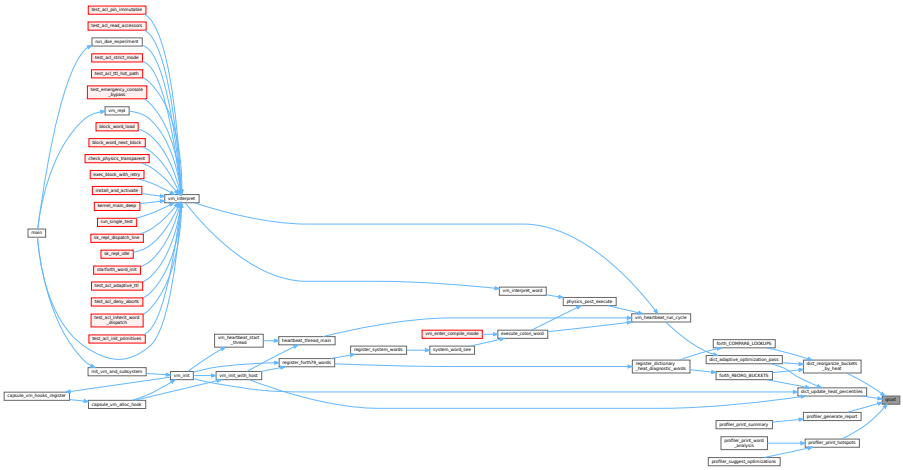


7.38.1.7 qsort()

```
void qsort (
            void * base,
            size_t nmemb,
            size_t size,
            int(*)(const void *, const void *) compar )
```

Referenced by dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), profiler_generate_report(), and profiler_print_hotspots().

Here is the caller graph for this function:

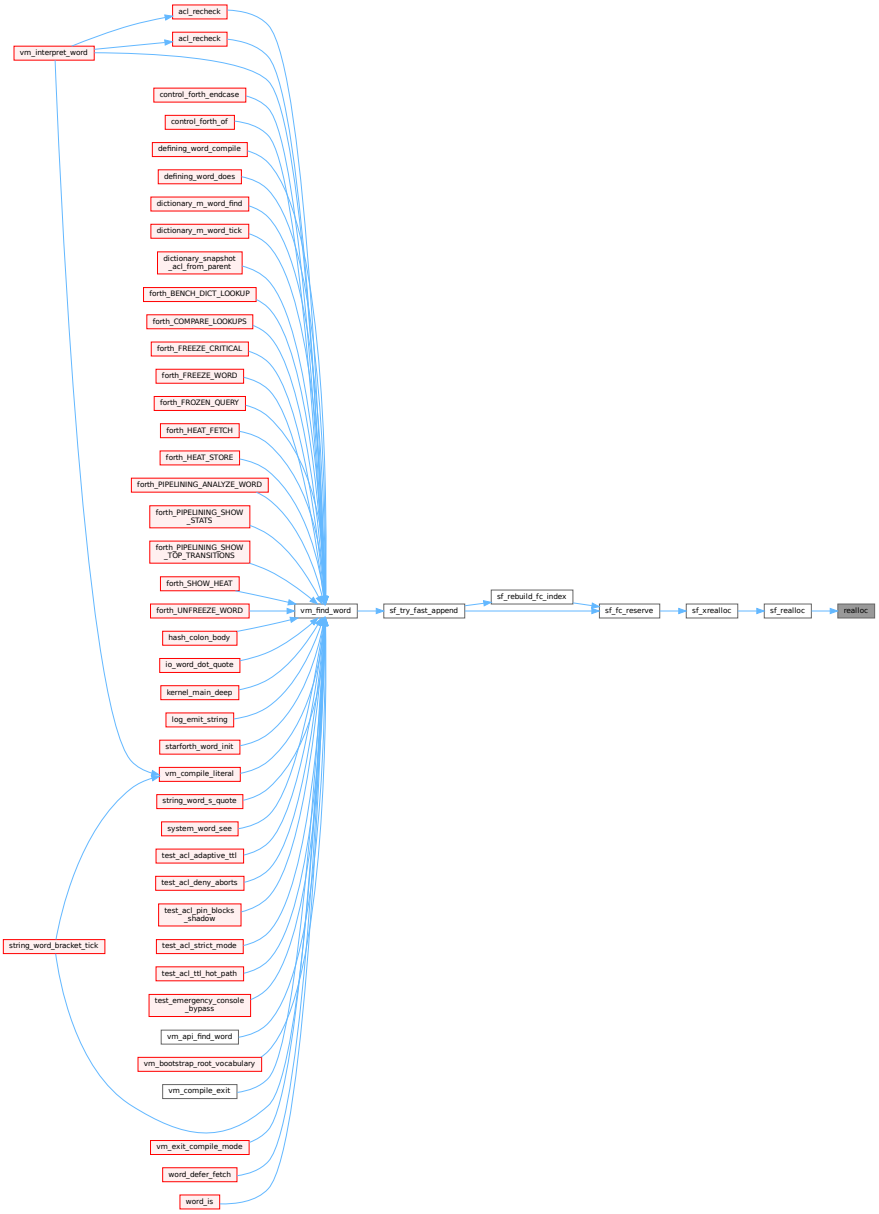


7.38.1.8 realloc()

```
void * realloc (
            void * ptr,
            size_t size )
```

Referenced by sf_realloc().

Here is the caller graph for this function:



7.38.1.9 strtol()

```
long strtol (
            const char * str,
            char ** endptr,
            int base )
```

Referenced by convert_string_to_number().

Here is the caller graph for this function:

7.38.1.10 strtoll()

```
long long strtoll (
            const char * str,
            char ** endptr,
            int base )
```

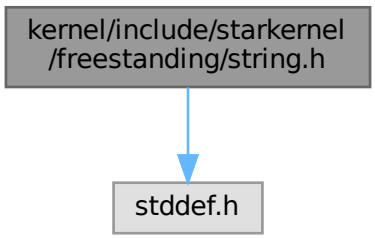
7.38.1.11 strtoul()

```
unsigned long strtoul (
            const char * str,
            char ** endptr,
            int base )
```

7.39 kernel/include/starkernel/freestanding/string.h File Reference

```
#include <stddef.h>
```

Include dependency graph for string.h:



Functions

- void ∗ memcpy (void ∗dest, const void ∗src, size_t n)
- void ∗ memmove (void ∗dest, const void ∗src, size_t n)
- void ∗ memset (void ∗s, int c, size_t n)
- int memcmp (const void ∗s1, const void ∗s2, size_t n)
- void ∗ memchr (const void ∗s, int c, size_t n)

- size_t strlen (const char *s)
- char * strcpy (char *dest, const char *src)
- char * strncpy (char *dest, const char *src, size_t n)
- int strcmp (const char *s1, const char *s2)
- int strncmp (const char *s1, const char *s2, size_t n)
- char * strchr (const char *s, int c)
- char * strrchr (const char *s, int c)
- char * strerror (int errnum)

7.39.1 Function Documentation

7.39.1.1 memchr()

```
void * memchr (
            const void * s,
            int c,
            size_t n )
```

Referenced by block_word_next_block().

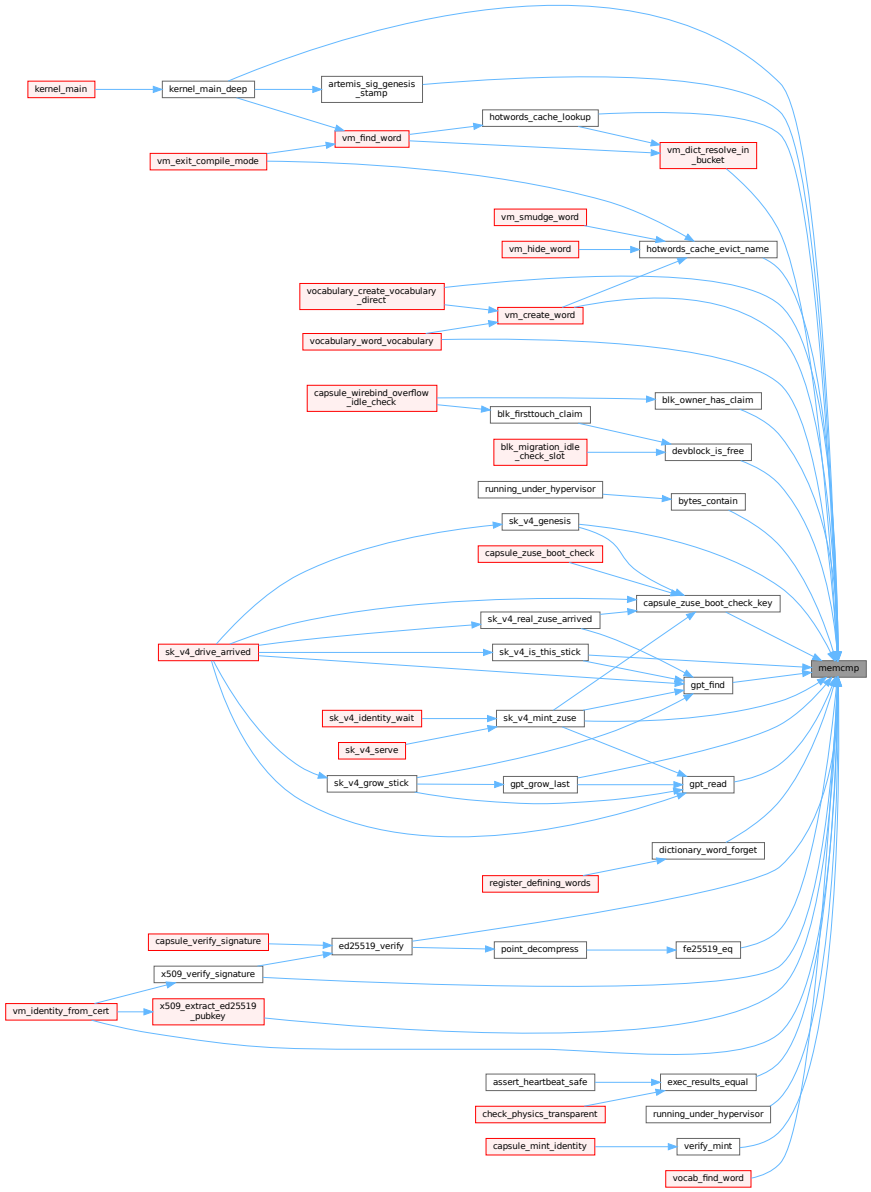
Here is the caller graph for this function:

7.39.1.2 memcmp()

```
int memcmp (
            const void * s1,
            const void * s2,
            size_t n )
```

Referenced by artemis_sig_genesis_stamp(), blk_owner_has_claim(), bytes_contain(), capsule_zuse_boot_check_key(), devblock_is_free(), dictionary_word_forget(), ed25519_verify(), exec_results_equal(), fe25519_eq(), gpt_find(), gpt_grow_last(), gpt_read(), hotwords_cache_evict_name(), hotwords_cache_lookup(), kernel_main_deep(), running_under_hypervisor(), sk_v4_genesis(), sk_v4_is_this_stick(), sk_v4_mint_zuse(), verify_mint(), vm_create_word(), vm_dict_resolve_in_bucket(), vm_identity_from_cert(), vocab_find_word(), vocabulary_create_vocabulary_direct(), vocabulary_word_vocabulary(), x509_extract_ed25519_pubkey(), and x509_verify_signature().

Here is the caller graph for this function:



7.39.1.3 memcpy()

```
void * memcpy (
            void * dest,
            const void * src,
            size_t n )
```

Referenced by artemis_sig_check(), artemis_sig_genesis_stamp(), blk_firsttouch_claim(), blk_meta_relocate_devblock(), blk_owner_fp_get(), blk_owner_fp_set(), blk_subsys_relocate_block(), blk_vm_evict(), blk_vm_flush_all(), blk_vm_load(), block_word_load(), block_word_next_block(), block_word_owner_fetch(), block_word_update(), capsule_console_birth(), capsule_mint_identity(), capsule_mint_identity_scratch(), capsule_runcap_birth(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), capsule_zuse_boot_check_key(),

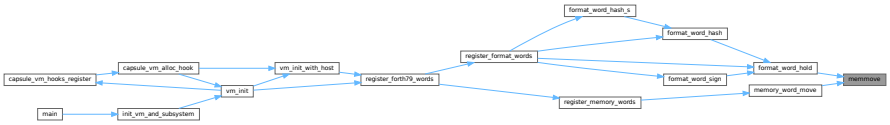
capsule_zuse_boot_root_pubkey(), capsule_zuse_boot_try_attach(), capture_exec_result(), compute_median_q48(), convert_string_to_number(), defining_runtime_does_rt(), der_put_tlv(), der_put_uint_integer(), dict_reorganize_buckets_by_heat(), ed25519_sign(), editor_word_edit(), editor_word_s(), expand_seed(), extract_name(), gpt_disk_guid(), gpt_grow_last(), gpt_read(), gpt_write_one(), homeblocks_sig_check(), install_and_activate(), io_write_block(), kernel_main_deep(), log_region_append(), log_region_read_slot(), log_region_write_slot(), meta_from_slice(), meta_to_slice(), physics_analytics_publish_event(), point_decompress(), put_sectors(), ram_read(), ram_write(), reloc_flush_to_disk(), reloc_load_from_disk(), rolling_window_export_execution_history(), rolling_window_publish_snapshot(), say_size(), scalar_reduce512(), sf_fc_reserve(), sha512_final(), sha512_update(), sk_repl_dispatch_line(), sk_v4_drive_arrived(), sk_v4_genesis(), sk_v4_login(), sk_v4_marker_read(), sk_v4_marker_write(), sk_v4_mint_zuse(), starforth_word_init(), string_word_word(), system_word_abort_quote(), usb_blk_read(), usb_blk_write(), vblk_read(), vblk_write(), virtio_rng_get_bytes(), vm_create_word(), vm_enter_compile_mode(), vm_identity_from_cert(), vm_input_load_line(), vm_load_cell(), vm_store_cell(), vm_zuse_cert_install(), vocabulary_create_vocabulary_direct(), volmeta_from_buf(), volmeta_to_buf(), x509_build_user_cert(), x509_extract_ed25519_pubkey(), x509_extract_serial(), xhci_ep0_bot_mass_storage_rese xhci_ep0_control_read(), xhci_ep0_control_write_nodata(), xhci_ep0_hid_set_leds(), and xhci_ep0_hid_set_protocol().

7.39.1.4 memmove()

```
void * memmove (
            void * dest,
            const void * src,
            size_t n )
```

Referenced by format_word_hold(), and memory_word_move().

Here is the caller graph for this function:



7.39.1.5 memset()

```
void * memset (
            void * s,
            int c,
            size_t n )
```

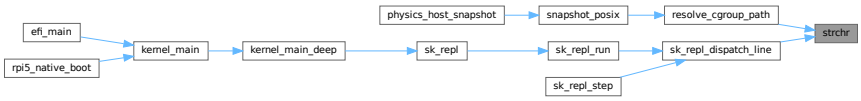
Referenced by analytics_heap_allocate(), artemis_sig_genesis_stamp(), bam_flush_to_disk(), bam_sync_from_disk(), blk_compute_fresh_geometry(), blk_flush(), blk_get_empty_buffer(), blk_get_meta(), blk_meta_relocate_devblock(), blk_subsys_init(), blk_subsys_shutdown(), blk_vm_assign(), blkio_file_init_state(), bucket_posterior(), cache_get_slot(), cache_load_devblock(), cache_posterior(), capsule_console_birth(), capsule_mint_identity(), capsule_mint_identity_scratch(), capsule_runcap_birth(), capsule_wirebind_verify_cert(), capsule_zuse_boot_try_attach(), capture_exec_result(), cd_classify_ids(), cd_config_table_free(), cd_config_table_init(), editor_word_edit(), editor_word_s(), empty_all_buffers(), fe25519_0(), fe25519_1(), format_word_begin_conversion(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), gpt_read(), gpt_write_one(), hotwords_cache_cleanup(), hotwords_cache_init(), infer_word_run(), io_init(), log_region_append(), log_region_ctrl_init_fresh(), log_region_write_slot(), memory_word_erase(), memory_word_fill(), meta_from_slice(), meta_to_slice(), physics_analytics_publish_event(), read_devblock_4k(), read_header_4k(), reloc_load_from_disk(), rolling_window_reset(), rolling_window_snapshot_view(), sf_alloc_init(), sk_repl_idle(), sk_v4_drive_left(), sk_v4_genesis(), sk_v4_idle(), sk_v4_marker_read(), sk_v4_marker_write(), sk_v4_mint_zuse(), sk_v4_real_zuse_arrived(), snapshot_posix(), starforth_word_init(), transition_metrics_reset(), usb_blk_state_for(), vblk_init_device(), vinput_init_device(), vm_create_word(), vm_debug_install_signal_handlers(), vm_host_calloc(), vm_init(), vm_init_with_host(), vm_input_ensure(), vm_input_load_line(), vm_tick_inference_engine(), volmeta_from_buf(), volmeta_to_buf(), vrng_init_device(), xhci_bringup(), and xhci_poll_one().

7.39.1.6 strchr()

```
char * strchr (
            const char * s,
            int c )
```

Referenced by resolve_cgroup_path(), and sk_repl_dispatch_line().

Here is the caller graph for this function:

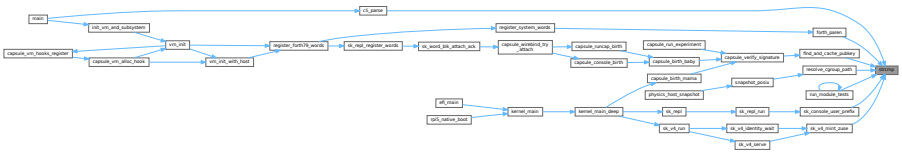


7.39.1.7 strcmp()

```
int strcmp (
            const char * s1,
            const char * s2 )
```

Referenced by cli_parse(), find_and_cache_pubkey(), forth_paren(), resolve_cgroup_path(), run_module_tests(), sk_console_user_prefix(), and sk_v4_mint_zuse().

Here is the caller graph for this function:



7.39.1.8 strcpy()

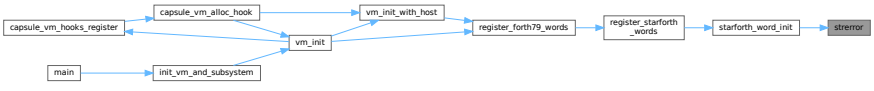
```
char * strcpy (
            char * dest,
            const char * src )
```

7.39.1.9 strerror()

```
char * strerror (
            int errnum )
```

Referenced by starforth_word_init().

Here is the caller graph for this function:

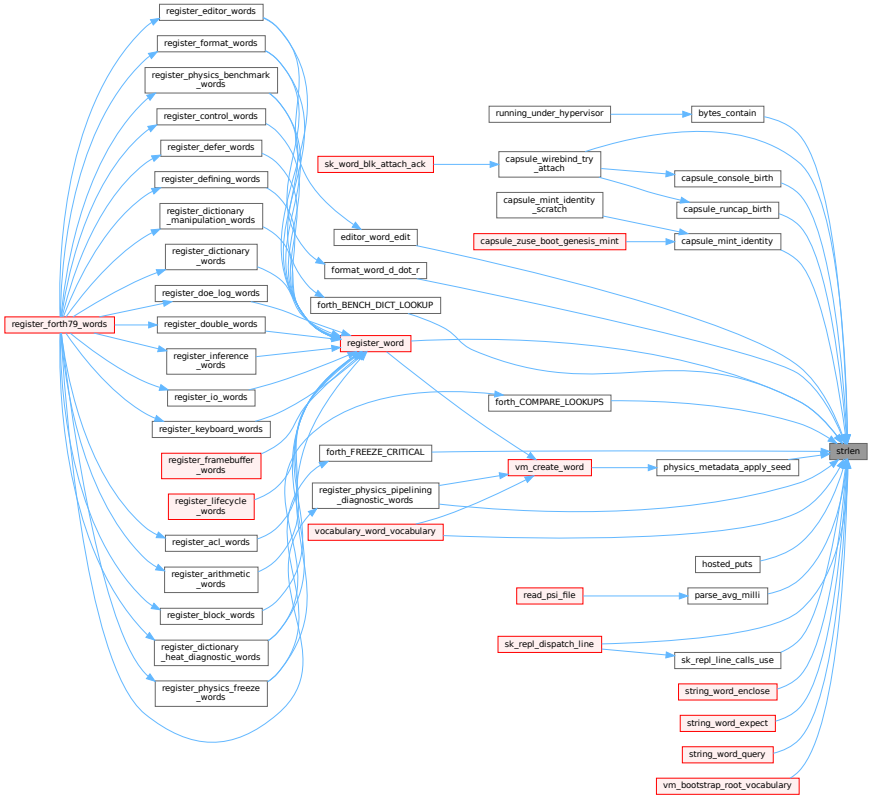


7.39.1.10 strlen()

```
size_t strlen (
            const char * s )
```

Referenced by bytes_contain(), capsule_console_birth(), capsule_mint_identity(), capsule_runcap_birth(), capsule_wirebind_try_attach(), editor_word_edit(), format_word_d_dot_r(), forth_BENCH_DICT_LOOKUP(), forth_COMPARE_LOOKUPS(), forth_FREEZE_CRITICAL(), hosted_puts(), parse_avg_milli(), physics_metadata_apply_seed(), register_physics_pipelining_diagnostic_words(), register_word(), sk_repl_dispatch_line(), sk_repl_line_calls_use(), string_word_enclose(), string_word_expect(), string_word_query(), vm_bootstrap_root_vocabulary(), and vocabulary_word_vocabulary().

Here is the caller graph for this function:

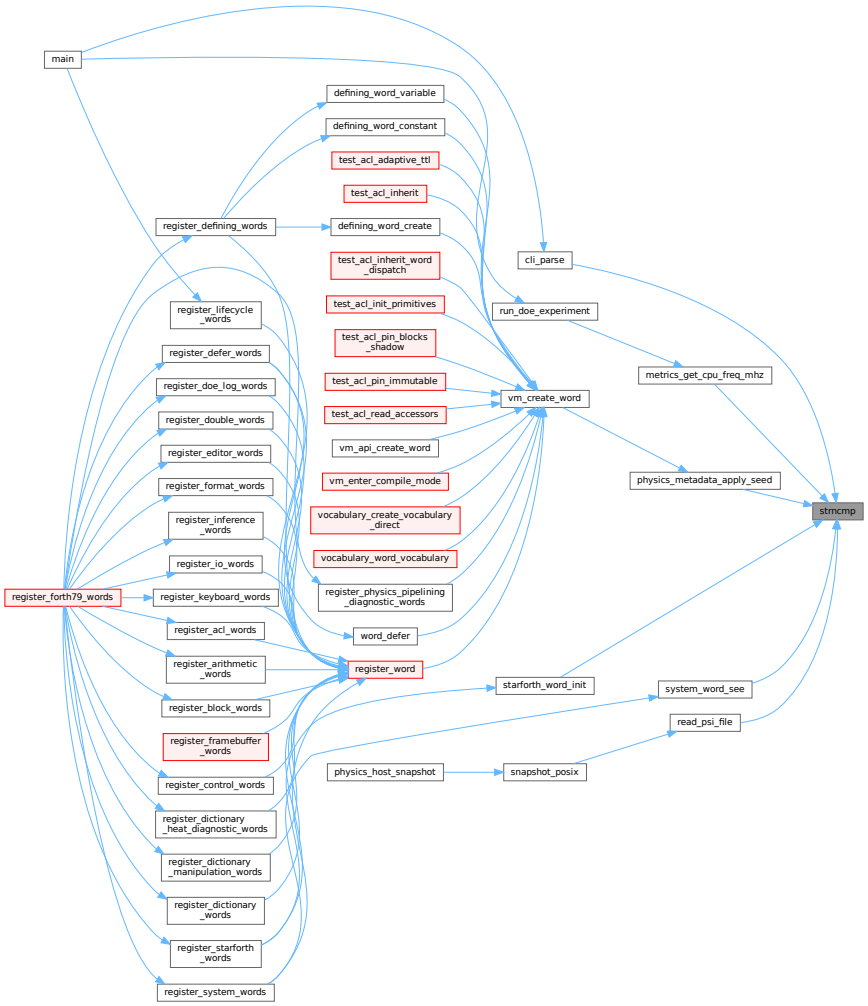


7.39.1.11 strncmp()

```
int strncmp (
            const char * s1,
            const char * s2,
            size_t n )
```

Referenced by cli_parse(), metrics_get_cpu_freq_mhz(), physics_metadata_apply_seed(), read_psi_file(), starforth_word_init(), and system_word_see().

Here is the caller graph for this function:



7.39.1.12 strncpy()

```
char * strncpy (
            char * dest,
            const char * src,
            size_t n )
```

Referenced by blk_compute_fresh_geometry(), and posix_format_timestamp().

Here is the caller graph for this function:

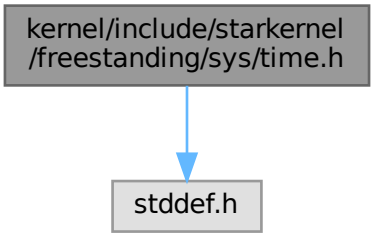
7.39.1.13 strrchr()

```
char * strrchr (
            const char * s,
            int c )
```

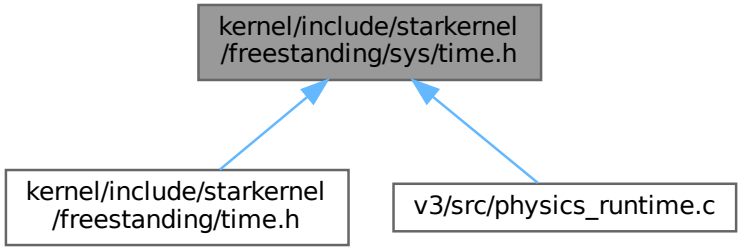
7.40 kernel/include/starkernel/freestanding/sys/time.h File Reference

```
#include <stddef.h>
```

Include dependency graph for time.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct timeval
- struct timezone

Typedefs

- typedef long time_t
- typedef long suseconds_t

Functions

- static int gettimeofday (struct timeval *tv, struct timezone *tz)

7.40.1 Typedef Documentation

7.40.1.1 suseconds_t

```
typedef long suseconds_t
```

7.40.1.2 time_t

```
typedef long time_t
```

7.40.2 Function Documentation

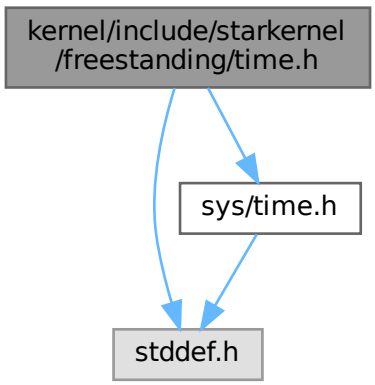
7.40.2.1 gettimeofday()

```
static int gettimeofday (
            struct timeval * tv,
            struct timezone * tz )  [inline], [static]
```

7.41 kernel/include/starkernel/freestanding/time.h File Reference

```
#include <stddef.h>
#include "sys/time.h"
```

Include dependency graph for time.h:



Data Structures

- struct timespec

Functions

- int nanosleep (const struct timespec *req, struct timespec *rem)

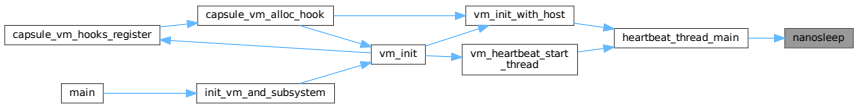
7.41.1 Function Documentation

7.41.1.1 nanosleep()

```
int nanosleep (
            const struct timespec * req,
            struct timespec * rem )
```

Referenced by heartbeat_thread_main().

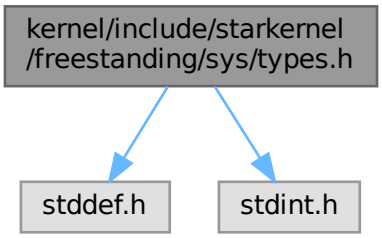
Here is the caller graph for this function:



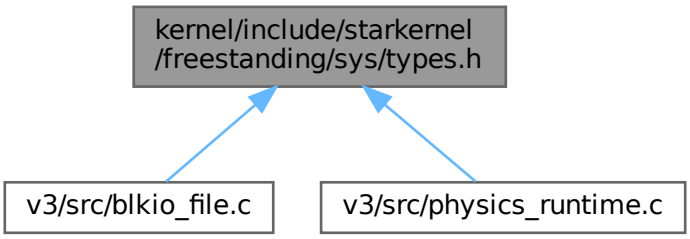
7.42 kernel/include/starkernel/freestanding/sys/types.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for types.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef long ssize_t
- typedef long off_t
- typedef unsigned int mode_t
- typedef unsigned int uid_t
- typedef unsigned int gid_t
- typedef int pid_t

7.42.1 Typedef Documentation

7.42.1.1 gid_t

```
typedef unsigned int gid_t
```

7.42.1.2 mode_t

```
typedef unsigned int mode_t
```

7.42.1.3 off_t

```
typedef long off_t
```

7.42.1.4 pid_t

```
typedef int pid_t
```

7.42.1.5 ssize_t

```
typedef long ssize_t
```

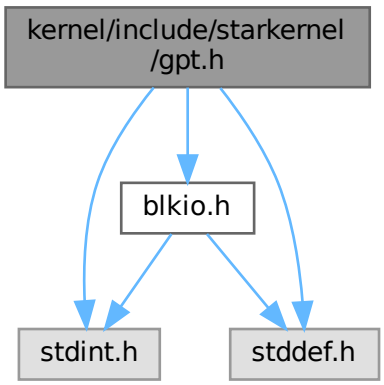
7.42.1.6 uid_t

```
typedef unsigned int uid_t
```

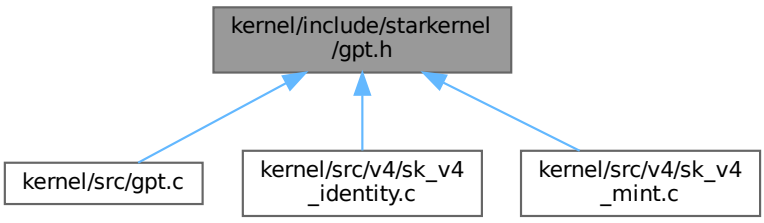
7.43 kernel/include/starkernel/gpt.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "blkio.h"
```

Include dependency graph for gpt.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct gpt_part_t
- struct gpt_part_dev_t

Macros

- #define GPT_OK 0 /* a valid GPT */
- #define GPT_NONE 1 /* not a GPT disk -- no protective MBR -- (or one whose sectors are not 512 bytes): read it raw */
- #define GPT_BAD 2 /* a protective MBR, but its header or entry array does not check out: not used */
- #define GPT_EIO 3 /* its first block could not be read: not used */

Functions

- int gpt_read (blkio_dev_t *disk, gpt_part_t *parts, unsigned max, unsigned *count)
- void gpt_disk_guid (uint8_t guid[16])
- uint64_t gpt_alternate_lba (void)
- int gpt_grow_last (blkio_dev_t *disk, const uint8_t type[16], uint64_t max_part_sectors, uint64_t *new_last_lba)
- int gpt_write_one (blkio_dev_t *disk, const uint8_t type[16], uint64_t part_sectors, uint8_t disk_guid[16])
- int gpt_find (const gpt_part_t *parts, unsigned count, const uint8_t type[16])
- int gpt_part_open (blkio_dev_t *dev, gpt_part_dev_t *st, blkio_dev_t *disk, const gpt_part_t *p)
- uint32_t gpt_crc32 (const void *buf, size_t len)

Variables

- const uint8_t gpt_type_starship_identity [16]
- const uint8_t gpt_type_starship_artemis [16]

7.43.1 Macro Definition Documentation

7.43.1.1 GPT_BAD

```
#define GPT_BAD 2 /* a protective MBR, but its header or entry array does not check out:  not used */
```

7.43.1.2 GPT_EIO

```
#define GPT_EIO 3 /* its first block could not be read:  not used */
```

7.43.1.3 GPT_NONE

```
#define GPT_NONE 1 /* not a GPT disk -- no protective MBR -- (or one whose sectors are not 512 bytes):  read it raw */
```

7.43.1.4 GPT_OK

```
#define GPT_OK 0 /* a valid GPT */
```

7.43.2 Function Documentation

7.43.2.1 gpt_alternate_lba()

```
uint64_t gpt_alternate_lba (
            void  )
```

References g_gpt_alternate_lba.

Referenced by sk_v4_grow_stick().

Here is the caller graph for this function:

7.43.2.2 gpt_crc32()

```
uint32_t gpt_crc32 (
            const void * buf,
            size_t len )
```

Referenced by gpt_grow_last(), gpt_read(), and gpt_write_one().

Here is the caller graph for this function:

7.43.2.3 gpt_disk_guid()

```
void gpt_disk_guid (
            uint8_t guid[16] )
```

References g_gpt_disk_guid, and memcpy().

Referenced by sk_v4_is_this_stick().

Here is the call graph for this function:

Here is the caller graph for this function:

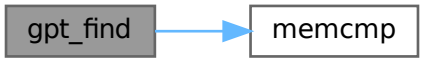
7.43.2.4 gpt_find()

```
int gpt_find (
            const gpt_part_t * parts,
            unsigned count,
            const uint8_t type[16] )
```

References count, and memcmp().

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), sk_v4_is_this_stick(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

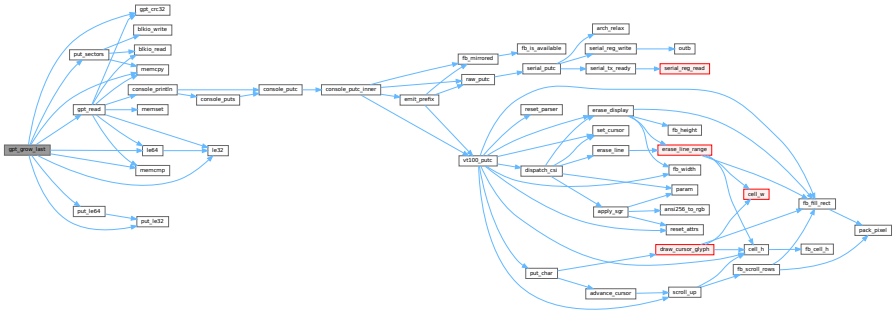
7.43.2.5 gpt_grow_last()

```
int gpt_grow_last (
            blkio_dev_t * disk,
            const uint8_t type[16],
            uint64_t max_part_sectors,
            uint64_t * new_last_lba )
```

References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, count, blkio_vtable_t::flush, g_gpt_entries, g_gpt_hdr_block, GPT_BAD, gpt_crc32(), GPT_EIO, GPT_NONE, GPT_OK, gpt_read(), GPT_SECTOR, le32(), le64(), memcmp(), memcpy(), put_le32(), put_le64(), put_sectors(), blkio_dev::total_blocks, blkio_dev::vt, and zero.

Referenced by sk_v4_grow_stick().

Here is the call graph for this function:



Here is the caller graph for this function:

7.43.2.6 gpt_part_open()

```
int gpt_part_open (
            blkio_dev_t * dev,
            gpt_part_dev_t * st,
            blkio_dev_t * disk,
            const gpt_part_t * p )
```

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, blkio_open(), gpt_part_dev_t::disk, gpt_part_dev_t::fblocks, gpt_part_dev_t::first_fblock, gpt_part_t::first_lba, blkio_params_t::forth_block_size, g_part_vt, gpt_part_t::last_lba, blkio_params_t::opaque, blkio_params_t::total_blocks, and blkio_dev::total_blocks.

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

Here is the call graph for this function:

Here is the caller graph for this function:

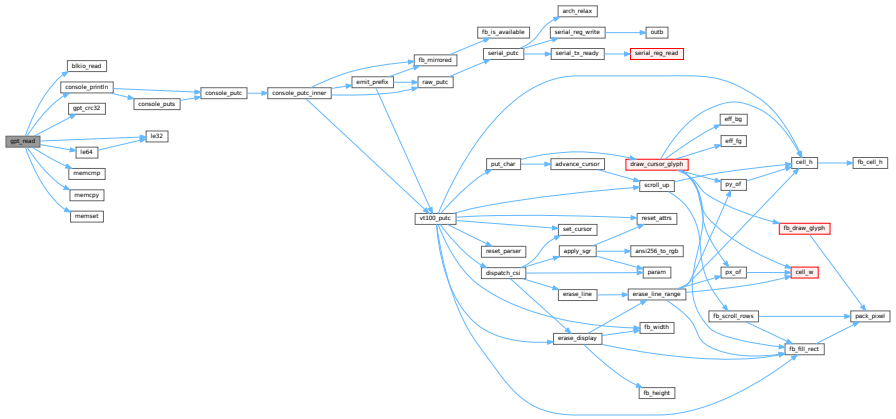
7.43.2.7 gpt_read()

```
int gpt_read (
            blkio_dev_t * disk,
            gpt_part_t * parts,
            unsigned max,
            unsigned * count )
```

References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_read(), console_println(), count, gpt_part_t::first_lba, blkio_dev::forth_block_size, g_gpt_alternate_lba, g_gpt_disk_guid, g_gpt_entries, g_gpt_hdr_block, GPT_BAD, gpt_crc32(), GPT_EIO, GPT_ENTRIES_MAX, GPT_NONE, GPT_OK, GPT_SECTOR, blkio_vtable_t::info, gpt_part_t::last_lba, le32(), le64(), memcmp(), memcpy(), memset(), blkio_info_t::phys_sector_size, blkio_dev::total_blocks, blkio_dev::vt, and zero.

Referenced by gpt_grow_last(), sk_v4_drive_arrived(), sk_v4_grow_stick(), and sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

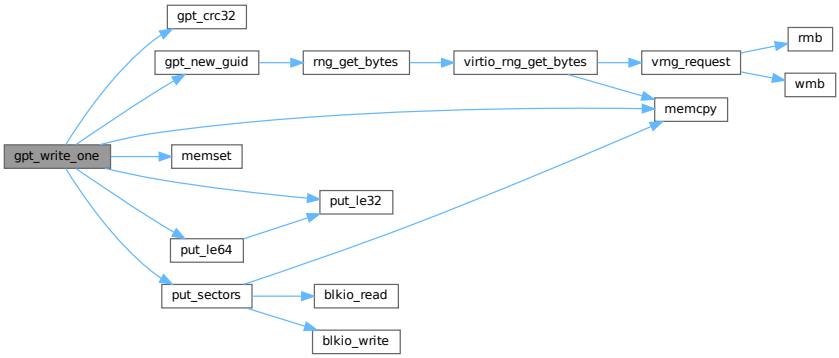
7.43.2.8 gpt_write_one()

```
int gpt_write_one (
            blkio_dev_t * disk,
            const uint8_t type[16],
            uint64_t part_sectors,
            uint8_t disk_guid[16] )
```

References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_vtable_t::flush, blkio_dev::forth_block_size, g_gpt_new_entries, GPT_BAD, gpt_crc32(), GPT_EIO, GPT_ENT_SECTORS, GPT_ESIZE, GPT_NENT, gpt_new_guid(), GPT_OK, GPT_SECTOR, blkio_vtable_t::info, memcpy(), memset(), name, blkio_info_t::phys_sector_size, put_le32(), put_le64(), put_sectors(), blkio_dev::total_blocks, blkio_dev::vt, and zero.

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

7.43.3 Variable Documentation

7.43.3.1 gpt_type_starship_artemis

```
const uint8_t gpt_type_starship_artemis[16]  [extern]
```

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), and sk_v4_is_this_stick().

7.43.3.2 gpt_type_starship_identity

```
const uint8_t gpt_type_starship_identity[16]  [extern]
```

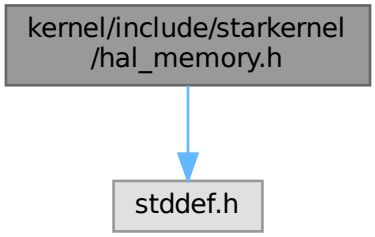
Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), sk_v4_is_this_stick(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

7.44 kernel/include/starkernel/hal/hal.h File Reference

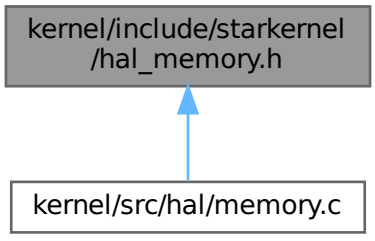
7.45 kernel/include/starkernel/hal_memory.h File Reference

```
#include <stddef.h>
```

Include dependency graph for hal_memory.h:



This graph shows which files directly or indirectly include this file:



Functions

- void * hal_mem_alloc (size_t size)

  hal_memory.h - HAL memory allocation wrappers
- void * hal_mem_alloc_aligned (size_t size, size_t align)
- void hal_mem_free (void *ptr)

7.45.1 Function Documentation

7.45.1.1 hal_mem_alloc()

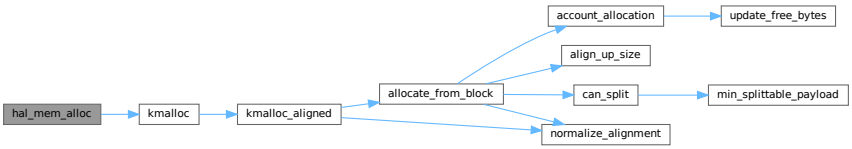
```
void * hal_mem_alloc (
            size_t size )
```

hal_memory.h - HAL memory allocation wrappers

hal_memory.h - HAL memory allocation wrappers

References kmalloc().

Here is the call graph for this function:

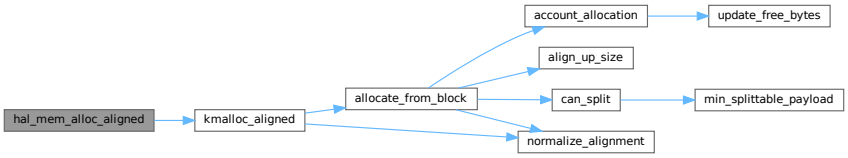


7.45.1.2 hal_mem_alloc_aligned()

```
void * hal_mem_alloc_aligned (
            size_t size,
            size_t align )
```

References kmalloc_aligned().

Here is the call graph for this function:

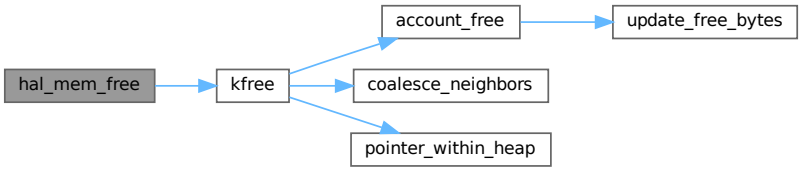


7.45.1.3 hal_mem_free()

```
void hal_mem_free (
            void * ptr )
```

References kfree().

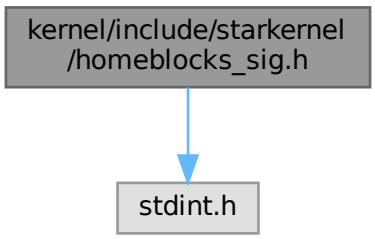
Here is the call graph for this function:



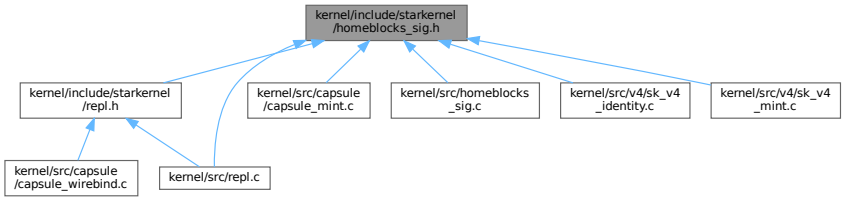
7.46 kernel/include/starkernel/homeblocks_sig.h File Reference

```
#include <stdint.h>
```

Include dependency graph for homeblocks_sig.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct homeblocks_sig_t

Macros

- #define HOMEBLOCKS_SIG_MAGIC

  homeblocks_sig.h - Home-blocks drive signature format (FABRIC-2.md, Milestone 4, Phase 8 kickoff; relocated + GPT dropped ?F.13/?F.8, 2026-08-28)
- #define HOMEBLOCKS_SIG_VERSION_0 0
- #define HOMEBLOCKS_SIG_PACK(ver) (HOMEBLOCKS_SIG_MAGIC | ((uint64_t)(ver) << 32))
- #define HOMEBLOCKS_SIG_GET_MAGIC(m) ((uint32_t)((m) & 0xFFFFFFFFULL))
- #define HOMEBLOCKS_SIG_GET_VERSION(m) ((uint8_t)(((m) >> 32) & 0xFF))
- #define HOMEBLOCKS_SIG_START_FBLOCK 4u

Typedefs

- typedef char homeblocks_sig_size_check[(sizeof(homeblocks_sig_t)==4096) ? 1 :-1]

Enumerations

- enum homeblocks_sig_result_t {
  HOMEBLOCKS_SIG_OK = 0 , HOMEBLOCKS_SIG_BLANK , HOMEBLOCKS_SIG_BAD_VERSION ,
  HOMEBLOCKS_SIG_BAD_CRC ,
  HOMEBLOCKS_SIG_READ_ERROR }

Functions

- homeblocks_sig_result_t homeblocks_sig_check (struct blkio_dev *dev, uint32_t sig_start_fblock, homeblocks_sig_t *out_sig)
- uint64_t homeblocks_sig_compute_crc (const homeblocks_sig_t *sig)

  homeblocks_sig.c - Drive signature check (FABRIC-2.md, Milestone 4).

7.46.1 Macro Definition Documentation

7.46.1.1 HOMEBLOCKS_SIG_GET_MAGIC

```
#define HOMEBLOCKS_SIG_GET_MAGIC(
            m ) ((uint32_t)((m) & 0xFFFFFFFFULL))
```

7.46.1.2 HOMEBLOCKS_SIG_GET_VERSION

```
#define HOMEBLOCKS_SIG_GET_VERSION(
            m ) ((uint8_t)(((m) >> 32) & 0xFF))
```

7.46.1.3 HOMEBLOCKS_SIG_MAGIC

```
#define HOMEBLOCKS_SIG_MAGIC
```

Value:

```
0x4248414CULL /* 'LAHB', same little-endian ASCII
* packing as CAPSULE_DESC_MAGIC's 'CAPS' */
```

homeblocks_sig.h - Home-blocks drive signature format (FABRIC-2.md, Milestone 4, Phase 8 kickoff; relocated + GPT dropped ?F.13/?F.8, 2026-08-28)

Identifies and authenticates a physical thumb drive as a legitimate LithosAnanke home-blocks drive, before any write path touches it. Lives at forth-block HOMEBLOCKS_SIG_START_FBLOCK (devblock 1) of the raw device – GPT partitioning was decided against permanently (?F.8): this is the real, final on-disk location, not an interim stand-in. Devblock 0 is left alone for the block-subsystem's own generic 'STFR'/v2 volume header (block_subsystem.h) – the two formats would otherwise collide (?F.13, found while scoping BMAPREAD).

Mirrors two existing precedents exactly, not invented fresh:

- CAPSULE_MAGIC_PACK's bit-packed magic (starkernel/capsule.h)
- blk_volume_meta_t's magic+version+fields+pad-to-4096 structural convention (block_subsystem.h)

Deliberately narrow in scope: this header identifies/authenticates the DRIVE only. It reserves offset/size pointers to where the cert (CERTVERIFY, ?F.7) and this identity's own personality/init source (RUNCAP/MINT, ?F.6/?F.8) attach, so this format doesn't need revisiting when those get built – it does not itself decide their content.

7.46.1.4 HOMEBLOCKS_SIG_PACK

```
#define HOMEBLOCKS_SIG_PACK(
              ver )   (HOMEBLOCKS_SIG_MAGIC | ((uint64_t)(ver) << 32))
```

7.46.1.5 HOMEBLOCKS_SIG_START_FBLOCK

```
#define HOMEBLOCKS_SIG_START_FBLOCK 4u
```

7.46.1.6 HOMEBLOCKS_SIG_VERSION_0

```
#define HOMEBLOCKS_SIG_VERSION_0 0
```

7.46.2 Typedef Documentation

7.46.2.1 homeblocks_sig_size_check

```
typedef char homeblocks_sig_size_check[(sizeof(homeblocks_sig_t)==4096) ?  1 :-1]
```

7.46.3 Enumeration Type Documentation

7.46.3.1 homeblocks_sig_result_t

```
enum homeblocks_sig_result_t
```

Enumerator

HOMEBLOCKS_SIG_OK	
HOMEBLOCKS_SIG_BLANK	
HOMEBLOCKS_SIG_BAD_VERSION	
HOMEBLOCKS_SIG_BAD_CRC	
HOMEBLOCKS_SIG_READ_ERROR	

7.46.4 Function Documentation

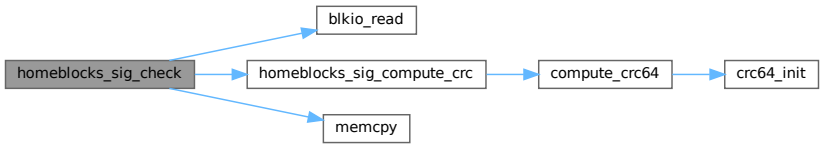
7.46.4.1 homeblocks_sig_check()

```
homeblocks_sig_result_t homeblocks_sig_check (
            struct blkio_dev * dev,
            uint32_t sig_start_fblock,
            homeblocks_sig_t * out_sig )
```

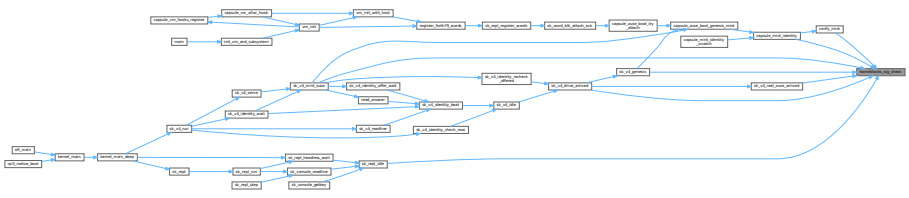
References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_read(), homeblocks_sig_t::hdr_crc, HOMEBLOCKS_SIG_BAD_CRC, HOMEBLOCKS_SIG_BAD_VERSION, HOMEBLOCKS_SIG_BLANK, homeblocks_sig_compute_crc(), HOMEBLOCKS_SIG_GET_M HOMEBLOCKS_SIG_GET_VERSION, HOMEBLOCKS_SIG_MAGIC, HOMEBLOCKS_SIG_OK, HOMEBLOCKS_SIG_READ_ERRO HOMEBLOCKS_SIG_VERSION_0, homeblocks_sig_t::magic, and memcpy().

Referenced by capsule_mint_identity(), sk_repl_idle(), sk_v4_drive_arrived(), sk_v4_genesis(), sk_v4_mint_zuse(), sk_v4_real_zuse_arrived(), and verify_mint().

Here is the call graph for this function:



Here is the caller graph for this function:



7.46.4.2 homeblocks_sig_compute_crc()

```
uint64_t homeblocks_sig_compute_crc (
            const homeblocks_sig_t * sig )
```

homeblocks_sig.c - Drive signature check (FABRIC-2.md, Milestone 4).

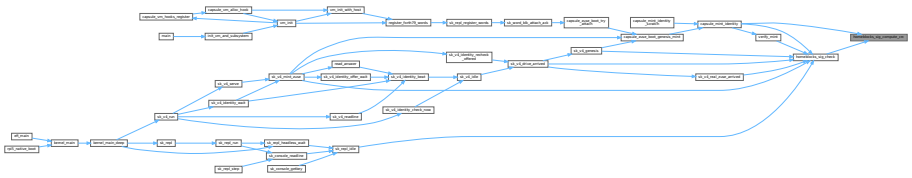
See starkernel/homeblocks_sig.h for the format and interface design.

References compute_crc64().

Referenced by capsule_mint_identity(), and homeblocks_sig_check().

Here is the call graph for this function:

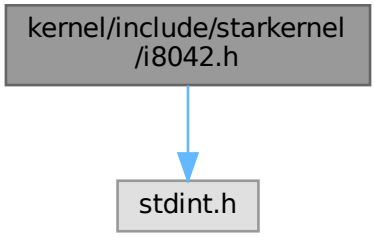
Here is the caller graph for this function:



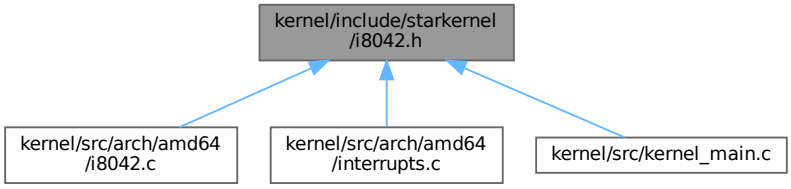
7.47 kernel/include/starkernel/i8042.h File Reference

```
#include <stdint.h>
```

Include dependency graph for i8042.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define I8042_KEYBOARD_VECTOR 0x21

  i8042.h - PS/2 keyboard controller interface (amd64 only)

Functions

- void i8042_init (void)

  Enable IRQ1 delivery in the i8042 controller's command byte.
- void i8042_drain_stale (void)

  Drain any byte sitting in the output buffer right before unmasking IRQ1.
- void i8042_handle_irq (void)

  Called from the keyboard IRQ dispatch path (interrupts.c) on every I8042_KEYBOARD_VECTOR interrupt.
- int i8042_pop_scancode (uint8_t *out)

  Pop one raw scancode off the ring buffer, from non-interrupt context.

Variables

- volatile uint32_t g_i8042_isr_count

7.47.1 Macro Definition Documentation

7.47.1.1 I8042_KEYBOARD_VECTOR

```
#define I8042_KEYBOARD_VECTOR 0x21
```

i8042.h - PS/2 keyboard controller interface (amd64 only)

Item 4.3.5 (FABRIC-0.md ?27.5). Interrupt-driven only — no polling of the status port (0x64) anywhere in this path. Groundwork only: this captures and prints raw scancodes. Scancode-to-keycode translation and a consumer API belong to the REPL keyboard-input work noted in FABRIC-0.md, not here.

7.47.2 Function Documentation

7.47.2.1 i8042_drain_stale()

```
void i8042_drain_stale (
            void  )
```

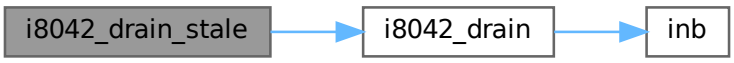
Drain any byte sitting in the output buffer right before unmasking IRQ1.

Edge-triggered lines only assert on a rising edge; if OBF is already set (from real wall-clock time elapsing between i8042_init() and unmask while OBF was already high), there is no edge left to fire on, ever.

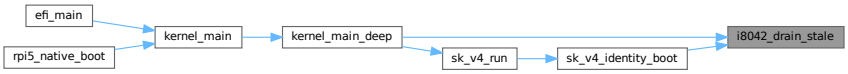
References g_i8042_present, and i8042_drain().

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.47.2.2 i8042_handle_irq()

```
void i8042_handle_irq (
            void  )
```

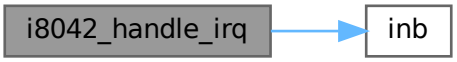
Called from the keyboard IRQ dispatch path (interrupts.c) on every I8042_KEYBOARD_VECTOR interrupt.

Trivial by design: reads the scancode from port 0x60 (must always be read, or the controller never clears OBF and stops delivering further interrupts) and pushes it onto a small ring buffer. No printing, no translation, no other work — that all happens outside interrupt context via i8042_pop_scancode(). Does not issue apic_eoi() — the caller does.

References g_i8042_isr_count, I8042_DATA_PORT, inb(), ring, ring_head, RING_SIZE, and ring_tail.

Referenced by isr_common_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

7.47.2.3 i8042_init()

```
void i8042_init (
            void  )
```

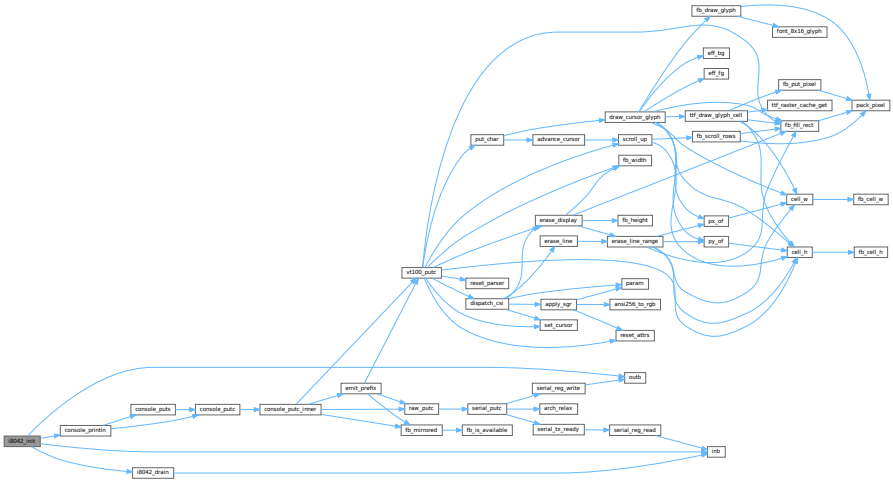
Enable IRQ1 delivery in the i8042 controller's command byte.

Assumes the controller was already brought up by firmware (OVMF) — does not run the 0xAA self-test or the full two-port init sequence, since that is more than this item's scope requires.

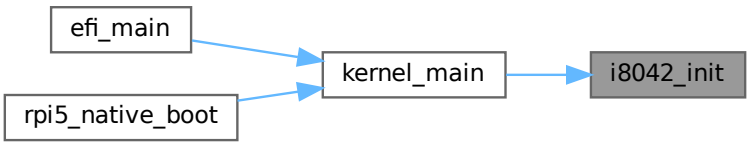
References console_println(), g_i8042_present, I8042_CMD_PORT, I8042_CMD_READ_CONFIG, I8042_CMD_WRITE_CONFIG, I8042_CONFIG_IRQ1_ENABLE, I8042_DATA_PORT, I8042_DEV_ACK, I8042_DEV_CMD_ENABLE_SCANNING, i8042_drain(), I8042_INIT_WAIT_ITERS, I8042_STATUS_INPUT_FULL, I8042_STATUS_OUTPUT_FULL, I8042_STATUS_PORT, inb(), and outb().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.47.2.4 i8042_pop_scancode()

```
int i8042_pop_scancode (
            uint8_t * out )
```

Pop one raw scancode off the ring buffer, from non-interrupt context.

Single producer (the ISR), single reader (this function) — same shape already established safe on one hart elsewhere in this tree (?21.1).

Parameters

out	Written with the popped scancode on success.

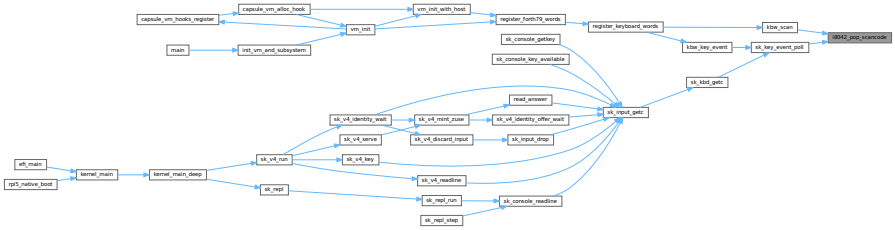
Returns

1 if a scancode was popped, 0 if the buffer was empty.

References ring, ring_head, RING_SIZE, and ring_tail.

Referenced by kbw_scan(), and sk_key_event_poll().

Here is the caller graph for this function:



7.47.3 Variable Documentation

7.47.3.1 g_i8042_isr_count

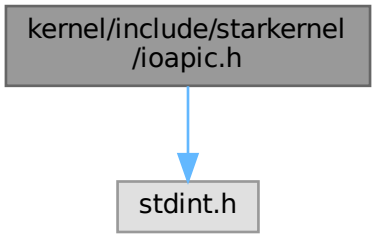
```
volatile uint32_t g_i8042_isr_count  [extern]
```

Referenced by i8042_handle_irq().

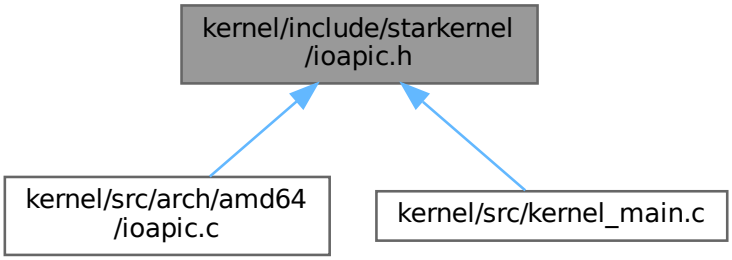
7.48 kernel/include/starkernel/ioapic.h File Reference

```
#include <stdint.h>
```

Include dependency graph for ioapic.h:



This graph shows which files directly or indirectly include this file:



Functions

- int ioapic_init (void *acpi_rsdp)

  ioapic.h - I/O APIC interface (amd64 only)
- int ioapic_route_legacy_irq (uint8_t isa_irq, uint8_t vector, uint8_t dest_apic_id)

  Program a redirection entry for a legacy ISA IRQ, initially masked.
- void ioapic_unmask_legacy_irq (uint8_t isa_irq)

  Clear the mask bit on a previously routed legacy IRQ's redirection entry.

7.48.1 Function Documentation

7.48.1.1 ioapic_init()

```
int ioapic_init (
            void * acpi_rsdp )
```

ioapic.h - I/O APIC interface (amd64 only)

Item 4.3.5 (FABRIC-0.md ?27.5): no I/O APIC driver existed anywhere in this tree before this item. The Local APIC (apic.h) self-interrupts for the timer and needs no routing; any *legacy* IRQ (i8042 keyboard's IRQ1 included) requires the I/O APIC to redirect it to a Local APIC vector. Locate and map the I/O APIC via the ACPI MADT (parsed from `acpi_rsdp`). Also captures any Interrupt Source Override entries so legacy ISA IRQs with a non-default GSI/polarity/trigger mapping route correctly.

Parameters

acpi_rsdp	BootInfo->acpi_table (RSDP), or NULL.

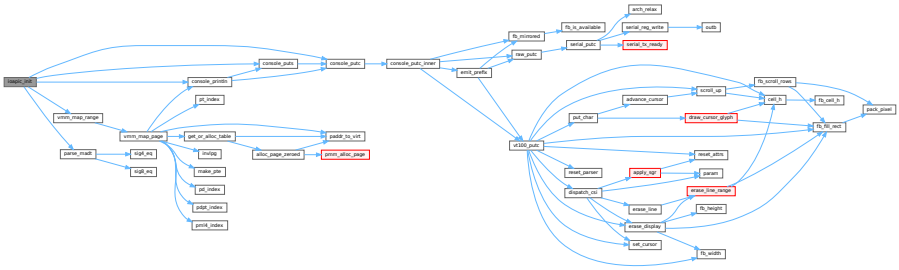
Returns

0 on success, -1 if no MADT/I/O APIC entry was found.

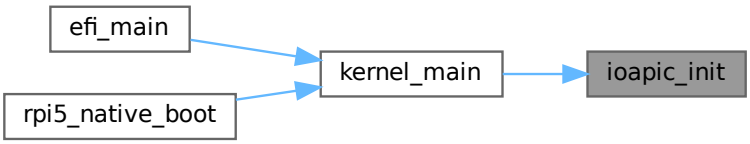
References console_println(), console_putc(), console_puts(), flags, g_ioapic_base, g_ioapic_gsi_base, g_ioapic_ready, g_override_count, g_overrides, parse_madt(), VMM_FLAG_CACHE_DISABLE, VMM_FLAG_WRITABLE, and vmm_map_range().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.48.1.2 ioapic_route_legacy_irq()

```
int ioapic_route_legacy_irq (
            uint8_t isa_irq,
            uint8_t vector,
            uint8_t dest_apic_id )
```

Program a redirection entry for a legacy ISA IRQ, initially masked.

Honors an Interrupt Source Override for `isa_irq` if the MADT had one (different GSI, polarity, or trigger mode); otherwise uses the ISA default (GSI == isa_irq, active-high, edge-triggered) per the ACPI specification's "conforms to bus" default.

Parameters

isa_irq	Legacy IRQ number (0-15).
vector	IDT vector to deliver (e.g. 0x21).
dest_apic↩ _id	Destination Local APIC ID (see apic_id()).

Returns

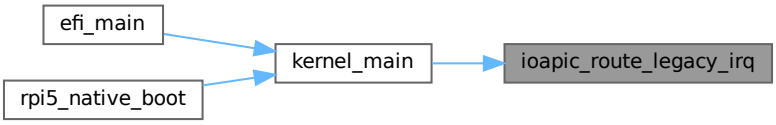
0 on success, -1 if ioapic_init() has not succeeded.

References flags, g_ioapic_gsi_base, g_ioapic_ready, g_override_count, g_overrides, g_redir_low, g_redir_valid, ioapic_write(), ISO_POLARITY_LOW, ISO_POLARITY_MASK, ISO_TRIGGER_LEVEL, ISO_TRIGGER_MASK, and MAX_REDIR_ENTRIES.

Referenced by kernel_main().

Here is the call graph for this function:

Here is the caller graph for this function:



7.48.1.3 ioapic_unmask_legacy_irq()

```
void ioapic_unmask_legacy_irq (
            uint8_t isa_irq )
```

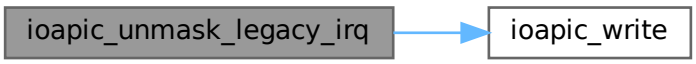
Clear the mask bit on a previously routed legacy IRQ's redirection entry.

Call only after ioapic_route_legacy_irq() has programmed it.

References g_ioapic_gsi_base, g_ioapic_ready, g_override_count, g_overrides, g_redir_low, g_redir_valid, ioapic_write(), and MAX_REDIR_ENTRIES.

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

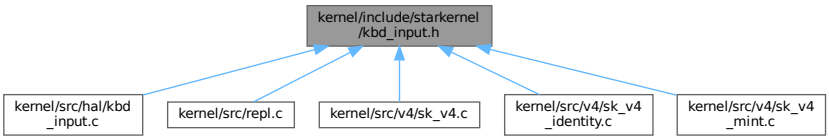
Here is the call graph for this function:



Here is the caller graph for this function:

7.49 kernel/include/starkernel/kbd_input.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int sk_input_getc (void)
- void sk_input_drop (void)

7.49.1 Function Documentation

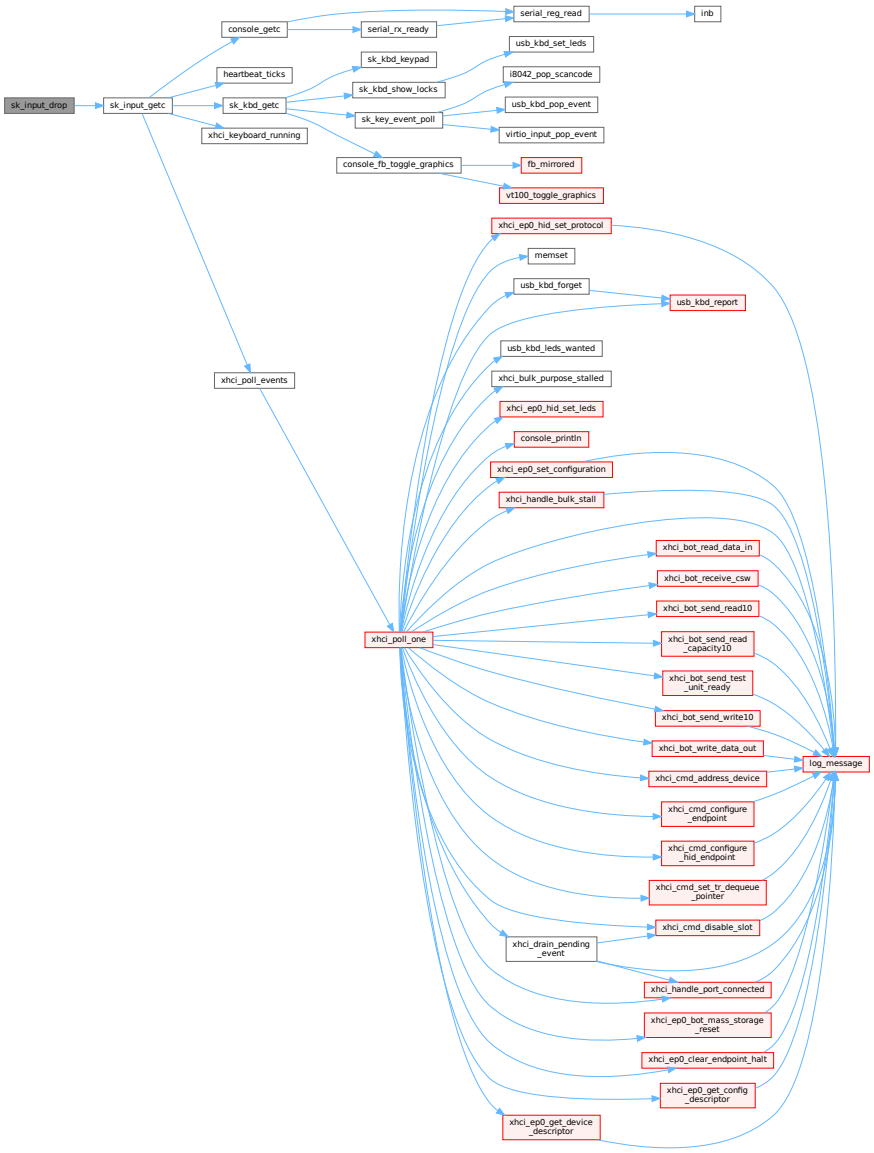
7.49.1.1 sk_input_drop()

```
void sk_input_drop (
            void  )
```

References sk_input_getc().

Referenced by sk_v4_discard_input().

Here is the call graph for this function:



Here is the caller graph for this function:

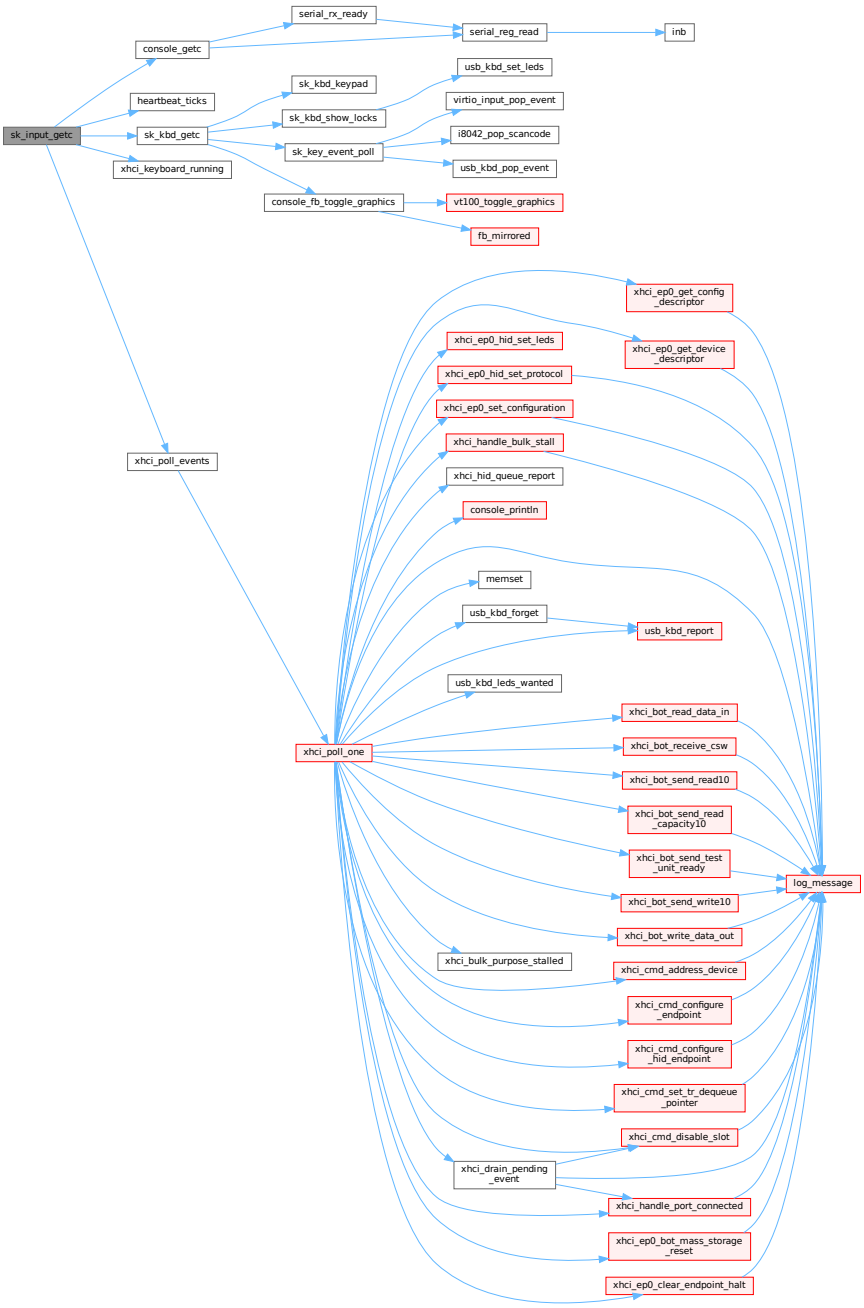
7.49.1.2 sk_input_getc()

```
int sk_input_getc (
            void  )
```

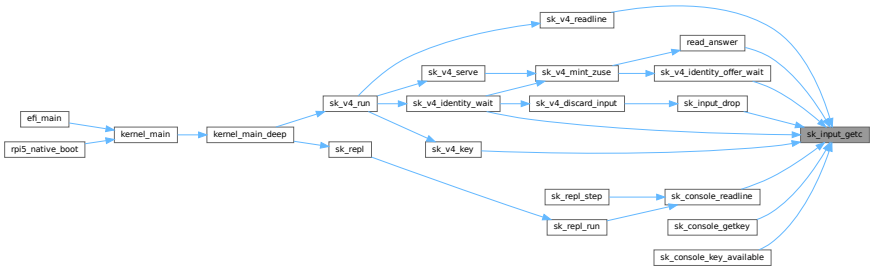
References console_getc(), g_kbd_poll_tick, heartbeat_ticks(), sk_kbd_getc(), xhci_keyboard_running(), and xhci_poll_events().

Referenced by read_answer(), sk_console_getkey(), sk_console_key_available(), sk_console_readline(), sk_input_drop(), sk_v4_identity_offer_wait(), sk_v4_identity_wait(), sk_v4_key(), and sk_v4_readline().

Here is the call graph for this function:

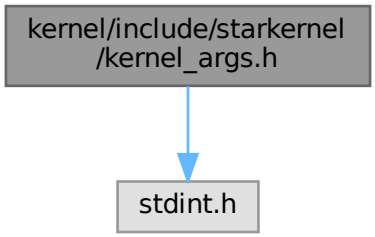


Here is the caller graph for this function:



7.50 kernel/include/starkernel/kernel_args.h File Reference

`#include <stdint.h>`
Include dependency graph for kernel_args.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct KernelArgs

Macros

- #define KERNEL_ARGS_CMDLINE_MAX 512
- #define KARGS_DEFAULT_STACK_SIZE (2ULL ∗ 1024ULL ∗ 1024ULL) /∗ 2 MB ∗/
- #define KARGS_DEFAULT_HEAP_SIZE (2ULL ∗ 1024ULL ∗ 1024ULL ∗ 1024ULL) /∗ 2 GB ∗/
- #define KARGS_DEFAULT_LOG_LEVEL 2 /∗ warn — quiet by default; --log-level=info/debug re-enables ECW word-execution tracing ∗/

- #define KARGS_DEFAULT_RUN_DOE 0
- #define KARGS_SYSTEM_DEFAULT 0
- #define KARGS_SYSTEM_V3 3
- #define KARGS_SYSTEM_V4 4
- #define KARGS_LOG_DEBUG 0
- #define KARGS_LOG_INFO 1
- #define KARGS_LOG_WARN 2
- #define KARGS_LOG_ERROR 3
- #define REBOOT_MAX_TRIES 3

7.50.1 Macro Definition Documentation

7.50.1.1 KARGS_DEFAULT_HEAP_SIZE

```
#define KARGS_DEFAULT_HEAP_SIZE (2ULL * 1024ULL * 1024ULL * 1024ULL) /* 2 GB */
```

7.50.1.2 KARGS_DEFAULT_LOG_LEVEL

```
#define KARGS_DEFAULT_LOG_LEVEL 2 /* warn -- quiet by default; --log-level=info/debug re-enables
ECW word-execution tracing */
```

7.50.1.3 KARGS_DEFAULT_RUN_DOE

```
#define KARGS_DEFAULT_RUN_DOE 0
```

7.50.1.4 KARGS_DEFAULT_STACK_SIZE

```
#define KARGS_DEFAULT_STACK_SIZE (2ULL * 1024ULL * 1024ULL) /* 2 MB */
```

7.50.1.5 KARGS_LOG_DEBUG

```
#define KARGS_LOG_DEBUG 0
```

7.50.1.6 KARGS_LOG_ERROR

```
#define KARGS_LOG_ERROR 3
```

7.50.1.7 KARGS_LOG_INFO

```
#define KARGS_LOG_INFO 1
```

7.50.1.8 KARGS_LOG_WARN

```
#define KARGS_LOG_WARN 2
```

7.50.1.9 KARGS_SYSTEM_DEFAULT

```
#define KARGS_SYSTEM_DEFAULT 0
```

7.50.1.10 KARGS_SYSTEM_V3

```
#define KARGS_SYSTEM_V3 3
```

7.50.1.11 KARGS_SYSTEM_V4

```
#define KARGS_SYSTEM_V4 4
```

7.50.1.12 KERNEL_ARGS_CMDLINE_MAX

```
#define KERNEL_ARGS_CMDLINE_MAX 512
```

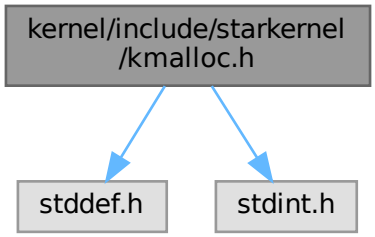
7.50.1.13 REBOOT_MAX_TRIES

```
#define REBOOT_MAX_TRIES 3
```

7.51 kernel/include/starkernel/kmalloc.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for kmalloc.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct kmalloc_stats_t

  kmalloc.h - Kernel heap allocator interface

Functions

- int kmalloc_init (uint64_t heap_size_bytes)
- int kmalloc_is_initialized (void)
- void * kmalloc (size_t size)
- void * kmalloc_aligned (size_t size, size_t align)
- void kfree (void *ptr)
- kmalloc_stats_t kmalloc_get_stats (void)
- uintptr_t kmalloc_heap_base_addr (void)
- uintptr_t kmalloc_heap_end_addr (void)
- void kmalloc_debug_census (size_t *out_largest_free, size_t *out_free_count, size_t *out_used_count, size_t *out_total_blocks)
- void kmalloc_debug_census_bytes (size_t *out_total_free_bytes, size_t *out_total_used_bytes)

7.51.1 Function Documentation

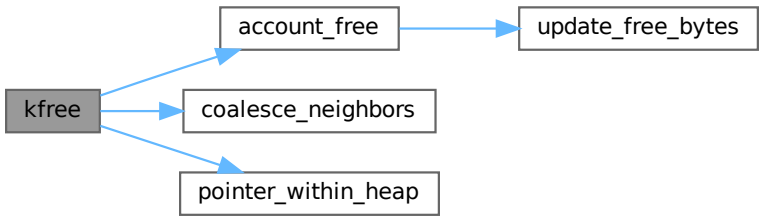
7.51.1.1 kfree()

```
void kfree (
            void * ptr )
```

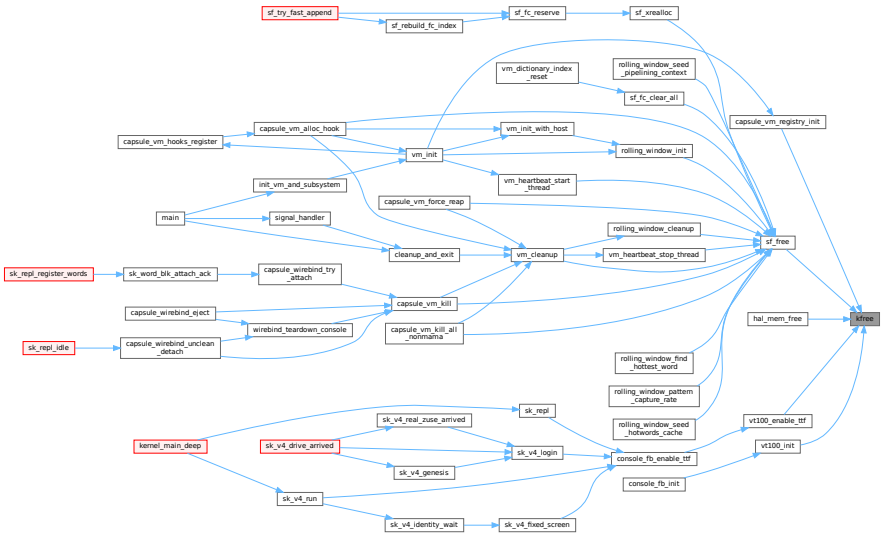
References account_free(), kmalloc_prefix_t::block, coalesce_neighbors(), heap_block::free, heap_initialized, KMALLOC_MAGIC, kmalloc_prefix_t::magic, pointer_within_heap(), and heap_block::requested.

Referenced by capsule_vm_registry_init(), hal_mem_free(), sf_free(), vt100_enable_ttf(), and vt100_init().

Here is the call graph for this function:



Here is the caller graph for this function:



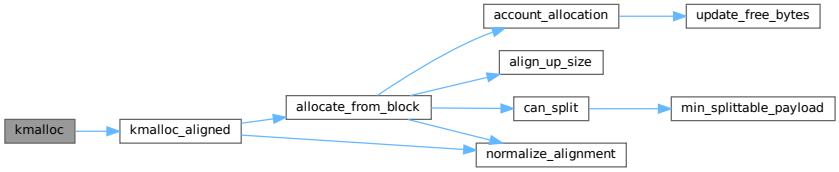
7.51.1.2 kmalloc()

```
void * kmalloc (
            size_t size )
```

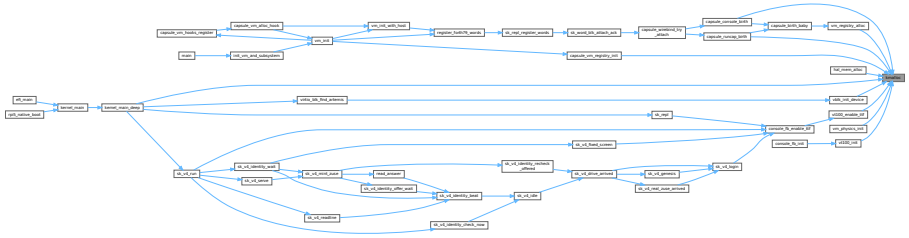
References kmalloc_aligned(), and KMALLOC_MIN_ALIGN.

Referenced by capsule_console_birth(), capsule_runcap_birth(), capsule_vm_registry_init(), hal_mem_alloc(), kernel_main_deep(), vblk_init_device(), vm_physics_init(), vm_registry_alloc(), vt100_enable_ttf(), and vt100_init().

Here is the call graph for this function:



Here is the caller graph for this function:



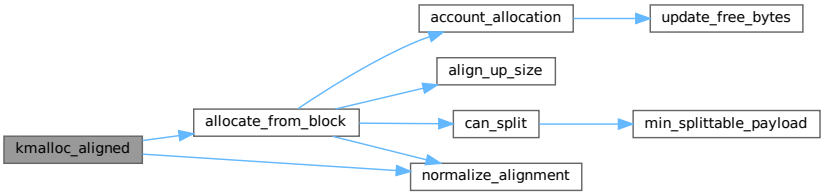
7.51.1.3 kmalloc_aligned()

```
void * kmalloc_aligned (
            size_t size,
            size_t align )
```

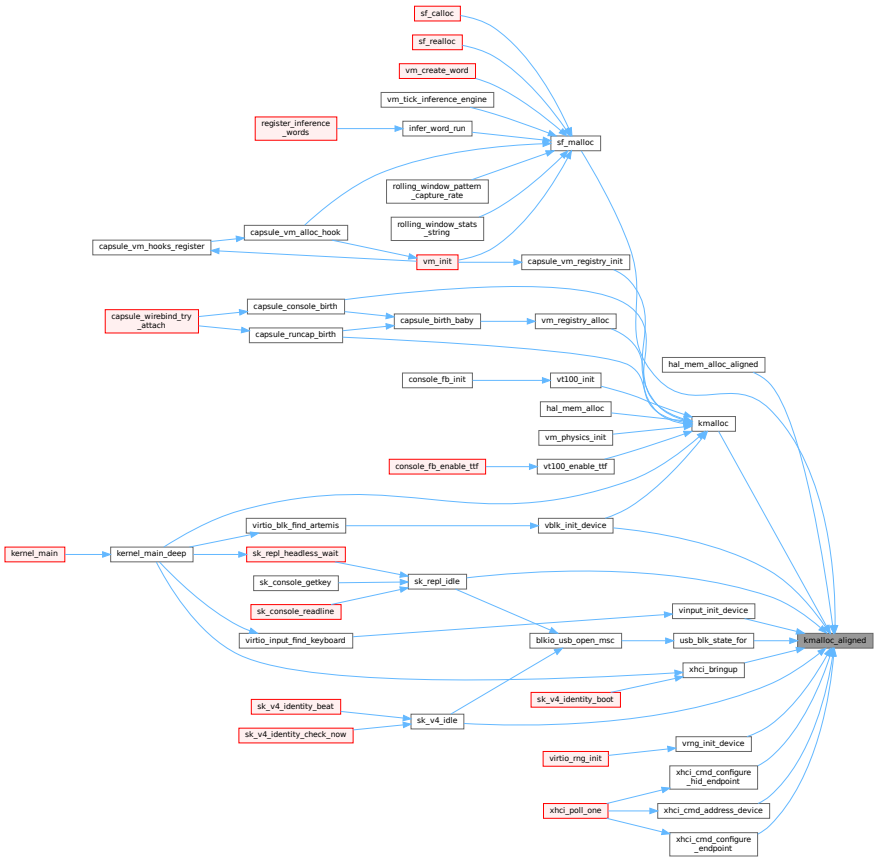
References allocate_from_block(), heap_block::free, heap_head, heap_initialized, heap_block::next, and normalize_alignment().

Referenced by hal_mem_alloc_aligned(), kmalloc(), sf_malloc(), sk_repl_idle(), sk_v4_idle(), usb_blk_state_for(), vblk_init_device(), vinput_init_device(), vrng_init_device(), xhci_bringup(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint and xhci_cmd_configure_hid_endpoint().

Here is the call graph for this function:



Here is the caller graph for this function:



7.51.1.4 kmalloc_debug_census()

```
void kmalloc_debug_census (
            size_t * out_largest_free,
            size_t * out_free_count,
            size_t * out_used_count,
            size_t * out_total_blocks )
```

References heap_block::free, heap_head, heap_block::next, and heap_block::size.

7.51.1.5 kmalloc_debug_census_bytes()

```
void kmalloc_debug_census_bytes (
            size_t * out_total_free_bytes,
            size_t * out_total_used_bytes )
```

References heap_block::free, heap_head, heap_block::next, and heap_block::size.

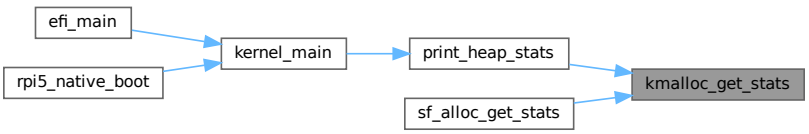
7.51.1.6 kmalloc_get_stats()

```
kmalloc_stats_t kmalloc_get_stats (
            void  )
```

References heap_stats.

Referenced by print_heap_stats(), and sf_alloc_get_stats().

Here is the caller graph for this function:



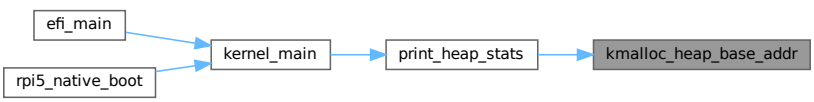
7.51.1.7 kmalloc_heap_base_addr()

```
uintptr_t kmalloc_heap_base_addr (
            void  )
```

References heap_base.

Referenced by print_heap_stats().

Here is the caller graph for this function:



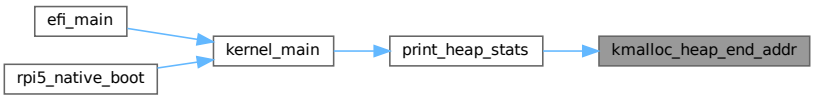
7.51.1.8 kmalloc_heap_end_addr()

```
uintptr_t kmalloc_heap_end_addr (
            void  )
```

References heap_base, and heap_size.

Referenced by print_heap_stats().

Here is the caller graph for this function:



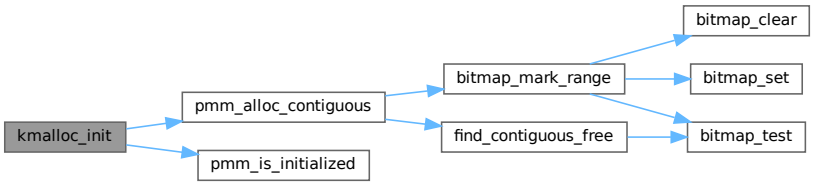
7.51.1.9 kmalloc_init()

```
int kmalloc_init (
            uint64_t heap_size_bytes )
```

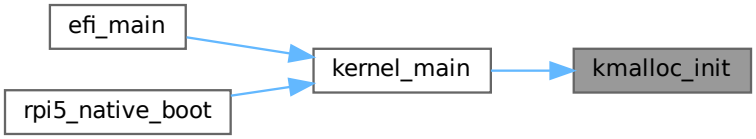
References heap_block::free, kmalloc_stats_t::free_bytes, heap_base, heap_head, heap_initialized, heap_size, heap_stats, KMALLOC_DEFAULT_HEAP_SIZE, KMALLOC_MIN_PAGES, heap_block::next, kmalloc_stats_t::peak_bytes, pmm_alloc_contiguous(), pmm_is_initialized(), PMM_PAGE_SIZE, heap_block::prev, heap_block::requested, heap_block::size, kmalloc_stats_t::total_bytes, and kmalloc_stats_t::used_bytes.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.51.1.10 kmalloc_is_initialized()

```
int kmalloc_is_initialized (
            void  )
```

References heap_initialized.

7.52 kernel/include/starkernel/log_attrib.h File Reference

Functions

- struct VM * vm_log_attributed_vm (void)

7.52.1 Function Documentation

7.52.1.1 vm_log_attributed_vm()

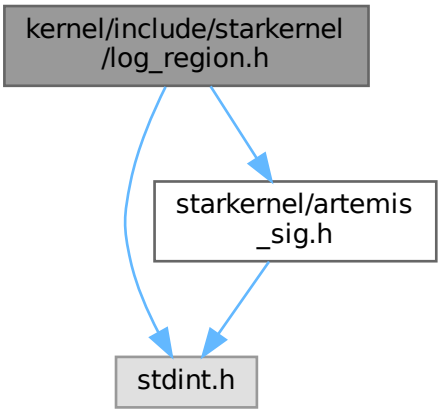
```
struct VM * vm_log_attributed_vm (
            void  )
```

References g_log_attrib_vm.

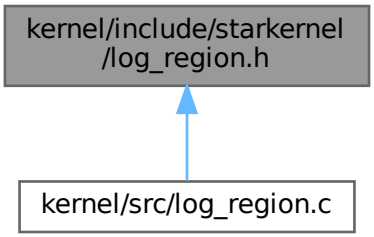
7.53 kernel/include/starkernel/log_region.h File Reference

```
#include <stdint.h>
#include "starkernel/artemis_sig.h"
```

Include dependency graph for log_region.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct log_region_ctrl_t
- struct log_slot_t

Macros

- #define LOG_REGION_DEVBLOCK_FROM_TOP_BASE (ARTEMIS_SIG_DEVBLOCK_FROM_TOP + 1u) /* 65 */

  log_region.h - Growable per-VM log-persistence ring on Artemis's own disk (FABRIC-3.md ?XXVI follow-on, Step 4, 2026-09-13)
- #define LOG_REGION_INITIAL_DEVBLOCKS 4u /* slot devblocks at first use, excludes control header */
- #define LOG_REGION_GROWTH_INCREMENT 4u
- #define LOG_REGION_MAX_DEVBLOCKS 32u /* ceiling -- devblock_from_top stays within [66,96] */
- #define LOG_REGION_PROTECTED_MAX_LEVEL 1u /* LOG_WARN and below (ERROR, WARN) */
- #define LOG_SLOTS_PER_DEVBLOCK 4u /* 4 x 1 KiB forth-blocks per 4 KiB devblock */
- #define LOG_SLOT_SIZE 1024u
- #define LOG_REGION_MAGIC 0x474C474Cull /* 'LGLG' */
- #define LOG_REGION_VERSION_0 0
- #define LOG_REGION_PACK(ver) (LOG_REGION_MAGIC | ((uint64_t)(ver) << 32))
- #define LOG_REGION_GET_MAGIC(m) ((uint32_t)((m) & 0xFFFFFFFFull))
- #define LOG_REGION_GET_VERSION(m) ((uint8_t)(((m) >> 32) & 0xFF))
- #define LOG_SLOT_SOURCE_MAX 16u /* NUL-padded VM/source tag, e.g. "Hera", "HADES" */
- #define LOG_SLOT_MSG_MAX (LOG_SLOT_SIZE - 8u - 2u - 1u - LOG_SLOT_SOURCE_MAX) /* 997 */

Typedefs

- typedef char log_region_ctrl_size_check[(sizeof(log_region_ctrl_t)==4096) ? 1 :-1]
- typedef char log_slot_size_check[(sizeof(log_slot_t)==1024u) ? 1 :-1]

Functions

- int log_region_append (uint8_t level, uint64_t timestamp, const char *source, uint32_t source_len, const char *msg, uint32_t msg_len)
- int log_region_read_recent (uint32_t back, log_slot_t *out)

  log_region_read_recent - read one record back (2026-10-14, for LOG-SHOW).

7.53.1 Macro Definition Documentation

7.53.1.1 LOG_REGION_DEVBLOCK_FROM_TOP_BASE

```
#define LOG_REGION_DEVBLOCK_FROM_TOP_BASE (ARTEMIS_SIG_DEVBLOCK_FROM_TOP + 1u) /* 65 */
```

log_region.h - Growable per-VM log-persistence ring on Artemis's own disk (FABRIC-3.md ?XXVI follow-on, Step 4, 2026-09-13)

Records real log_message() output (INFO and above, independent of the console's own current_level filter) so a run survives past the QEMU serial log – the motivating case is comparing bare-metal and QEMU results directly as a DoE factor once bare-metal boot lands, where the only artifact both platforms share is whatever got persisted to disk.

Lives in the SAME top-of-device system-metadata fence artemis_sig_t and Zuse's genesis marker already uses (block_subsystem.h's blk_meta_zone_read()/write(), devblock_from_top addressing) – reached the SAME way, deliberately NOT through artemis_sig.c's own independent blkio_info()-based arithmetic. That arithmetic exists only because artemis_sig_t must be discoverable on a device that isn't attached yet (bus-agnostic discovery, repl.c's idle-loop USB-MSC scan); this ring is only ever read or written once Artemis's disk is already attached and formatted (LOG-APPEND runs inside Artemis's own already-live dictionary context, dispatched by a sender's MSG-SEND/↩ MSG-TICK), so blk_meta_zone_∗() – which requires exactly that already-attached state – is the correct, simpler, already-tested tool, not a limitation to work around.

Fixed devblock_from_top allocation (NOT persisted in artemis_sig_t's own log_region_offset/log_region_devblocks fields – those stay at their genesis-time value of 0/0, "informational only, control header below is authoritative," documented in artemis_sig.h; two writers of the same fact was rejected as unnecessary drift risk): 64 – artemis_sig_t itself (ARTEMIS_SIG_DEVBLOCK_FROM_TOP) 65 – this ring's control header (LOG_REGION_DEVBLOCK_↩ FROM_TOP_BASE) 66-96 – this ring's slot devblocks, growable up to LOG_REGION_MAX_DEVBLOCKS Chosen clear of Zuse's genesis marker (devblock_from_top=0) and of devblocks 1-63, reserved and unused since Zuse's eligibility list was removed (2026-10-14, FABRIC-3.md SXXXVII).

Ring granularity is one whole devblock-quarter (LOG_SLOT_SIZE, 1024 bytes = exactly one blkio forth-block) per record – avoids any partial-forth-block read-modify-write for the write itself (a slot write is one aligned blkio_write()-equivalent-sized unit); the surrounding devblock (4 slots) still needs a read-modify-write via blk_meta_zone_read()/write() since that accessor's own unit is one full 4 KiB devblock, but that cost is the same regardless of slot size. Records are truncated to fit one slot rather than spanning multiple – simple, real, and sufficient for what's actually persisted (see vm_log_buffer.h's own tighter caps, chosen to fit a whole batch of several records inside one VM-EXEC/MSG-SEND line, INPUT_BUFFER_SIZE=1025).

7.53.1.2 LOG_REGION_GET_MAGIC

```
#define LOG_REGION_GET_MAGIC(
              m ) ((uint32_t)((m) & 0xFFFFFFFFull))
```

7.53.1.3 LOG_REGION_GET_VERSION

```
#define LOG_REGION_GET_VERSION(
              m ) ((uint8_t)(((m) >> 32) & 0xFF))
```

7.53.1.4 LOG_REGION_GROWTH_INCREMENT

```
#define LOG_REGION_GROWTH_INCREMENT 4u
```

7.53.1.5 LOG_REGION_INITIAL_DEVBLOCKS

```
#define LOG_REGION_INITIAL_DEVBLOCKS 4u /* slot devblocks at first use, excludes control header
*/
```

7.53.1.6 LOG_REGION_MAGIC

```
#define LOG_REGION_MAGIC 0x474C474Cull /* 'LGLG' */
```

7.53.1.7 LOG_REGION_MAX_DEVBLOCKS

```
#define LOG_REGION_MAX_DEVBLOCKS 32u /* ceiling -- devblock_from_top stays within [66,96] */
```

7.53.1.8 LOG_REGION_PACK

```
#define LOG_REGION_PACK(
             ver )  (LOG_REGION_MAGIC | ((uint64_t)(ver) << 32))
```

7.53.1.9 LOG_REGION_PROTECTED_MAX_LEVEL

```
#define LOG_REGION_PROTECTED_MAX_LEVEL 1u /* LOG_WARN and below (ERROR, WARN) */
```

7.53.1.10 LOG_REGION_VERSION_0

```
#define LOG_REGION_VERSION_0 0
```

7.53.1.11 LOG_SLOT_MSG_MAX

```
#define LOG_SLOT_MSG_MAX (LOG_SLOT_SIZE - 8u - 2u - 1u - LOG_SLOT_SOURCE_MAX) /* 997 */
```

7.53.1.12 LOG_SLOT_SIZE

```
#define LOG_SLOT_SIZE 1024u
```

7.53.1.13 LOG_SLOT_SOURCE_MAX

```
#define LOG_SLOT_SOURCE_MAX 16u /* NUL-padded VM/source tag, e.g.  "Hera", "HADES" */
```

7.53.1.14 LOG_SLOTS_PER_DEVBLOCK

```
#define LOG_SLOTS_PER_DEVBLOCK 4u /* 4 x 1 KiB forth-blocks per 4 KiB devblock */
```

7.53.2 Typedef Documentation

7.53.2.1 log_region_ctrl_size_check

```
typedef char log_region_ctrl_size_check[(sizeof(log_region_ctrl_t)==4096) ?  1 :-1]
```

7.53.2.2 log_slot_size_check

```
typedef char log_slot_size_check[(sizeof(log_slot_t)== 1024u) ?  1 :-1]
```

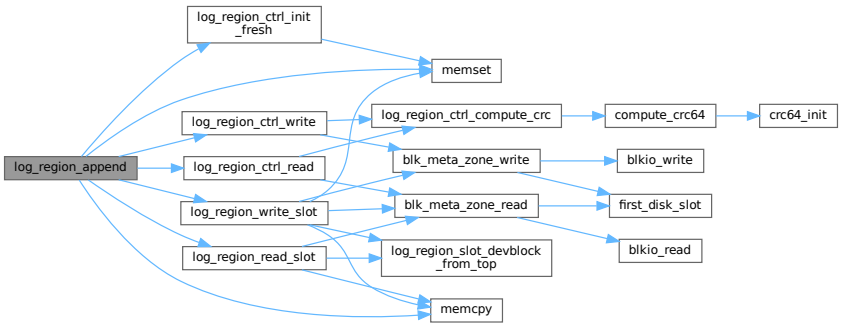
7.53.3 Function Documentation

7.53.3.1 log_region_append()

```
int log_region_append (
            uint8_t level,
            uint64_t timestamp,
            const char * source,
            uint32_t source_len,
            const char * msg,
            uint32_t msg_len )
```

References log_region_ctrl_t::devblocks, log_region_ctrl_t::head_slot, log_slot_t::level, level, log_region_ctrl_init_fresh(), log_region_ctrl_read(), log_region_ctrl_write(), LOG_REGION_GROWTH_INCREMENT, LOG_REGION_MAX_DEVBLOCKS, LOG_REGION_PROTECTED_MAX_LEVEL, log_region_read_slot(), log_region_write_slot(), LOG_SLOT_MSG_MAX, LOG_SLOT_SOURCE_MAX, LOG_SLOTS_PER_DEVBLOCK, memcpy(), memset(), log_slot_t::msg, log_slot_t::msg_len, log_region_ctrl_t::record_count, log_slot_t::source, log_region_ctrl_t::tail_slot, and log_slot_t::timestamp.

Here is the call graph for this function:



7.53.3.2 log_region_read_recent()

```
int log_region_read_recent (
            uint32_t back,
            log_slot_t * out )
```

log_region_read_recent - read one record back (2026-10-14, for LOG-SHOW).

Parameters

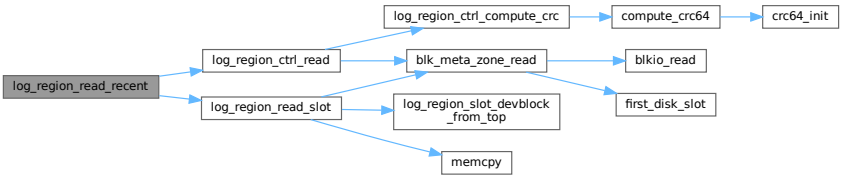
back	0 for the newest record, 1 for the one before it, ...
out	Filled with the record.

Returns

0 on success, 1 if there is no such record (the ring holds fewer than back+1, or none at all), -1 on a read failure or no ring.

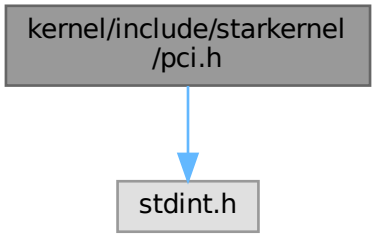
References log_region_ctrl_t::devblocks, log_region_ctrl_read(), log_region_read_slot(), LOG_SLOTS_PER_DEVBLOCK, log_region_ctrl_t::record_count, and log_region_ctrl_t::tail_slot.

Here is the call graph for this function:

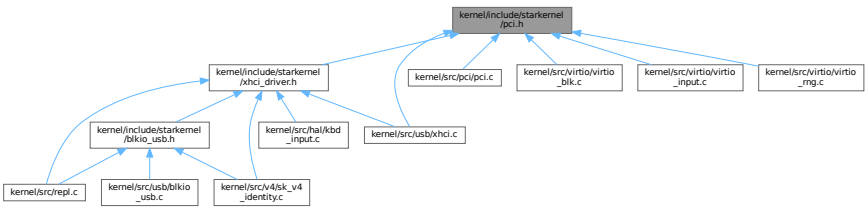


7.54 kernel/include/starkernel/pci.h File Reference

`#include <stdint.h>`
Include dependency graph for pci.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct PciDevice

Macros

- #define PCI_CFG_VENDOR_ID 0x00
- #define PCI_CFG_DEVICE_ID 0x02
- #define PCI_CFG_COMMAND 0x04
- #define PCI_CFG_STATUS 0x06
- #define PCI_CFG_CLASS_REV 0x08
- #define PCI_CFG_HEADER_TYPE 0x0E
- #define PCI_CFG_BAR0 0x10
- #define PCI_CFG_BAR1 0x14
- #define PCI_CFG_BAR2 0x18
- #define PCI_CFG_BAR3 0x1C
- #define PCI_CFG_BAR4 0x20
- #define PCI_CFG_BAR5 0x24
- #define PCI_CFG_CAP_PTR 0x34
- #define PCI_CFG_INT_LINE 0x3C
- #define PCI_CFG_INT_PIN 0x3D /∗ 0=none, 1=INTA .. 4=INTD (item 4.3.5c/4.3.5d) ∗/
- #define PCI_CMD_IO_SPACE (1u << 0)
- #define PCI_CMD_MEM_SPACE (1u << 1)
- #define PCI_CMD_BUS_MASTER (1u << 2)
- #define PCI_CMD_INTX_DISABLE
- #define PCI_BAR_TYPE_MASK 0x1u
- #define PCI_BAR_IO 0x1u
- #define PCI_BAR_MEM_64 0x4u /∗ bits [2:1] == 10 in memory BAR ∗/

Functions

- int pci_init (void ∗rsdp)
- int pci_find_first (uint16_t vendor_id, uint16_t device_id, PciDevice ∗out)
- int pci_find_class (uint8_t class_code, uint8_t subclass, uint8_t prog_if, unsigned index, PciDevice ∗out)
- uint32_t pci_read32 (const PciDevice ∗d, uint16_t offset)
- uint16_t pci_read16 (const PciDevice ∗d, uint16_t offset)
- uint8_t pci_read8 (const PciDevice ∗d, uint16_t offset)
- void pci_write32 (const PciDevice ∗d, uint16_t offset, uint32_t val)
- void pci_write16 (const PciDevice ∗d, uint16_t offset, uint16_t val)
- void pci_write8 (const PciDevice ∗d, uint16_t offset, uint8_t val)
- void pci_enable (const PciDevice ∗d)
- uint64_t pci_bar (const PciDevice ∗d, int bar_idx)
- uint64_t pci_bar_size (const PciDevice ∗d, int bar_idx)
- int pci_map_bar (uint64_t phys_addr, uint64_t size)

7.54.1 Macro Definition Documentation

7.54.1.1 PCI_BAR_IO

```
#define PCI_BAR_IO 0x1u
```

7.54.1.2 PCI_BAR_MEM_64

```
#define PCI_BAR_MEM_64 0x4u /* bits [2:1] == 10 in memory BAR */
```

7.54.1.3 PCI_BAR_TYPE_MASK

```
#define PCI_BAR_TYPE_MASK 0x1u
```

7.54.1.4 PCI_CFG_BAR0

```
#define PCI_CFG_BAR0 0x10
```

7.54.1.5 PCI_CFG_BAR1

```
#define PCI_CFG_BAR1 0x14
```

7.54.1.6 PCI_CFG_BAR2

```
#define PCI_CFG_BAR2 0x18
```

7.54.1.7 PCI_CFG_BAR3

```
#define PCI_CFG_BAR3 0x1C
```

7.54.1.8 PCI_CFG_BAR4

```
#define PCI_CFG_BAR4 0x20
```

7.54.1.9 PCI_CFG_BAR5

```
#define PCI_CFG_BAR5 0x24
```

7.54.1.10 PCI_CFG_CAP_PTR

```
#define PCI_CFG_CAP_PTR 0x34
```

7.54.1.11 PCI_CFG_CLASS_REV

```
#define PCI_CFG_CLASS_REV 0x08
```

7.54.1.12 PCI_CFG_COMMAND

```
#define PCI_CFG_COMMAND 0x04
```

7.54.1.13 PCI_CFG_DEVICE_ID

```
#define PCI_CFG_DEVICE_ID 0x02
```

7.54.1.14 PCI_CFG_HEADER_TYPE

```
#define PCI_CFG_HEADER_TYPE 0x0E
```

7.54.1.15 PCI_CFG_INT_LINE

```
#define PCI_CFG_INT_LINE 0x3C
```

7.54.1.16 PCI_CFG_INT_PIN

```
#define PCI_CFG_INT_PIN 0x3D /* 0=none, 1=INTA ..  4=INTD (item 4.3.5c/4.3.5d) */
```

7.54.1.17 PCI_CFG_STATUS

```
#define PCI_CFG_STATUS 0x06
```

7.54.1.18 PCI_CFG_VENDOR_ID

```
#define PCI_CFG_VENDOR_ID 0x00
```

7.54.1.19 PCI_CMD_BUS_MASTER

```
#define PCI_CMD_BUS_MASTER (1u << 2)
```

7.54.1.20 PCI_CMD_INTX_DISABLE

```
#define PCI_CMD_INTX_DISABLE
```

Value:

```
(1u « 10)  /* item 4.3.5c/4.3.5e: pci_enable() never
* clears this -- if firmware left it set,
* INTx never asserts. Check explicitly. */
```

7.54.1.21 PCI_CMD_IO_SPACE

```
#define PCI_CMD_IO_SPACE (1u << 0)
```

7.54.1.22 PCI_CMD_MEM_SPACE

```
#define PCI_CMD_MEM_SPACE (1u << 1)
```

7.54.2 Function Documentation

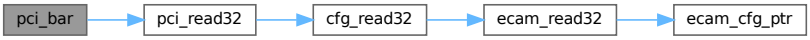
7.54.2.1 pci_bar()

```
uint64_t pci_bar (
            const PciDevice * d,
            int bar_idx )
```

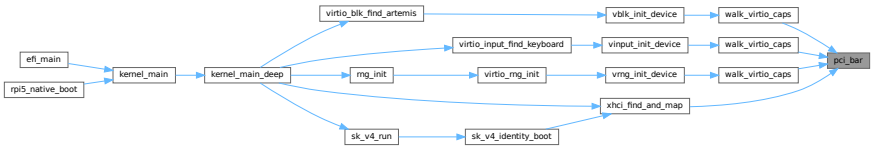
References PCI_BAR_MEM_64, PCI_BAR_TYPE_MASK, PCI_CFG_BAR0, and pci_read32().

Referenced by walk_virtio_caps(), walk_virtio_caps(), walk_virtio_caps(), and xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:



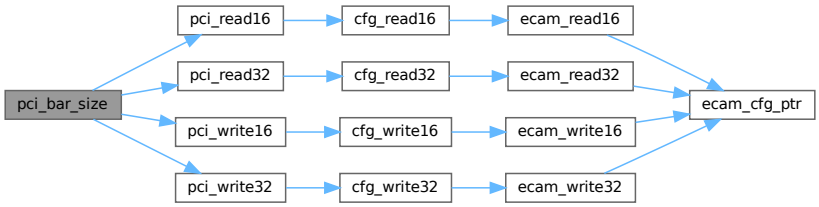
7.54.2.2 pci_bar_size()

```
uint64_t pci_bar_size (
            const PciDevice * d,
            int bar_idx )
```

References PCI_BAR_MEM_64, PCI_BAR_TYPE_MASK, PCI_CFG_BAR0, PCI_CFG_COMMAND, PCI_CMD_MEM_SPACE, pci_read16(), pci_read32(), pci_write16(), and pci_write32().

Referenced by xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:

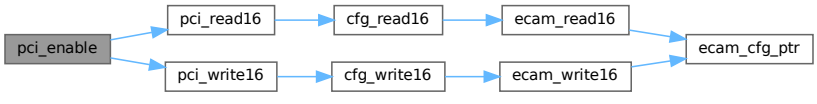
7.54.2.3 pci_enable()

```
void pci_enable (
            const PciDevice * d )
```

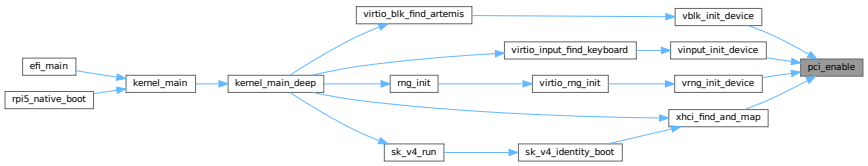
References PCI_CFG_COMMAND, PCI_CMD_BUS_MASTER, PCI_CMD_IO_SPACE, PCI_CMD_MEM_SPACE, pci_read16(), and pci_write16().

Referenced by vblk_init_device(), vinput_init_device(), vrng_init_device(), and xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:



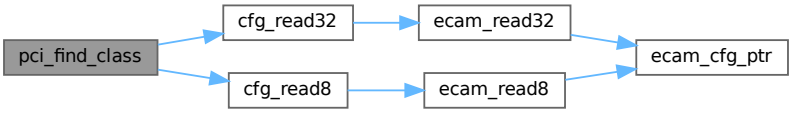
7.54.2.4 pci_find_class()

```
int pci_find_class (
            uint8_t class_code,
            uint8_t subclass,
            uint8_t prog_if,
            unsigned index,
            PciDevice * out )
```

References PciDevice::bus, cfg_read32(), cfg_read8(), PciDevice::device, PciDevice::device_id, PciDevice::function, g_bus_first, g_bus_last, g_ecam_base, PCI_CFG_CLASS_REV, PCI_CFG_HEADER_TYPE, PCI_CFG_VENDOR_ID, and PciDevice::vendor_id.

Referenced by xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:

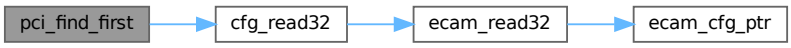
7.54.2.5 pci_find_first()

```
int pci_find_first (
            uint16_t vendor_id,
            uint16_t device_id,
            PciDevice * out )
```

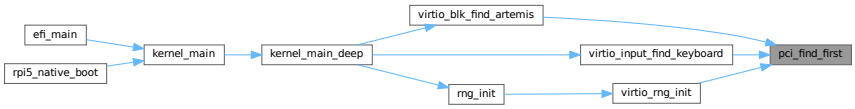
References PciDevice::bus, cfg_read32(), PciDevice::device, PciDevice::device_id, PciDevice::function, PCI_CFG_VENDOR_ID, and PciDevice::vendor_id.

Referenced by virtio_blk_find_artemis(), virtio_input_find_keyboard(), and virtio_rng_init().

Here is the call graph for this function:



Here is the caller graph for this function:



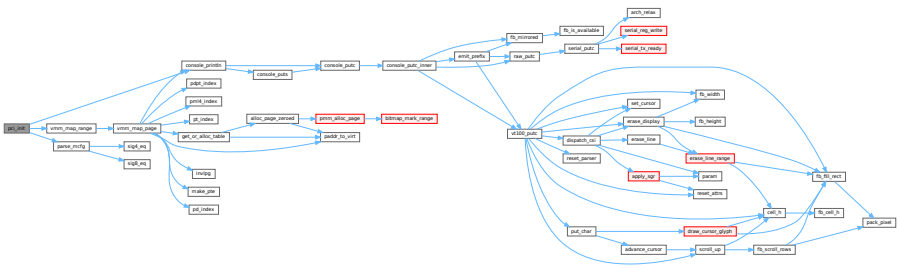
7.54.2.6 pci_init()

```
int pci_init (
            void * rsdp )
```

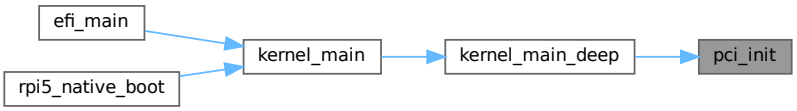
References console_println(), g_ecam_base, parse_mcfg(), VMM_FLAG_CACHE_DISABLE, VMM_FLAG_WRITABLE, and vmm_map_range().

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



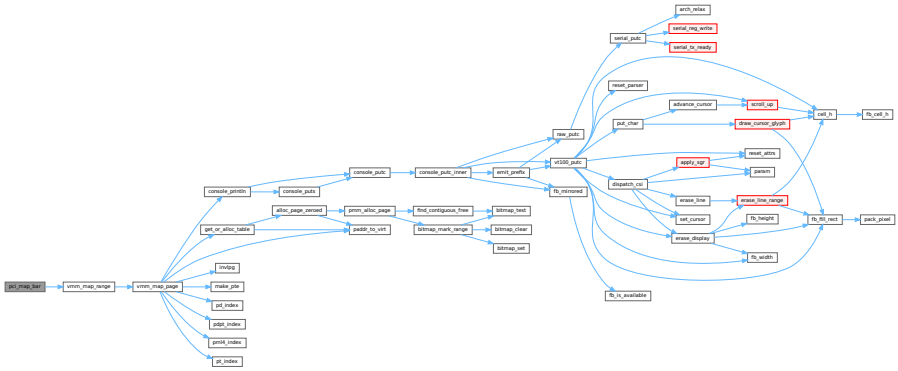
7.54.2.7 pci_map_bar()

```
int pci_map_bar (
            uint64_t phys_addr,
            uint64_t size )
```

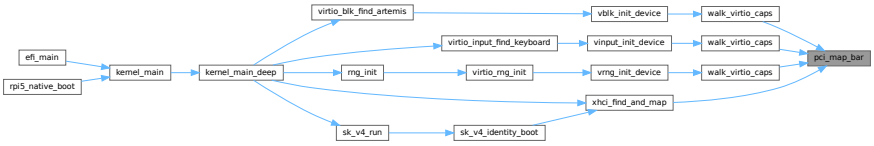
References VMM_FLAG_CACHE_DISABLE, VMM_FLAG_WRITABLE, and vmm_map_range().

Referenced by walk_virtio_caps(), walk_virtio_caps(), walk_virtio_caps(), and xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:



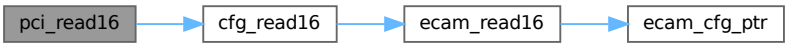
7.54.2.8 pci_read16()

```
uint16_t pci_read16 (
            const PciDevice * d,
            uint16_t offset )
```

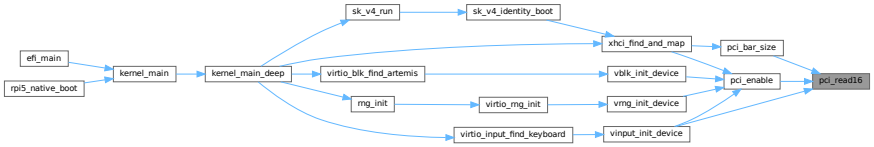
References PciDevice::bus, cfg_read16(), PciDevice::device, and PciDevice::function.

Referenced by pci_bar_size(), pci_enable(), and vinput_init_device().

Here is the call graph for this function:



Here is the caller graph for this function:



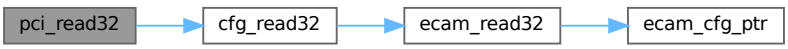
7.54.2.9 pci_read32()

```
uint32_t pci_read32 (
            const PciDevice * d,
            uint16_t offset )
```

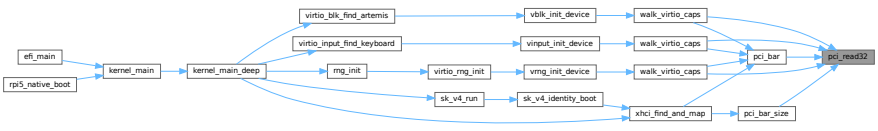
References PciDevice::bus, cfg_read32(), PciDevice::device, and PciDevice::function.

Referenced by pci_bar(), pci_bar_size(), walk_virtio_caps(), walk_virtio_caps(), and walk_virtio_caps().

Here is the call graph for this function:



Here is the caller graph for this function:



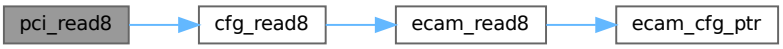
7.54.2.10 pci_read8()

```
uint8_t pci_read8 (
            const PciDevice * d,
            uint16_t offset )
```

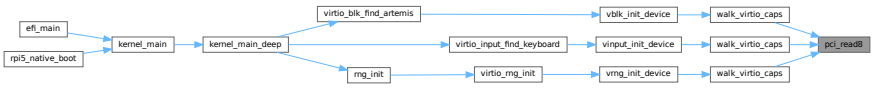
References PciDevice::bus, cfg_read8(), PciDevice::device, and PciDevice::function.

Referenced by walk_virtio_caps(), walk_virtio_caps(), and walk_virtio_caps().

Here is the call graph for this function:



Here is the caller graph for this function:



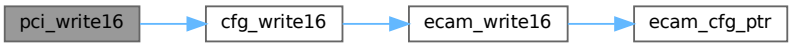
7.54.2.11 pci_write16()

```
void pci_write16 (
            const PciDevice * d,
            uint16_t offset,
            uint16_t val )
```

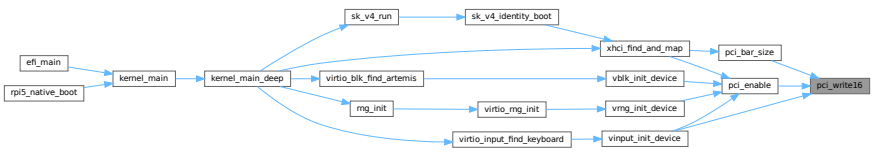
References PciDevice::bus, cfg_write16(), PciDevice::device, and PciDevice::function.

Referenced by pci_bar_size(), pci_enable(), and vinput_init_device().

Here is the call graph for this function:



Here is the caller graph for this function:



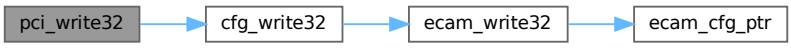
7.54.2.12 pci_write32()

```
void pci_write32 (
            const PciDevice * d,
            uint16_t offset,
            uint32_t val )
```

References PciDevice::bus, cfg_write32(), PciDevice::device, and PciDevice::function.

Referenced by pci_bar_size().

Here is the call graph for this function:



Here is the caller graph for this function:

7.54.2.13 pci_write8()

```
void pci_write8 (
            const PciDevice * d,
            uint16_t offset,
            uint8_t val )
```

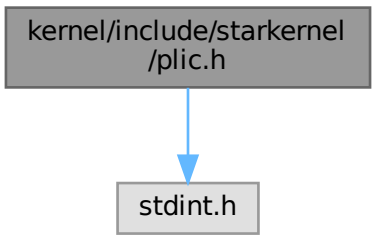
References PciDevice::bus, cfg_write8(), PciDevice::device, and PciDevice::function.

Here is the call graph for this function:

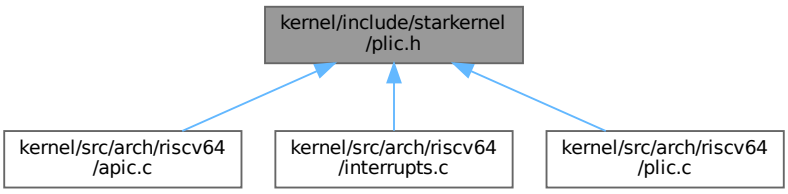
7.55 kernel/include/starkernel/pie_hidden.h File Reference

7.56 kernel/include/starkernel/plic.h File Reference

`#include <stdint.h>`
Include dependency graph for plic.h:



This graph shows which files directly or indirectly include this file:



Functions

- int plic_init (const void *dtb)

  plic.h - Platform-Level Interrupt Controller interface (riscv64 only)
- void plic_set_priority (uint32_t irq, uint32_t priority)

  Set a source's interrupt priority (1-7; 0 means "never interrupt").
- void plic_enable (uint32_t irq)

  Enable a source for the S-mode hart-0 context.
- void plic_disable (uint32_t irq)

  Disable a source for the S-mode hart-0 context.
- uint32_t plic_claim (void)

  Claim the highest-priority pending interrupt for the S-mode hart-0 context.
- void plic_complete (uint32_t irq)

  Signal completion of the source previously returned by plic_claim().

7.56.1 Function Documentation

7.56.1.1 plic_claim()

```
uint32_t plic_claim (
            void  )
```

Claim the highest-priority pending interrupt for the S-mode hart-0 context.

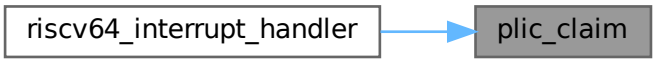
Returns the source ID, or 0 if none is pending.

References PLIC_CLAIM, PLIC_CONTEXT_S, and reg32().

Referenced by riscv64_interrupt_handler().

Here is the call graph for this function:

Here is the caller graph for this function:



7.56.1.2 plic_complete()

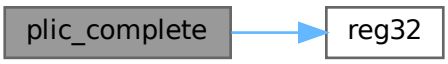
```
void plic_complete (
            uint32_t irq )
```

Signal completion of the source previously returned by plic_claim().

References PLIC_CLAIM, PLIC_CONTEXT_S, and reg32().

Referenced by riscv64_interrupt_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

7.56.1.3 plic_disable()

```
void plic_disable (
            uint32_t irq )
```

Disable a source for the S-mode hart-0 context.

References PLIC_CONTEXT_S, PLIC_ENABLE, and reg32().

Here is the call graph for this function:

7.56.1.4 plic_enable()

```
void plic_enable (
            uint32_t irq )
```

Enable a source for the S-mode hart-0 context.

References PLIC_CONTEXT_S, PLIC_ENABLE, and reg32().

Here is the call graph for this function:

7.56.1.5 plic_init()

```
int plic_init (
            const void * dtb )
```

plic.h - Platform-Level Interrupt Controller interface (riscv64 only)

Item 4.3.5a (FABRIC-0.md ?27.5): Phase 0 (0.2/0.3) only ever enabled the S-mode *timer* interrupt (sie.STIE). External interrupts (sie.SEIE) were never touched, and the PLIC – the only external-interrupt path on RISC-V, there is no legacy PIC or I/O APIC equivalent – had no driver at all before this item. Pure substrate: this item wires the mechanism (claim/dispatch/complete) with no permanent source enabled by default; a real consumer (4.3.5b, virtio-keyboard) enables its own source later.

`plic_init()` takes the boot DTB as of FABRIC-3.md ?V.3 item 3's fix, 2026-09-05: the base address was previously a QEMU-virt-specific hardcoded constant, unconditionally wrong on the Milk-V Mars's real JH7110 PLIC. See `plic.c`'s file header for the discovery mechanism. Map the PLIC and set the S-mode hart-0 context's priority threshold to 0 (maximally permissive – safe because nothing is enabled at any source by default; enabling a source is what actually lets it reach claim()). Also sets sie.SEIE. Does not touch sstatus.SIE – arch_enable_interrupts() still owns that, same as the timer.

Parameters

dtb	Candidate devicetree blob (`BootInfo->dtb`); NULL-safe. When a real `"sifive,plic-1.0.0"` node is found, the PLIC base address is read from its `reg` property; otherwise falls back to the QEMU-virt-machine constant, unchanged from this function's previous unconditional behaviour.

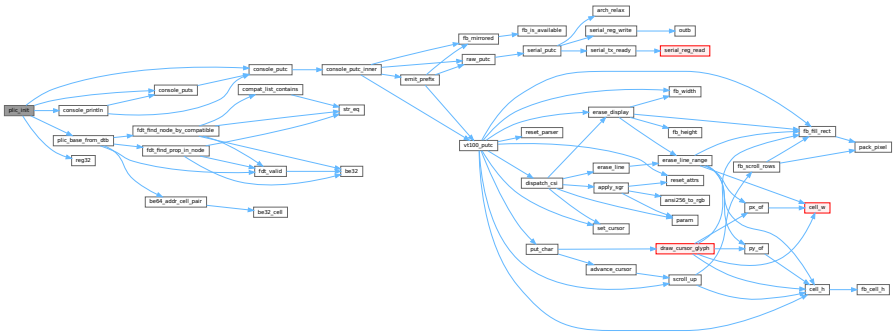
Returns

0 on success.

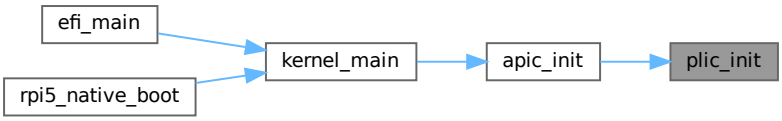
References console_println(), console_putc(), console_puts(), plic_base_from_dtb(), PLIC_CONTEXT_S, PLIC_THRESHOLD, reg32(), and s_plic_base.

Referenced by apic_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.56.1.6 plic_set_priority()

```
void plic_set_priority (
            uint32_t irq,
            uint32_t priority )
```

Set a source's interrupt priority (1-7; 0 means "never interrupt").

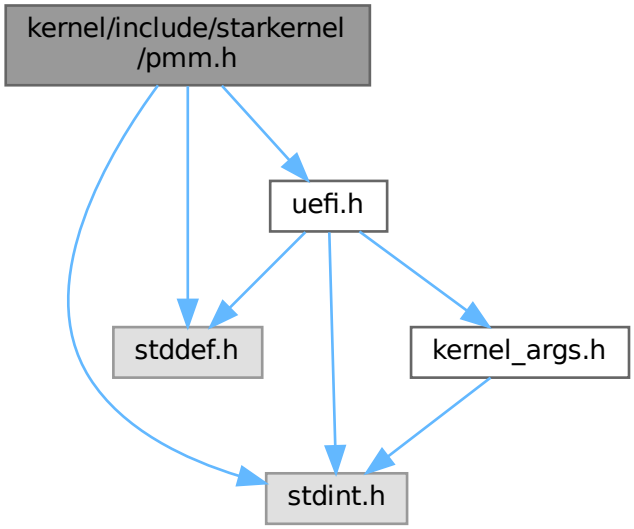
References PLIC_PRIORITY, and reg32().

Here is the call graph for this function:

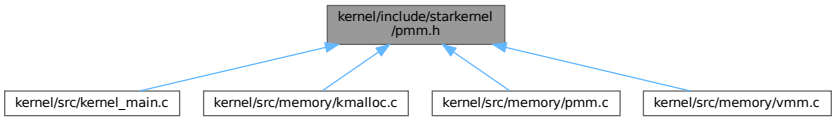
7.57 kernel/include/starkernel/pmm.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "uefi.h"
```

Include dependency graph for pmm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct pmm_stats_t

Macros

- #define PMM_PAGE_SIZE 4096u

  pmm.h - Physical Memory Manager interface for StarKernel

Functions

- int pmm_init (BootInfo *boot_info)
- int pmm_is_initialized (void)
- uint64_t pmm_alloc_page (void)
- uint64_t pmm_alloc_contiguous (uint64_t num_pages)
- void pmm_free_page (uint64_t paddr)
- void pmm_free_contiguous (uint64_t paddr, uint64_t num_pages)
- pmm_stats_t pmm_get_stats (void)

7.57.1 Macro Definition Documentation

7.57.1.1 PMM_PAGE_SIZE

```
#define PMM_PAGE_SIZE 4096u
```

pmm.h - Physical Memory Manager interface for StarKernel

Bitmap-based allocator for 4KB physical pages. Initialization parses the UEFI memory map provided in BootInfo and exposes simple allocation and statistics helpers for early kernel bring-up.

7.57.2 Function Documentation

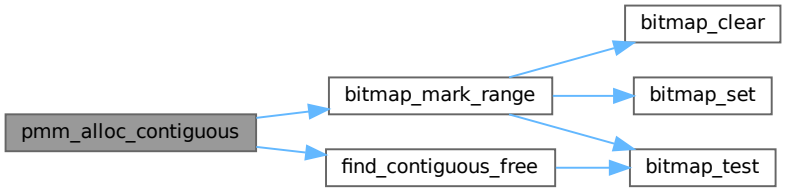
7.57.2.1 pmm_alloc_contiguous()

```
uint64_t pmm_alloc_contiguous (
            uint64_t num_pages )
```

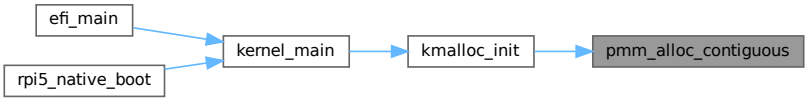
References bitmap_mark_range(), find_contiguous_free(), pmm_initialized, and PMM_PAGE_SIZE.

Referenced by kmalloc_init().

Here is the call graph for this function:



Here is the caller graph for this function:



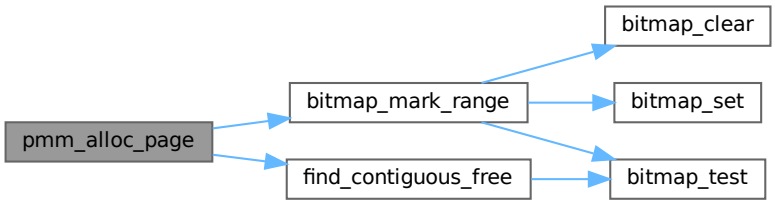
7.57.2.2 pmm_alloc_page()

```
uint64_t pmm_alloc_page (
            void  )
```

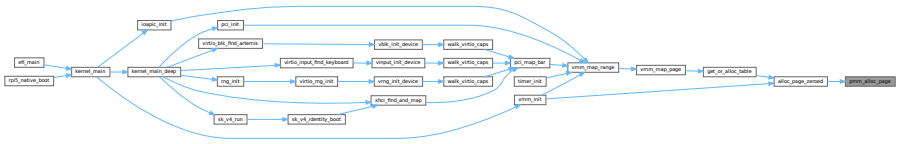
References bitmap_mark_range(), find_contiguous_free(), pmm_initialized, and PMM_PAGE_SIZE.

Referenced by alloc_page_zeroed().

Here is the call graph for this function:



Here is the caller graph for this function:



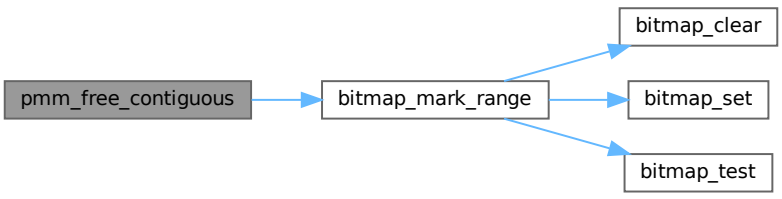
7.57.2.3 pmm_free_contiguous()

```
void pmm_free_contiguous (
            uint64_t paddr,
            uint64_t num_pages )
```

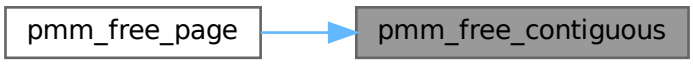
References bitmap_mark_range(), pmm_initialized, PMM_PAGE_SIZE, and pmm_tracked_pages.

Referenced by pmm_free_page().

Here is the call graph for this function:



Here is the caller graph for this function:

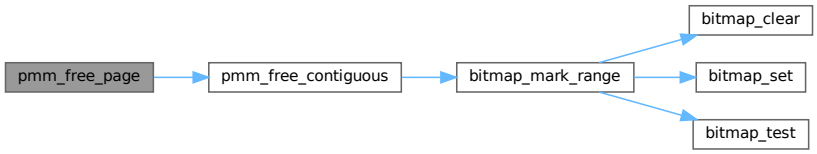


7.57.2.4 pmm_free_page()

```
void pmm_free_page (
            uint64_t paddr )
```

References pmm_free_contiguous(), pmm_initialized, and PMM_PAGE_SIZE.

Here is the call graph for this function:



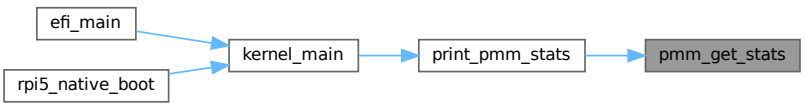
7.57.2.5 pmm_get_stats()

```
pmm_stats_t pmm_get_stats (
            void  )
```

References pmm_stats_t::free_bytes, pmm_stats_t::free_pages, pmm_free_pages, PMM_PAGE_SIZE, pmm_total_pages, pmm_stats_t::total_bytes, pmm_stats_t::total_pages, pmm_stats_t::used_bytes, and pmm_stats_t::used_pages.

Referenced by print_pmm_stats().

Here is the caller graph for this function:



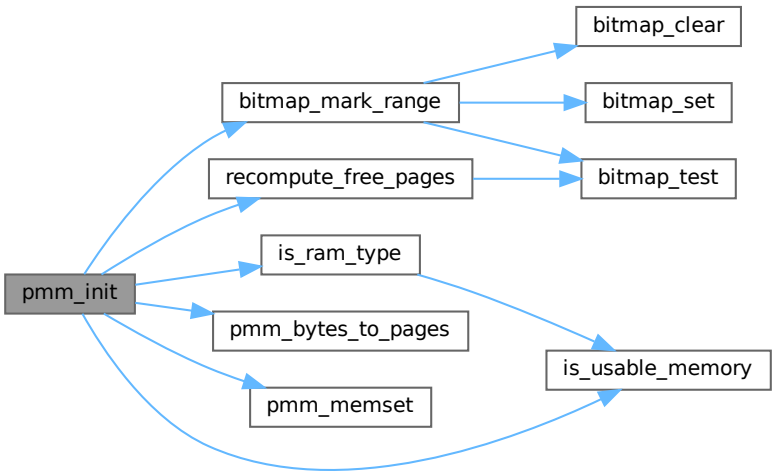
7.57.2.6 pmm_init()

```
int pmm_init (
            BootInfo * boot_info )
```

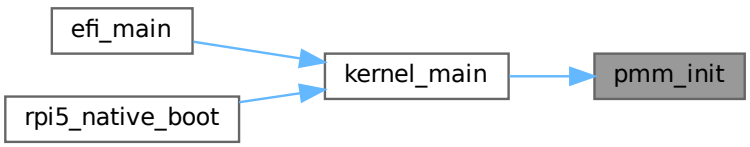
References __kernel_end, __kernel_start, bitmap_mark_range(), is_ram_type(), is_usable_memory(), BootInfo::memory_map, BootInfo::memory_map_descriptor_size, BootInfo::memory_map_size, EFI_MEMORY_DESCRIPTOR::NumberOfPages, EFI_MEMORY_DESCRIPTOR::PhysicalStart, pmm_alloc_floor_page, pmm_bitmap, PMM_BITMAP_BITS, pmm_bytes_to_pages(), pmm_free_pages, pmm_initialized, pmm_memset(), PMM_PAGE_SIZE, pmm_total_pages, pmm_tracked_pages, recompute_free_pages(), and EFI_MEMORY_DESCRIPTOR::Type.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



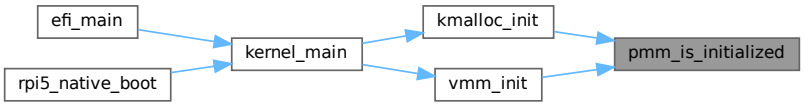
7.57.2.7 pmm_is_initialized()

```
int pmm_is_initialized (
            void  )
```

References pmm_initialized.

Referenced by kmalloc_init(), and vmm_init().

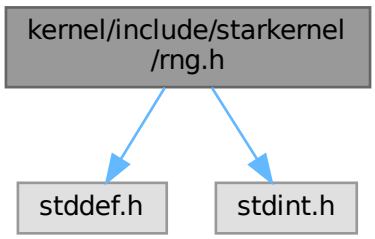
Here is the caller graph for this function:



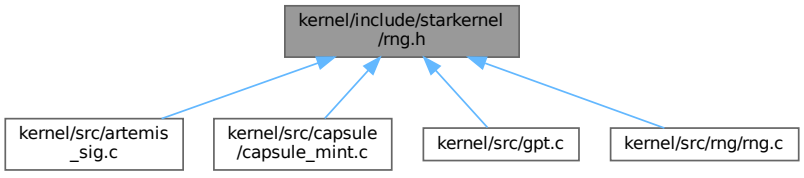
7.58 kernel/include/starkernel/rng.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for rng.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define RNG_ERR_NO_BACKEND (-1) /∗ rng_init() found no working entropy source ∗/

Functions

- int rng_init (void)
- int rng_ready (void)
- int rng_get_bytes (uint8_t ∗buf, size_t n)

7.58.1 Macro Definition Documentation

7.58.1.1 RNG_ERR_NO_BACKEND

```
#define RNG_ERR_NO_BACKEND (-1) /* rng_init() found no working entropy source */
```

7.58.2 Function Documentation

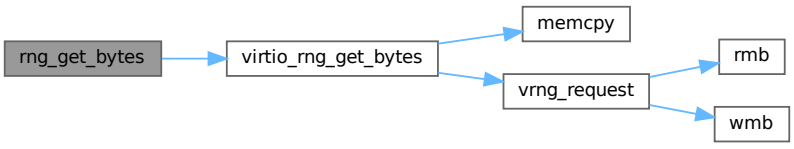
7.58.2.1 rng_get_bytes()

```
int rng_get_bytes (
            uint8_t * buf,
            size_t n )
```

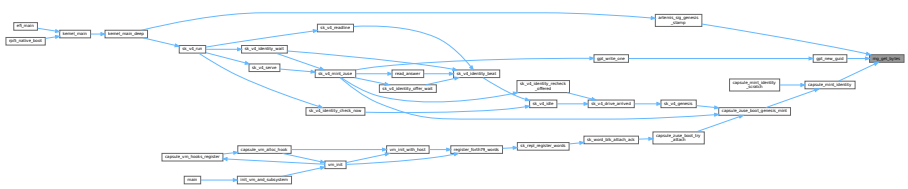
References g_rng_backend, RNG_BACKEND_VIRTIO, RNG_ERR_NO_BACKEND, and virtio_rng_get_bytes().

Referenced by artemis_sig_genesis_stamp(), capsule_mint_identity(), and gpt_new_guid().

Here is the call graph for this function:



Here is the caller graph for this function:



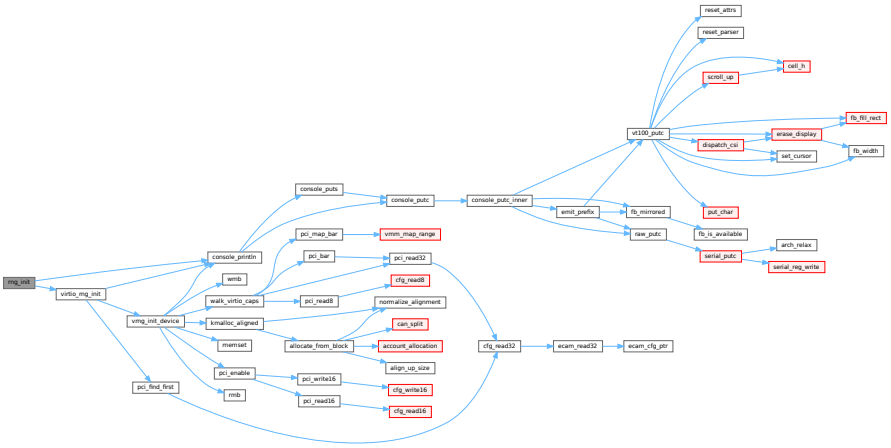
7.58.2.2 rng_init()

```
int rng_init (
              void  )
```

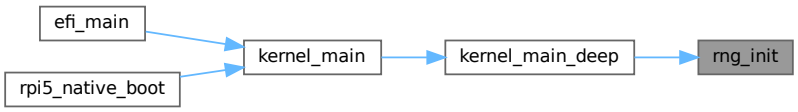
References console_println(), g_rng_backend, RNG_BACKEND_NONE, RNG_BACKEND_VIRTIO, and virtio_rng_init().

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.58.2.3 rng_ready()

```
int rng_ready (
              void  )
```

References g_rng_backend, and RNG_BACKEND_NONE.

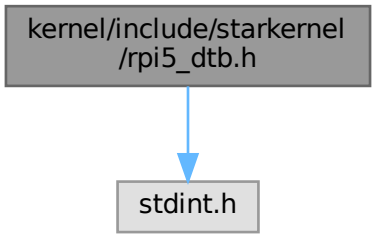
Referenced by capsule_mint_identity().

Here is the caller graph for this function:

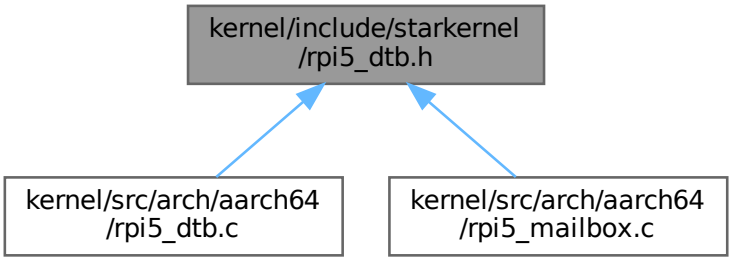
7.59 kernel/include/starkernel/rpi5_dtb.h File Reference

```
#include <stdint.h>
```

Include dependency graph for rpi5_dtb.h:



This graph shows which files directly or indirectly include this file:



Functions

- uint64_t rpi5_uart_base (const void *dtb)

  rpi5_dtb.h - Raspberry Pi 5 (BCM2712) devicetree-based peripheral discovery, for the native (non-UEFI) boot path (FABRIC-3.md ?IV.3).
- uint64_t rpi5_mailbox_base (const void *dtb)

  Find the VideoCore mailbox property interface's CPU-physical base address from the DTB.

7.59.1 Function Documentation

7.59.1.1 rpi5_mailbox_base()

```
uint64_t rpi5_mailbox_base (
            const void * dtb )
```

Find the VideoCore mailbox property interface's CPU-physical base address from the DTB.

Parameters

dtb	Devicetree blob; NULL is safe.

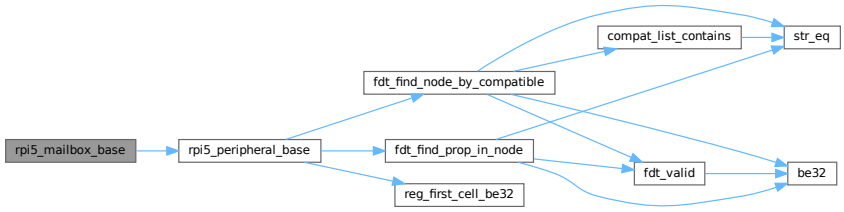
Returns

Final CPU-physical MMIO base address, or 0 if the node is absent, malformed, or `dtb` is invalid.

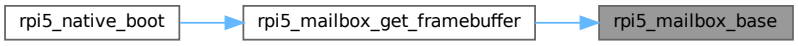
References rpi5_peripheral_base().

Referenced by rpi5_mailbox_get_framebuffer().

Here is the call graph for this function:



Here is the caller graph for this function:



7.59.1.2 rpi5_uart_base()

```
uint64_t rpi5_uart_base (
            const void * dtb )
```

rpi5_dtb.h - Raspberry Pi 5 (BCM2712) devicetree-based peripheral discovery, for the native (non-UEFI) boot path (FABRIC-3.md ?IV.3).

Two lookups: the PL011 UART (early console) and the VideoCore mailbox property interface (framebuffer setup, ?IV.3 item 3). Both peripherals live under BCM2712's own devicetree "soc" simple-bus node, which applies one fixed address translation to every child `reg` value — see `rpi5_dtb.c`'s own doc comment for the confirmed offset and where it was verified. `fdt.c`'s reader deliberately does not do general `ranges`-property translation (it is "not a general devicetree library"); this file applies the one, fixed, SoC-wide offset by name instead of teaching `fdt.c` a general mechanism for a single known hardware fact.

Find the PL011 UART's CPU-physical base address from the DTB.

Parameters

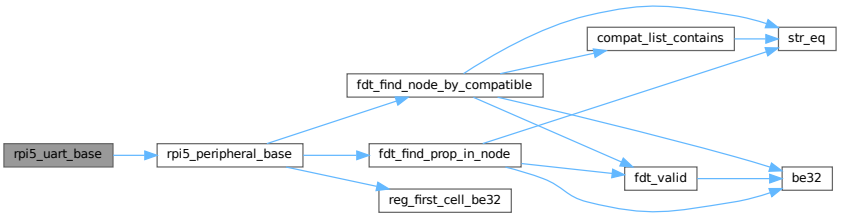
dtb	Devicetree blob, as passed to the native boot entry (or `BootInfo->dtb`); NULL is safe.

Returns

Final CPU-physical MMIO base address, or 0 if the node is absent, malformed, or `dtb` is invalid.

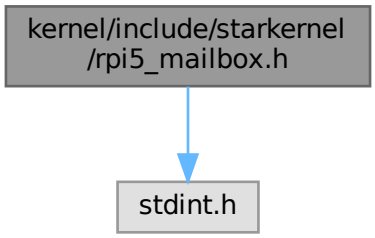
References rpi5_peripheral_base().

Here is the call graph for this function:

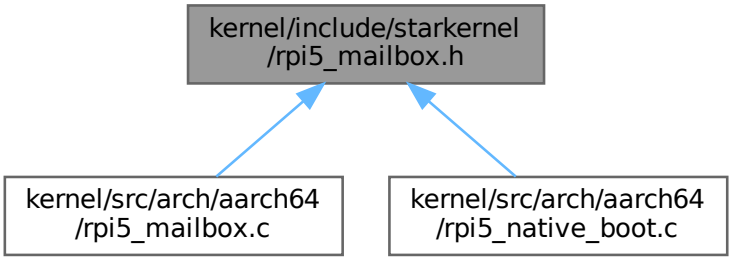


7.60 kernel/include/starkernel/rpi5_mailbox.h File Reference

`#include <stdint.h>`
Include dependency graph for rpi5_mailbox.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Rpi5FramebufferInfo

  rpi5_mailbox.h - VideoCore mailbox property-interface framebuffer setup, for the native (non-UEFI) Raspberry Pi 5 boot path (FABRIC-3.md ?IV.3 item 3).

Functions

- int rpi5_mailbox_get_framebuffer (const void *dtb, uint32_t width, uint32_t height, uint32_t bpp, Rpi5FramebufferInfo *out)

  Request a framebuffer from the VideoCore firmware via the mailbox property interface, at the requested resolution/depth.

7.60.1 Function Documentation

7.60.1.1 rpi5_mailbox_get_framebuffer()

```
int rpi5_mailbox_get_framebuffer (
            const void * dtb,
            uint32_t width,
            uint32_t height,
            uint32_t bpp,
            Rpi5FramebufferInfo * out )
```

Request a framebuffer from the VideoCore firmware via the mailbox property interface, at the requested resolution/depth.

Sends one buffer with all six setup tags (physical size, virtual size, depth, pixel order, virtual offset, allocate) plus a get-pitch tag, in one request/response round trip.

Parameters

dtb	Devicetree blob (passed to `rpi5_mailbox_base()`).
width	Requested physical+virtual width, pixels.
height	Requested physical+virtual height, pixels.
bpp	Requested bits per pixel (32 is the only depth this driver has been designed against; others are not refused outright but are unverified).
out	Populated on success; untouched on failure.

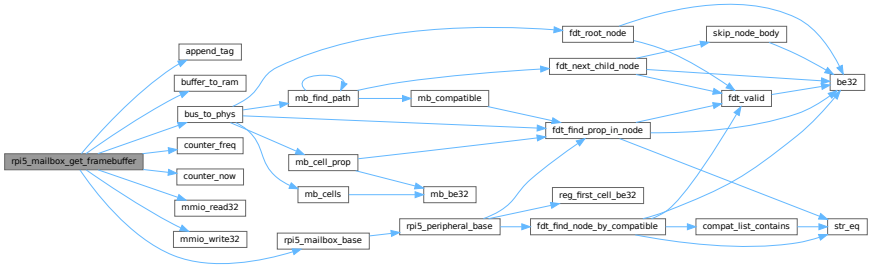
Returns

0 on success, negative on failure: -2 no mailbox node in the DTB, -3 the VC firmware rejected the request, -4 a response came back empty, -5 no answer within one second (every wait is bounded), -6 the devicetree's dma-ranges cannot place the framebuffer's bus address.

References append_tag(), Rpi5FramebufferInfo::base, base, buffer_to_ram(), bus_to_phys(), counter_freq(), counter_now(), Rpi5FramebufferInfo::height, limit, MBOX_CH_PROPERTY, MBOX_CODE_REQUEST, MBOX_CODE_RESPONSE_, MBOX_EMPTY, MBOX_FULL, MBOX_OFF_READ, MBOX_OFF_STATUS0, MBOX_OFF_STATUS1, MBOX_OFF_WRITE, mmio_read32(), mmio_write32(), Rpi5FramebufferInfo::pixel_format, PIXEL_ORDER_RGB, Rpi5FramebufferInfo::pixels_per_scanline, rpi5_mailbox_base(), Rpi5FramebufferInfo::size, TAG_ALLOCATE_BUFFER, TAG_END, TAG_GET_PITCH, TAG_SET_DEPTH, TAG_SET_PHYS_SIZE, TAG_SET_PIXEL_ORDER, TAG_SET_VIRT_SIZE, TAG_SET_VIRTUAL_OFFS, v2, and Rpi5FramebufferInfo::width.

Referenced by rpi5_native_boot().

Here is the call graph for this function:

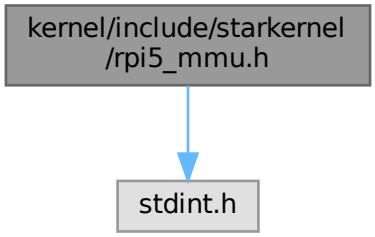


Here is the caller graph for this function:

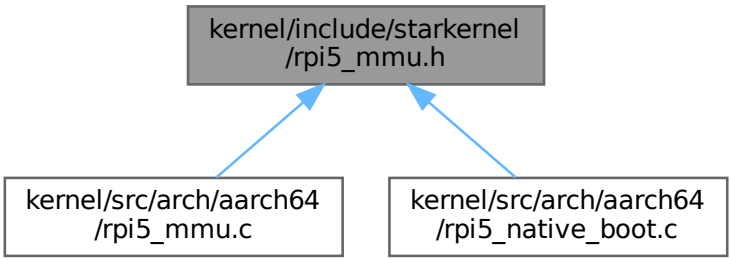
7.61 kernel/include/starkernel/rpi5_mmu.h File Reference

```
#include <stdint.h>
```

Include dependency graph for rpi5_mmu.h:



This graph shows which files directly or indirectly include this file:



Functions

- int rpi5_mmu_map_ram (const void *dtb)

  rpi5_mmu.h - the Pi 5 native path's identity map, once C runs (rpi5_mmu.c; native_rpi5_entry.S turns the MMU on).
- int rpi5_mmu_uncached (uint64_t addr, uint64_t size)

7.61.1 Function Documentation

7.61.1.1 rpi5_mmu_map_ram()

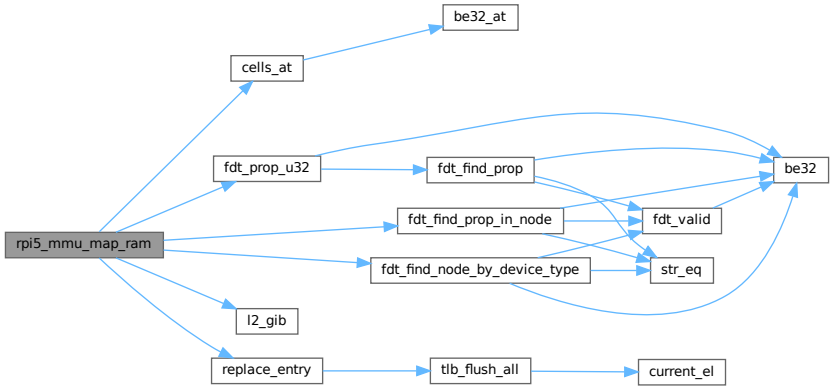
```
int rpi5_mmu_map_ram (
            const void * dtb )
```

rpi5_mmu.h - the Pi 5 native path's identity map, once C runs (rpi5_mmu.c; native_rpi5_entry.S turns the MMU on).

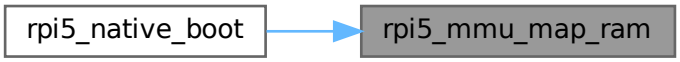
References BLOCK_NORMAL_WB, cells_at(), entry, fdt_find_node_by_device_type(), fdt_find_prop_in_node(), fdt_prop_u32(), g_rpi5_l1, GIB, l2_gib(), and replace_entry().

Referenced by rpi5_native_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



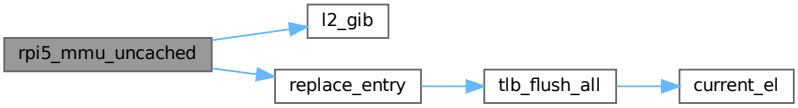
7.61.1.2 rpi5_mmu_uncached()

```
int rpi5_mmu_uncached (
            uint64_t addr,
            uint64_t size )
```

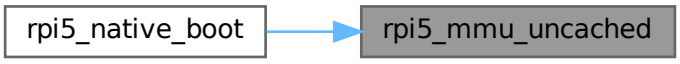
References __kernel_end, __kernel_start, BLOCK_NORMAL_NC, g_rpi5_l1, g_rpi5_l2, l2_gib(), MIB2, and replace_entry().

Referenced by rpi5_native_boot().

Here is the call graph for this function:

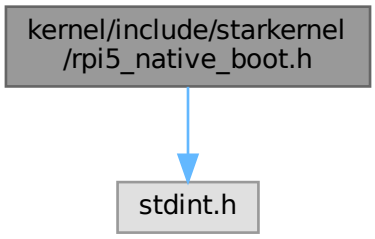


Here is the caller graph for this function:

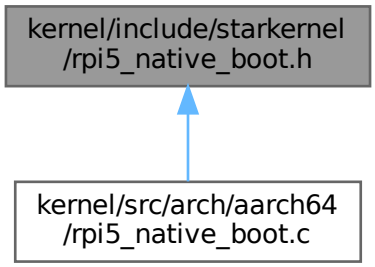


7.62 kernel/include/starkernel/rpi5_native_boot.h File Reference

`#include <stdint.h>`
Include dependency graph for rpi5_native_boot.h:



This graph shows which files directly or indirectly include this file:



Functions

- void rpi5_native_boot (uint64_t dtb) __attribute__((noreturn))
  rpi5_native_boot.h - Raspberry Pi 5 DTB->BootInfo constructor (FABRIC-3.md ?IV.3 item 2).

7.62.1 Function Documentation

7.62.1.1 rpi5_native_boot()

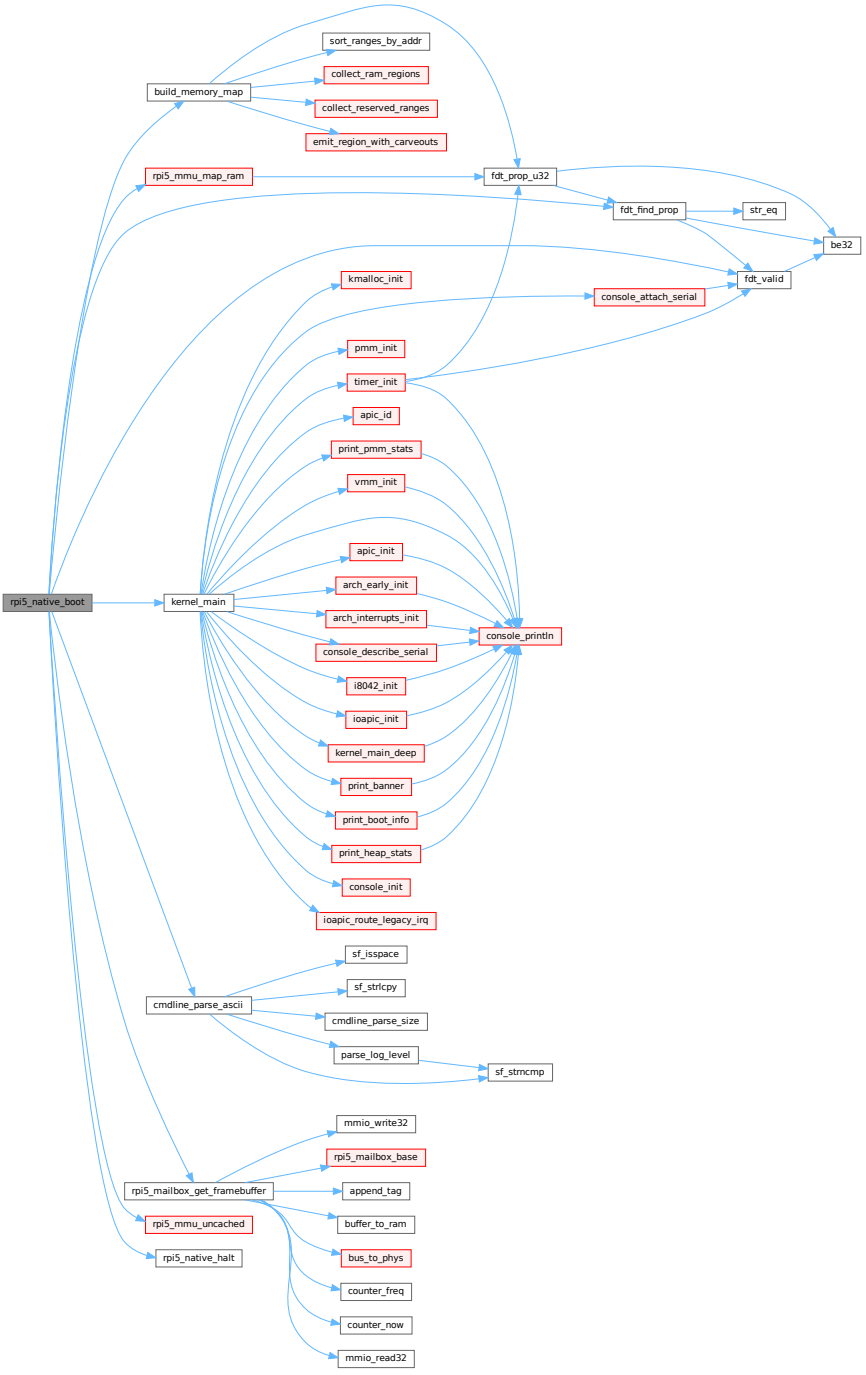
```
void rpi5_native_boot (
            uint64_t dtb )
```

rpi5_native_boot.h - Raspberry Pi 5 DTB->BootInfo constructor (FABRIC-3.md ?IV.3 item 2).

`native_rpi5_entry.S`'s `rpi5_native_start` tail-calls `rpi5_native_boot()` with the masked DTB pointer it captured. This function builds the *existing*, unmodified `BootInfo` struct (`include/starkernel/uefi.h`) from the devicetree instead of UEFI protocols, then calls the *existing*, unmodified `kernel_main()` — see this file's own `.c` for exactly what is and is not populated, and why.

References BootInfo::acpi_table, BootInfo::args, Rpi5FramebufferInfo::base, FramebufferInfo::base, build_memory_map(), cmdline_parse_ascii(), BootInfo::dtb, fdt_find_prop(), fdt_valid(), BootInfo::framebuffer, g_rpi5_boot_info, g_rpi5_memmap, Rpi5FramebufferInfo::height, FramebufferInfo::height, kernel_main(), BootInfo::kernel_stack_base, BootInfo::kernel_stack_size, BootInfo::memory_map, BootInfo::memory_map_descriptor_size, BootInfo::memory_map_size, Rpi5FramebufferInfo::pixel_format, FramebufferInfo::pixel_format, Rpi5FramebufferInfo::pixels_per_scanline, FramebufferInfo::pixels_per_scanline, RPI5_FB_DEFAULT_BPP, RPI5_FB_DEFAULT_HEIGHT, RPI5_FB_DEFAULT_WIDTH, rpi5_mailbox_get_framebuffer(), rpi5_mmu_map_ram(), rpi5_mmu_uncached(), rpi5_native_halt(), BootInfo::runtime_services, Rpi5FramebufferInfo::size, FramebufferInfo::size, BootInfo::uefi_boot_services_exited, Rpi5FramebufferInfo::width, and FramebufferInfo::width.

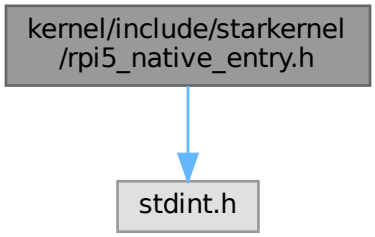
Here is the call graph for this function:



7.63 kernel/include/starkernel/rpi5_native_entry.h File Reference

```
#include <stdint.h>
```

Include dependency graph for rpi5_native_entry.h:



Variables

- uint64_t g_rpi5_dtb_ptr

  rpi5_native_entry.h - Raspberry Pi 5 native (non-UEFI) boot entry point (FABRIC-3.md ?IV.3 item 1).

7.63.1 Variable Documentation

7.63.1.1 g_rpi5_dtb_ptr

```
uint64_t g_rpi5_dtb_ptr  [extern]
```

rpi5_native_entry.h - Raspberry Pi 5 native (non-UEFI) boot entry point (FABRIC-3.md ?IV.3 item 1).

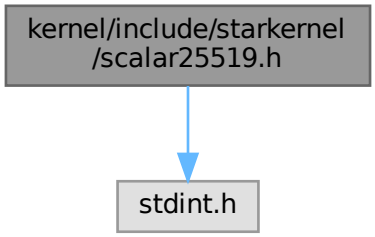
`native_rpi5_entry.S`'s `rpi5_native_start` is the very first code that runs on this path – entered directly by Pi 5 firmware, no UEFI, no ACPI, none of `boot/uefi_loader.c`'s PE-loader shape applies. It masks the firmware's raw entry register down to the documented 32-bit DTB pointer (?IV.1: upper 32 bits of the 64-bit register are unspecified) and stores it here, then establishes its own dedicated stack (this path has no EDK2 boot stack to inherit) and halts.

`g_rpi5_dtb_ptr` is this stub's one real output – the still-open DTB->BootInfo constructor (?IV.3 item 2) reads it from here once it exists; nothing calls that constructor yet, so `rpi5_native_start` halts rather than tail-calling into a function that isn't real.

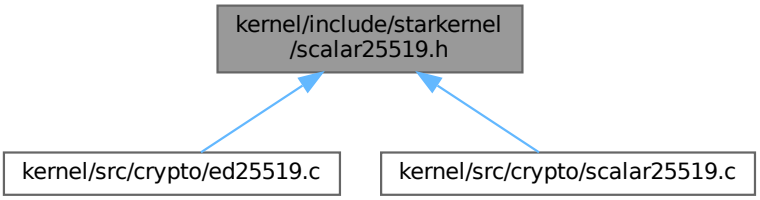
7.64 kernel/include/starkernel/scalar25519.h File Reference

```
#include <stdint.h>
```

Include dependency graph for scalar25519.h:



This graph shows which files directly or indirectly include this file:



Functions

- void scalar_reduce512 (uint8_t out[32], const uint8_t in[64])
- int scalar_lt_L (const uint8_t s[32])
- void scalar_muladd (uint8_t out[32], const uint8_t a[32], const uint8_t b[32], const uint8_t c[32])

7.64.1 Function Documentation

7.64.1.1 scalar_lt_L()

```
int scalar_lt_L (
            const uint8_t s[32] )
```

References cmp32(), and L_BYTES.

Referenced by ed25519_verify().

Here is the call graph for this function:

Here is the caller graph for this function:

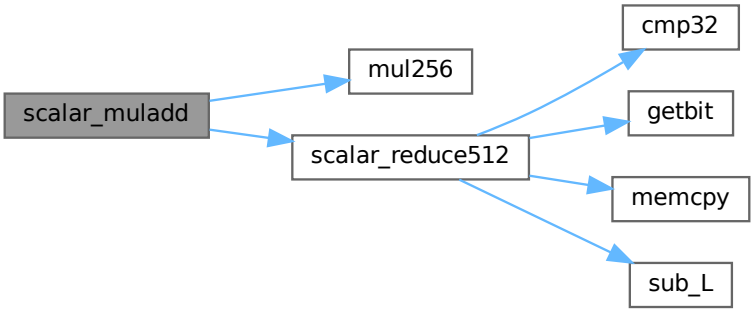
7.64.1.2 scalar_muladd()

```
void scalar_muladd (
            uint8_t out[32],
            const uint8_t a[32],
            const uint8_t b[32],
            const uint8_t c[32] )
```

References mul256(), and scalar_reduce512().

Referenced by ed25519_sign().

Here is the call graph for this function:



Here is the caller graph for this function:

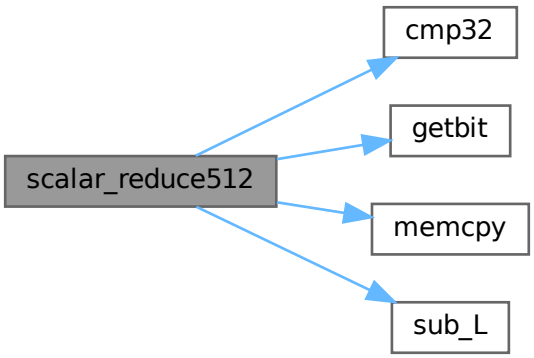
7.64.1.3 scalar_reduce512()

```
void scalar_reduce512 (
            uint8_t out[32],
            const uint8_t in[64] )
```

References cmp32(), getbit(), L_BYTES, memcpy(), and sub_L().

Referenced by ed25519_sign(), ed25519_verify(), and scalar_muladd().

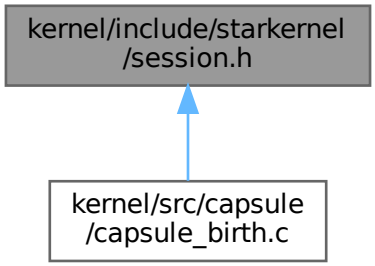
Here is the call graph for this function:



Here is the caller graph for this function:

7.65 kernel/include/starkernel/session.h File Reference

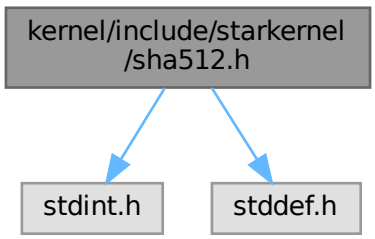
This graph shows which files directly or indirectly include this file:



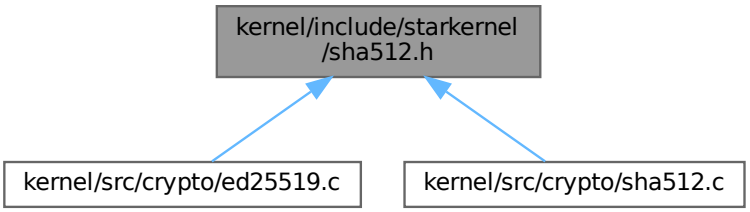
7.66 kernel/include/starkernel/sha512.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for sha512.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct sha512_ctx_t

Functions

- void sha512_init (sha512_ctx_t *ctx)
- void sha512_update (sha512_ctx_t *ctx, const uint8_t *data, size_t len)
- void sha512_final (sha512_ctx_t *ctx, uint8_t out[64])
- void sha512 (const uint8_t *data, size_t len, uint8_t out[64])

7.66.1 Function Documentation

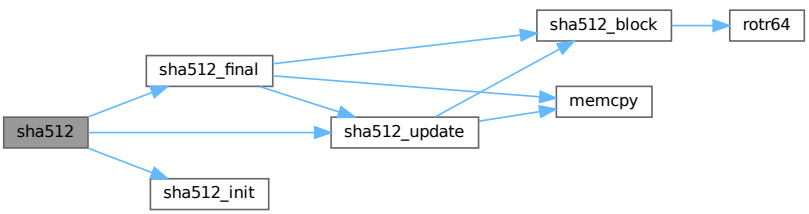
7.66.1.1 sha512()

```
void sha512 (
            const uint8_t * data,
            size_t len,
            uint8_t out[64] )
```

References sha512_final(), sha512_init(), and sha512_update().

Referenced by expand_seed().

Here is the call graph for this function:



Here is the caller graph for this function:

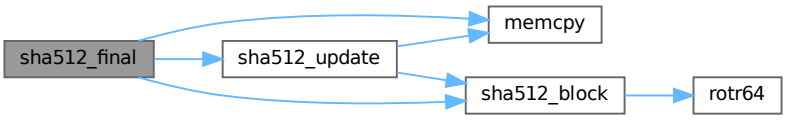
7.66.1.2 sha512_final()

```
void sha512_final (
            sha512_ctx_t * ctx,
            uint8_t out[64] )
```

References sha512_ctx_t::bitlen, sha512_ctx_t::buf, sha512_ctx_t::buf_len, memcpy(), sha512_block(), sha512_update(), sha512_ctx_t::state, and zero.

Referenced by ed25519_sign(), ed25519_verify(), and sha512().

Here is the call graph for this function:



Here is the caller graph for this function:

7.66.1.3 sha512_init()

```
void sha512_init (
            sha512_ctx_t * ctx )
```

References sha512_ctx_t::bitlen, sha512_ctx_t::buf_len, and sha512_ctx_t::state.

Referenced by ed25519_sign(), ed25519_verify(), and sha512().

Here is the caller graph for this function:

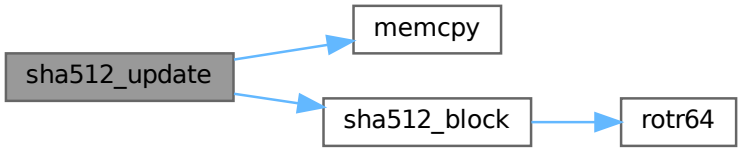
7.66.1.4 sha512_update()

```
void sha512_update (
            sha512_ctx_t * ctx,
            const uint8_t * data,
            size_t len )
```

References sha512_ctx_t::bitlen, sha512_ctx_t::buf, sha512_ctx_t::buf_len, memcpy(), and sha512_block().

Referenced by ed25519_sign(), ed25519_verify(), sha512(), and sha512_final().

Here is the call graph for this function:

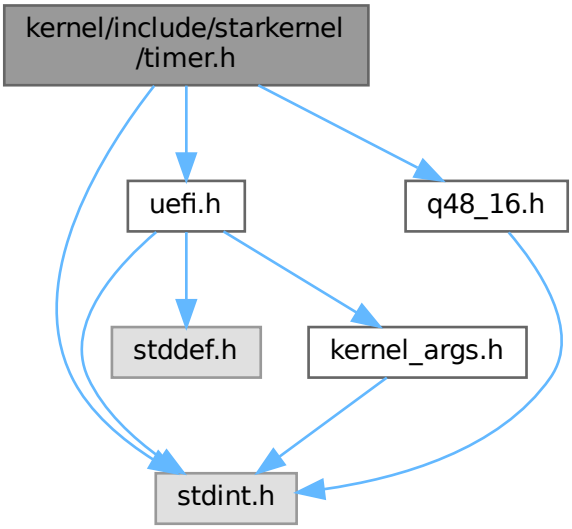


Here is the caller graph for this function:

7.67 kernel/include/starkernel/timer.h File Reference

```
#include <stdint.h>
#include "uefi.h"
#include "q48_16.h"
```

Include dependency graph for timer.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct time_window
- struct time_trust_state
- struct timer_calibration_record

Macros

- #define TIME_WINDOW_SIZE 64
- #define TIME_TRUST_HIGH Q48_ONE /∗ 1.0 = full confidence ∗/
- #define TIME_TRUST_LOW (Q48_ONE >> 2) /∗ 0.25 = low confidence ∗/

Typedefs

- typedef q48_16_t time_trust_t

  timer.h - Timer and Heartbeat Interface
- typedef struct time_window TimeWindow
- typedef struct time_trust_state TimeTrustState
- typedef enum timer_trust_level timer_trust_level_t
- typedef struct timer_calibration_record timer_calibration_record_t

Enumerations

- enum timer_trust_level { TIMER_TRUST_NONE = 0 , TIMER_TRUST_RELATIVE = 1 , TIMER_TRUST_ABSOLUTE = 2 }

Functions

- int timer_init (BootInfo *boot_info)

  Initialise the kernel timer subsystem (M5).
- uint64_t timer_tsc_hz (void)

  Return the system counter frequency in Hz.
- uint64_t timer_now_ns (void)

  Return a monotonic nanosecond timestamp.
- int timer_check_drift_now (void)

  Check for timer drift (AArch64 stub — always returns 0).
- const timer_calibration_record_t * timer_calibration_record (void)

  Return a pointer to the timer calibration record.
- void heartbeat_init (uint64_t tsc_hz, uint64_t tick_hz)

  Initialise the M5 heartbeat / TIME-TRUST subsystem.
- void heartbeat_tick (void)

  Top half.
- void heartbeat_service (void)

  Bottom half (punch-list item 0.8).
- uint64_t heartbeat_read_counter (void)

  Read the raw hardware counter this architecture's heartbeat is paced against – the same clock `timer_now_ns()` and calibration already use internally (rdtsc on amd64, the `time` CSR on riscv64, CNTPCT_EL0 on aarch64), not a separate/different source.
- uint64_t heartbeat_ticks (void)

  Get current TIME-TICKS (monotonic heartbeat count).
- time_trust_t heartbeat_trust (void)

  Get current TIME-TRUST (Q48.16 confidence metric).
- const TimeTrustState * heartbeat_state (void)

  Get pointer to full heartbeat state (for diagnostics).
- void heartbeat_set_adaptive_period_ns (uint64_t ns)

  Set the adaptive re-arm period, in nanoseconds (punch-list item 0.8, FABRIC-0.md ?26).
- uint64_t heartbeat_next_period_ns (void)

  Read the period the next hardware re-arm should use.

7.67.1 Macro Definition Documentation

7.67.1.1 TIME_TRUST_HIGH

```
#define TIME_TRUST_HIGH Q48_ONE /* 1.0 = full confidence */
```

7.67.1.2 TIME_TRUST_LOW

```
#define TIME_TRUST_LOW (Q48_ONE >> 2) /* 0.25 = low confidence */
```

7.67.1.3 TIME_WINDOW_SIZE

```
#define TIME_WINDOW_SIZE 64
```

7.67.2 Typedef Documentation

7.67.2.1 time_trust_t

```
typedef q48_16_t time_trust_t
```

timer.h - Timer and Heartbeat Interface

M5 Time Model:

- TIME-TICKS (Q64.0): Monotonic heartbeat counter, never decreases
- TIME-TRUST (Q48.16): Continuous confidence metric [0.0, 1.0]
- No discrete modes (NONE/REL/ABS are legacy, being phased out)
- Trust is a measurement, never gates execution

7.67.2.2 timer_calibration_record_t

```
typedef struct timer_calibration_record timer_calibration_record_t
```

7.67.2.3 timer_trust_level_t

```
typedef enum timer_trust_level timer_trust_level_t
```

7.67.2.4 TimeTrustState

```
typedef struct time_trust_state TimeTrustState
```

7.67.2.5 TimeWindow

```
typedef struct time_window TimeWindow
```

7.67.3 Enumeration Type Documentation

7.67.3.1 timer_trust_level

```
enum timer_trust_level
```

Enumerator

TIMER_TRUST_NONE	
TIMER_TRUST_RELATIVE	
TIMER_TRUST_ABSOLUTE	

7.67.4 Function Documentation

7.67.4.1 heartbeat_init()

```
void heartbeat_init (
            uint64_t tsc_hz,
            uint64_t tick_hz )
```

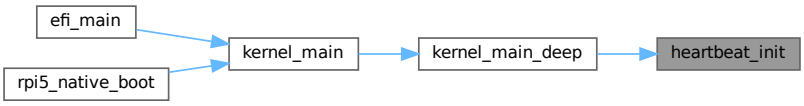
Initialise the M5 heartbeat / TIME-TRUST subsystem.

Shared across all three architectures – the fallback and calibration formula never differed between them, only what heartbeat_service() does with the resulting window did.

References time_window::count, time_window::deltas, time_trust_state::expected_delta, g_adaptive_period_ns, g_heartbeat, g_pending_valid, HEARTBEAT_BASE_PERIOD_NS, time_trust_state::last_tsc, time_window::pos, Q48_ONE, time_trust_state::ticks, TIME_WINDOW_SIZE, time_trust_state::total_samples, time_trust_state::trust, time_trust_state::variance, and time_trust_state::window.

Referenced by kernel_main_deep().

Here is the caller graph for this function:



7.67.4.2 heartbeat_next_period_ns()

```
uint64_t heartbeat_next_period_ns (
            void  )
```

Read the period the next hardware re-arm should use.

Called from interrupt context by each architecture's re-arm function in place of a fixed constant.

Called from interrupt context by each architecture's re-arm function (apic_timer_rearm(), riscv64_timer_rearm(), the AArch64 equivalent) in place of the fixed constant those functions used before item 0.8.

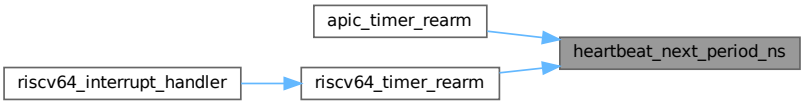
Returns

Current adaptive period in nanoseconds.

References g_adaptive_period_ns.

Referenced by apic_timer_rearm(), and riscv64_timer_rearm().

Here is the caller graph for this function:



7.67.4.3 heartbeat_read_counter()

```
uint64_t heartbeat_read_counter (
            void  )
```

Read the raw hardware counter this architecture's heartbeat is paced against – the same clock `timer_now_ns()` and calibration already use internally (rdtsc on amd64, the `time` CSR on riscv64, CNTPCT_EL0 on aarch64), not a separate/different source.

Implemented once per architecture in that architecture's timer.c; consumed only by `heartbeat_tick()` in the shared heartbeat.c.

Read the raw hardware counter this architecture's heartbeat is paced against – the same clock `timer_now_ns()` and calibration already use internally (rdtsc on amd64, the `time` CSR on riscv64, CNTPCT_EL0 on aarch64), not a separate/different source.

Item 0.8 (FABRIC-0.md ?25.1): the shared heartbeat.c now owns heartbeat_init()/heartbeat_tick()/heartbeat_↩service()/heartbeat_ticks()/ heartbeat_trust()/heartbeat_state() and the per-arch `g_heartbeat` state that used to live in this file. This is the one piece that stays per-architecture – the same `CNTPCT_EL0` the rest of this file's calibration already reads via `cntpct_read()`.

Returns

Current `CNTPCT_EL0` value.

Read the raw hardware counter this architecture's heartbeat is paced against – the same clock `timer_now_ns()` and calibration already use internally (rdtsc on amd64, the `time` CSR on riscv64, CNTPCT_EL0 on aarch64), not a separate/different source.

Item 0.8 (FABRIC-0.md ?25.1): the shared heartbeat.c owns heartbeat_init()/heartbeat_tick()/heartbeat_↩service()/heartbeat_ticks()/ heartbeat_trust()/heartbeat_state() and the variance/trust math that used to live in this file. This is the one piece that stays per-architecture – the same `rdtsc()` the rest of this file's TSC calibration already uses, not a separate/different source.

Returns

Current TSC value.

Read the raw hardware counter this architecture's heartbeat is paced against – the same clock `timer_now_ns()` and calibration already use internally (rdtsc on amd64, the `time` CSR on riscv64, CNTPCT_EL0 on aarch64), not a separate/different source.

Item 0.8 (FABRIC-0.md ?25.1): the shared heartbeat.c now owns heartbeat_init()/heartbeat_tick()/heartbeat_↩ service()/heartbeat_ticks()/ heartbeat_trust()/heartbeat_state(). This is the one piece that stays per-architecture – the same `rdtime()` the timer deadline is armed against, not `rdcycle()` or any other source. Must read the same counter the deadline was programmed against: `expected_delta` is derived from timebase-frequency and is therefore in `time` units; measuring the interval with `cycle` instead would difference two unrelated clocks.

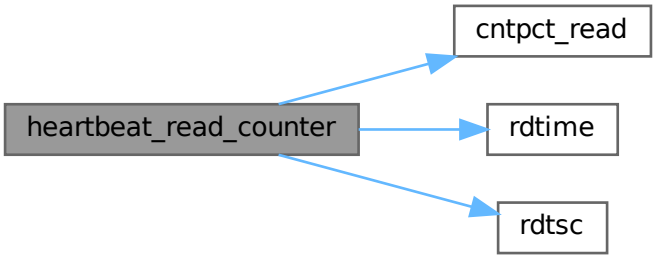
Returns

Current `time` CSR value.

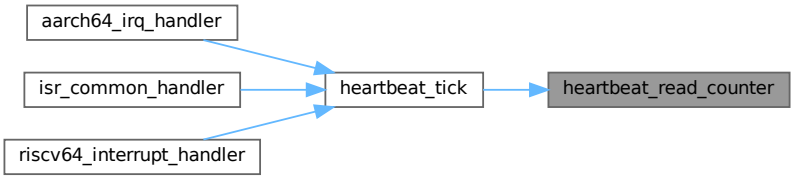
References cntpct_read(), rdtime(), and rdtsc().

Referenced by heartbeat_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.67.4.4 heartbeat_service()

```
void heartbeat_service (
            void  )
```

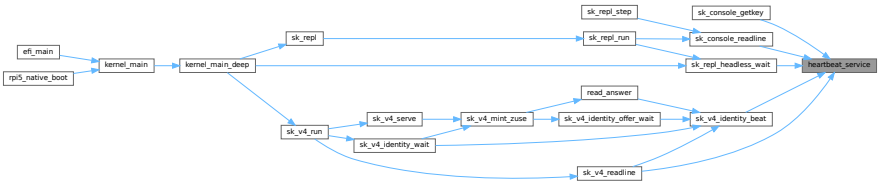
Bottom half (punch-list item 0.8).

Services one pending sample if `heartbeat_tick()` has latched one since the last call: computes the inter-tick deviation, updates the rolling window, and (architecture permitting – see `heartbeat.c`) recomputes variance and TIME-TRUST. Never runs in interrupt context. Call from the mainline, as frequently as convenient – a stale/skipped service call degrades the window's fidelity but affects nothing else, since TIME-TRUST is diagnostic only and never gates execution.

References time_trust_state::expected_delta, g_heartbeat, g_pending_counter, g_pending_valid, time_trust_state::last_tsc, Q48_ONE, time_trust_state::total_samples, time_trust_state::trust, time_trust_state::variance, and time_trust_state::window.

Referenced by sk_console_getkey(), sk_console_readline(), sk_repl_headless_wait(), sk_v4_identity_beat(), and sk_v4_readline().

Here is the caller graph for this function:



7.67.4.5 heartbeat_set_adaptive_period_ns()

```
void heartbeat_set_adaptive_period_ns (
            uint64_t ns )
```

Set the adaptive re-arm period, in nanoseconds (punch-list item 0.8, FABRIC-0.md ?26).

Called from the mainline execution path only (Loop #7's site in vm_runtime.c) – never from interrupt context. Clamped to [1/4x, 4x] of the kernel's base period internally; a caller need not pre-clamp.

Set the adaptive re-arm period, in nanoseconds (punch-list item 0.8, FABRIC-0.md ?26).

Called from vm_runtime.c's Loop #7 site on the mainline execution path (never interrupt context) with a value already rescaled to this file's HEARTBEAT_BASE_PERIOD_NS. Clamped defensively to the same [1/4x, 4x] band Loop #7 itself enforces around its own base, so a caller forwarding an unclamped or wrongly-scaled value cannot run the ISR away.

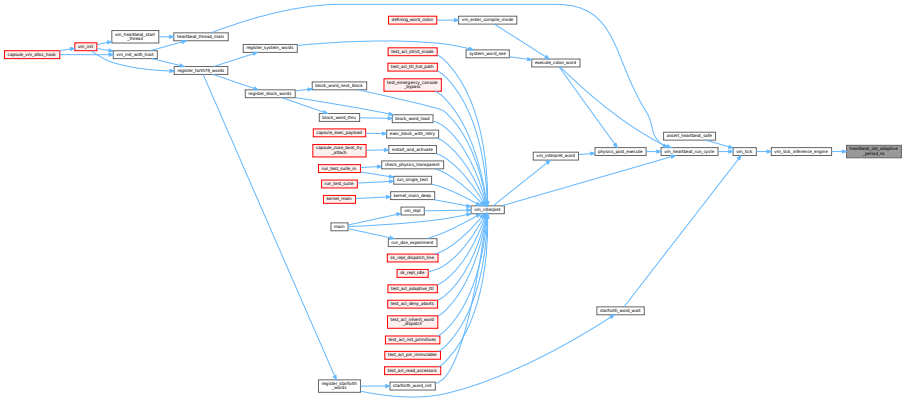
Parameters

ns	Desired period in nanoseconds for the next re-arm.

References g_adaptive_period_ns, and HEARTBEAT_BASE_PERIOD_NS.

Referenced by vm_tick_inference_engine().

Here is the caller graph for this function:



7.67.4.6 heartbeat_state()

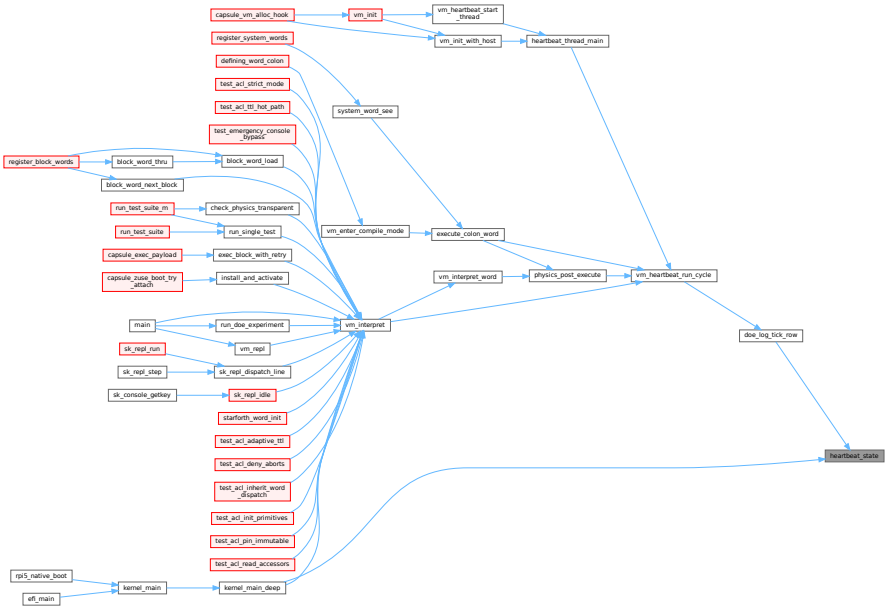
```
const TimeTrustState * heartbeat_state (
            void  )
```

Get pointer to full heartbeat state (for diagnostics).

References g_heartbeat.

Referenced by doe_log_tick_row(), and kernel_main_deep().

Here is the caller graph for this function:



7.67.4.7 heartbeat_tick()

```
void heartbeat_tick (
            void  )
```

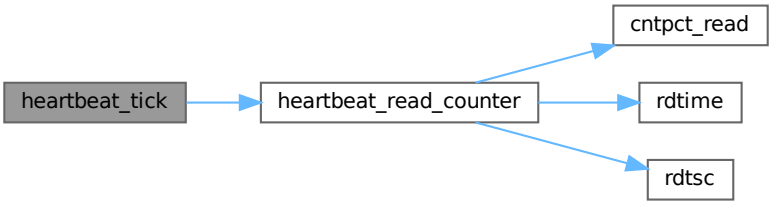
Top half.

Called directly from each architecture's ISR (punch-list item 0.8) – one call site per architecture, unchanged from before this item. Does exactly three things: reads the raw counter via `heartbeat_read_counter()`, increments TIME-TICKS, and latches the sample for `heartbeat_service()` to pick up. No window math, no variance, no loops – this must stay cheap enough for interrupt context.

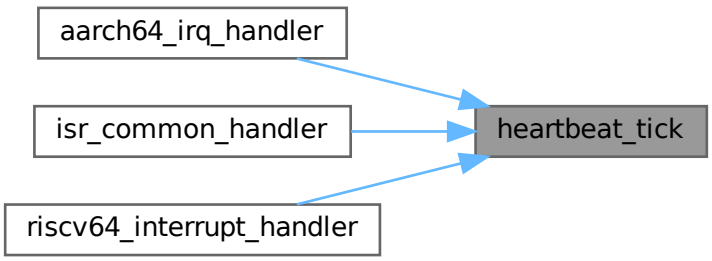
References g_heartbeat, g_pending_counter, g_pending_valid, heartbeat_read_counter(), and time_trust_state::ticks.

Referenced by aarch64_irq_handler(), isr_common_handler(), and riscv64_interrupt_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.67.4.8 heartbeat_ticks()

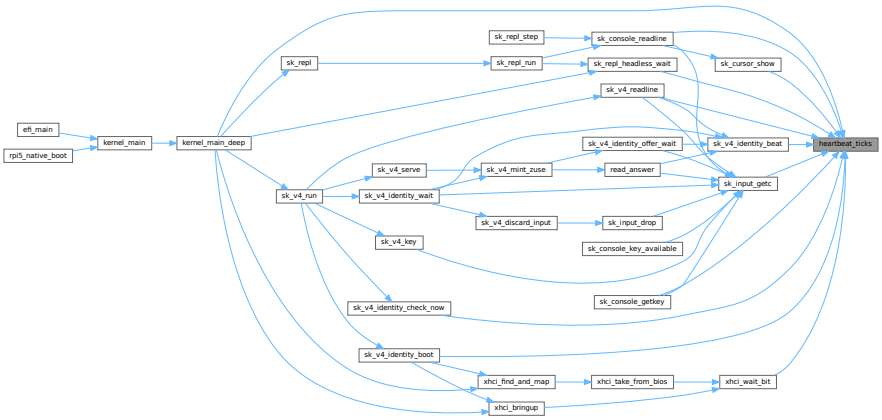
```
uint64_t heartbeat_ticks (
            void  )
```

Get current TIME-TICKS (monotonic heartbeat count).

References g_heartbeat, and time_trust_state::ticks.

Referenced by kernel_main_deep(), sk_console_getkey(), sk_console_readline(), sk_cursor_show(), sk_input_getc(), sk_repl_headless_wait(), sk_v4_identity_beat(), sk_v4_identity_boot(), sk_v4_identity_check_now(), sk_v4_readline(), and xhci_wait_bit().

Here is the caller graph for this function:



7.67.4.9 heartbeat_trust()

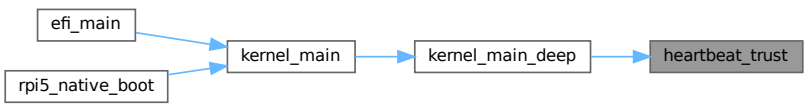
```
time_trust_t heartbeat_trust (
            void  )
```

Get current TIME-TRUST (Q48.16 confidence metric).

References g_heartbeat, and time_trust_state::trust.

Referenced by kernel_main_deep().

Here is the caller graph for this function:



7.67.4.10 timer_calibration_record()

```
const timer_calibration_record_t * timer_calibration_record (
            void  )
```

Return a pointer to the timer calibration record.

Returns `&s_cal` which is populated by `timer_init()` with:

- `tsc_hz_mean` = `CNTFRQ_EL0` (or 62.5 MHz fallback)
- `hpet_hz` = 0 (no HPET on AArch64)
- `pit_hz_mean` = 0 (no PIT on AArch64)
- `converged` = 1 (Generic Timer is already calibrated by firmware)
- `vm_mode` = real hypervisor-vs-hardware detection as of 2026-09-04 (ACPI RSDP OEM ID check, running_under_hypervisor() above) – the Generic Timer itself works identically either way, so this field's *timing* meaning here is purely informational, unlike amd64 where vm_mode gates a real calibration-path choice
- `trust` = `TIMER_TRUST_ABSOLUTE`

The record is exposed to `kernel_main()` and `timer.h` consumers for boot-log reporting and for the heartbeat subsystem's trust initialisation.

Returns

Pointer to the module-static calibration record; valid for the lifetime of the kernel.

Return a pointer to the timer calibration record.

Provides read-only access to the `calib_record` populated by `timer_init()`. Callers can inspect fields such as `trust`, `tsc_hz_mean`, `cv_hpet_ppm`, and `converged` for diagnostic output or to determine whether absolute timing guarantees are available.

Returns

Pointer to the static `timer_calibration_record_t` (never NULL).

Returns `&s_cal`, populated by `timer_init()` with the discovered (or fallback) frequency and the corresponding trust level — `TIMER_TRUST_ABSOLUTE` when `timebase-frequency` came from the devicetree, `TIMER_TRUST_RELATIVE` on the fallback. Consumed by `kernel_main()` for boot-log reporting and by the heartbeat subsystem's trust init.

Returns

Pointer to the module-static calibration record; valid for the lifetime of the kernel.

References calib_record, and s_cal.

7.67.4.11 timer_check_drift_now()

```
int timer_check_drift_now (
            void  )
```

Check for timer drift (AArch64 stub — always returns 0).

On x86-64 this function cross-checks the TSC against the HPET or PM Timer to detect frequency drift. The AArch64 Generic Timer is architecturally invariant and synchronised across all cores in the system; it cannot drift against itself. This stub always returns 0 (no drift detected) to satisfy the common `timer_check_drift_now()` call site.

Returns

0 always.

Check for timer drift (AArch64 stub — always returns 0).

In bare-metal mode delegates to `timer_drift_check_bare_metal()`. In VM mode returns 0 immediately: VM drift checking requires a VM-friendly reference (such as pvclock) that is not yet implemented.

Returns

0 if drift is acceptable or in VM mode, -1 on bare-metal measurement failure.

Check for timer drift (AArch64 stub — always returns 0).

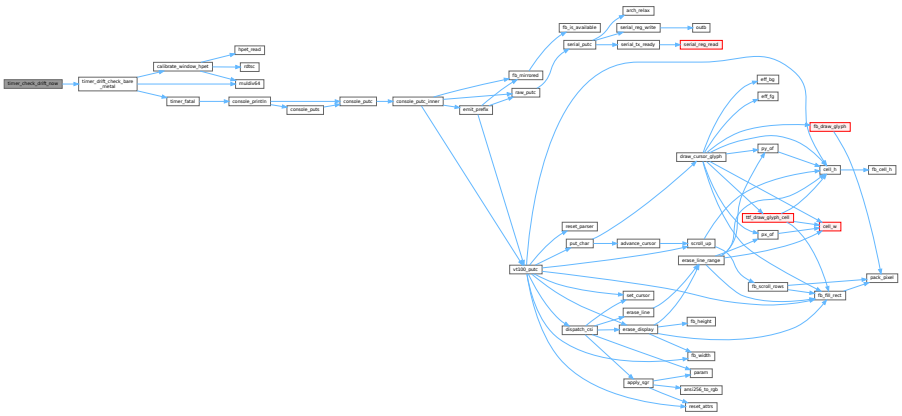
On x86-64 this cross-checks the TSC against the HPET. The RISC-V `time` counter has no independent reference to check against at this milestone — and per the privileged spec it is architecturally required to be a fixed, synchronised rate across harts, unlike `cycle`. Always returns 0 (no drift) to satisfy the common call site.

Returns

0 always.

References calib_record, timer_drift_check_bare_metal(), and timer_calibration_record::vm_mode.

Here is the call graph for this function:



7.67.4.12 timer_init()

```
int timer_init (
            BootInfo * boot_info )
```

Initialise the kernel timer subsystem (M5).

Detects the execution environment (bare metal vs. hypervisor) and follows the appropriate calibration path:

- **VM** / **hypervisor path** (CPUID.1:ECX[31] set): calls `init_timer_vm_mode()` — HPET MMIO disabled, TSC from CPUID/PM Timer.
- **Bare metal path**: maps HPET MMIO, enables the counter, optionally probes the PIT, then calls `converge_tsc_bare_metal()` until TSC Hz converges within `CV_EPSILON_PPM` ppm. Calls `timer_fatal()` on any hard failure.

On success, `tsc_hz_locked` holds the calibrated TSC frequency and `calib_record` is fully populated. `timer_now_ns()` is safe to call after this function returns.

Parameters

boot_info	BootInfo pointer from the UEFI loader; `acpi_table` is used to discover the ACPI PM_TMR_BLK port at runtime (see `fadt_find_pm_tmr_port()`) rather than assuming a fixed machine type's legacy address.

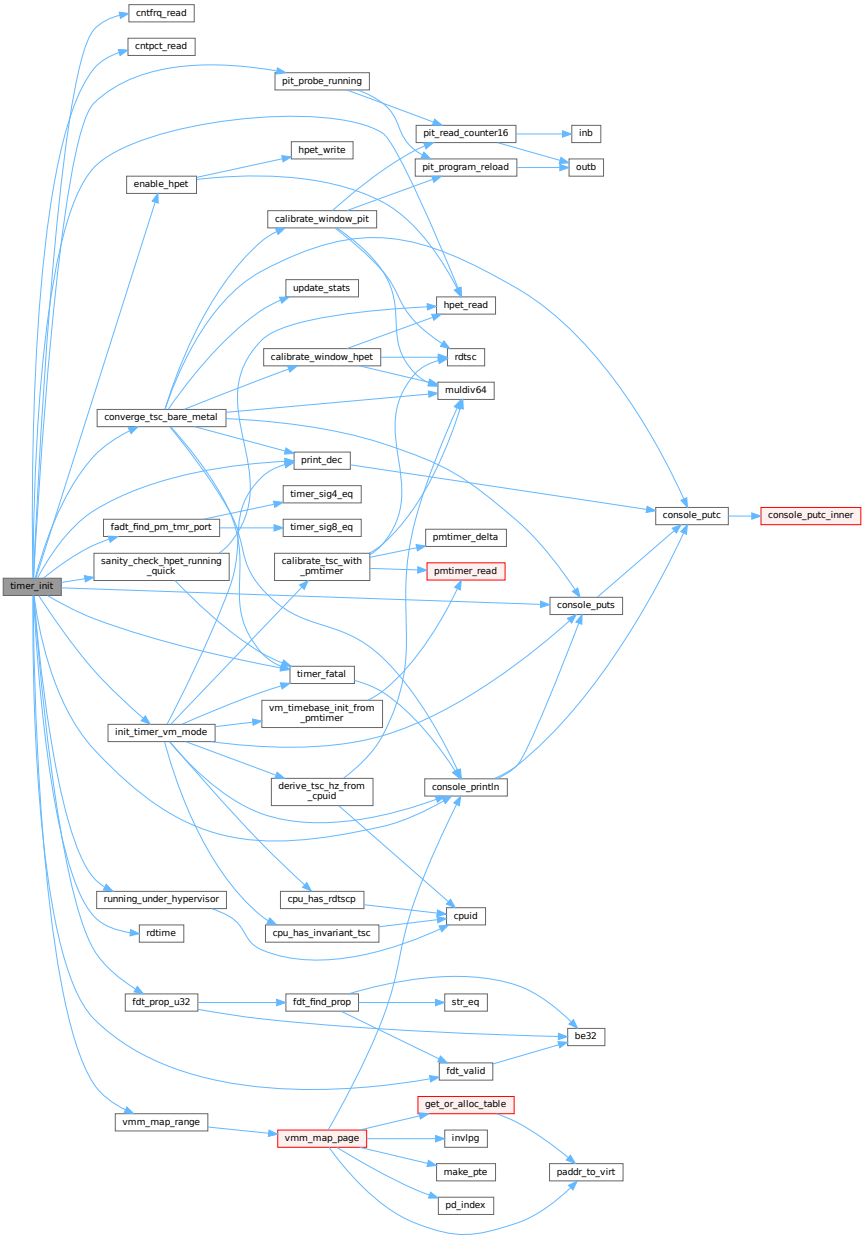
Returns

0 on success; calls `timer_fatal()` (never returns) on hard failure.

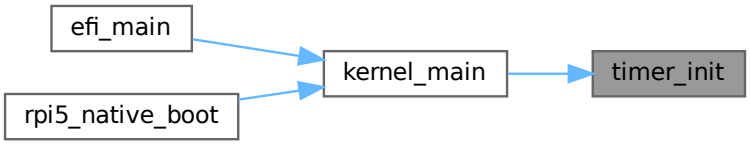
References BootInfo::acpi_table, calib_record, cntfrq_read(), cntpct_read(), console_println(), console_puts(), converge_tsc_bare_metal(), timer_calibration_record::converged, BootInfo::dtb, enable_hpet(), fadt_find_pm_tmr_port(), fdt_prop_u32(), fdt_valid(), hpet_freq_hz, HPET_GEN_CAP_ID, timer_calibration_record::hpet_hz, HPET_PHYS_BASE, hpet_read(), hpet_regs, HPET_VIRT_BASE, init_timer_vm_mode(), pit_available, timer_calibration_record::pit_hz_mean, pit_probe_running(), pit_tsc_hz_mean, pmtimer_port, print_dec(), rdtime(), running_under_hypervisor(), s_base_count, s_base_ns, s_cal, s_counter_hz, s_ns_per_tick, sanity_check_hpet_running_quick(), timer_fatal(), TIMER_REQUIRE_PIT, TIMER_TRUST_ABSOLUTE, TIMER_TRUST_NONE, TIMER_TRUST_RELATIVE, timer_calibration_record::trust, tsc_hz_locked, timer_calibration_record::tsc_hz_mean, timer_calibration_record::vm_mode, VMM_FLAG_WRITABLE, and vmm_map_range().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.67.4.13 timer_now_ns()

```
uint64_t timer_now_ns (
            void  )
```

Return a monotonic nanosecond timestamp.

Reads CNTPCT_EL0, computes the delta from s_base_count (captured at timer_init()), and converts to nanoseconds using the Q16.16 fixed-point s_ns_per_tick:
ns = s_base_ns + ((delta * s_ns_per_tick) » 16)

The multiplication uses 64-bit arithmetic; at typical counter frequencies ($\leq$ 100 MHz) and reasonable uptime intervals (days to weeks) there is no overflow risk in the delta-to-ns conversion.

Returns

Monotonic nanosecond count since timer_init(), starting at 0.

Return a monotonic nanosecond timestamp.

Selects the best available time source based on the calibration record:

- TIMER_TRUST_NONE: returns 0 (no time base yet).
- VM mode + trust $<$ TIMER_TRUST_ABSOLUTE: returns PM Timer relative ns.
- Absolute mode (tsc_hz_locked $>$ 0): reads TSC (RDTSCP if supported, else RDTSC) and converts to ns via muldiv64().

Called by the HAL (sk_hal_time_ns()) to feed the physics engine.

Returns

Nanoseconds since boot, or 0 if the timer is not yet calibrated.

Reads rdtime(), computes the elapsed delta from s_base_count (captured at timer_init()), and converts to nanoseconds using the Q16.16 fixed-point s_ns_per_tick:
ns = s_base_ns + ((delta * s_ns_per_tick) » 16)

The 64-bit product of delta $\times$ s_ns_per_tick wraps once elapsed real time reaches 2^48 ns ($\sim$3.26 days) — this bound is **independent of s_counter_hz**, since s_ns_per_tick scales inversely with frequency and the two effects cancel. Verified by direct computation, not carried over from the earlier cycle-based comment here, which claimed "$\sim$18 seconds" and did not match the arithmetic either. Beyond $\sim$3.26 days uptime timer_now_ns() wraps and produces incorrect values — a theoretical concern only for this single-hart POST-only kernel.

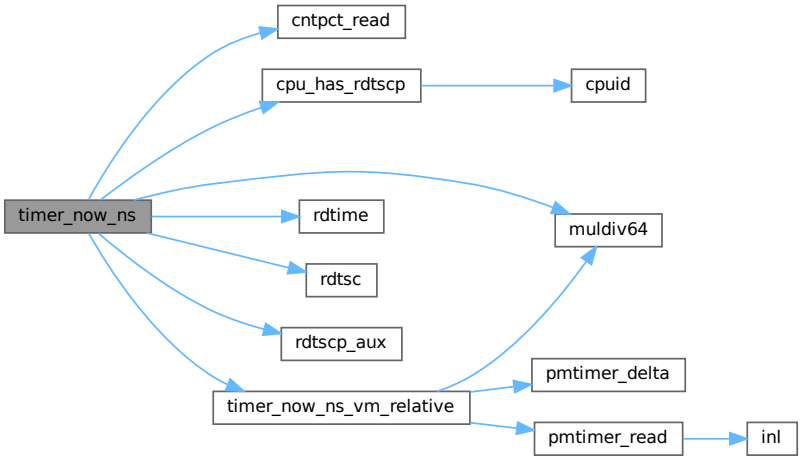
Returns

Monotonic nanosecond count since `timer_init()`, starting at 0.

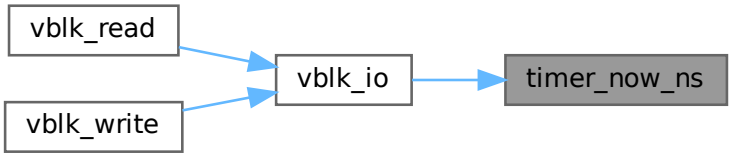
References calib_record, cntpct_read(), cpu_has_rdtscp(), muldiv64(), rdtime(), rdtsc(), rdtscp_aux(), s_base_count, s_base_ns, s_ns_per_tick, timer_now_ns_vm_relative(), TIMER_TRUST_ABSOLUTE, TIMER_TRUST_NONE, timer_calibration_record::trust, tsc_hz_locked, and timer_calibration_record::vm_mode.

Referenced by vblk_io().

Here is the call graph for this function:



Here is the caller graph for this function:



7.67.4.14 timer_tsc_hz()

```
uint64_t timer_tsc_hz (
            void  )
```

Return the system counter frequency in Hz.

Returns `s_counter_hz` as set from `CNTFRQ_EL0` by `timer_init()`. If `timer_init()` has not been called yet (or firmware reported 0 and the fallback was not applied), returns 62,500,000 Hz as a safe default.

Used by `apic_timer_init()` to compute `s_timer_period_tsc` and by the shim time backend to convert counter ticks to nanoseconds before the full timer subsystem is available.

Returns

System counter frequency in Hz; at least 62,500,000.

Return the system counter frequency in Hz.

Returns the value established by `timer_init()`. Zero means calibration has not completed or failed to derive a TSC frequency (PM Timer relative mode only).

Returns

TSC frequency in Hz, or 0 if unknown.

Return the system counter frequency in Hz.

Returns `s_counter_hz` — discovered from the devicetree's `timebase-frequency` when available, otherwise `RISCV_TIMEBASE_HZ_FALLBACK`. Used by `apic_timer_init()` to compute the expected inter-tick period and by the shim time backend to convert `time` counts to nanoseconds.

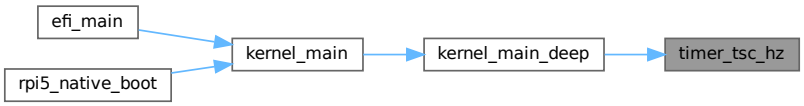
Returns

`time` counter frequency in Hz.

References s_counter_hz, and tsc_hz_locked.

Referenced by kernel_main_deep().

Here is the caller graph for this function:

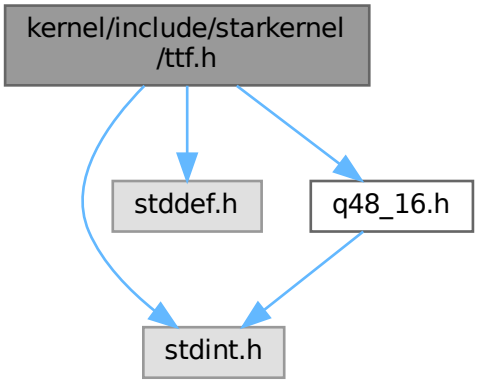


7.68 kernel/include/starkernel/ttf.h File Reference

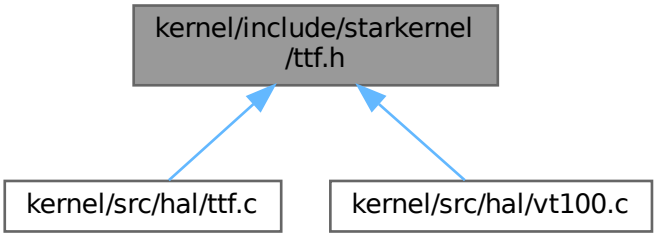
```
#include <stdint.h>
#include <stddef.h>
```

```
#include "q48_16.h"
```

Include dependency graph for ttf.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct ttf_font_t

  Parsed font handle.
- struct ttf_glyph_header_t

  Raw glyf record header, per ?27.6/4.3.7's "done when" clause.
- struct ttf_point_t

  One outline point, in raw font design units (NOT scaled by unitsPerEm — that's the caller's job, same convention as ttf_glyph_header_t's bbox), expressed in Q48.16.
- struct ttf_outline_t

  Simple- and composite-glyph outline, flattened to one point list plus per-contour end indices (TrueType convention: contour_ends[c] is the index of the LAST point of contour c, inclusive; points are shared across contours only in the sense that contour c+1 starts right after contour_ends[c]).
- struct ttf_bitmap_t

  A caller-owned 8-bit-per-pixel bitmap.
- struct ttf_raster_cache_slot_t
- struct ttf_raster_cache_t

Macros

- #define TTF_OK 0

  ttf.h - TrueType font parser core (Freestanding)
- #define TTF_ERR_BAD_SFNT -1
- #define TTF_ERR_TABLE_MISSING -2
- #define TTF_ERR_BAD_TABLE -3
- #define TTF_ERR_BAD_GLYPH_INDEX -4
- #define TTF_ERR_OUT_OF_BOUNDS -5
- #define TTF_ERR_TOO_MANY_POINTS -6
- #define TTF_ERR_TOO_MANY_CONTOURS -7
- #define TTF_ERR_TOO_DEEP -8 /∗ composite glyph nesting exceeded TTF_MAX_COMPOSITE_DEPTH ∗/
- #define TTF_ERR_UNSUPPORTED
- #define TTF_MAX_COMPOSITE_DEPTH 8
- #define TTF_GLYPH_MISSING 0

  Glyph index returned for "no mapping" by ttf_codepoint_to_glyph().
- #define TTF_CACHE_MAX_SIZE_PX 64
- #define TTF_CACHE_MARGIN 8
- #define TTF_CACHE_BITMAP_DIM (TTF_CACHE_MAX_SIZE_PX + TTF_CACHE_MARGIN ∗ 2)
- #define TTF_CACHE_BITMAP_BYTES (TTF_CACHE_BITMAP_DIM ∗ TTF_CACHE_BITMAP_DIM)

Functions

- int ttf_parse (const uint8_t ∗data, uint32_t size, ttf_font_t ∗out)

  ttf_parse - Locate and validate the sfnt directory and the head/maxp/ loca/glyf tables (cmap is optional; ttf_codepoint_to_glyph() fails cleanly if absent).
- uint32_t ttf_codepoint_to_glyph (const ttf_font_t ∗font, uint32_t codepoint)

  ttf_codepoint_to_glyph - Resolve a Unicode codepoint via the font's format-4 cmap subtable.
- int ttf_glyph_header (const ttf_font_t ∗font, uint32_t glyph_index, ttf_glyph_header_t ∗out)

  ttf_glyph_header - Read a glyph's outline header (contour count, bounding box) via loca + glyf.
- uint16_t ttf_glyph_advance_width (const ttf_font_t ∗font, uint32_t glyph_index)

  ttf_glyph_advance_width - Horizontal advance width via hmtx, in raw font design units (NOT scaled by unitsPerEm – caller's job, e.g.
- int ttf_glyph_outline (const ttf_font_t ∗font, uint32_t glyph_index, ttf_outline_t ∗out)

  ttf_glyph_outline - Extract a glyph's outline (simple or composite, recursively resolving composite components) into caller-supplied buffers.
- int ttf_rasterize_glyph (const ttf_font_t ∗font, uint32_t glyph_index, q48_16_t scale, q48_16_t origin_↩ x, q48_16_t origin_y, uint8_t fill_value, ttf_bitmap_t ∗out)

  ttf_rasterize_glyph - Flatten a glyph's outline (quadratic Bezier contours, fixed segment count per curve — see ttf.c's file header comment) and fill it into `out` using the even-odd rule (FABRIC-0.md item 4.3.7c; see that item's completion note for why even-odd rather than nonzero winding — correct for the v1 glyph repertoire's non-self- intersecting nested contours, not necessarily for an arbitrary font).
- void ttf_raster_cache_init (ttf_raster_cache_t ∗cache, ttf_raster_cache_slot_t ∗slots, uint32_t slot_count)

  ttf_raster_cache_init - Bind a caller-supplied slot array to `cache` and mark every slot empty.
- int ttf_raster_cache_get (ttf_raster_cache_t ∗cache, const ttf_font_t ∗font, uint32_t codepoint, uint32_t size↩ _px, ttf_bitmap_t ∗out, int ∗was_hit)

  ttf_raster_cache_get - Look up (font, codepoint, size_px); on a miss, rasterize and insert (evicting round-robin if every slot is full).

7.68.1 Macro Definition Documentation

7.68.1.1 TTF_CACHE_BITMAP_BYTES

```
#define TTF_CACHE_BITMAP_BYTES (TTF_CACHE_BITMAP_DIM * TTF_CACHE_BITMAP_DIM)
```

7.68.1.2 TTF_CACHE_BITMAP_DIM

```
#define TTF_CACHE_BITMAP_DIM (TTF_CACHE_MAX_SIZE_PX + TTF_CACHE_MARGIN * 2)
```

7.68.1.3 TTF_CACHE_MARGIN

```
#define TTF_CACHE_MARGIN 8
```

7.68.1.4 TTF_CACHE_MAX_SIZE_PX

```
#define TTF_CACHE_MAX_SIZE_PX 64
```

7.68.1.5 TTF_ERR_BAD_GLYPH_INDEX

```
#define TTF_ERR_BAD_GLYPH_INDEX -4
```

7.68.1.6 TTF_ERR_BAD_SFNT

```
#define TTF_ERR_BAD_SFNT -1
```

7.68.1.7 TTF_ERR_BAD_TABLE

```
#define TTF_ERR_BAD_TABLE -3
```

7.68.1.8 TTF_ERR_OUT_OF_BOUNDS

```
#define TTF_ERR_OUT_OF_BOUNDS -5
```

7.68.1.9 TTF_ERR_TABLE_MISSING

```
#define TTF_ERR_TABLE_MISSING -2
```

7.68.1.10 TTF_ERR_TOO_DEEP

```
#define TTF_ERR_TOO_DEEP -8 /* composite glyph nesting exceeded TTF_MAX_COMPOSITE_DEPTH */
```

7.68.1.11 TTF_ERR_TOO_MANY_CONTOURS

```
#define TTF_ERR_TOO_MANY_CONTOURS -7
```

7.68.1.12 TTF_ERR_TOO_MANY_POINTS

```
#define TTF_ERR_TOO_MANY_POINTS -6
```

7.68.1.13 TTF_ERR_UNSUPPORTED

```
#define TTF_ERR_UNSUPPORTED
```

Value:

```
-9 /* e.g. a non-identity composite transform or point-matched
* component args -- see ttf.c's file header comment; not a
* malformed font, just a code path this parser doesn't
* implement yet */
```

7.68.1.14 TTF_GLYPH_MISSING

```
#define TTF_GLYPH_MISSING 0
```

Glyph index returned for "no mapping" by ttf_codepoint_to_glyph().

7.68.1.15 TTF_MAX_COMPOSITE_DEPTH

```
#define TTF_MAX_COMPOSITE_DEPTH 8
```

7.68.1.16 TTF_OK

```
#define TTF_OK 0
```

ttf.h - TrueType font parser core (Freestanding)

FABRIC-0.md item 4.3.7. Reads a TTF's sfnt directory plus head/maxp/loca/ glyf/cmap tables, resolving a Unicode codepoint to a glyph index and its outline header (contour count, bounding box). Does NOT extract outline points or rasterize — that is 4.3.7a/4.3.7c. No floating point; all fields read here are raw integers straight from the font's own big-endian on-disk format (see FABRIC-0.md ?27.7 decision #2 for why the Q48.16-vs-float call was made, and why it doesn't bind this file, which never scales anything).

cmap: only format 4 (Windows/Unicode BMP) subtables are resolved. This covers ASCII and all of the BMP, which is what the v1 glyph repertoire (?27.6.4) needs. Format 12 (supplementary planes) is deferred — no v1 glyph requires it.

7.68.2 Function Documentation

7.68.2.1 ttf_codepoint_to_glyph()

```
uint32_t ttf_codepoint_to_glyph (
            const ttf_font_t * font,
            uint32_t codepoint )
```

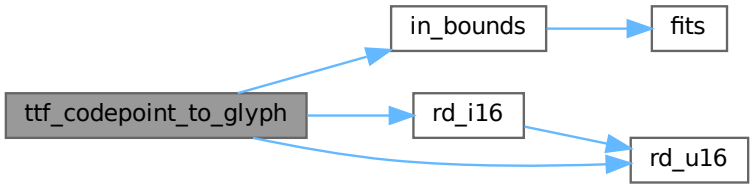
ttf_codepoint_to_glyph - Resolve a Unicode codepoint via the font's format-4 cmap subtable.

Returns

glyph index, or TTF_GLYPH_MISSING if unmapped or no cmap

References ttf_font_t::cmap_subtable_off, ttf_font_t::data, ttf_font_t::has_cmap, in_bounds(), rd_i16(), rd_u16(), and TTF_GLYPH_MISSING.

Here is the call graph for this function:



7.68.2.2 ttf_glyph_advance_width()

```
uint16_t ttf_glyph_advance_width (
            const ttf_font_t * font,
            uint32_t glyph_index )
```

ttf_glyph_advance_width - Horizontal advance width via hmtx, in raw font design units (NOT scaled by unitsPerEm – caller's job, e.g.

`q48_mul(q48_from_u64(width), scale)`; always non-negative, so the shared q48_mul is fine here unlike the signed cases documented in ttf.c's rasterizer). Glyphs past hmtx's numberOfHMetrics entries share the last entry's width, per spec.

Parameters

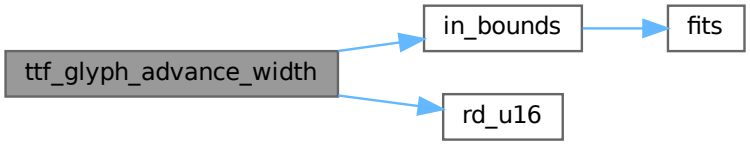
glyph_index	As returned by ttf_codepoint_to_glyph()

Returns

advance width, or 0 if glyph_index is out of range

References ttf_font_t::data, ttf_font_t::hmtx_off, in_bounds(), ttf_font_t::num_h_metrics, and rd_u16().

Here is the call graph for this function:



7.68.2.3 ttf_glyph_header()

```
int ttf_glyph_header (
            const ttf_font_t * font,
            uint32_t glyph_index,
            ttf_glyph_header_t * out )
```

ttf_glyph_header - Read a glyph's outline header (contour count, bounding box) via loca + glyf.

Does not extract contour points.

Parameters

glyph_index	As returned by ttf_codepoint_to_glyph()
out	Header to populate

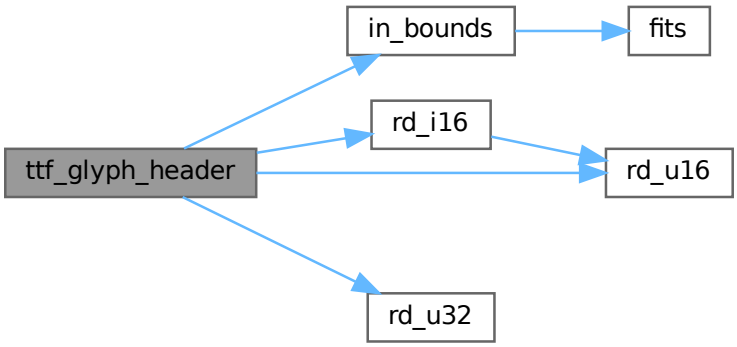
Returns

TTF_OK, or a TTF_ERR_∗ code

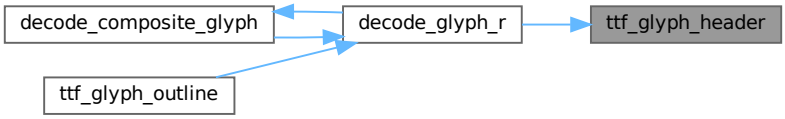
References ttf_font_t::data, ttf_glyph_header_t::glyf_length, ttf_font_t::glyf_off, ttf_glyph_header_t::glyf_offset, in_bounds(), ttf_font_t::index_to_loc_format, ttf_font_t::loca_off, ttf_glyph_header_t::num_contours, ttf_font_t::num_glyphs, rd_i16(), rd_u16(), rd_u32(), TTF_ERR_BAD_GLYPH_INDEX, TTF_ERR_BAD_TABLE, TTF_ERR_OUT_OF_BOUNDS, TTF_OK, ttf_glyph_header_t::x_max, ttf_glyph_header_t::x_min, ttf_glyph_header_t::y_max, and ttf_glyph_header_t::y_min.

Referenced by decode_glyph_r().

Here is the call graph for this function:



Here is the caller graph for this function:



7.68.2.4 ttf_glyph_outline()

```
int ttf_glyph_outline (
            const ttf_font_t * font,
            uint32_t glyph_index,
            ttf_outline_t * out )
```

ttf_glyph_outline - Extract a glyph's outline (simple or composite, recursively resolving composite components) into caller-supplied buffers.

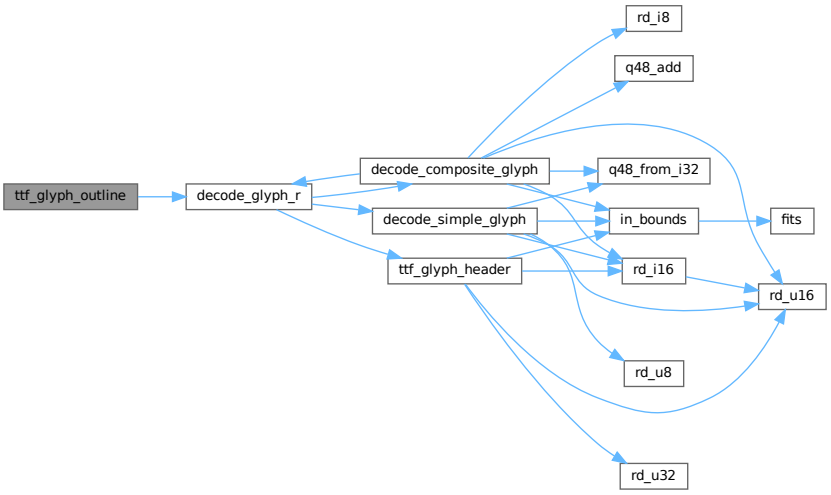
Composite components with a non-identity transform (any scale/rotation/ skew, i.e. anything but a pure (dx,dy) translation) or with point-matched (rather than xy-offset) placement args return TTF_ERR_UNSUPPORTED rather than silently producing a wrong outline — see ttf.c's file header comment for why, and check that limitation before relying on this for an arbitrary font.

Returns

TTF_OK, or a TTF_ERR_∗ code (including TTF_ERR_TOO_MANY_POINTS/ _CONTOURS if a caller buffer is too small, and TTF_ERR_TOO_DEEP if composite nesting exceeds TTF_MAX_COMPOSITE_DEPTH)

References ttf_outline_t::contour_count, decode_glyph_r(), ttf_outline_t::point_count, and TTF_ERR_BAD_TABLE.

Here is the call graph for this function:



7.68.2.5 ttf_parse()

```
int ttf_parse (
            const uint8_t * data,
            uint32_t size,
            ttf_font_t * out )
```

ttf_parse - Locate and validate the sfnt directory and the head/maxp/ loca/glyf tables (cmap is optional; ttf_codepoint_to_glyph() fails cleanly if absent).

Does not copy `data` — `out` borrows it.

Parameters

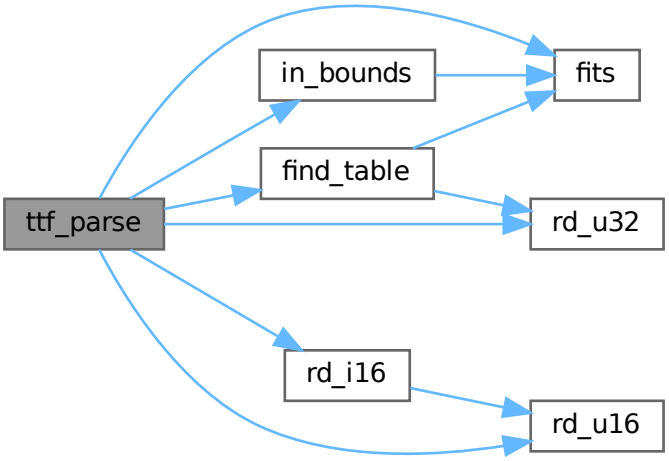
data	Whole .ttf file contents
size	Length of data in bytes
out	Parsed handle to populate

Returns

TTF_OK, or a TTF_ERR_∗ code

References ttf_font_t::cmap_subtable_off, ttf_font_t::data, find_table(), fits(), ttf_font_t::glyf_len, ttf_font_t::glyf_off, ttf_font_t::has_cmap, ttf_font_t::head_off, ttf_font_t::hmtx_off, in_bounds(), ttf_font_t::index_to_loc_format, ttf_font_t::loca_len, ttf_font_t::loca_off, ttf_font_t::maxp_off, ttf_font_t::num_glyphs, ttf_font_t::num_h_metrics, rd_i16(), rd_u16(), rd_u32(), ttf_font_t::size, TAG, TTF_ERR_BAD_SFNT, TTF_ERR_BAD_TABLE, TTF_ERR_OUT_OF_BOUNDS, TTF_OK, and ttf_font_t::units_per_em.

Here is the call graph for this function:



7.68.2.6 ttf_raster_cache_get()

```
int ttf_raster_cache_get (
            ttf_raster_cache_t * cache,
            const ttf_font_t * font,
            uint32_t codepoint,
            uint32_t size_px,
            ttf_bitmap_t * out,
            int * was_hit )
```

ttf_raster_cache_get - Look up (font, codepoint, size_px); on a miss, rasterize and insert (evicting round-robin if every slot is full).

`out` borrows the winning slot's buffer directly – valid until that slot is evicted by a later call.

Parameters

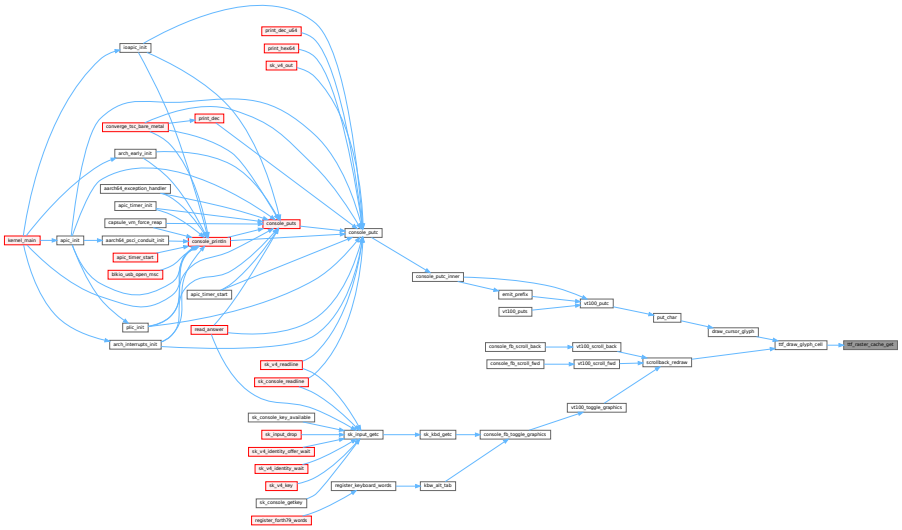
was_hit	If non-NULL, set to 1 on a cache hit, 0 if this call rasterized and inserted

Returns

TTF_OK, TTF_ERR_UNSUPPORTED if size_px > TTF_CACHE_MAX_SIZE_PX, or a ttf_rasterize_glyph() TTF_ERR_∗ code on a miss that failed to rasterize

Referenced by ttf_draw_glyph_cell().

Here is the caller graph for this function:



7.68.2.7 ttf_raster_cache_init()

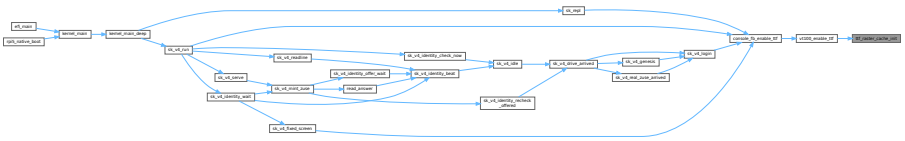
```
void ttf_raster_cache_init (
            ttf_raster_cache_t * cache,
            ttf_raster_cache_slot_t * slots,
            uint32_t slot_count )
```

ttf_raster_cache_init - Bind a caller-supplied slot array to `cache` and mark every slot empty.

References ttf_raster_cache_t::evict_next, ttf_raster_cache_t::slot_count, ttf_raster_cache_t::slots, and slots.

Referenced by vt100_enable_ttf().

Here is the caller graph for this function:



7.68.2.8 ttf_rasterize_glyph()

```
int ttf_rasterize_glyph (
            const ttf_font_t * font,
            uint32_t glyph_index,
            q48_16_t scale,
            q48_16_t origin_x,
            q48_16_t origin_y,
```

```
            uint8_t fill_value,
            ttf_bitmap_t * out )
```

ttf_rasterize_glyph - Flatten a glyph's outline (quadratic Bezier contours, fixed segment count per curve — see ttf.c's file header comment) and fill it into `out` using the even-odd rule (FABRIC-0.md item 4.3.7c; see that item's completion note for why even-odd rather than nonzero winding — correct for the v1 glyph repertoire's non-self-intersecting nested contours, not necessarily for an arbitrary font).

No antialiasing (explicitly deferred, per 4.3.7c's own "done when" clause). Never allocates — flattening uses a fixed-size local buffer bounded by TTF_RASTER_MAX_POINTS/TTF_RASTER_MAX_CONTOURS.

Parameters

scale	Font-design-units-to-pixels scale, Q48.16, e.g. q48_div(q48_from_u64(size_px), q48_from_u64(font->units_per_em))
origin_x	Pixel-space X (Q48.16) of the glyph's (0,0) font origin within `out`
origin_y	Pixel-space Y (Q48.16) of the glyph's baseline (font y=0) within `out`; font Y-up is flipped to raster Y-down internally

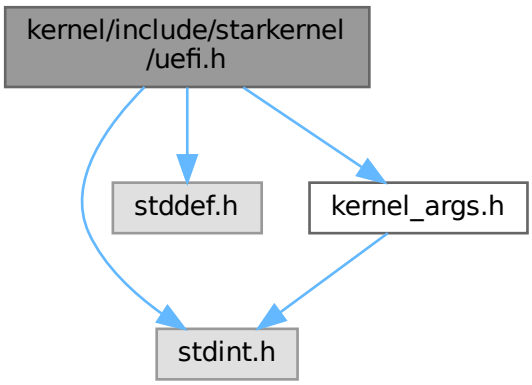
Returns

TTF_OK, or a TTF_ERR_* code (including TTF_ERR_TOO_MANY_POINTS/ _CONTOURS if the flattened outline exceeds the local buffer)

7.69 kernel/include/starkernel/uefi.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "kernel_args.h"
```

Include dependency graph for uefi.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct EFI_GUID
- struct EFI_MEMORY_DESCRIPTOR
- struct EFI_TABLE_HEADER
- struct _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL
- struct EFI_DEVICE_PATH_PROTOCOL
- struct _EFI_BOOT_SERVICES
- struct _EFI_RUNTIME_SERVICES
- struct _EFI_SYSTEM_TABLE
- struct EFI_CONFIGURATION_TABLE
- struct EFI_PIXEL_BITMASK
- struct EFI_GRAPHICS_OUTPUT_MODE_INFORMATION
- struct EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE
- struct _EFI_GRAPHICS_OUTPUT_PROTOCOL
- struct FramebufferInfo
- struct EFI_BLOCK_IO_MEDIA
- struct _EFI_BLOCK_IO_PROTOCOL
- struct _EFI_FILE_PROTOCOL
- struct _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL
- struct EFI_TIME
- struct EFI_FILE_INFO
- struct EFI_LOADED_IMAGE_PROTOCOL
- struct BootDisk
- struct BootInfo

Macros

- #define TRUE ((BOOLEAN)1)
- #define FALSE ((BOOLEAN)0)
- #define TPL_APPLICATION 4
- #define TPL_CALLBACK 8
- #define TPL_NOTIFY 16
- #define TPL_HIGH_LEVEL 31
- #define EFIAPI
- #define EFI_SUCCESS 0
- #define EFI_LOAD_ERROR (1 | (1ULL << 63))
- #define EFI_INVALID_PARAMETER (2 | (1ULL << 63))
- #define EFI_UNSUPPORTED (3 | (1ULL << 63))
- #define EFI_BAD_BUFFER_SIZE (4 | (1ULL << 63))
- #define EFI_BUFFER_TOO_SMALL (5 | (1ULL << 63))
- #define EFI_NOT_READY (6 | (1ULL << 63))
- #define EFI_DEVICE_ERROR (7 | (1ULL << 63))
- #define EFI_WRITE_PROTECTED (8 | (1ULL << 63))
- #define EFI_OUT_OF_RESOURCES (9 | (1ULL << 63))
- #define EFI_NOT_FOUND (14 | (1ULL << 63))
- #define EFI_ABORTED (21 | (1ULL << 63))

- #define EFI_MEMORY_UC 0x0000000000000001ULL
- #define EFI_MEMORY_WC 0x0000000000000002ULL
- #define EFI_MEMORY_WT 0x0000000000000004ULL
- #define EFI_MEMORY_WB 0x0000000000000008ULL
- #define EFI_MEMORY_UCE 0x0000000000000010ULL
- #define EFI_MEMORY_WP 0x0000000000001000ULL
- #define EFI_MEMORY_RP 0x0000000000002000ULL
- #define EFI_MEMORY_XP 0x0000000000004000ULL
- #define EFI_MEMORY_RUNTIME 0x8000000000000000ULL
- #define EFI_DEVICE_PATH_TYPE_MEDIA 0x04
- #define EFI_DEVICE_PATH_SUBTYPE_HARDDRIVE 0x01
- #define EFI_DEVICE_PATH_TYPE_END 0x7F
- #define EFI_DEVICE_PATH_SUBTYPE_END_ENTIRE 0xFF
- #define EFI_FILE_MODE_READ 0x0000000000000001ULL
- #define EFI_VARIABLE_NON_VOLATILE 0x00000001U
- #define EFI_VARIABLE_BOOTSERVICE_ACCESS 0x00000002U
- #define EFI_VARIABLE_RUNTIME_ACCESS 0x00000004U
- #define STARFORTH_VENDOR_GUID
- #define SF_VAR_BOOT_ARGS L"StarForthBootArgs"
- #define SF_VAR_REBOOT_TRIES L"StarForthRebootTries"
- #define SF_VAR_ZUSE_CERT

Typedefs

- typedef uint64_t UINTN

  uefi.h - UEFI definitions for StarKernel Minimal UEFI interface for bare-metal boot
- typedef int64_t INTN
- typedef uint8_t UINT8
- typedef uint16_t UINT16
- typedef uint32_t UINT32
- typedef uint64_t UINT64
- typedef int8_t INT8
- typedef int16_t INT16
- typedef int32_t INT32
- typedef int64_t INT64
- typedef uint8_t BOOLEAN
- typedef void VOID
- typedef uint16_t CHAR16
- typedef UINTN EFI_TPL
- typedef UINTN EFI_STATUS
- typedef void * EFI_HANDLE
- typedef UINT64 EFI_PHYSICAL_ADDRESS
- typedef EFI_STATUS(* EFI_TEXT_STRING) (struct _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *This, CHAR16 *String)
- typedef EFI_STATUS(* EFI_TEXT_RESET) (struct _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *This, BOOLEAN ExtendedVerification)
- typedef struct _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL
- typedef EFI_STATUS(* EFI_GET_MEMORY_MAP) (UINTN *MemoryMapSize, EFI_MEMORY_DESCRIPTOR *MemoryMap, UINTN *MapKey, UINTN *DescriptorSize, UINT32 *DescriptorVersion)
- typedef EFI_STATUS(* EFI_EXIT_BOOT_SERVICES) (EFI_HANDLE ImageHandle, UINTN MapKey)
- typedef EFI_STATUS(* EFI_ALLOCATE_POOL) (EFI_MEMORY_TYPE PoolType, UINTN Size, void **Buffer)
- typedef EFI_STATUS(* EFI_FREE_POOL) (void *Buffer)

- typedef EFI_STATUS(∗ EFI_ALLOCATE_PAGES) (EFI_ALLOCATE_TYPE Type, EFI_MEMORY_TYPE MemoryType, UINTN Pages, EFI_PHYSICAL_ADDRESS ∗Memory)
- typedef EFI_STATUS(∗ EFI_FREE_PAGES) (EFI_PHYSICAL_ADDRESS Memory, UINTN Pages)
- typedef EFI_STATUS(∗ EFI_HANDLE_PROTOCOL) (EFI_HANDLE Handle, EFI_GUID ∗Protocol, void ∗∗Interface)
- typedef EFI_STATUS(∗ EFI_STALL) (UINTN Microseconds)
- typedef EFI_STATUS(∗ EFI_LOCATE_HANDLE) (EFI_LOCATE_SEARCH_TYPE SearchType, EFI_GUID ∗Protocol, void ∗SearchKey, UINTN ∗BufferSize, EFI_HANDLE ∗Buffer)
- typedef EFI_STATUS(∗ EFI_LOCATE_PROTOCOL) (EFI_GUID ∗Protocol, void ∗Registration, void ∗∗Interface)
- typedef EFI_STATUS(∗ EFI_LOCATE_DEVICE_PATH) (EFI_GUID ∗Protocol, EFI_DEVICE_PATH_PROTOCOL ∗∗DevicePath, EFI_HANDLE ∗Device)
- typedef EFI_TPL(∗ EFI_RAISE_TPL) (EFI_TPL NewTpl)
- typedef void(∗ EFI_RESTORE_TPL) (EFI_TPL OldTpl)
- typedef struct _EFI_BOOT_SERVICES EFI_BOOT_SERVICES
- typedef struct _EFI_RUNTIME_SERVICES EFI_RUNTIME_SERVICES
- typedef struct _EFI_SYSTEM_TABLE EFI_SYSTEM_TABLE
- typedef EFI_STATUS(∗ EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE) (struct _EFI_GRAPHICS_OUTPUT_PROTO ∗This, UINT32 ModeNumber, UINTN ∗SizeOfInfo, EFI_GRAPHICS_OUTPUT_MODE_INFORMATION ∗∗Info)
- typedef EFI_STATUS(∗ EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE) (struct _EFI_GRAPHICS_OUTPUT_PROTOCO ∗This, UINT32 ModeNumber)
- typedef struct _EFI_GRAPHICS_OUTPUT_PROTOCOL EFI_GRAPHICS_OUTPUT_PROTOCOL
- typedef EFI_STATUS(∗ EFI_BLOCK_READ) (struct _EFI_BLOCK_IO_PROTOCOL ∗This, UINT32 MediaId, UINT64 Lba, UINTN BufferSize, void ∗Buffer)
- typedef struct _EFI_BLOCK_IO_PROTOCOL EFI_BLOCK_IO_PROTOCOL
- typedef EFI_STATUS(∗ EFI_FILE_OPEN) (struct _EFI_FILE_PROTOCOL ∗This, struct _EFI_FILE_PROTOCOL ∗∗NewHandle, CHAR16 ∗FileName, UINT64 OpenMode, UINT64 Attributes)
- typedef EFI_STATUS(∗ EFI_FILE_CLOSE) (struct _EFI_FILE_PROTOCOL ∗This)
- typedef EFI_STATUS(∗ EFI_FILE_READ) (struct _EFI_FILE_PROTOCOL ∗This, UINTN ∗BufferSize, VOID ∗Buffer)
- typedef EFI_STATUS(∗ EFI_FILE_GET_INFO) (struct _EFI_FILE_PROTOCOL ∗This, EFI_GUID ∗InformationType, UINTN ∗BufferSize, VOID ∗Buffer)
- typedef struct _EFI_FILE_PROTOCOL EFI_FILE_PROTOCOL
- typedef EFI_STATUS(∗ EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME) (struct _EFI_SIMPLE_FILE_SYSTEM_ ∗This, EFI_FILE_PROTOCOL ∗∗Root)
- typedef struct _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL EFI_SIMPLE_FILE_SYSTEM_PROTOCOL
- typedef EFI_STATUS(∗ EFI_GET_VARIABLE) (CHAR16 ∗VariableName, EFI_GUID ∗VendorGuid, UINT32 ∗Attributes, UINTN ∗DataSize, void ∗Data)
- typedef EFI_STATUS(∗ EFI_SET_VARIABLE) (CHAR16 ∗VariableName, EFI_GUID ∗VendorGuid, UINT32 Attributes, UINTN DataSize, void ∗Data)
- typedef void(∗ EFI_RESET_SYSTEM) (EFI_RESET_TYPE ResetType, EFI_STATUS ResetStatus, UINTN DataSize, void ∗ResetData)

Enumerations

- enum EFI_ALLOCATE_TYPE { AllocateAnyPages , AllocateMaxAddress , AllocateAddress , MaxAllocateType }
- enum EFI_MEMORY_TYPE { EfiReservedMemoryType , EfiLoaderCode , EfiLoaderData , EfiBootServicesCode , EfiBootServicesData , EfiRuntimeServicesCode , EfiRuntimeServicesData , EfiConventionalMemory , EfiUnusableMemory , EfiACPIReclaimMemory , EfiACPIMemoryNVS , EfiMemoryMappedIO , EfiMemoryMappedIOPortSpace , EfiPalCode , EfiPersistentMemory , EfiMaxMemoryType }
- enum EFI_LOCATE_SEARCH_TYPE { AllHandles , ByRegisterNotify , ByProtocol }

- enum EFI_GRAPHICS_PIXEL_FORMAT { PixelRedGreenBlueReserved8BitPerColor , PixelBlueGreenRedReserved8BitPerColor , PixelBitMask , PixelBltOnly , PixelFormatMax }
- enum EFI_RESET_TYPE { EfiResetCold , EfiResetWarm , EfiResetShutdown }

Variables

- static const EFI_GUID EFI_ACPI_20_TABLE_GUID = {0x8868e871,0xe4f1,0x11d3,{0xbc,0x22,0x00,0x80,0xc7,0x3c,0x88,0x8
- static const EFI_GUID EFI_ACPI_TABLE_GUID = {0xeb9d2d30,0x2d88,0x11d3,{0x9a,0x16,0x00,0x90,0x27,0x3f,0xc1,0x4d}}
- static const EFI_GUID EFI_DTB_TABLE_GUID
- static const EFI_GUID EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID
- static const EFI_GUID EFI_LOADED_IMAGE_PROTOCOL_GUID
- static const EFI_GUID EFI_DEVICE_PATH_PROTOCOL_GUID
- static const EFI_GUID EFI_BLOCK_IO_PROTOCOL_GUID
- static const EFI_GUID EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID
- static const EFI_GUID EFI_FILE_INFO_GUID

7.69.1 Macro Definition Documentation

7.69.1.1 EFI_ABORTED

```
#define EFI_ABORTED (21 | (1ULL << 63))
```

7.69.1.2 EFI_BAD_BUFFER_SIZE

```
#define EFI_BAD_BUFFER_SIZE (4 | (1ULL << 63))
```

7.69.1.3 EFI_BUFFER_TOO_SMALL

```
#define EFI_BUFFER_TOO_SMALL (5 | (1ULL << 63))
```

7.69.1.4 EFI_DEVICE_ERROR

```
#define EFI_DEVICE_ERROR (7 | (1ULL << 63))
```

7.69.1.5 EFI_DEVICE_PATH_SUBTYPE_END_ENTIRE

```
#define EFI_DEVICE_PATH_SUBTYPE_END_ENTIRE 0xFF
```

7.69.1.6 EFI_DEVICE_PATH_SUBTYPE_HARDDRIVE

```
#define EFI_DEVICE_PATH_SUBTYPE_HARDDRIVE 0x01
```

7.69.1.7 EFI_DEVICE_PATH_TYPE_END

```
#define EFI_DEVICE_PATH_TYPE_END 0x7F
```

7.69.1.8 EFI_DEVICE_PATH_TYPE_MEDIA

```
#define EFI_DEVICE_PATH_TYPE_MEDIA 0x04
```

7.69.1.9 EFI_FILE_MODE_READ

```
#define EFI_FILE_MODE_READ 0x0000000000000001ULL
```

7.69.1.10 EFI_INVALID_PARAMETER

```
#define EFI_INVALID_PARAMETER (2 | (1ULL << 63))
```

7.69.1.11 EFI_LOAD_ERROR

```
#define EFI_LOAD_ERROR (1 | (1ULL << 63))
```

7.69.1.12 EFI_MEMORY_RP

```
#define EFI_MEMORY_RP 0x0000000000002000ULL
```

7.69.1.13 EFI_MEMORY_RUNTIME

```
#define EFI_MEMORY_RUNTIME 0x8000000000000000ULL
```

7.69.1.14 EFI_MEMORY_UC

```
#define EFI_MEMORY_UC 0x0000000000000001ULL
```

7.69.1.15 EFI_MEMORY_UCE

```
#define EFI_MEMORY_UCE 0x0000000000000010ULL
```

7.69.1.16 EFI_MEMORY_WB

```
#define EFI_MEMORY_WB 0x0000000000000008ULL
```

7.69.1.17 EFI_MEMORY_WC

```
#define EFI_MEMORY_WC 0x0000000000000002ULL
```

7.69.1.18 EFI_MEMORY_WP

```
#define EFI_MEMORY_WP 0x0000000000001000ULL
```

7.69.1.19 EFI_MEMORY_WT

```
#define EFI_MEMORY_WT 0x0000000000000004ULL
```

7.69.1.20 EFI_MEMORY_XP

```
#define EFI_MEMORY_XP 0x0000000000004000ULL
```

7.69.1.21 EFI_NOT_FOUND

```
#define EFI_NOT_FOUND (14 | (1ULL << 63))
```

7.69.1.22 EFI_NOT_READY

```
#define EFI_NOT_READY (6 | (1ULL << 63))
```

7.69.1.23 EFI_OUT_OF_RESOURCES

```
#define EFI_OUT_OF_RESOURCES (9 | (1ULL << 63))
```

7.69.1.24 EFI_SUCCESS

```
#define EFI_SUCCESS 0
```

7.69.1.25 EFI_UNSUPPORTED

```
#define EFI_UNSUPPORTED (3 | (1ULL << 63))
```

7.69.1.26 EFI_VARIABLE_BOOTSERVICE_ACCESS

```
#define EFI_VARIABLE_BOOTSERVICE_ACCESS 0x00000002U
```

7.69.1.27 EFI_VARIABLE_NON_VOLATILE

```
#define EFI_VARIABLE_NON_VOLATILE 0x00000001U
```

7.69.1.28 EFI_VARIABLE_RUNTIME_ACCESS

```
#define EFI_VARIABLE_RUNTIME_ACCESS 0x00000004U
```

7.69.1.29 EFI_WRITE_PROTECTED

```
#define EFI_WRITE_PROTECTED (8 | (1ULL << 63))
```

7.69.1.30 EFIAPI

```
#define EFIAPI
```

7.69.1.31 FALSE

```
#define FALSE ((BOOLEAN)0)
```

7.69.1.32 SF_VAR_BOOT_ARGS

```
#define SF_VAR_BOOT_ARGS L"StarForthBootArgs"
```

7.69.1.33 SF_VAR_REBOOT_TRIES

```
#define SF_VAR_REBOOT_TRIES L"StarForthRebootTries"
```

7.69.1.34 SF_VAR_ZUSE_CERT

```
#define SF_VAR_ZUSE_CERT
```

Value:

```
L"StarForthZuseCert" /* 64 bytes: 32-byte Ed25519
* seed || 32-byte pubkey.
* Written exactly once
* (Phase 8 first-boot mint,
* see FABRIC-2.md) --
* presence means the fuse
* is already blown. */
```

7.69.1.35 STARFORTH_VENDOR_GUID

```
#define STARFORTH_VENDOR_GUID
```

Value:

```
    {0xb7e42c1aU, 0x4f3dU, 0x4e8bU, \
     {0x9cU, 0x5aU, 0x1dU, 0x2fU, 0x6aU, 0x8bU, 0x3eU, 0x70U}}
```

7.69.1.36 TPL_APPLICATION

```
#define TPL_APPLICATION 4
```

7.69.1.37 TPL_CALLBACK

```
#define TPL_CALLBACK 8
```

7.69.1.38 TPL_HIGH_LEVEL

```
#define TPL_HIGH_LEVEL 31
```

7.69.1.39 TPL_NOTIFY

```
#define TPL_NOTIFY 16
```

7.69.1.40 TRUE

```
#define TRUE ((BOOLEAN)1)
```

7.69.2 Typedef Documentation

7.69.2.1 BOOLEAN

```
typedef uint8_t BOOLEAN
```

7.69.2.2 CHAR16

```
typedef uint16_t CHAR16
```

7.69.2.3 EFI_ALLOCATE_PAGES

```
typedef EFI_STATUS( * EFI_ALLOCATE_PAGES) (EFI_ALLOCATE_TYPE Type, EFI_MEMORY_TYPE MemoryType,
UINTN Pages, EFI_PHYSICAL_ADDRESS *Memory)
```

7.69.2.4 EFI_ALLOCATE_POOL

```
typedef EFI_STATUS( * EFI_ALLOCATE_POOL) (EFI_MEMORY_TYPE PoolType, UINTN Size, void **Buffer)
```

7.69.2.5 EFI_BLOCK_IO_PROTOCOL

```
typedef struct _EFI_BLOCK_IO_PROTOCOL EFI_BLOCK_IO_PROTOCOL
```

7.69.2.6 EFI_BLOCK_READ

```
typedef EFI_STATUS( * EFI_BLOCK_READ) (struct _EFI_BLOCK_IO_PROTOCOL *This, UINT32 MediaId,
UINT64 Lba, UINTN BufferSize, void *Buffer)
```

7.69.2.7 EFI_BOOT_SERVICES

```
typedef struct _EFI_BOOT_SERVICES EFI_BOOT_SERVICES
```

7.69.2.8 EFI_EXIT_BOOT_SERVICES

```
typedef EFI_STATUS( * EFI_EXIT_BOOT_SERVICES) (EFI_HANDLE ImageHandle, UINTN MapKey)
```

7.69.2.9 EFI_FILE_CLOSE

```
typedef EFI_STATUS( * EFI_FILE_CLOSE) (struct _EFI_FILE_PROTOCOL *This)
```

7.69.2.10 EFI_FILE_GET_INFO

```
typedef EFI_STATUS( * EFI_FILE_GET_INFO) (struct _EFI_FILE_PROTOCOL *This, EFI_GUID *Information↩
Type, UINTN *BufferSize, VOID *Buffer)
```

7.69.2.11 EFI_FILE_OPEN

```
typedef EFI_STATUS( * EFI_FILE_OPEN) (struct _EFI_FILE_PROTOCOL *This, struct _EFI_FILE_PROTOCOL
**NewHandle, CHAR16 *FileName, UINT64 OpenMode, UINT64 Attributes)
```

7.69.2.12 EFI_FILE_PROTOCOL

```
typedef struct _EFI_FILE_PROTOCOL EFI_FILE_PROTOCOL
```

7.69.2.13 EFI_FILE_READ

```
typedef EFI_STATUS( * EFI_FILE_READ) (struct _EFI_FILE_PROTOCOL *This, UINTN *BufferSize, VOID
*Buffer)
```

7.69.2.14 EFI_FREE_PAGES

```
typedef EFI_STATUS( * EFI_FREE_PAGES) (EFI_PHYSICAL_ADDRESS Memory, UINTN Pages)
```

7.69.2.15 EFI_FREE_POOL

```
typedef EFI_STATUS( * EFI_FREE_POOL) (void *Buffer)
```

7.69.2.16 EFI_GET_MEMORY_MAP

```
typedef EFI_STATUS( * EFI_GET_MEMORY_MAP) (UINTN *MemoryMapSize, EFI_MEMORY_DESCRIPTOR *Memory↩
Map, UINTN *MapKey, UINTN *DescriptorSize, UINT32 *DescriptorVersion)
```

7.69.2.17 EFI_GET_VARIABLE

```
typedef EFI_STATUS( * EFI_GET_VARIABLE) (CHAR16 *VariableName, EFI_GUID *VendorGuid, UINT32
*Attributes, UINTN *DataSize, void *Data)
```

7.69.2.18 EFI_GRAPHICS_OUTPUT_PROTOCOL

```
typedef struct _EFI_GRAPHICS_OUTPUT_PROTOCOL EFI_GRAPHICS_OUTPUT_PROTOCOL
```

7.69.2.19 EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE

```
typedef EFI_STATUS( * EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE) (struct _EFI_GRAPHICS_OUTPUT_PROTOCOL
*This, UINT32 ModeNumber, UINTN *SizeOfInfo, EFI_GRAPHICS_OUTPUT_MODE_INFORMATION **Info)
```

7.69.2.20 EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE

```
typedef EFI_STATUS( * EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE) (struct _EFI_GRAPHICS_OUTPUT_PROTOCOL
*This, UINT32 ModeNumber)
```

7.69.2.21 EFI_HANDLE

```
typedef void* EFI_HANDLE
```

7.69.2.22 EFI_HANDLE_PROTOCOL

```
typedef EFI_STATUS( * EFI_HANDLE_PROTOCOL) (EFI_HANDLE Handle, EFI_GUID *Protocol, void **Interface)
```

7.69.2.23 EFI_LOCATE_DEVICE_PATH

```
typedef EFI_STATUS( * EFI_LOCATE_DEVICE_PATH) (EFI_GUID *Protocol, EFI_DEVICE_PATH_PROTOCOL
**DevicePath, EFI_HANDLE *Device)
```

7.69.2.24 EFI_LOCATE_HANDLE

```
typedef EFI_STATUS( * EFI_LOCATE_HANDLE) (EFI_LOCATE_SEARCH_TYPE SearchType, EFI_GUID *Protocol,
void *SearchKey, UINTN *BufferSize, EFI_HANDLE *Buffer)
```

7.69.2.25 EFI_LOCATE_PROTOCOL

```
typedef EFI_STATUS( * EFI_LOCATE_PROTOCOL) (EFI_GUID *Protocol, void *Registration, void **Interface)
```

7.69.2.26 EFI_PHYSICAL_ADDRESS

```
typedef UINT64 EFI_PHYSICAL_ADDRESS
```

7.69.2.27 EFI_RAISE_TPL

```
typedef EFI_TPL( * EFI_RAISE_TPL) (EFI_TPL NewTpl)
```

7.69.2.28 EFI_RESET_SYSTEM

```
typedef void( * EFI_RESET_SYSTEM) (EFI_RESET_TYPE ResetType, EFI_STATUS ResetStatus, UINTN
DataSize, void *ResetData)
```

7.69.2.29 EFI_RESTORE_TPL

```
typedef void( * EFI_RESTORE_TPL) (EFI_TPL OldTpl)
```

7.69.2.30 EFI_RUNTIME_SERVICES

```
typedef struct _EFI_RUNTIME_SERVICES EFI_RUNTIME_SERVICES
```

7.69.2.31 EFI_SET_VARIABLE

```
typedef EFI_STATUS( * EFI_SET_VARIABLE) (CHAR16 *VariableName, EFI_GUID *VendorGuid, UINT32
Attributes, UINTN DataSize, void *Data)
```

7.69.2.32 EFI_SIMPLE_FILE_SYSTEM_PROTOCOL

```
typedef struct _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL EFI_SIMPLE_FILE_SYSTEM_PROTOCOL
```

7.69.2.33 EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME

```
typedef EFI_STATUS( * EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME) (struct _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL
*This, EFI_FILE_PROTOCOL **Root)
```

7.69.2.34 EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL

```
typedef struct _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL
```

7.69.2.35 EFI_STALL

```
typedef EFI_STATUS( * EFI_STALL) (UINTN Microseconds)
```

7.69.2.36 EFI_STATUS

```
typedef UINTN EFI_STATUS
```

7.69.2.37 EFI_SYSTEM_TABLE

```
typedef struct _EFI_SYSTEM_TABLE EFI_SYSTEM_TABLE
```

7.69.2.38 EFI_TEXT_RESET

```
typedef EFI_STATUS( * EFI_TEXT_RESET) (struct _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *This, BOOLEAN
ExtendedVerification)
```

7.69.2.39 EFI_TEXT_STRING

```
typedef EFI_STATUS( * EFI_TEXT_STRING) (struct _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *This, CHAR16
*String)
```

7.69.2.40 EFI_TPL

```
typedef UINTN EFI_TPL
```

7.69.2.41 INT16

```
typedef int16_t INT16
```

7.69.2.42 INT32

```
typedef int32_t INT32
```

7.69.2.43 INT64

```
typedef int64_t INT64
```

7.69.2.44 INT8

```
typedef int8_t INT8
```

7.69.2.45 INTN

```
typedef int64_t INTN
```

7.69.2.46 UINT16

```
typedef uint16_t UINT16
```

7.69.2.47 UINT32

```
typedef uint32_t UINT32
```

7.69.2.48 UINT64

```
typedef uint64_t UINT64
```

7.69.2.49 UINT8

```
typedef uint8_t UINT8
```

7.69.2.50 UINTN

```
typedef uint64_t UINTN
```

uefi.h - UEFI definitions for StarKernel Minimal UEFI interface for bare-metal boot

7.69.2.51 VOID

```
typedef void VOID
```

7.69.3 Enumeration Type Documentation

7.69.3.1 EFI_ALLOCATE_TYPE

```
enum EFI_ALLOCATE_TYPE
```

Enumerator

AllocateAnyPages	
AllocateMaxAddress	
AllocateAddress	
MaxAllocateType	

7.69.3.2 EFI_GRAPHICS_PIXEL_FORMAT

enum EFI_GRAPHICS_PIXEL_FORMAT

Enumerator

PixelRedGreenBlueReserved8BitPerColor	
PixelBlueGreenRedReserved8BitPerColor	
PixelBitMask	
PixelBltOnly	
PixelFormatMax	

7.69.3.3 EFI_LOCATE_SEARCH_TYPE

enum EFI_LOCATE_SEARCH_TYPE

Enumerator

AllHandles	
ByRegisterNotify	
ByProtocol	

7.69.3.4 EFI_MEMORY_TYPE

enum EFI_MEMORY_TYPE

Enumerator

EfiReservedMemoryType	
EfiLoaderCode	
EfiLoaderData	
EfiBootServicesCode	
EfiBootServicesData	
EfiRuntimeServicesCode	
EfiRuntimeServicesData	
EfiConventionalMemory	
EfiUnusableMemory	
EfiACPIReclaimMemory	
EfiACPIMemoryNVS	
EfiMemoryMappedIO	
EfiMemoryMappedIOPortSpace	
EfiPalCode	
EfiPersistentMemory	
EfiMaxMemoryType	

7.69.3.5 EFI_RESET_TYPE

```
enum EFI_RESET_TYPE
```

Enumerator

EfiResetCold	
EfiResetWarm	
EfiResetShutdown	

7.69.4 Variable Documentation

7.69.4.1 EFI_ACPI_20_TABLE_GUID

```
const EFI_GUID EFI_ACPI_20_TABLE_GUID = {0x8868e871,0xe4f1,0x11d3,{0xbc,0x22,0x00,0x80,0xc7,0x3c,0x88,0x81}}
[static]
```

Referenced by efi_main().

7.69.4.2 EFI_ACPI_TABLE_GUID

```
const EFI_GUID EFI_ACPI_TABLE_GUID = {0xeb9d2d30,0x2d88,0x11d3,{0x9a,0x16,0x00,0x90,0x27,0x3f,0xc1,0x4d}}
[static]
```

Referenced by efi_main().

7.69.4.3 EFI_BLOCK_IO_PROTOCOL_GUID

```
const EFI_GUID EFI_BLOCK_IO_PROTOCOL_GUID  [static]
```

Initial value:

```
=
    {0x964e5b21,0x6459,0x11d2, {0x8e,0x39,0x00,0xa0,0xc9,0x69,0x72,0x3b}}
```

7.69.4.4 EFI_DEVICE_PATH_PROTOCOL_GUID

```
const EFI_GUID EFI_DEVICE_PATH_PROTOCOL_GUID  [static]
```

Initial value:

```
=
    {0x09576e91,0x6d3f,0x11d2, {0x8e,0x39,0x00,0xa0,0xc9,0x69,0x72,0x3b}}
```

7.69.4.5 EFI_DTB_TABLE_GUID

```
const EFI_GUID EFI_DTB_TABLE_GUID  [static]
```

Initial value:

```
= {
    0xb1b621d5, 0xf19c, 0x41a5, {0x83, 0x0b, 0xd9, 0x15, 0x2c, 0x69, 0xaa, 0xe0}
}
```

Referenced by efi_main().

7.69.4.6 EFI_FILE_INFO_GUID

```
const EFI_GUID EFI_FILE_INFO_GUID  [static]
```

Initial value:

```
=
    {0x09576e92,0x6d3f,0x11d2, {0x8e,0x39,0x00,0xa0,0xc9,0x69,0x72,0x3b}}
```

Referenced by load_kernel_from_esp().

7.69.4.7 EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID

```
const EFI_GUID EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID  [static]
```

Initial value:

```
=
    {0x9042a9de, 0x23dc, 0x4a38, {0x96, 0xfb, 0x7a, 0xde, 0xd0, 0x80, 0x51, 0x6a}}
```

Referenced by efi_main().

7.69.4.8 EFI_LOADED_IMAGE_PROTOCOL_GUID

```
const EFI_GUID EFI_LOADED_IMAGE_PROTOCOL_GUID  [static]
```

Initial value:

```
=
    {0x5B1B31A1,0x9562,0x11d2, {0x8E,0x3F,0x00,0xA0,0xC9,0x69,0x72,0x3B}}
```

Referenced by efi_main(), load_cfg_from_esp(), and load_kernel_from_esp().

7.69.4.9 EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID

```
const EFI_GUID EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID  [static]
```

Initial value:

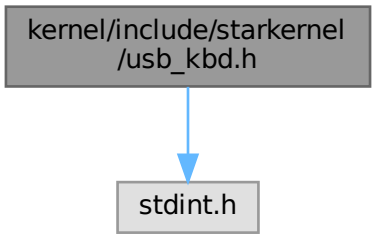
```
=
    {0x0964e5b22,0x6459,0x11d2, {0x8e,0x39,0x00,0xa0,0xc9,0x69,0x72,0x3b}}
```

Referenced by load_cfg_from_esp(), and load_kernel_from_esp().

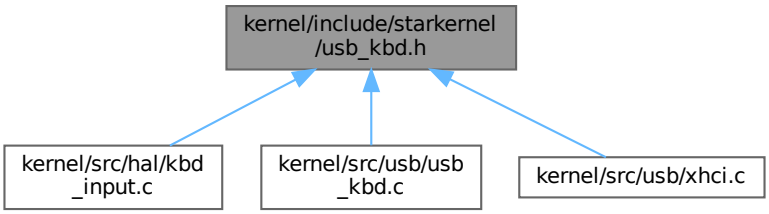
7.70 kernel/include/starkernel/usb_kbd.h File Reference

```
#include <stdint.h>
```

Include dependency graph for usb_kbd.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define USB_KBD_LED_NUM 0x01u
- #define USB_KBD_LED_CAPS 0x02u
- #define USB_KBD_LED_SCROLL 0x04u

Functions

- void usb_kbd_report (uint8_t prev[8], const uint8_t *report, uint32_t len)

  usb_kbd.h - a USB boot-protocol keyboard's keys, as key events (2026-10-14, boards/BRINGUP.md A3; usb_kbd.c).
- void usb_kbd_forget (uint8_t prev[8])
- int usb_kbd_pop_event (uint16_t *keycode, int *pressed)
- void usb_kbd_set_leds (uint8_t leds)
- uint8_t usb_kbd_leds_wanted (void)

7.70.1 Macro Definition Documentation

7.70.1.1 USB_KBD_LED_CAPS

```
#define USB_KBD_LED_CAPS 0x02u
```

7.70.1.2 USB_KBD_LED_NUM

```
#define USB_KBD_LED_NUM 0x01u
```

7.70.1.3 USB_KBD_LED_SCROLL

```
#define USB_KBD_LED_SCROLL 0x04u
```

7.70.2 Function Documentation

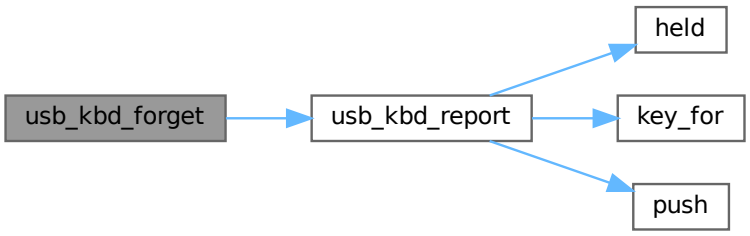
7.70.2.1 usb_kbd_forget()

```
void usb_kbd_forget (
            uint8_t prev[8] )
```

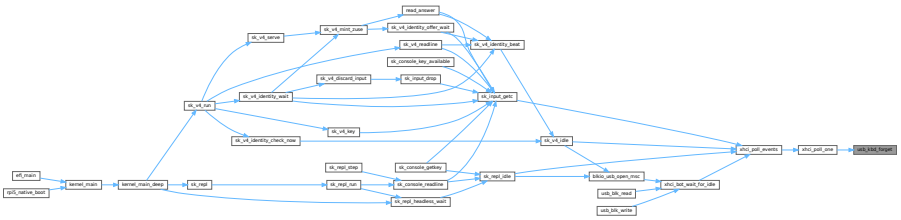
References usb_kbd_report().

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



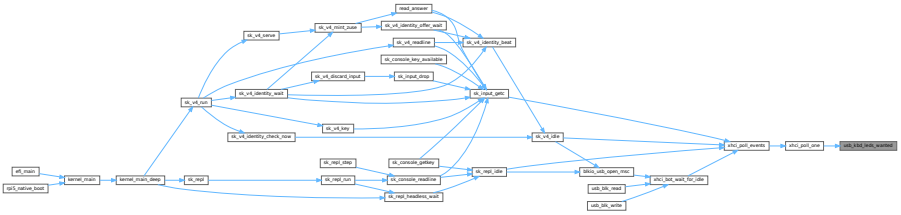
7.70.2.2 usb_kbd_leds_wanted()

```
uint8_t usb_kbd_leds_wanted (
            void  )
```

References g_leds_want.

Referenced by xhci_poll_one().

Here is the caller graph for this function:



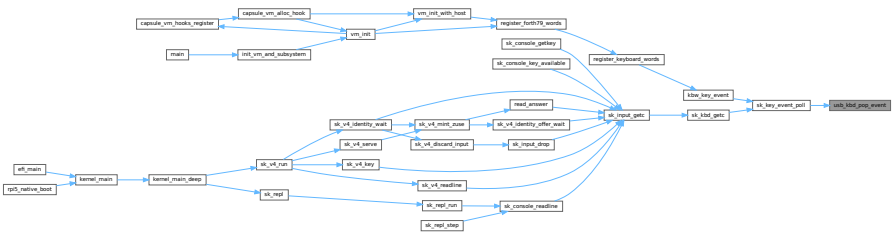
7.70.2.3 usb_kbd_pop_event()

```
int usb_kbd_pop_event (
            uint16_t * keycode,
            int * pressed )
```

References g_count, g_events, g_head, KBD_EVENTS, keycode, and pressed.

Referenced by sk_key_event_poll().

Here is the caller graph for this function:



7.70.2.4 usb_kbd_report()

```
void usb_kbd_report (
            uint8_t prev[8],
            const uint8_t * report,
            uint32_t len )
```

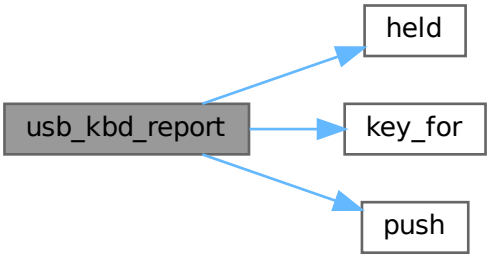
usb_kbd.h - a USB boot-protocol keyboard's keys, as key events (2026-10-14, boards/BRINGUP.md A3; usb_kbd.c).

The xHCI driver hands each 8-byte boot report here; the keys pressed and released since the report before come out as Linux input keycodes, the namespace every keyboard in this kernel already speaks (sk_key_event_poll(), keyboard_words.c).

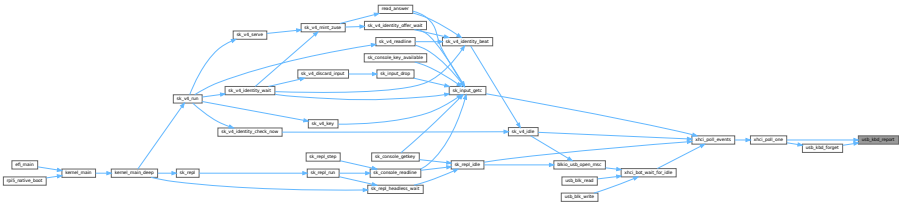
References held(), key_for(), modifier_key, and push().

Referenced by usb_kbd_forget(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



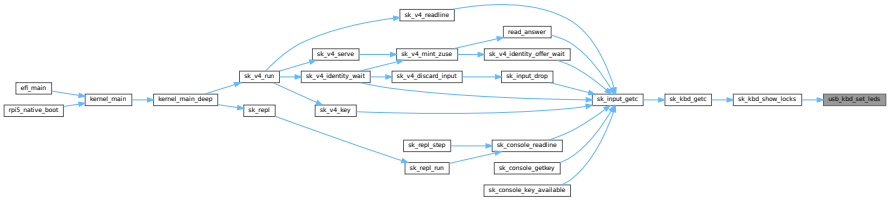
7.70.2.5 usb_kbd_set_leds()

```
void usb_kbd_set_leds (
            uint8_t leds )
```

References g_leds_want.

Referenced by sk_kbd_show_locks().

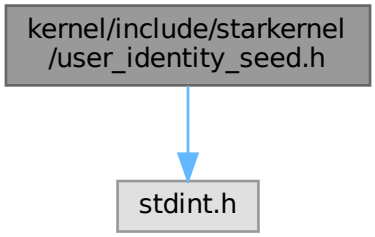
Here is the caller graph for this function:



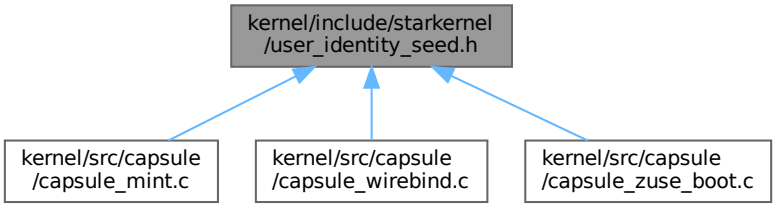
7.71 kernel/include/starkernel/user_identity_seed.h File Reference

```
#include <stdint.h>
```

Include dependency graph for user_identity_seed.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct user_identity_seed_t

Macros

- #define USER_IDENTITY_SEED_MAGIC ((uint32_t)'U' | ((uint32_t)'I' << 8) | ((uint32_t)'D' << 16) | ((uint32_t)'S' << 24))
- #define USER_IDENTITY_SEED_VERSION 2u
- #define USER_IDENTITY_FULL_NAME_MAX 64u
- #define USER_IDENTITY_USERNAME_MAX 32u
- #define USER_IDENTITY_EMAIL_MAX 64u
- #define USER_IDENTITY_PHONE_MAX 24u

Typedefs

- typedef char user_identity_seed_size_check[(sizeof(user_identity_seed_t)==4096) ? 1 :-1]

7.71.1 Macro Definition Documentation

7.71.1.1 USER_IDENTITY_EMAIL_MAX

```
#define USER_IDENTITY_EMAIL_MAX 64u
```

7.71.1.2 USER_IDENTITY_FULL_NAME_MAX

```
#define USER_IDENTITY_FULL_NAME_MAX 64u
```

7.71.1.3 USER_IDENTITY_PHONE_MAX

```
#define USER_IDENTITY_PHONE_MAX 24u
```

7.71.1.4 USER_IDENTITY_SEED_MAGIC

```
#define USER_IDENTITY_SEED_MAGIC  ((uint32_t)'U' | ((uint32_t)'I' << 8) | ((uint32_t)'D' <<
16) | ((uint32_t)'S' << 24))
```

7.71.1.5 USER_IDENTITY_SEED_VERSION

```
#define USER_IDENTITY_SEED_VERSION 2u
```

7.71.1.6 USER_IDENTITY_USERNAME_MAX

```
#define USER_IDENTITY_USERNAME_MAX 32u
```

7.71.2 Typedef Documentation

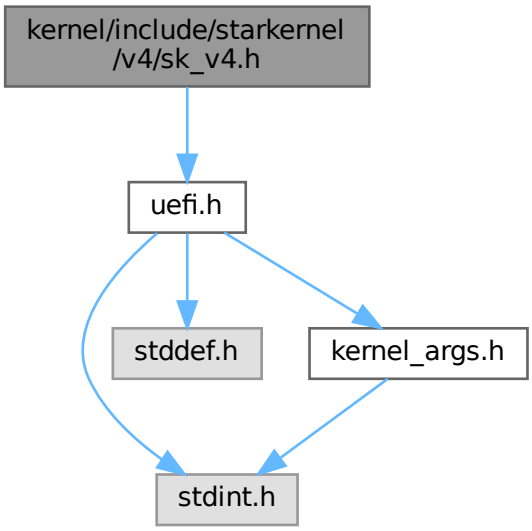
7.71.2.1 user_identity_seed_size_check

```
typedef char user_identity_seed_size_check[(sizeof(user_identity_seed_t)==4096) ?  1 :-1]
```

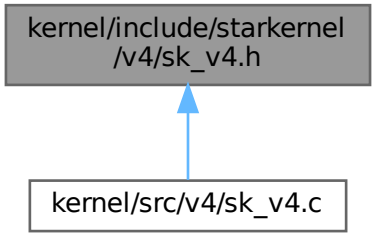
7.72 kernel/include/starkernel/v4/sk_v4.h File Reference

```
#include "uefi.h"
```

Include dependency graph for sk_v4.h:



This graph shows which files directly or indirectly include this file:



Functions

- void sk_v4_run (void(∗chain)(void), const BootDisk ∗boot_disk)

7.72.1 Function Documentation

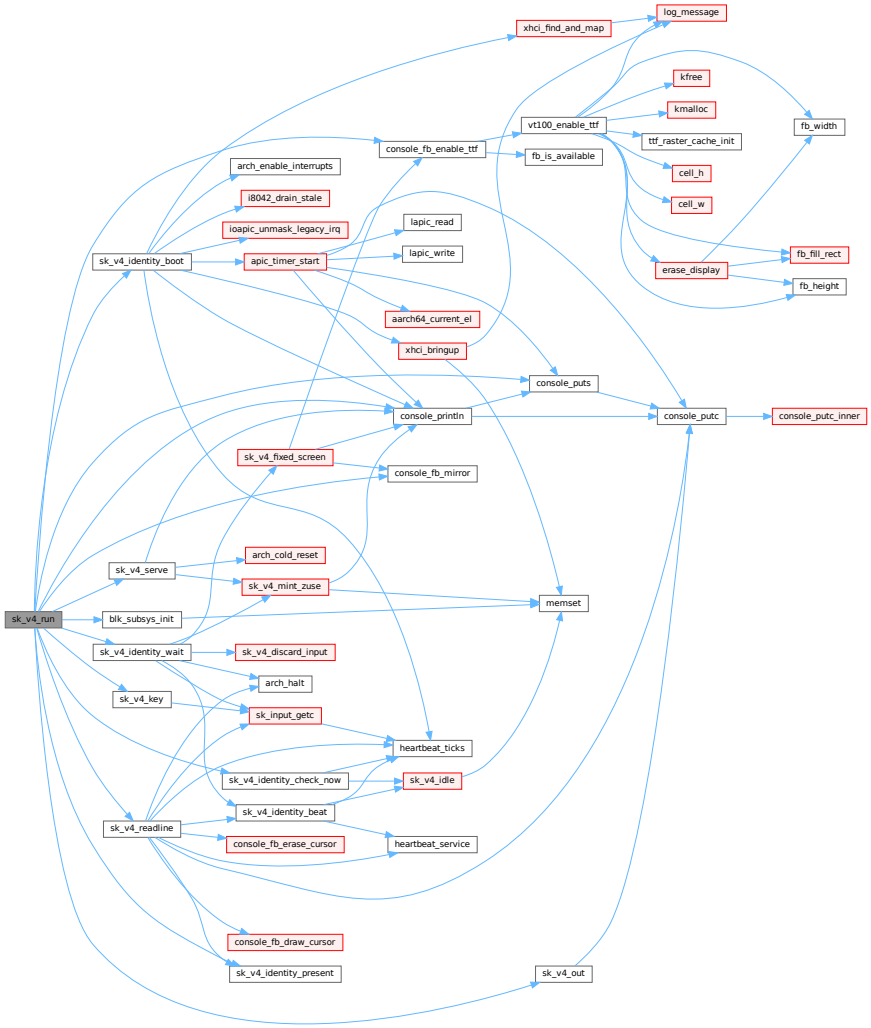
7.72.1.1 sk_v4_run()

```
void sk_v4_run (
            void(*)(void) chain,
            const BootDisk * boot_disk )
```

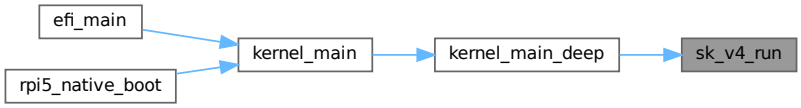
References blk_subsys_init(), console_fb_enable_ttf(), console_fb_mirror(), console_println(), console_puts(), sk_v4_es, sk_v4_heat, sk_v4_identity_boot(), sk_v4_identity_check_now(), sk_v4_identity_present(), sk_v4_identity_wait(), sk_v4_image, sk_v4_key(), sk_v4_node, sk_v4_out(), sk_v4_post_blk_ram, SK_V4_PROMPT, sk_v4_readline(), sk_v4_serve(), sk_v4_text, and sk_v4_words.

Referenced by kernel_main_deep().

Here is the call graph for this function:



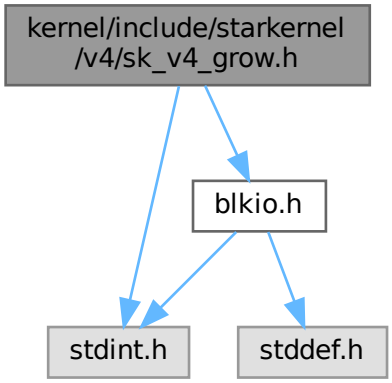
Here is the caller graph for this function:



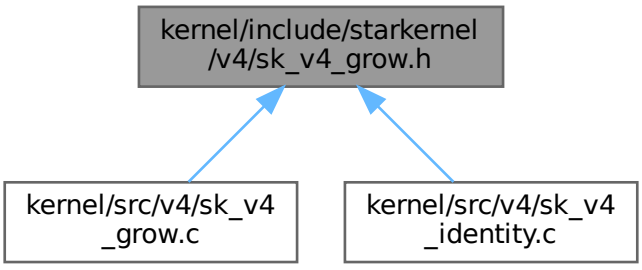
7.73 kernel/include/starkernel/v4/sk_v4_grow.h File Reference

```
#include <stdint.h>
#include "blkio.h"
```

Include dependency graph for sk_v4_grow.h:



This graph shows which files directly or indirectly include this file:



Functions

- uint64_t sk_v4_grow_cap_sectors (blkio_dev_t *art, uint64_t cap_sectors)
- int sk_v4_grow_artemis (blkio_dev_t *art)

7.73.1 Function Documentation

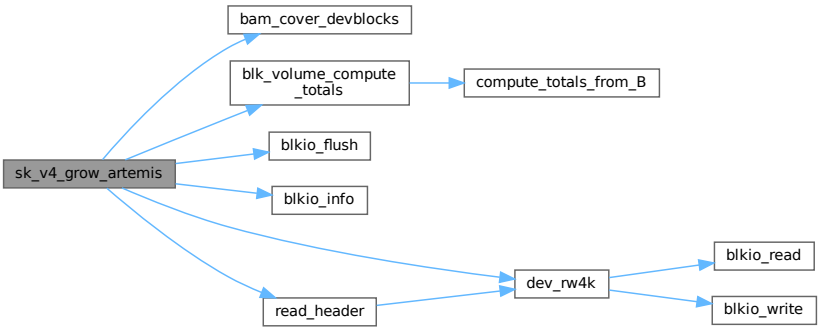
7.73.1.1 sk_v4_grow_artemis()

```
int sk_v4_grow_artemis (
            blkio_dev_t * art )
```

References bam_cover_devblocks(), blk_volume_compute_totals(), blkio_flush(), blkio_info(), BLKIO_OK, dev_rw4k(), g_hdr, g_move, read_header(), and blkio_info_t::total_blocks.

Referenced by sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

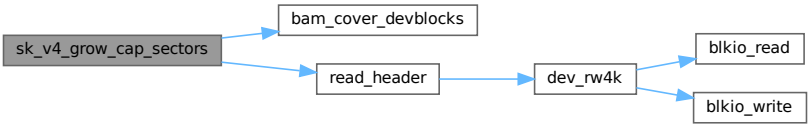
7.73.1.2 sk_v4_grow_cap_sectors()

```
uint64_t sk_v4_grow_cap_sectors (
            blkio_dev_t * art,
            uint64_t cap_sectors )
```

References bam_cover_devblocks(), g_hdr, and read_header().

Referenced by sk_v4_grow_stick().

Here is the call graph for this function:

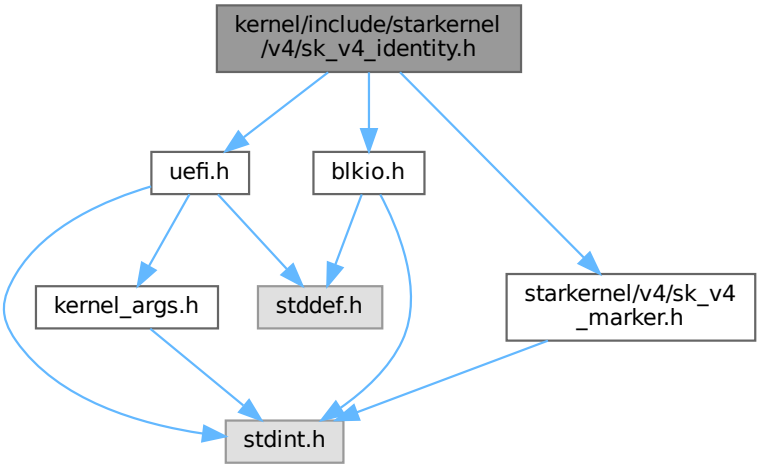


Here is the caller graph for this function:

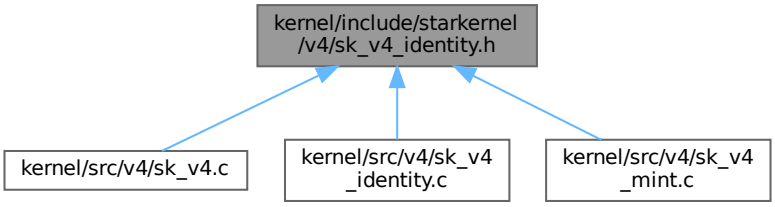
7.74 kernel/include/starkernel/v4/sk_v4_identity.h File Reference

```
#include "uefi.h"
#include "blkio.h"
#include "starkernel/v4/sk_v4_marker.h"
```

Include dependency graph for sk_v4_identity.h:



This graph shows which files directly or indirectly include this file:



Functions

- void sk_v4_identity_boot (const BootDisk *boot_disk)
- int sk_v4_identity_present (void)
- void sk_v4_identity_beat (void)
- void sk_v4_identity_check_now (void)
- int sk_v4_identity_offer_wait (blkio_dev_t **raw, uint64_t *bytes)
- int sk_v4_identity_stick_in (void)
- int sk_v4_identity_offered_present (void)
- void sk_v4_identity_release_offered (void)
- const sk_v4_marker_t * sk_v4_identity_marker (void)
- void sk_v4_identity_set_marker (const sk_v4_marker_t *m)
- void sk_v4_identity_recheck_offered (void)
- void sk_v4_identity_wait (void)

7.74.1 Function Documentation

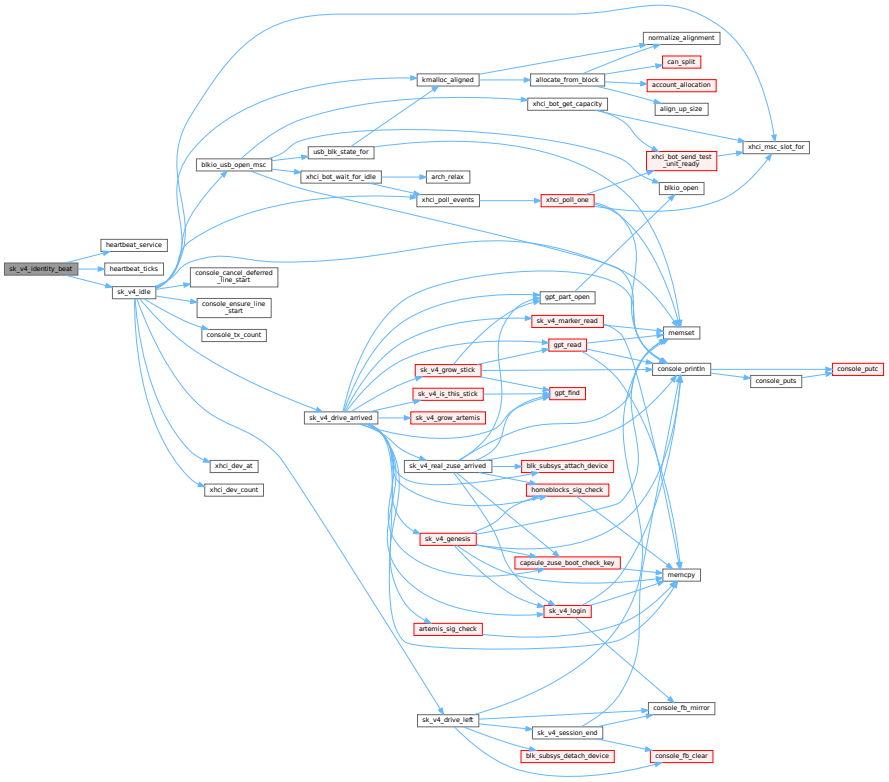
7.74.1.1 sk_v4_identity_beat()

```
void sk_v4_identity_beat (
            void  )
```

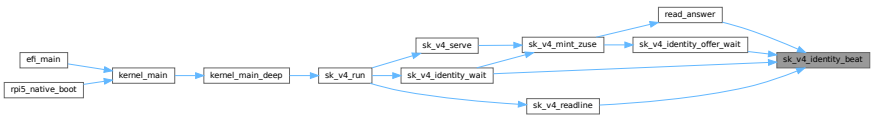
References g_last_beat, heartbeat_service(), heartbeat_ticks(), sk_v4_idle(), and SK_V4_IDLE_BEAT.

Referenced by read_answer(), sk_v4_identity_offer_wait(), sk_v4_identity_wait(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:



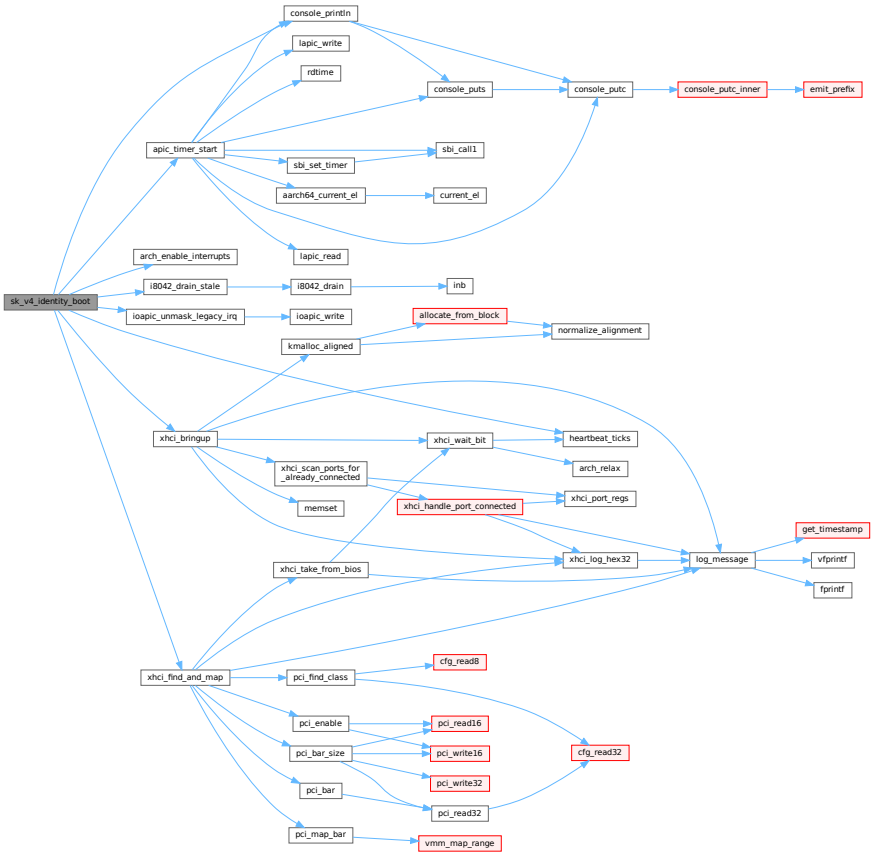
7.74.1.2 sk_v4_identity_boot()

```
void sk_v4_identity_boot (
            const BootDisk * boot_disk )
```

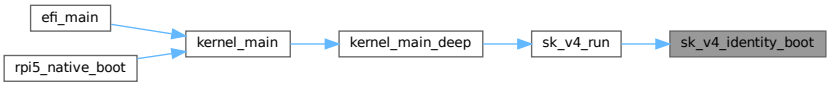
References apic_timer_start(), arch_enable_interrupts(), console_println(), g_boot_disk, g_last_beat, heartbeat_ticks(), i8042_drain_stale(), ioapic_unmask_legacy_irq(), xhci_bringup(), xhci_find_and_map(), and XHCI_MAX_CONTROLLERS.

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



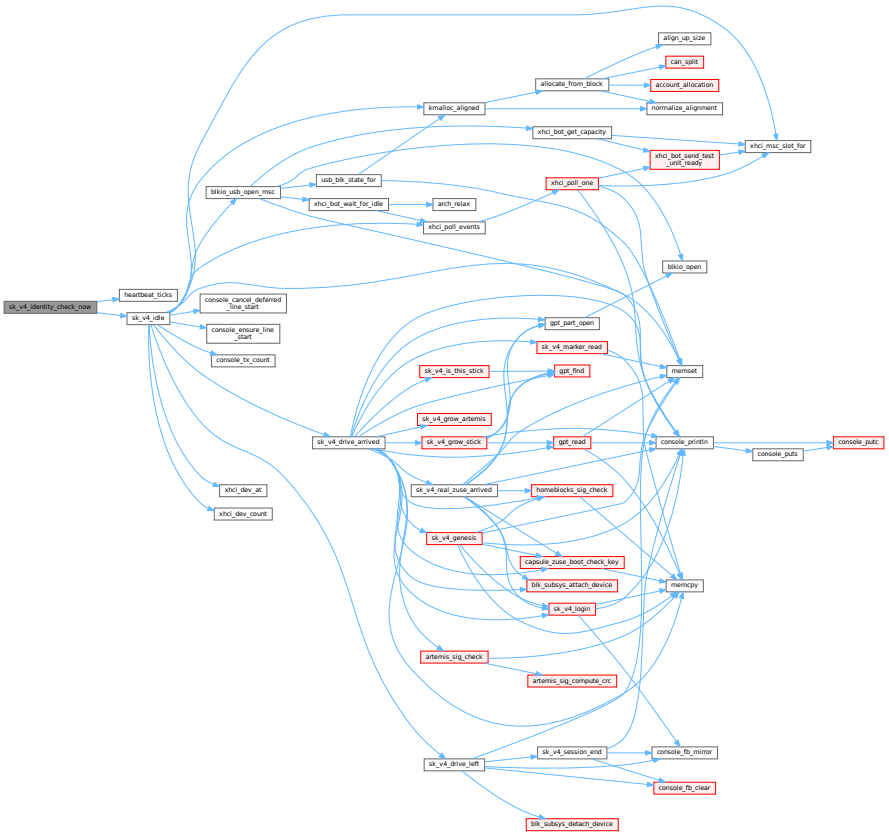
7.74.1.3 sk_v4_identity_check_now()

```
void sk_v4_identity_check_now (
            void  )
```

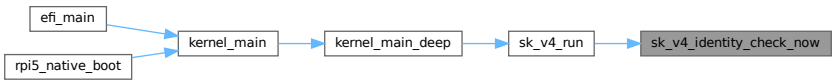
References g_last_beat, heartbeat_ticks(), and sk_v4_idle().

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.74.1.4 sk_v4_identity_marker()

```
const sk_v4_marker_t * sk_v4_identity_marker (
            void  )
```

References g_marker.

Referenced by sk_v4_mint_zuse().

Here is the caller graph for this function:

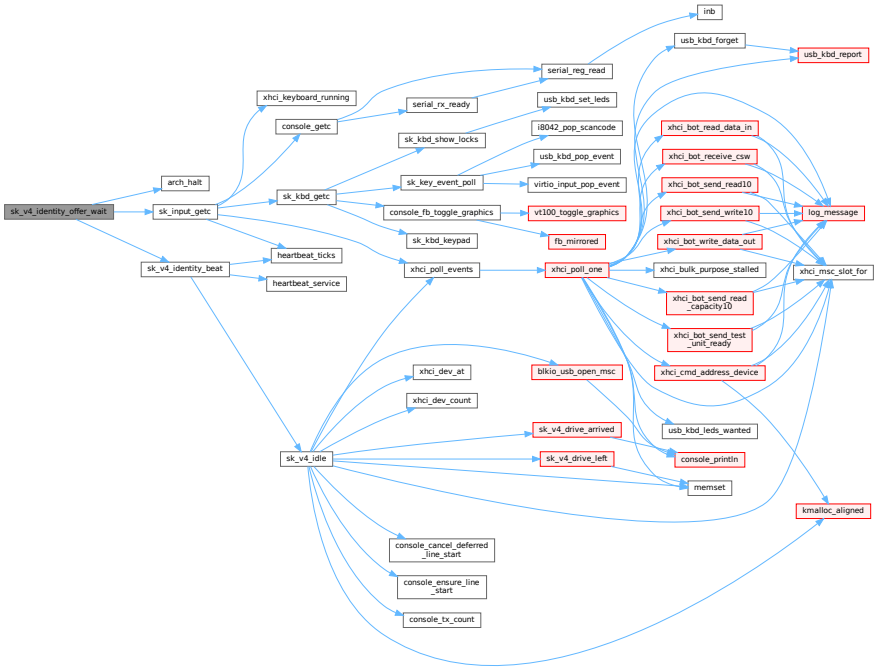
7.74.1.5 sk_v4_identity_offer_wait()

```
int sk_v4_identity_offer_wait (
            blkio_dev_t ** raw,
            uint64_t * bytes )
```

References arch_halt(), BLKIO_FORTH_BLOCK_SIZE, bytes, g_mint_waiting, g_offered, g_spare, g_stick_in, sk_v4_drive_t::raw, sk_input_getc(), sk_v4_identity_beat(), and blkio_dev::total_blocks.

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

7.74.1.6 sk_v4_identity_offered_present()

```
int sk_v4_identity_offered_present (
            void  )
```

References g_offered.

Referenced by read_answer(), and sk_v4_mint_zuse().

Here is the caller graph for this function:

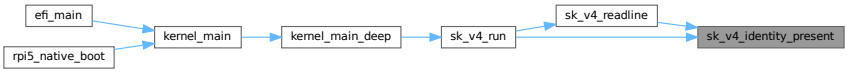
7.74.1.7 sk_v4_identity_present()

```
int sk_v4_identity_present (
            void  )
```

References g_zuse_session.

Referenced by sk_v4_readline(), and sk_v4_run().

Here is the caller graph for this function:



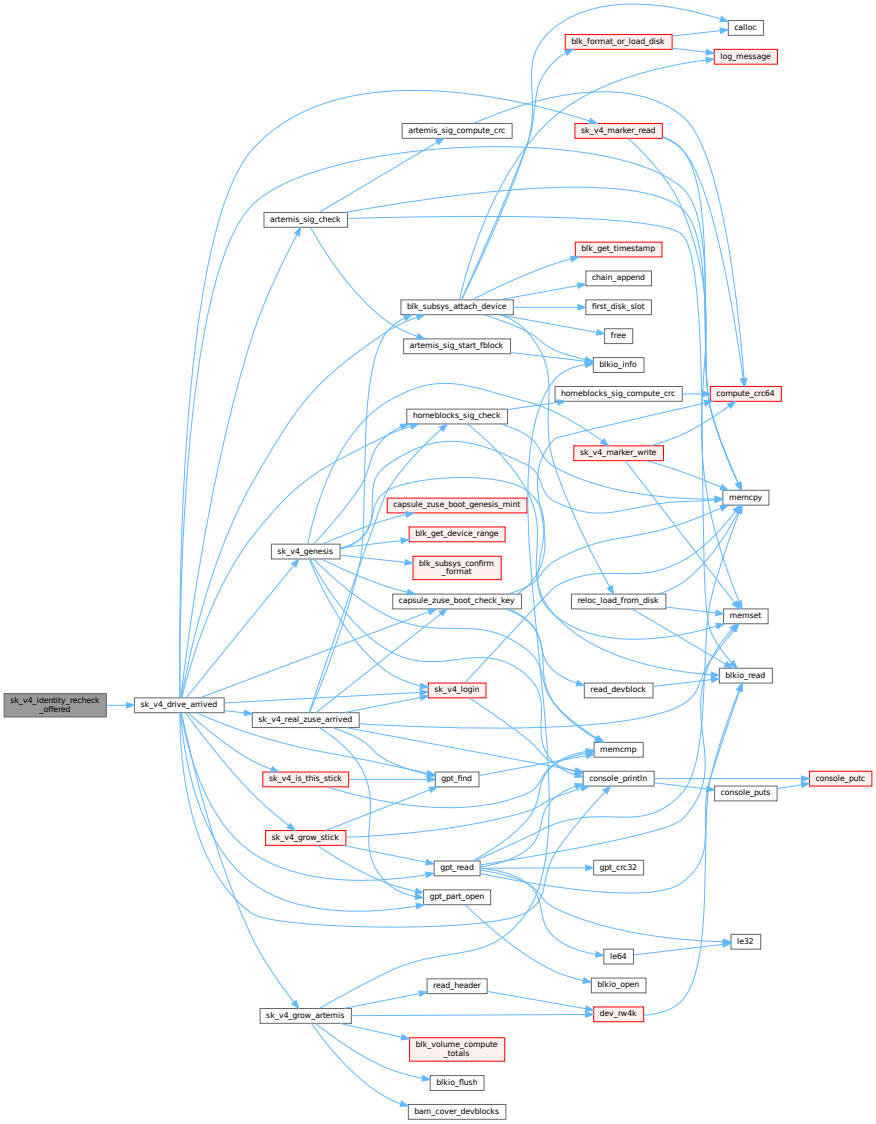
7.74.1.8 sk_v4_identity_recheck_offered()

```
void sk_v4_identity_recheck_offered (
            void  )
```

References g_offered, and sk_v4_drive_arrived().

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

7.74.1.9 sk_v4_identity_release_offered()

```
void sk_v4_identity_release_offered (
            void  )
```

References g_offered, and g_spare.

Referenced by failed(), and sk_v4_mint_zuse().

Here is the caller graph for this function:

7.74.1.10 sk_v4_identity_set_marker()

```
void sk_v4_identity_set_marker (
            const sk_v4_marker_t * m )
```

References g_marker.

Referenced by sk_v4_mint_zuse().

Here is the caller graph for this function:

7.74.1.11 sk_v4_identity_stick_in()

```
int sk_v4_identity_stick_in (
            void  )
```

References g_stick_in.

Referenced by read_answer().

Here is the caller graph for this function:

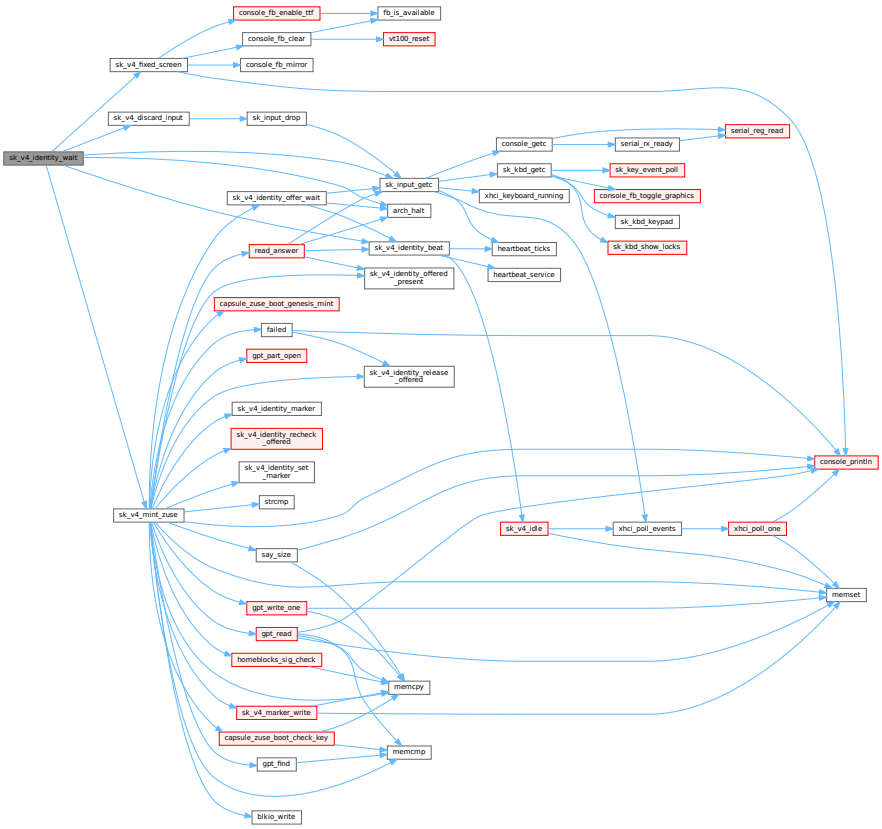
7.74.1.12 sk_v4_identity_wait()

```
void sk_v4_identity_wait (
            void  )
```

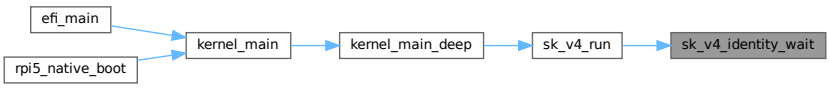
References arch_halt(), g_proxy_fallback, g_zuse_session, sk_input_getc(), sk_v4_discard_input(), sk_v4_fixed_screen(), sk_v4_identity_beat(), and sk_v4_mint_zuse().

Referenced by sk_v4_run().

Here is the call graph for this function:



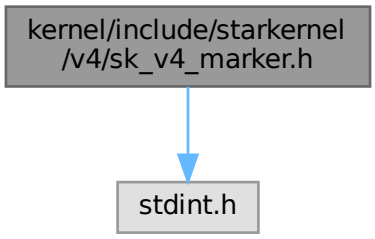
Here is the caller graph for this function:



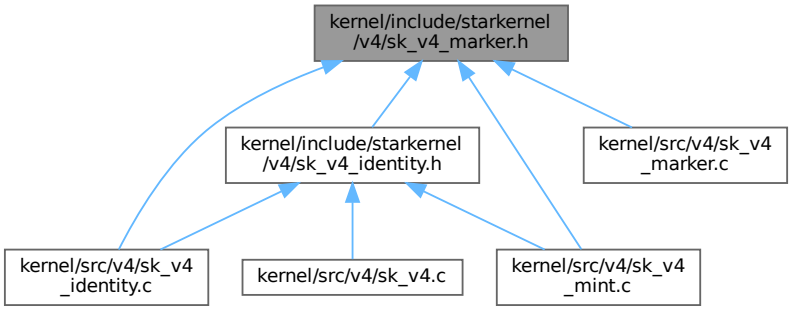
7.75 kernel/include/starkernel/v4/sk_v4_marker.h File Reference

```
#include <stdint.h>
```

Include dependency graph for sk_v4_marker.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct sk_v4_marker_t

Functions

- int sk_v4_marker_read (sk_v4_marker_t *m)
- int sk_v4_marker_write (const sk_v4_marker_t *m)

7.75.1 Function Documentation

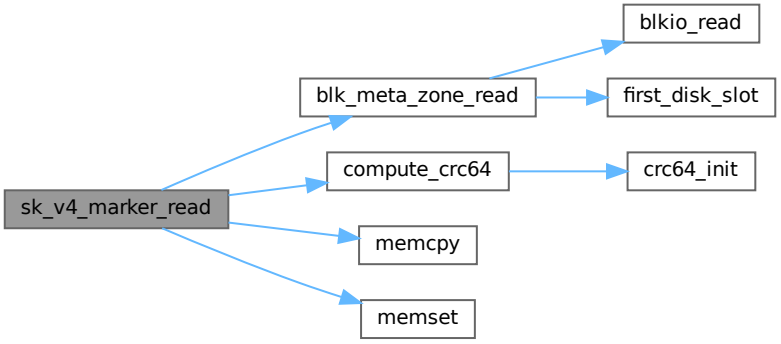
7.75.1.1 sk_v4_marker_read()

```
int sk_v4_marker_read (
            sk_v4_marker_t * m )
```

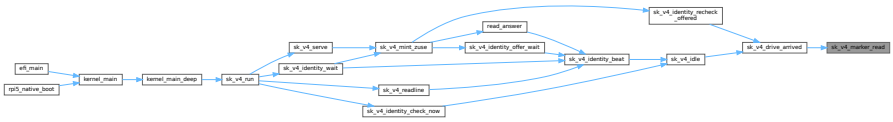
References blk_meta_zone_read(), compute_crc64(), g_buf, sk_v4_marker_t::has_real, memcpy(), memset(), sk_v4_marker_t::present, sk_v4_marker_t::proxy, sk_v4_marker_t::real, ZUSE_GENESIS_MARKER_MAGIC, ZUSE_GENESIS_MARKER_VERSION, and ZUSE_GENESIS_MARKER_VERSION_2.

Referenced by sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:



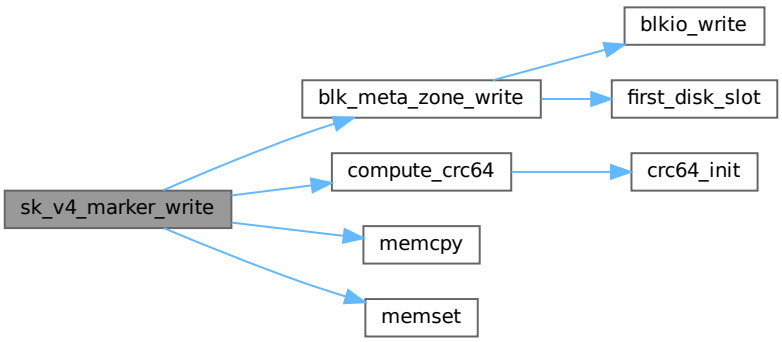
7.75.1.2 sk_v4_marker_write()

```
int sk_v4_marker_write (
            const sk_v4_marker_t * m )
```

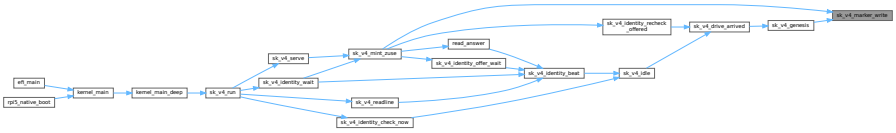
References blk_meta_zone_write(), compute_crc64(), g_buf, sk_v4_marker_t::has_real, memcpy(), memset(), sk_v4_marker_t::proxy, sk_v4_marker_t::real, ZUSE_GENESIS_MARKER_MAGIC, and ZUSE_GENESIS_MARKER_VERSION_2.

Referenced by sk_v4_genesis(), and sk_v4_mint_zuse().

Here is the call graph for this function:

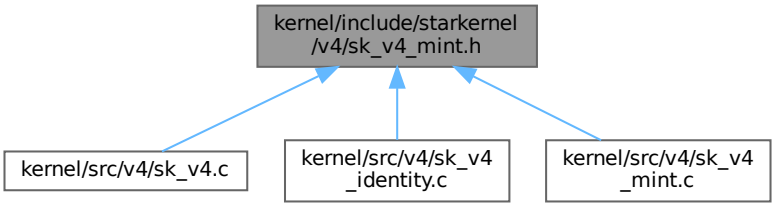


Here is the caller graph for this function:



7.76 kernel/include/starkernel/v4/sk_v4_mint.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int sk_v4_mint_zuse (void)

7.76.1 Function Documentation

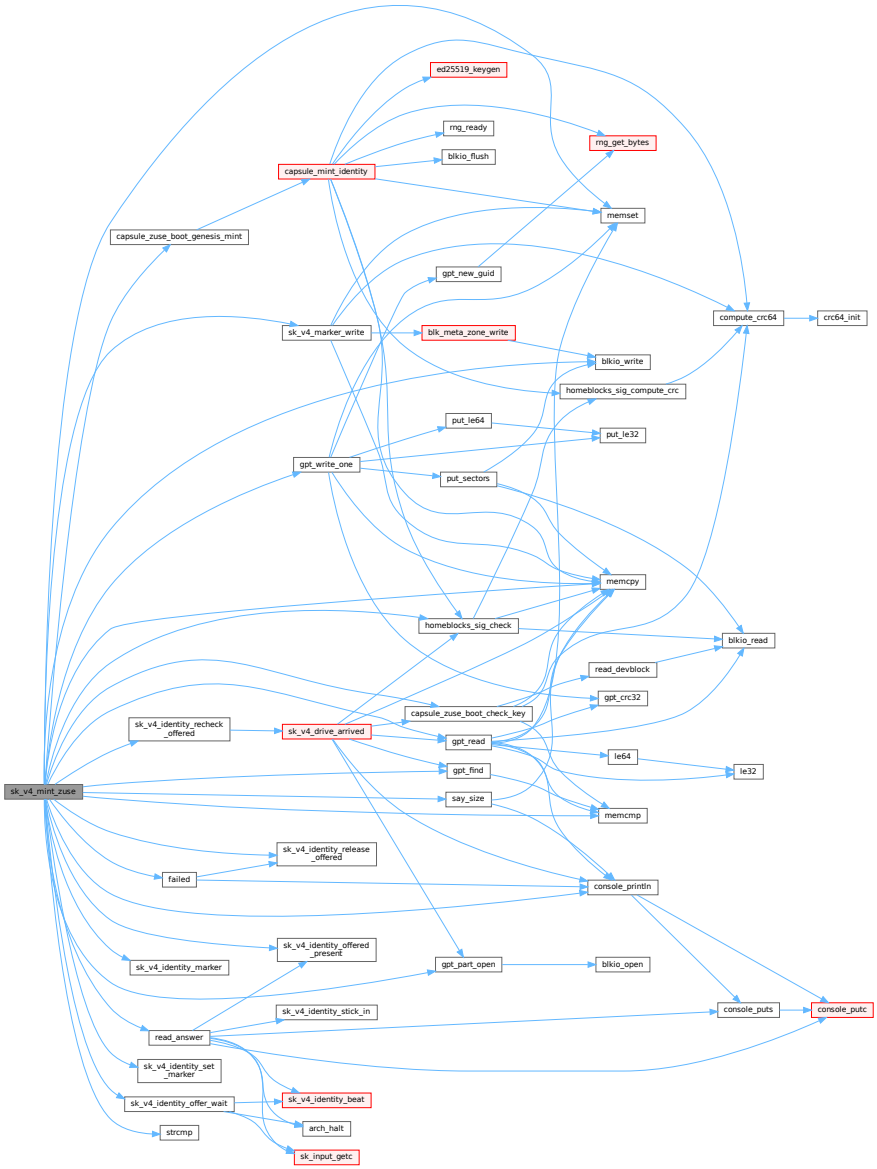
7.76.1.1 sk_v4_mint_zuse()

```
int sk_v4_mint_zuse (
            void  )
```

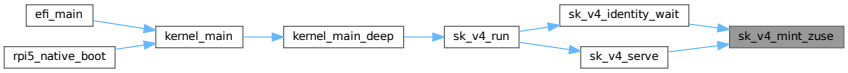
References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_write(), bytes, capsule_zuse_boot_check_key(), capsule_zuse_boot_genesis_mint(), console_println(), failed(), GPT_BAD, gpt_find(), GPT_OK, gpt_part_open(), gpt_read(), gpt_type_starship_identity, gpt_write_one(), sk_v4_marker_t::has_real, homeblocks_sig_check(), HOMEBLOCKS_SIG_START_FBLOCK, memcmp(), memcpy(), memset(), MINT_IDENT_SECTORS, MINT_MIN_BYTES, MINT_ZERO_FBLOCKS, sk_v4_marker_t::present, read_answer(), sk_v4_marker_t::real, say_size(), sk_v4_identity_marker(), sk_v4_identity_offer_wait(), sk_v4_identity_offered_present(), sk_v4_identity_recheck_offered(), sk_v4_identity_release_offered(), sk_v4_identity_set_marker(), sk_v4_marker_write(), strcmp(), and zero.

Referenced by sk_v4_identity_wait(), and sk_v4_serve().

Here is the call graph for this function:



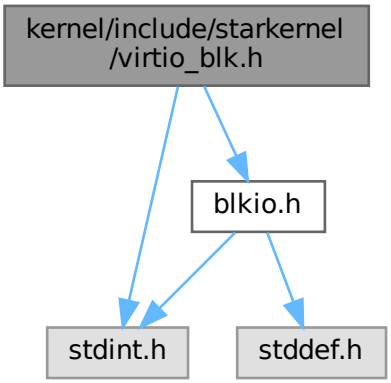
Here is the caller graph for this function:



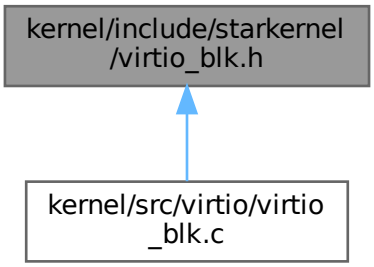
7.77 kernel/include/starkernel/virtio_blk.h File Reference

```
#include <stdint.h>
#include "blkio.h"
```

Include dependency graph for virtio_blk.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define VBLK_SECTOR_SIZE 512u
- #define VBLK_SECTORS_PER_BLOK 2u /* 1 KiB Forth block = 2 × 512-byte sectors */
- #define VBLK_ARTEMIS_SECTORS 61440u /* 30 MB */
- #define VIRTIO_PCI_VENDOR_ID 0x1AF4u
- #define VIRTIO_BLK_DEVICE_MODERN 0x1042u
- #define VIRTIO_BLK_DEVICE_LEGACY 0x1001u
- #define VIRTIO_BLK_T_IN 0u
- #define VIRTIO_BLK_T_OUT 1u
- #define VIRTIO_BLK_S_OK 0u
- #define VIRTIO_BLK_S_IOERR 1u
- #define VIRTIO_BLK_S_UNSUPP 2u

Functions

- int virtio_blk_find_artemis (blkio_dev_t *dev_out)

7.77.1 Macro Definition Documentation

7.77.1.1 VBLK_ARTEMIS_SECTORS

```
#define VBLK_ARTEMIS_SECTORS 61440u /* 30 MB */
```

7.77.1.2 VBLK_SECTOR_SIZE

```
#define VBLK_SECTOR_SIZE 512u
```

7.77.1.3 VBLK_SECTORS_PER_BLOK

```
#define VBLK_SECTORS_PER_BLOK 2u /* 1 KiB Forth block = 2 × 512-byte sectors */
```

7.77.1.4 VIRTIO_BLK_DEVICE_LEGACY

```
#define VIRTIO_BLK_DEVICE_LEGACY 0x1001u
```

7.77.1.5 VIRTIO_BLK_DEVICE_MODERN

```
#define VIRTIO_BLK_DEVICE_MODERN 0x1042u
```

7.77.1.6 VIRTIO_BLK_S_IOERR

```
#define VIRTIO_BLK_S_IOERR 1u
```

7.77.1.7 VIRTIO_BLK_S_OK

```
#define VIRTIO_BLK_S_OK 0u
```

7.77.1.8 VIRTIO_BLK_S_UNSUPP

```
#define VIRTIO_BLK_S_UNSUPP 2u
```

7.77.1.9 VIRTIO_BLK_T_IN

```
#define VIRTIO_BLK_T_IN 0u
```

7.77.1.10 VIRTIO_BLK_T_OUT

```
#define VIRTIO_BLK_T_OUT 1u
```

7.77.1.11 VIRTIO_PCI_VENDOR_ID

```
#define VIRTIO_PCI_VENDOR_ID 0x1AF4u
```

7.77.2 Function Documentation

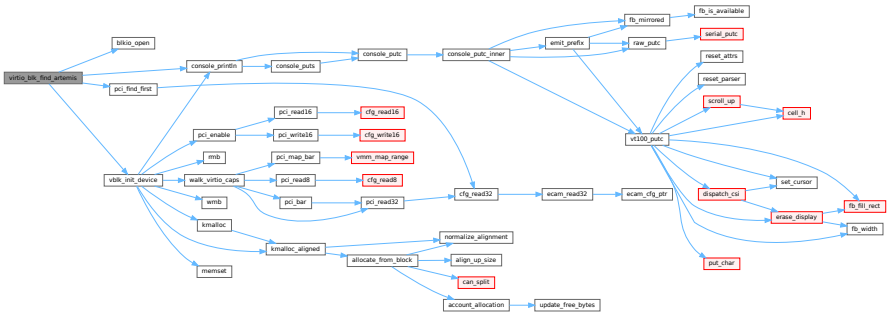
7.77.2.1 virtio_blk_find_artemis()

```
int virtio_blk_find_artemis (
            blkio_dev_t * dev_out )
```

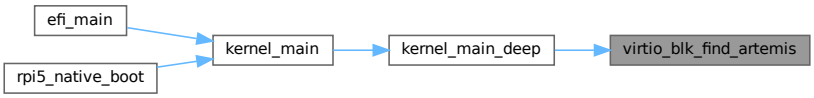
References BLKIO_FORTH_BLOCK_SIZE, blkio_open(), console_println(), blkio_params_t::forth_block_size, g_vblk, g_vblk_ready, g_vblk_vtable, blkio_params_t::opaque, pci_find_first(), blkio_params_t::total_blocks, VirtBlkState::total_forth_blocks, vblk_init_device(), VIRTIO_BLK_DEVICE_LEGACY, VIRTIO_BLK_DEVICE_MODERN, and VIRTIO_PCI_VENDOR_ID.

Referenced by kernel_main_deep().

Here is the call graph for this function:



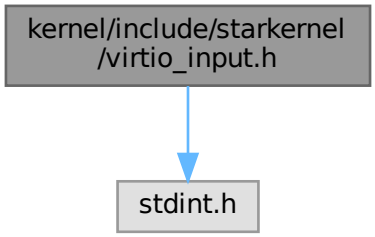
Here is the caller graph for this function:



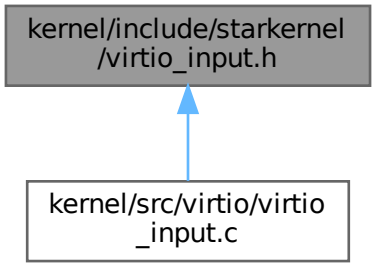
7.78 kernel/include/starkernel/virtio_input.h File Reference

```
#include <stdint.h>
```

Include dependency graph for virtio_input.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct VirtioInputEvent

Macros

- #define VIRTIO_INPUT_PCI_VENDOR_ID 0x1AF4u
- #define VIRTIO_INPUT_PCI_DEVICE_ID 0x1052u
- #define VIRTIO_INPUT_EV_KEY 0x01u

Functions

- int virtio_input_find_keyboard (void)
- void virtio_input_isr (void)
- int virtio_input_pop_event (uint16_t *code, uint32_t *value)

7.78.1 Macro Definition Documentation

7.78.1.1 VIRTIO_INPUT_EV_KEY

```
#define VIRTIO_INPUT_EV_KEY 0x01u
```

7.78.1.2 VIRTIO_INPUT_PCI_DEVICE_ID

```
#define VIRTIO_INPUT_PCI_DEVICE_ID 0x1052u
```

7.78.1.3 VIRTIO_INPUT_PCI_VENDOR_ID

```
#define VIRTIO_INPUT_PCI_VENDOR_ID 0x1AF4u
```

7.78.2 Function Documentation

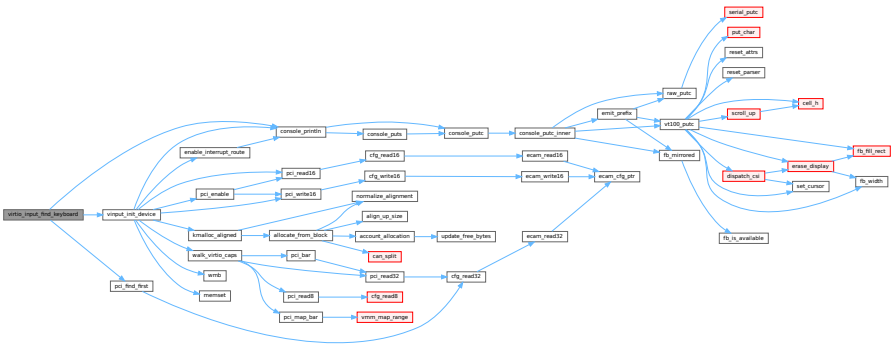
7.78.2.1 virtio_input_find_keyboard()

```
int virtio_input_find_keyboard (
            void  )
```

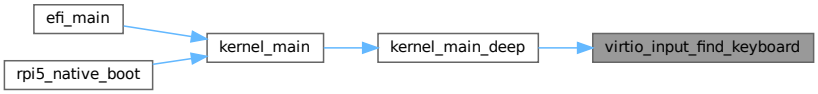
References console_println(), g_vinput_ready, pci_find_first(), vinput_init_device(), VIRTIO_INPUT_PCI_DEVICE_ID, and VIRTIO_INPUT_PCI_VENDOR_ID.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:

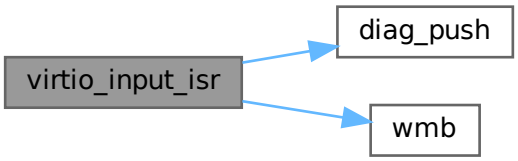


7.78.2.2 virtio_input_isr()

```
void virtio_input_isr (
            void  )
```

References VirtInputState::avail, VirtInputState::avail_idx, VirtInputState::bufs, VirtioInputEvent::code, diag_push(), EVENTQ_SIZE, g_vinput, g_vinput_ready, g_virtio_input_isr_count, VirtqUsedElem::id, VirtqAvail::idx, VirtqUsed::idx, VirtInputState::isr_status, VirtInputState::last_used_idx, VirtInputState::notify, VirtInputState::notify_off_mult, VirtInputState::queue_notify_off, VirtqAvail::ring, VirtqUsed::ring, VirtioInputEvent::type, VirtInputState::used, VirtioInputEvent::value, VIRTIO_INPUT_EV_KEY, and wmb().

Here is the call graph for this function:



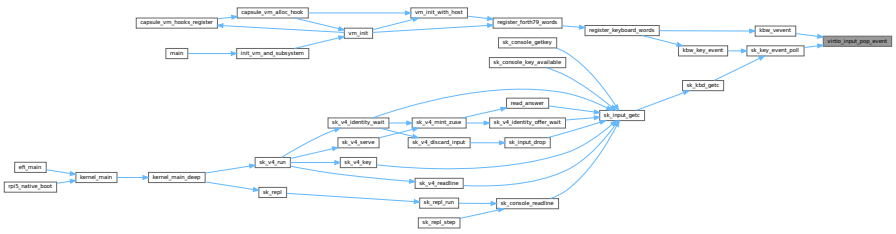
7.78.2.3 virtio_input_pop_event()

```
int virtio_input_pop_event (
            uint16_t * code,
            uint32_t * value )
```

References DiagEvent::code, DIAG_RING_SIZE, g_diag_head, g_diag_ring, g_diag_tail, and DiagEvent::value.

Referenced by kbw_vevent(), and sk_key_event_poll().

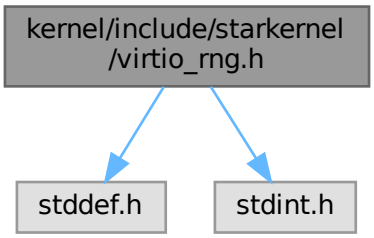
Here is the caller graph for this function:



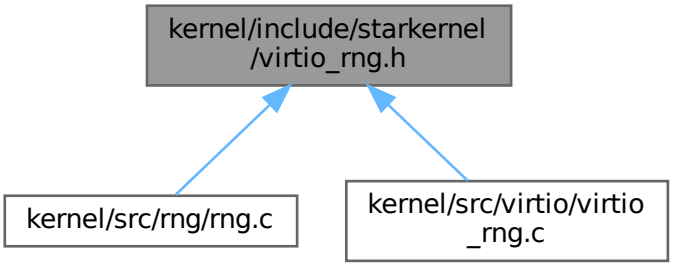
7.79 kernel/include/starkernel/virtio_rng.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for virtio_rng.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define VIRTIO_RNG_PCI_VENDOR_ID 0x1AF4u
- #define VIRTIO_RNG_DEVICE_MODERN 0x1044u
- #define VIRTIO_RNG_DEVICE_LEGACY 0x1005u

Functions

- int virtio_rng_init (void)
- int virtio_rng_ready (void)
- int virtio_rng_get_bytes (uint8_t *buf, size_t n)

7.79.1 Macro Definition Documentation

7.79.1.1 VIRTIO_RNG_DEVICE_LEGACY

```
#define VIRTIO_RNG_DEVICE_LEGACY 0x1005u
```

7.79.1.2 VIRTIO_RNG_DEVICE_MODERN

```
#define VIRTIO_RNG_DEVICE_MODERN 0x1044u
```

7.79.1.3 VIRTIO_RNG_PCI_VENDOR_ID

```
#define VIRTIO_RNG_PCI_VENDOR_ID 0x1AF4u
```

7.79.2 Function Documentation

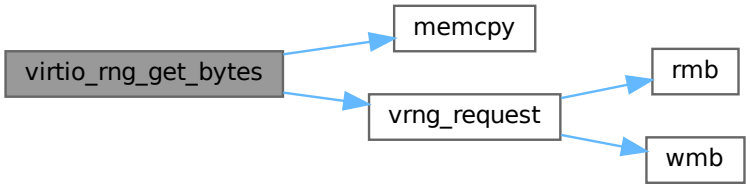
7.79.2.1 virtio_rng_get_bytes()

```
int virtio_rng_get_bytes (
            uint8_t * buf,
            size_t n )
```

References VirtRngState::data_buf, g_vrng, g_vrng_ready, memcpy(), and vrng_request().

Referenced by rng_get_bytes().

Here is the call graph for this function:



Here is the caller graph for this function:

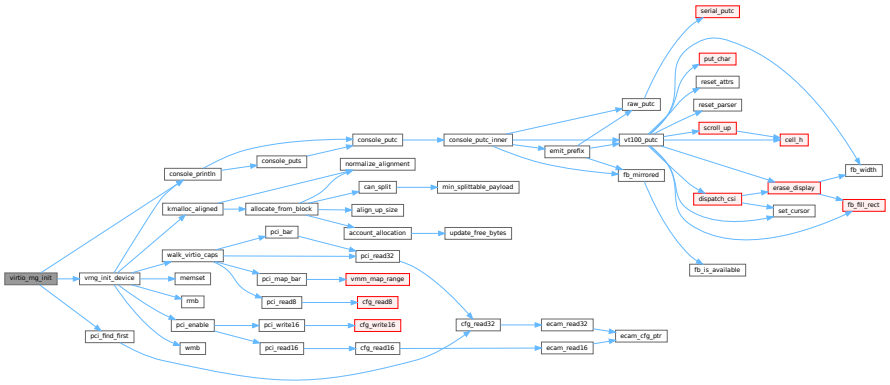
7.79.2.2 virtio_rng_init()

```
int virtio_rng_init (
            void  )
```

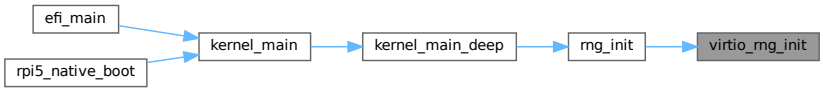
References console_println(), g_vrng_ready, pci_find_first(), VIRTIO_RNG_DEVICE_LEGACY, VIRTIO_RNG_DEVICE_MODERN, VIRTIO_RNG_PCI_VENDOR_ID, and vrng_init_device().

Referenced by rng_init().

Here is the call graph for this function:



Here is the caller graph for this function:



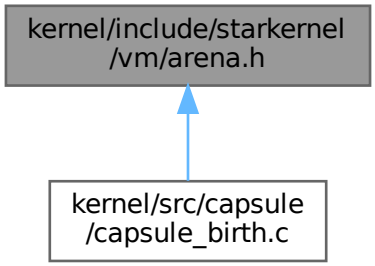
7.79.2.3 virtio_rng_ready()

```
int virtio_rng_ready (
            void  )
```

References g_vrng_ready.

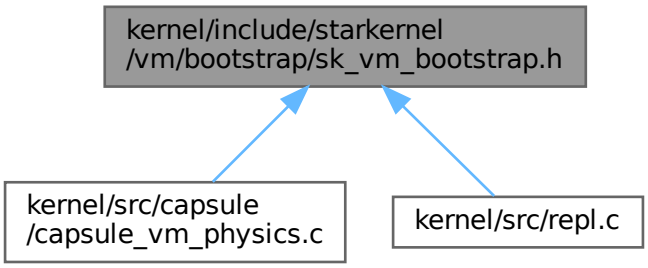
7.80 kernel/include/starkernel/vm/arena.h File Reference

This graph shows which files directly or indirectly include this file:



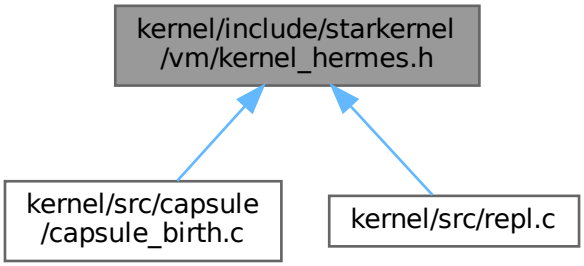
7.81 kernel/include/starkernel/vm/bootstrap/sk_vm_bootstrap.h File Reference

This graph shows which files directly or indirectly include this file:



7.82 kernel/include/starkernel/vm/kernel_hermes.h File Reference

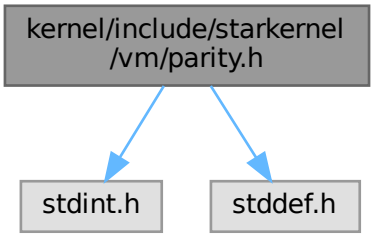
This graph shows which files directly or indirectly include this file:



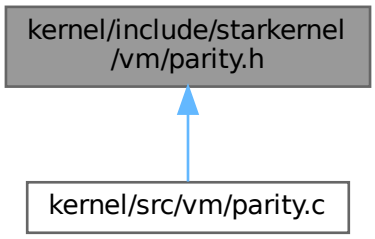
7.83 kernel/include/starkernel/vm/parity.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for parity.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct ParityPacket

  ParityPacket - Summary of VM state for comparison.

Macros

- #define SK_BOOTSTRAP_OK 0

  parity.h - Parity packet for VM validation
- #define SK_BOOTSTRAP_ARENA_FAIL 1
- #define SK_BOOTSTRAP_INIT_FAIL 2
- #define SK_BOOTSTRAP_DICT_FAIL 3
- #define FNV1A_64_OFFSET_BASIS 0xCBF29CE484222325ULL

  FNV-1a constants.
- #define FNV1A_64_PRIME 0x100000001B3ULL

Functions

- void sk_parity_collect (struct VM ∗vm, ParityPacket ∗out)

  Collect M7 parity data from the VM into a `ParityPacket`.
- void sk_parity_print (const ParityPacket ∗pkt)

  Print a `ParityPacket` to the kernel console or stdout.
- uint64_t sk_dict_canonical_hash (struct VM ∗vm)

  Compute the canonical FNV-1a 64-bit hash of the VM dictionary.
- uint32_t sk_dict_word_count (struct VM ∗vm)

  Count the number of entries in the VM dictionary.
- uint64_t fnv1a_64 (const uint8_t ∗data, size_t len, uint64_t hash)

  FNV-1a 64-bit incremental hash over a byte buffer.

7.83.1 Macro Definition Documentation

7.83.1.1 FNV1A_64_OFFSET_BASIS

```
#define FNV1A_64_OFFSET_BASIS 0xCBF29CE484222325ULL
```

FNV-1a constants.

7.83.1.2 FNV1A_64_PRIME

```
#define FNV1A_64_PRIME 0x100000001B3ULL
```

7.83.1.3 SK_BOOTSTRAP_ARENA_FAIL

```
#define SK_BOOTSTRAP_ARENA_FAIL 1
```

7.83.1.4 SK_BOOTSTRAP_DICT_FAIL

```
#define SK_BOOTSTRAP_DICT_FAIL 3
```

7.83.1.5 SK_BOOTSTRAP_INIT_FAIL

```
#define SK_BOOTSTRAP_INIT_FAIL 2
```

7.83.1.6 SK_BOOTSTRAP_OK

```
#define SK_BOOTSTRAP_OK 0
```

parity.h - Parity packet for VM validation

Defines structures and functions for comparing hosted vs kernel VM state. Parity is verified via canonical dictionary hash, not raw memory comparison.

M7 Normative Rules:

- word_id is monotonic creation index starting at 0
- Dictionary traversal is creation order (oldest to newest)
- Hash excludes pointers, padding, runtime fields
- Colon bodies are hashed as word_id sequences, not addresses Bootstrap result codes

7.83.2 Function Documentation

7.83.2.1 fnv1a_64()

```
uint64_t fnv1a_64 (
            const uint8_t * data,
            size_t len,
            uint64_t hash )
```

FNV-1a 64-bit incremental hash over a byte buffer.

Updates a running FNV-1a hash by folding `len` bytes from `data` into `hash`. The caller supplies the current hash state so that multiple buffers can be chained: pass `FNV1A_64_OFFSET_BASIS` for the first call and the returned value as `hash` for subsequent calls.

Algorithm: for each byte `b`, `hash` ^= b then `hash` *= FNV1A_64_PRIME.

Parameters

data	Pointer to the byte buffer to hash.
len	Number of bytes to consume.
hash	Running hash state; seed with `FNV1A_64_OFFSET_BASIS`.

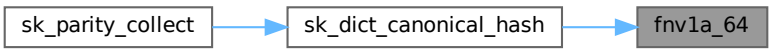
Returns

Updated hash state after processing all `len` bytes.

References FNV1A_64_PRIME.

Referenced by sk_dict_canonical_hash().

Here is the caller graph for this function:



7.83.2.2 sk_dict_canonical_hash()

```
uint64_t sk_dict_canonical_hash (
            struct VM * vm )
```

Compute the canonical FNV-1a 64-bit hash of the VM dictionary.

Implements M7 Rule 3: dictionary traversal is in creation order (oldest word first). Because `vm->latest` is the newest word, the function first builds a reverse array of entries and then hashes them in reverse-array order (oldest → newest).

For each entry the following structural fields are folded into the hash:

1. `flags` (1 byte)
2. `name_len` (1 byte)
3. `name` (name_len bytes)
4. `acl_default` (1 byte)
5. `word_id` (4 bytes, little-endian)
6. Colon body (via `hash_colon_body()`) if `WORD_COMPILED` is set

Pointer-valued fields (`func`, `link`) are intentionally excluded so the hash is stable across different build addresses and address-space layouts.

Traversal is capped at `MAX_DICT_ENTRIES` (2048) entries to guard against corruption.

Parameters

vm	Active VM whose dictionary will be hashed; returns 0 if NULL.

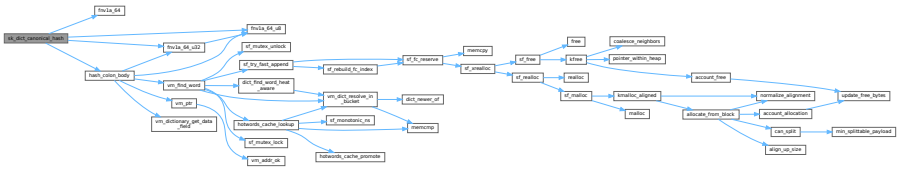
Returns

64-bit FNV-1a canonical hash of the dictionary structure.

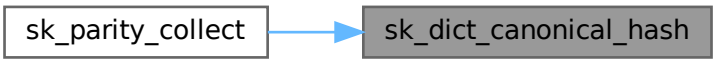
References DictEntry::acl_default, count, DictEntry::flags, fnv1a_64(), FNV1A_64_OFFSET_BASIS, fnv1a_64_u32(), fnv1a_64_u8(), DictEntry::func, hash_colon_body(), VM::latest, MAX_DICT_ENTRIES, DictEntry::name, DictEntry::name_len, sk_parity_debug_check_ptr, sk_parity_debug_log_msg, WORD_COMPILED, and DictEntry::word_id.

Referenced by sk_parity_collect().

Here is the call graph for this function:



Here is the caller graph for this function:



7.83.2.3 sk_dict_word_count()

```
uint32_t sk_dict_word_count (
            struct VM * vm )
```

Count the number of entries in the VM dictionary.

Walks the linked list starting at `vm->latest`, following each entry's `link` pointer until `NULL`. Stops early and returns `MAX_DICT_ENTRIES` if the list exceeds that limit, preventing runaway traversal on a corrupt dictionary.

Parameters

vm	Active VM whose dictionary will be counted; returns 0 if NULL.

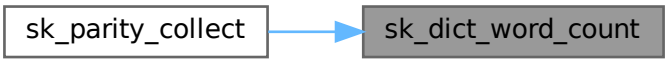
Returns

Number of dictionary entries found, capped at `MAX_DICT_ENTRIES`.

References count, VM::latest, and MAX_DICT_ENTRIES.

Referenced by sk_parity_collect().

Here is the caller graph for this function:



7.83.2.4 sk_parity_collect()

```
void sk_parity_collect (
            struct VM * vm,
            ParityPacket * out )
```

Collect M7 parity data from the VM into a `ParityPacket`.

Populates `out` with a snapshot of the VM's structural state for offline determinism verification. The packet contains:

- M7.1a fields: `word_count`, `here_offset`, `latest_word_id`, and the canonical dictionary hash (`header_hash64`) computed by `sk_dict_canonical_hash()`.
- M7.1b fields: test result counters (`tests_total` / `tests_passed` / `tests_failed` / `tests_skipped` / `tests_errors`) populated from `global_test_stats` when `STARFORTH_ENABLE_TESTS` is defined.

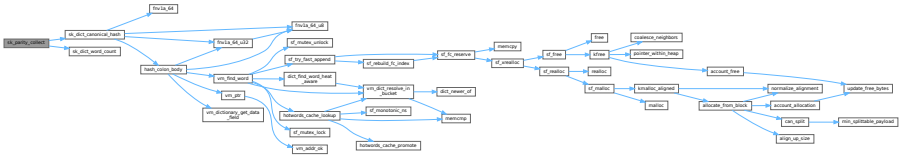
If `vm` is `NULL`, sets `out->bootstrap_result` to `SK_BOOTSTRAP_INIT_FAIL` and returns with all other fields zeroed. If `out` is `NULL` the function is a no-op.

Parameters

vm	Active VM to collect parity data from; may be `NULL`.
out	Output `ParityPacket` to fill; caller must allocate.

References ParityPacket::bootstrap_result, global_test_stats, ParityPacket::header_hash64, VM::here, ParityPacket::here_offset, VM::latest, ParityPacket::latest_word_id, SK_BOOTSTRAP_INIT_FAIL, SK_BOOTSTRAP_OK, sk_dict_canonical_hash(), sk_dict_word_count(), sk_parity_debug_log_msg, ParityPacket::tests_errors, ParityPacket::tests_failed, ParityPacket::tests_passed, ParityPacket::tests_skipped, ParityPacket::tests_total, TestStats::total_error, TestStats::total_fail, TestStats::total_pass, TestStats::total_skip, TestStats::total_tests, ParityPacket::window_hash64, ParityPacket::word_count, and DictEntry::word_id.

Here is the call graph for this function:



7.83.2.5 sk_parity_print()

```
void sk_parity_print (
            const ParityPacket * pkt )
```

Print a `ParityPacket` to the kernel console or stdout.

Emits two or three structured lines that can be captured in the QEMU serial log for offline determinism verification:

- `"PARITY:M7.1a word_count=N here=0xH latest_id=N hash=0xH"` — always emitted; contains the structural dictionary fingerprint.
- `"PARITY:M7.1b tests=N pass=N fail=N skip=N err=N"` — emitted only when `pkt->tests_total` > 0; contains POST result counts.
- `"PARITY:OK"` or `"PARITY:FAIL code=N"` — result verdict line.

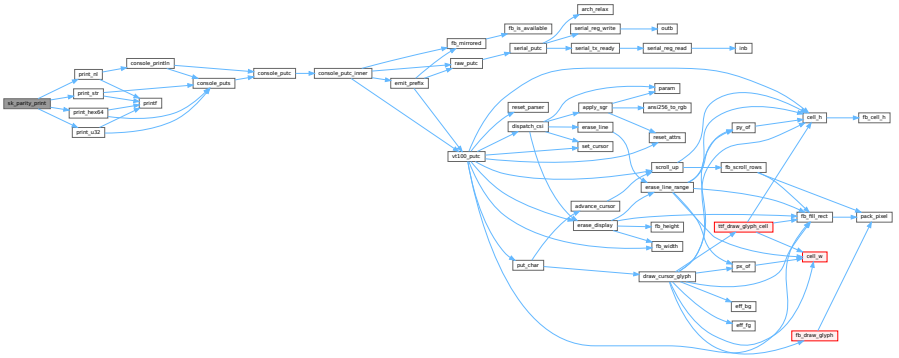
All output goes through `print_str()` / `print_u32()` / `print_hex64()` / `print_nl()` so the function compiles cleanly in both kernel and hosted builds. No-op if `pkt` is `NULL`.

Parameters

pkt	Pointer to the `ParityPacket` to display; may be `NULL`.

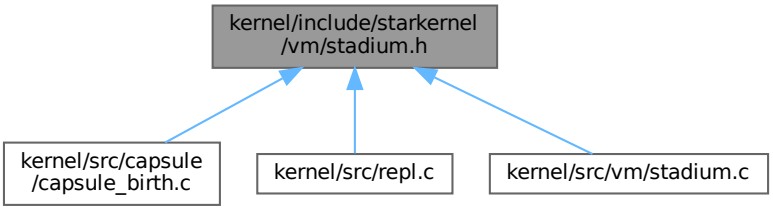
References ParityPacket::bootstrap_result, ParityPacket::header_hash64, ParityPacket::here_offset, ParityPacket::latest_word_id, print_hex64(), print_nl(), print_str(), print_u32(), SK_BOOTSTRAP_OK, ParityPacket::tests_errors, ParityPacket::tests_failed, ParityPacket::tests_passed, ParityPacket::tests_skipped, ParityPacket::tests_total, and ParityPacket::word_count.

Here is the call graph for this function:



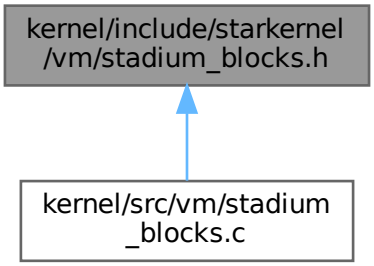
7.84 kernel/include/starkernel/vm/stadium.h File Reference

This graph shows which files directly or indirectly include this file:



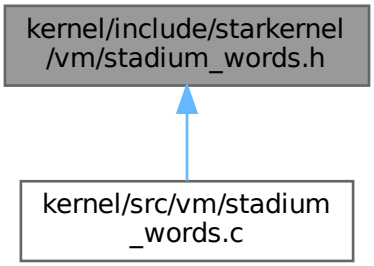
7.85 kernel/include/starkernel/vm/stadium_blocks.h File Reference

This graph shows which files directly or indirectly include this file:



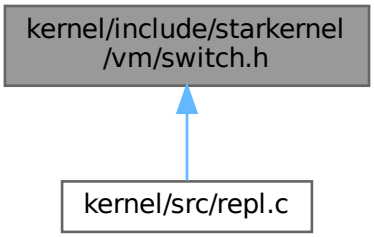
7.86 kernel/include/starkernel/vm/stadium_words.h File Reference

This graph shows which files directly or indirectly include this file:



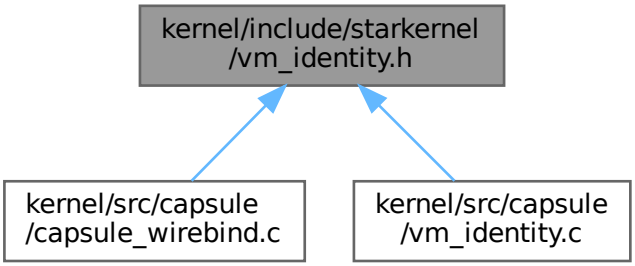
7.87 kernel/include/starkernel/vm/switch.h File Reference

This graph shows which files directly or indirectly include this file:



7.88 kernel/include/starkernel/vm_identity.h File Reference

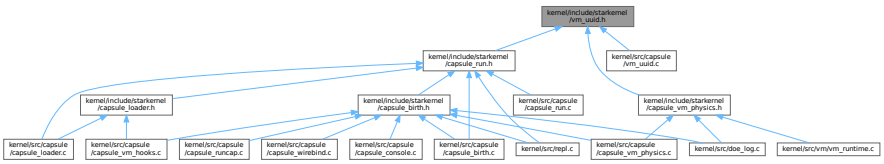
This graph shows which files directly or indirectly include this file:



7.89 kernel/include/starkernel/vm_log_buffer.h File Reference

7.90 kernel/include/starkernel/vm_uuid.h File Reference

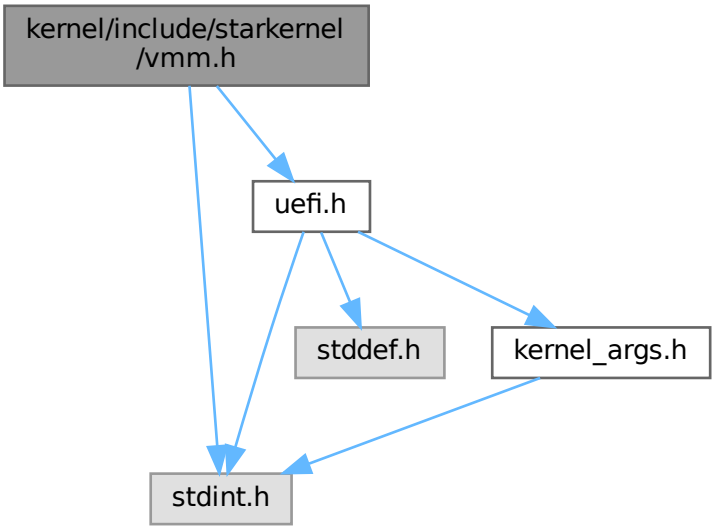
This graph shows which files directly or indirectly include this file:



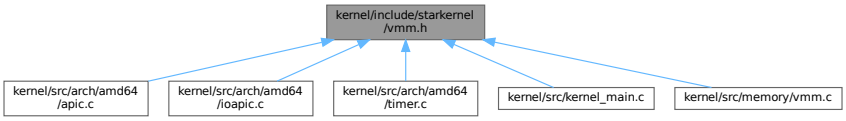
7.91 kernel/include/starkernel/vmm.h File Reference

```
#include <stdint.h>
#include "uefi.h"
```

Include dependency graph for vmm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct vmm_page_info_t

Macros

- #define VMM_PAGE_SIZE 4096ull

  vmm.h - Virtual Memory Manager interface (x86_64 4-level paging)
- #define VMM_FLAG_PRESENT (1ull << 0)
- #define VMM_FLAG_WRITABLE (1ull << 1)
- #define VMM_FLAG_USER (1ull << 2)
- #define VMM_FLAG_CACHE_DISABLE (1ull << 4) /* maps to PTE_PCD — required for MMIO */
- #define VMM_FLAG_NX (1ull << 63)

Functions

- int vmm_init (BootInfo *boot_info)
- int vmm_map_page (uint64_t vaddr, uint64_t paddr, uint64_t flags)
- int vmm_unmap_page (uint64_t vaddr)
- uint64_t vmm_get_paddr (uint64_t vaddr)
- int vmm_map_range (uint64_t vaddr, uint64_t paddr, uint64_t size, uint64_t flags)
- int vmm_query_page (uint64_t vaddr, vmm_page_info_t *info)

7.91.1 Macro Definition Documentation

7.91.1.1 VMM_FLAG_CACHE_DISABLE

```
#define VMM_FLAG_CACHE_DISABLE (1ull << 4) /* maps to PTE_PCD -- required for MMIO */
```

7.91.1.2 VMM_FLAG_NX

```
#define VMM_FLAG_NX (1ull << 63)
```

7.91.1.3 VMM_FLAG_PRESENT

```
#define VMM_FLAG_PRESENT (1ull << 0)
```

7.91.1.4 VMM_FLAG_USER

```
#define VMM_FLAG_USER (1ull << 2)
```

7.91.1.5 VMM_FLAG_WRITABLE

```
#define VMM_FLAG_WRITABLE (1ull << 1)
```

7.91.1.6 VMM_PAGE_SIZE

```
#define VMM_PAGE_SIZE 4096ull
```

vmm.h - Virtual Memory Manager interface (x86_64 4-level paging)

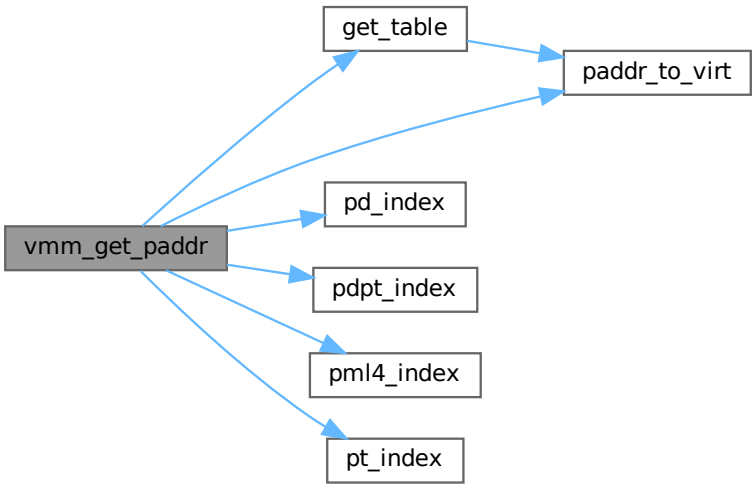
7.91.2 Function Documentation

7.91.2.1 vmm_get_paddr()

```
uint64_t vmm_get_paddr (
            uint64_t vaddr )
```

References ADDR_MASK, get_table(), paddr_to_virt(), pd_index(), pdpt_index(), pml4_index(), pt_index(), PTE_PRESENT, and vmm_root_pml4_phys.

Here is the call graph for this function:



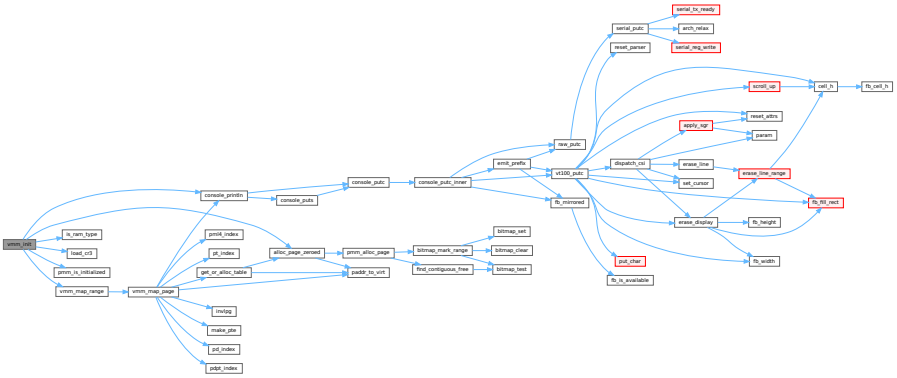
7.91.2.2 vmm_init()

```
int vmm_init (
            BootInfo * boot_info )
```

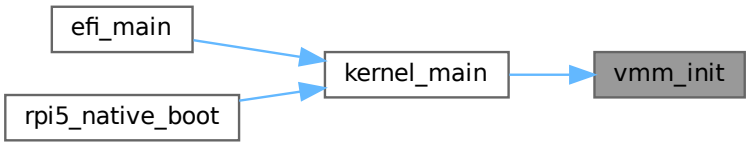
References alloc_page_zeroed(), FramebufferInfo::base, console_println(), BootInfo::framebuffer, hpet_phys_base, is_ram_type(), lapic_phys_base, load_cr3(), BootInfo::memory_map, BootInfo::memory_map_descriptor_size, BootInfo::memory_map_size, EFI_MEMORY_DESCRIPTOR::NumberOfPages, EFI_MEMORY_DESCRIPTOR::PhysicalStart, pmm_is_initialized(), FramebufferInfo::size, EFI_MEMORY_DESCRIPTOR::Type, VMM_FLAG_CACHE_DISABLE, VMM_FLAG_WRITABLE, vmm_map_range(), VMM_PAGE_SIZE, and vmm_root_pml4_phys.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



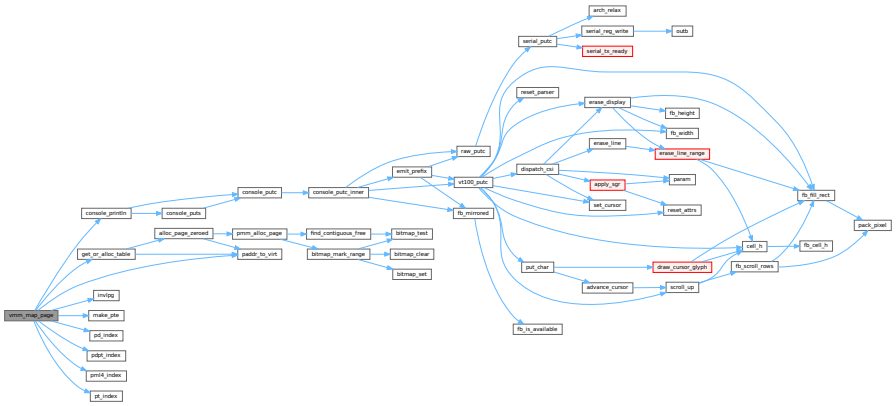
7.91.2.3 vmm_map_page()

```
int vmm_map_page (
            uint64_t vaddr,
            uint64_t paddr,
            uint64_t flags )
```

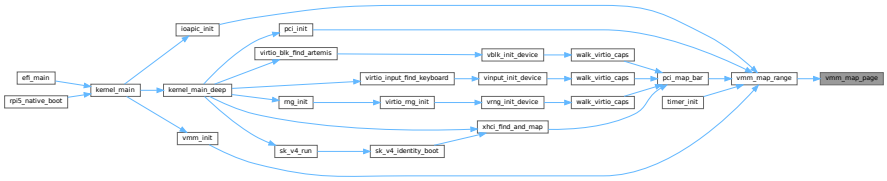
References ADDR_MASK, console_println(), flags, get_or_alloc_table(), invlpg(), make_pte(), paddr_to_virt(), pd_index(), pdpt_index(), pml4_index(), pt_index(), PTE_PRESENT, PTE_WRITABLE, VMM_FLAG_PRESENT, and vmm_root_pml4_phys.

Referenced by vmm_map_range().

Here is the call graph for this function:



Here is the caller graph for this function:



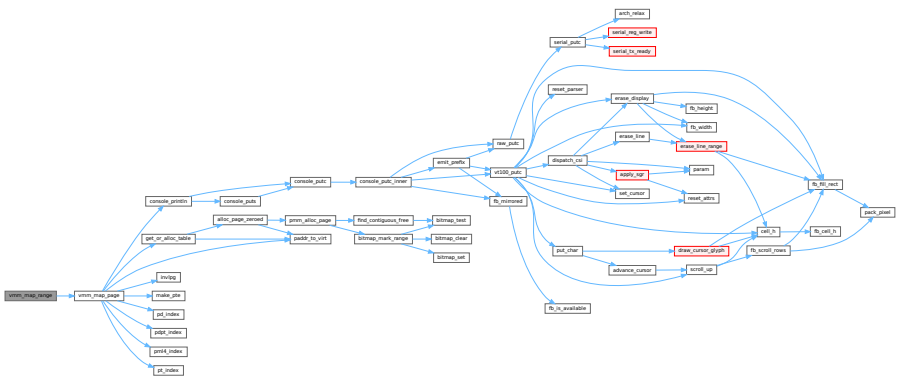
7.91.2.4 vmm_map_range()

```
int vmm_map_range (
            uint64_t vaddr,
            uint64_t paddr,
            uint64_t size,
            uint64_t flags )
```

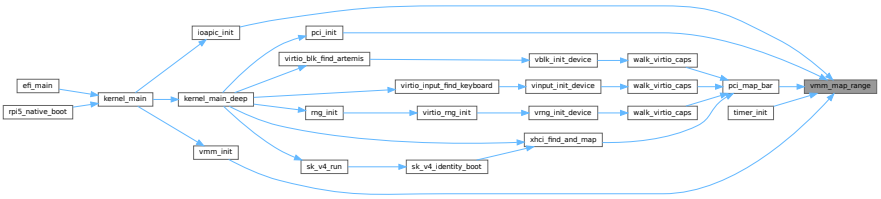
References flags, vmm_map_page(), and VMM_PAGE_SIZE.

Referenced by ioapic_init(), pci_init(), pci_map_bar(), timer_init(), and vmm_init().

Here is the call graph for this function:



Here is the caller graph for this function:

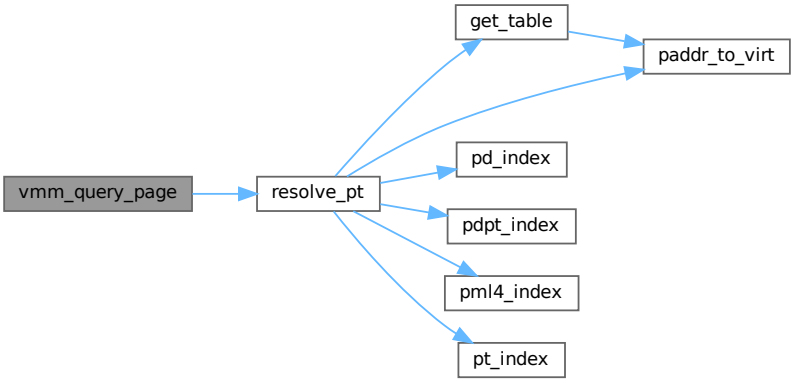


7.91.2.5 vmm_query_page()

```
int vmm_query_page (
            uint64_t vaddr,
            vmm_page_info_t * info )
```

References entry, vmm_page_info_t::executable, vmm_page_info_t::present, PTE_NX, PTE_PRESENT, PTE_WRITABLE, resolve_pt(), and vmm_page_info_t::writable.

Here is the call graph for this function:

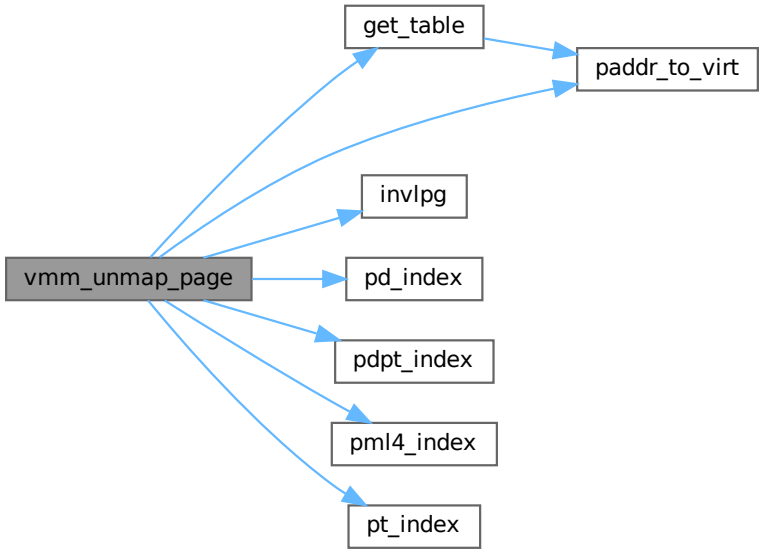


7.91.2.6 vmm_unmap_page()

```
int vmm_unmap_page (
            uint64_t vaddr )
```

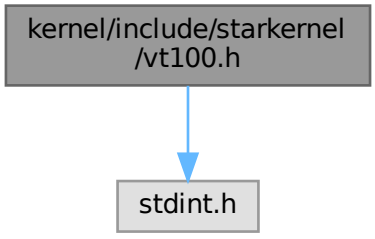
References get_table(), invlpg(), paddr_to_virt(), pd_index(), pdpt_index(), pml4_index(), pt_index(), PTE_PRESENT, and vmm_root_pml4_phys.

Here is the call graph for this function:

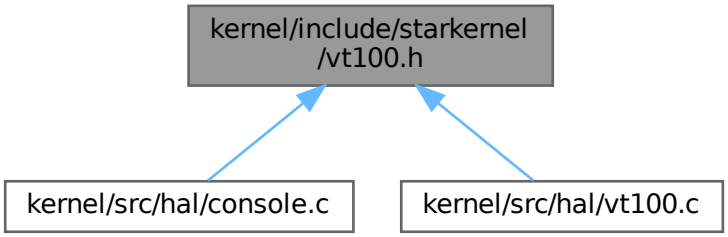


7.92 kernel/include/starkernel/vt100.h File Reference

`#include <stdint.h>`
Include dependency graph for vt100.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define VT100_DEFAULT_FG FB_RGB(0xAA, 0xAA, 0xAA) /∗ light gray ∗/

  vt100.h — Full-color ANSI / VT100 terminal state machine
- #define VT100_DEFAULT_BG FB_RGB(0x00, 0x00, 0x00) /∗ black ∗/

Functions

- void vt100_init (void)

  Initialize the VT100 terminal.
- void vt100_enable_ttf (void)

  FABRIC-0.md item 4.4j: switch the glyph-draw backend from font_8x16.c to TTF-TEXT's rasterizer for everything drawn from this call onward – boot/POST output before this call stays font_8x16.c, unaffected.
- void vt100_scroll_back (uint32_t n)

  FABRIC-0.md item 4.4q: move the REPL scrollback view back/forward by n lines and redraw.
- void vt100_scroll_fwd (uint32_t n)
- void vt100_putc (char c)

  Feed one byte into the terminal (handles escape sequences transparently).
- void vt100_puts (const char ∗s)

  Feed a null-terminated string.
- void vt100_toggle_graphics (void)

  FABRIC-0.md item 4.4y-revised: toggle the full-screen vt100 terminal between visible (normal operation) and hidden (graphics mode – the terminal stops drawing, letting direct framebuffer/TTF-TEXT calls show through undisturbed).
- void vt100_reset (void)
- void vt100_draw_cursor (void)

  Draw a solid block cursor at the terminal's current position.
- void vt100_erase_cursor (void)

  Erases whatever vt100_draw_cursor() last drew, restoring the cell to plain background.
- uint32_t vt100_cols (void)

  Terminal width in character columns.
- uint32_t vt100_rows (void)

  Terminal height in character rows.

7.92.1 Macro Definition Documentation

7.92.1.1 VT100_DEFAULT_BG

```
#define VT100_DEFAULT_BG FB_RGB(0x00, 0x00, 0x00) /* black */
```

7.92.1.2 VT100_DEFAULT_FG

```
#define VT100_DEFAULT_FG FB_RGB(0xAA, 0xAA, 0xAA) /* light gray */
```

vt100.h — Full-color ANSI / VT100 terminal state machine

Sits on top of the framebuffer driver. Call vt100_init() once the framebuffer is ready, then route all character output through vt100_putc().

Supported escape sequences ——————— Cursor movement ESC[H ESC[n;mH ESC[nA ESC[nB ESC[nC ESC[nD Cursor save/restore ESC[s ESC[u (also ESC7 / ESC8) Erase ESC[2J ESC[K ESC[0K ESC[1K ESC[2K SGR attributes ESC[...m (see below)

SGR codes ——— 0 reset all attributes 1 bold (doubles fg brightness) 4 underline 7 reverse video 22 normal intensity 24 underline off 27 reverse off 30–37 set fg to standard ANSI color 0–7 38;2;r;g;b set fg to 24-bit RGB truecolor 38;5;n set fg to 256-color index 39 reset fg to default 40–47 set bg to standard ANSI color 0–7 48;2;r;g;b set bg to 24-bit RGB truecolor 48;5;n set bg to 256-color index 49 reset bg to default 90–97 set fg to bright ANSI color 8–15 100–107 set bg to bright ANSI color 8–15

7.92.2 Function Documentation

7.92.2.1 vt100_cols()

```
uint32_t vt100_cols (
            void  )
```

Terminal width in character columns.

References VT100State::cols, and g_vt.

7.92.2.2 vt100_draw_cursor()

```
void vt100_draw_cursor (
            void  )
```

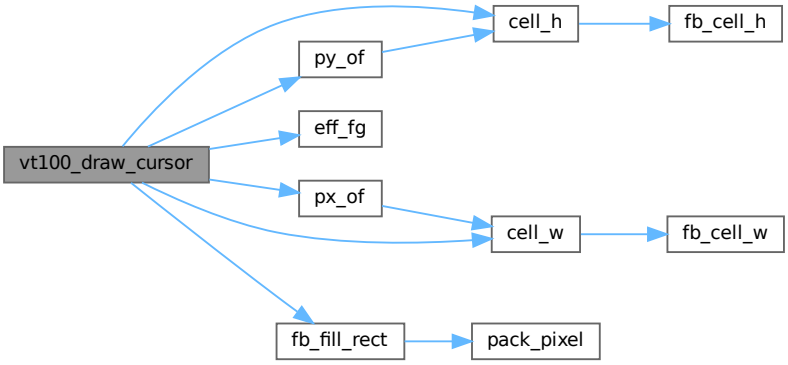
Draw a solid block cursor at the terminal's current position.

Static, not blinking. Idempotent – safe to call after every keystroke, since whatever gets typed next naturally overwrites it. No-op before TTF mode is active or while in graphics mode (vt100_toggle_graphics()).

References cell_h(), cell_w(), VT100State::cx, VT100State::cy, eff_fg(), fb_fill_rect(), g_glyph_mode, g_terminal_visible, g_vt, px_of(), py_of(), and VT_GLYPH_TTF.

Referenced by console_fb_draw_cursor().

Here is the call graph for this function:



Here is the caller graph for this function:

7.92.2.3 vt100_enable_ttf()

```
void vt100_enable_ttf (
            void  )
```

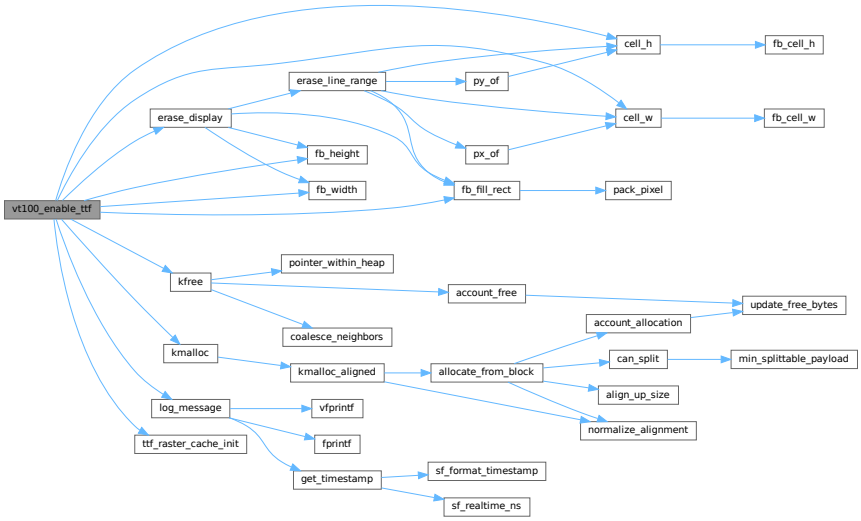
FABRIC-0.md item 4.4j: switch the glyph-draw backend from font_8x16.c to TTF-TEXT's rasterizer for everything drawn from this call onward – boot/POST output before this call stays font_8x16.c, unaffected.

Lazily loads the font capsule and its raster cache on first call (no-op on later calls). Recomputes cols/rows for the new cell size and clears the screen, since the two glyph backends use different cell dimensions. TTF point size/cell dimensions decided final by FABRIC-0.md item 4.4m (20px text, 96px REPL strip). No-op if the font capsule fails to load (stays on font_8x16.c; logged, not fatal).

References cell_h(), cell_w(), VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, erase_display(), fb_fill_rect(), fb_height(), fb_width(), g_boot_line_stride, g_boot_ring, g_boot_ring_cap, g_boot_ring_count, g_boot_ring_head, g_boot_rows, g_boot_shadow, g_glyph_mode, g_line_stride, g_ring, g_ring_cap, g_ring_count, g_ring_head, g_scroll_offset, g_shadow, g_ttf_cache, g_ttf_font, g_ttf_ready, g_vt, VT100State::initialized, kfree(), kmalloc(), LOG_ERROR, log_message(), VT100State::rows, slots, TTF_OK, ttf_raster_cache_init(), VT100_SCROLLBACK_MAX_LINES, VT100_TTF_CACHE_SLOTS, VT100_TTF_FONT_CAPSULE, and VT_GLYPH_TTF.

Referenced by console_fb_enable_ttf().

Here is the call graph for this function:



Here is the caller graph for this function:

7.92.2.4 vt100_erase_cursor()

```
void vt100_erase_cursor (
            void  )
```

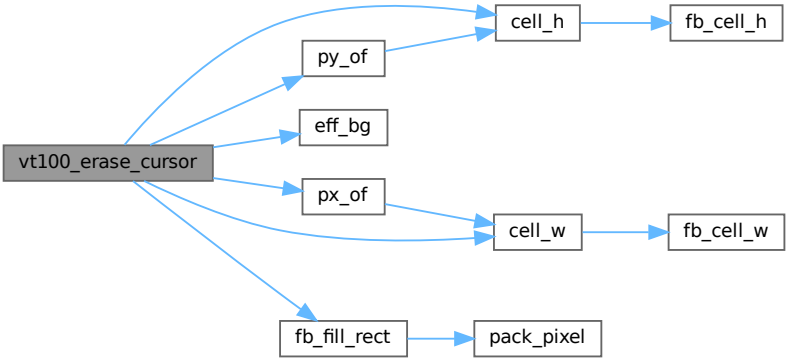
Erases whatever vt100_draw_cursor() last drew, restoring the cell to plain background.

Call before moving away from a cursor-drawn cell without also drawing a character there (e.g. Enter/newline). No-op before TTF mode is active or while in graphics mode.

References cell_h(), cell_w(), VT100State::cx, VT100State::cy, eff_bg(), fb_fill_rect(), g_glyph_mode, g_terminal_visible, g_vt, px_of(), py_of(), and VT_GLYPH_TTF.

Referenced by console_fb_erase_cursor().

Here is the call graph for this function:



Here is the caller graph for this function:

7.92.2.5 vt100_init()

```
void vt100_init (
            void  )
```

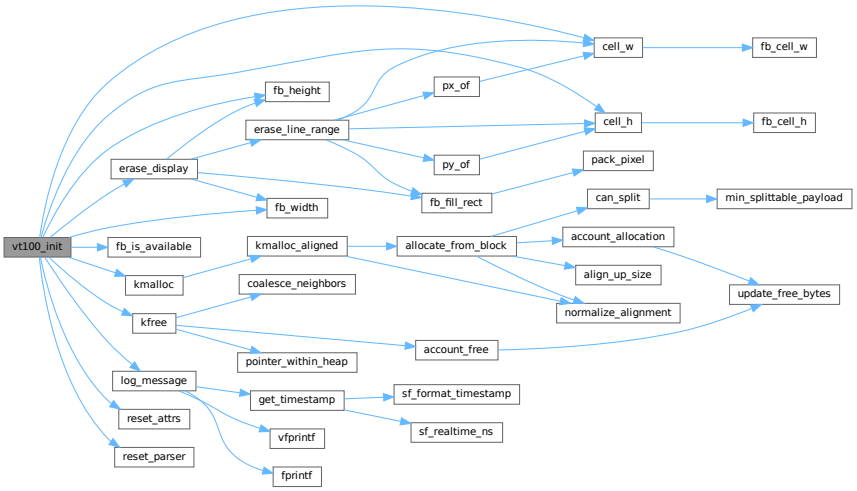
Initialize the VT100 terminal.

Must be called after fb_init(). Clears the screen and homes the cursor.

References cell_h(), cell_w(), VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, VT100State::def_fg, erase_display(), fb_height(), fb_is_available(), fb_width(), g_line_stride, g_ring, g_ring_cap, g_ring_count, g_ring_head, g_shadow, g_vt, VT100State::initialized, kfree(), kmalloc(), LOG_ERROR, log_message(), reset_attrs(), reset_parser(), VT100State::rows, VT100State::saved_cx, VT100State::saved_cy, VT100_BOOT_SCROLLBACK_MAX_LINES, VT100_DEFAULT_BG, and VT100_DEFAULT_FG.

Referenced by console_fb_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.92.2.6 vt100_putc()

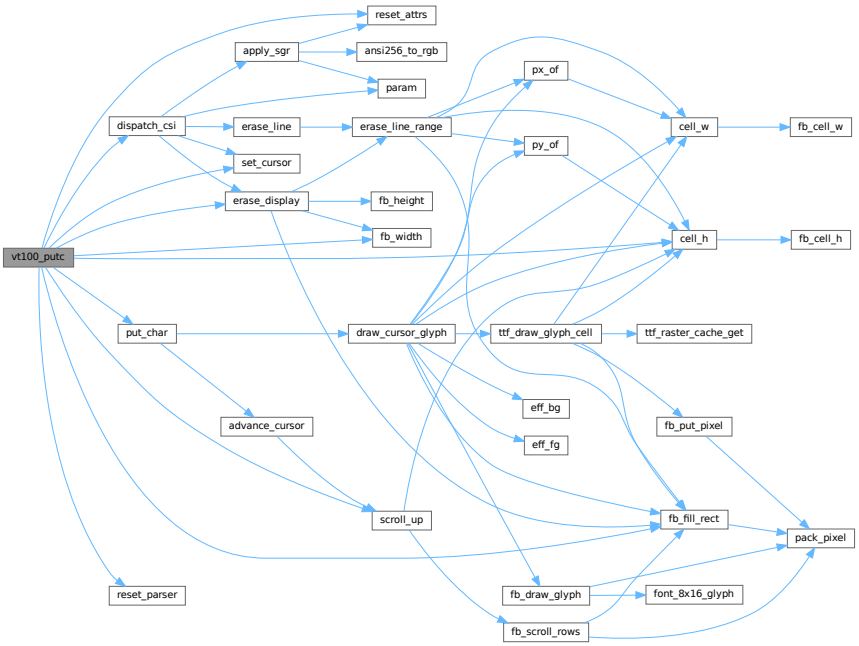
```
void vt100_putc (
            char c )
```

Feed one byte into the terminal (handles escape sequences transparently).

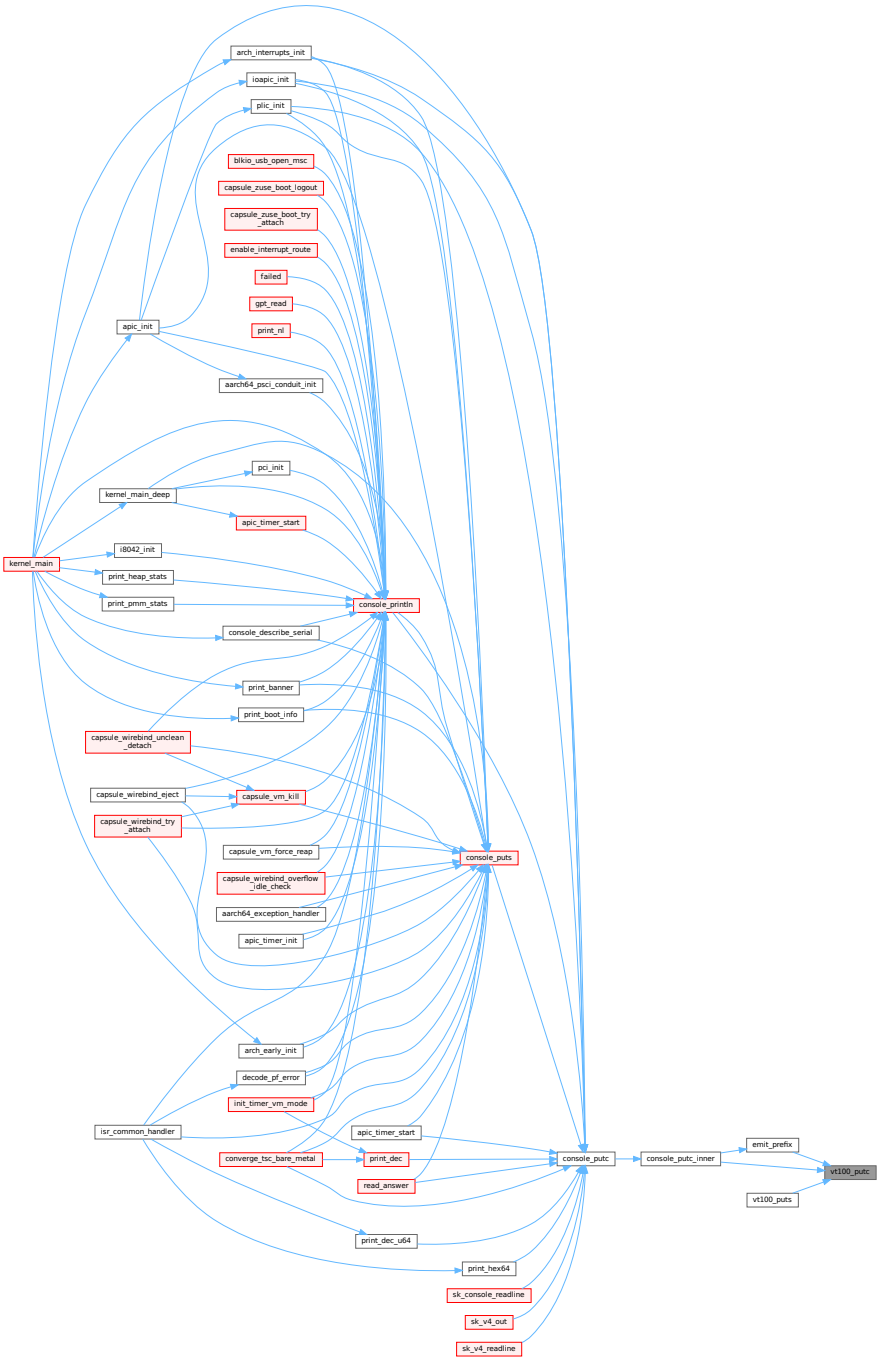
References cell_h(), VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, dispatch_csi(), erase_display(), fb_fill_rect(), fb_width(), g_vt, VT100State::initialized, VT100State::param_count, VT100State::param_has_value, VT100State::params, put_char(), reset_attrs(), reset_parser(), VT100State::rows, VT100State::saved_cx, VT100State::saved_cy, scroll_up(), set_cursor(), ST_CSI, ST_ESC, ST_ESC_CHAR, ST_ESC_HASH, ST_NORMAL, VT100State::state, and VT100_MAX_PARAMS.

Referenced by console_putc_inner(), emit_prefix(), and vt100_puts().

Here is the call graph for this function:



Here is the caller graph for this function:



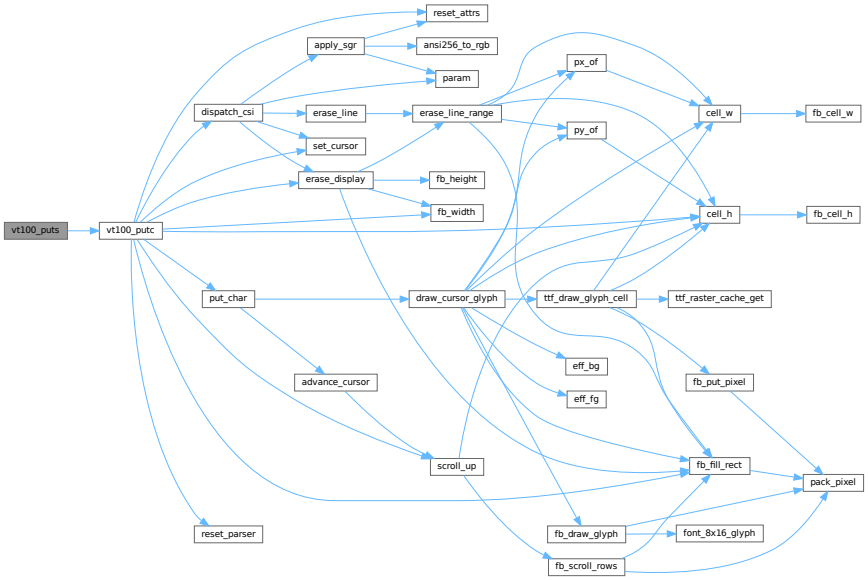
7.92.2.7 vt100_puts()

```
void vt100_puts (
            const char * s )
```

Feed a null-terminated string.

References vt100_putc().

Here is the call graph for this function:



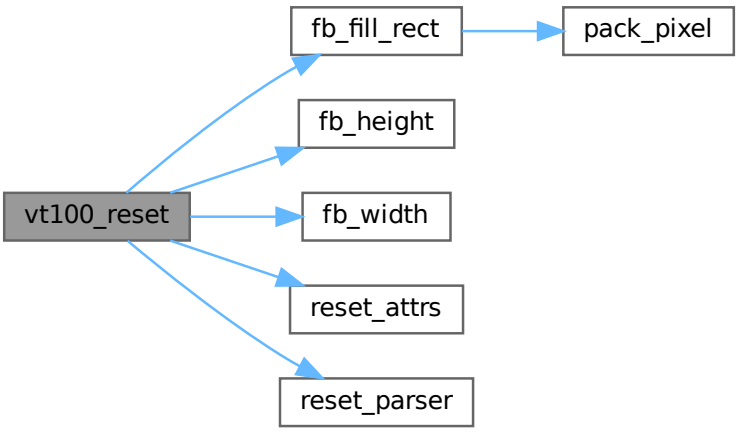
7.92.2.8 vt100_reset()

```
void vt100_reset (
            void  )
```

References VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, fb_fill_rect(), fb_height(), fb_width(), g_boot_ring_count, g_boot_rows, g_line_stride, g_ring_count, g_ring_head, g_scroll_offset, g_shadow, g_terminal_visible, g_vt, VT100State::initialized, reset_attrs(), reset_parser(), VT100State::rows, VT100State::saved_cx, and VT100State::saved_cy.

Referenced by console_fb_clear().

Here is the call graph for this function:



Here is the caller graph for this function:

7.92.2.9 vt100_rows()

```
uint32_t vt100_rows (
            void  )
```

Terminal height in character rows.

References g_vt, and VT100State::rows.

7.92.2.10 vt100_scroll_back()

```
void vt100_scroll_back (
            uint32_t n )
```

FABRIC-0.md item 4.4q: move the REPL scrollback view back/forward by `n` lines and redraw.

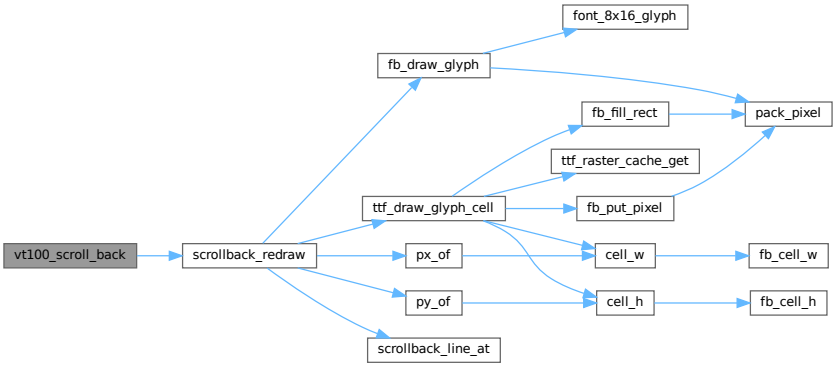
Offset 0 (the default, and where every call eventually returns to) is the live view – the same content already on screen. Clamped at both ends: back cannot pass the oldest stored line, forward cannot pass the live view. No-op (including no redraw) if the scrollback buffers failed to allocate at vt100_enable_ttf() time, or before TTF mode is active at all.

Known limitation: redraw uses the terminal's current default fg/bg, not each line's original SGR color state at the time it was printed (colors aren't stored per-cell) – text content is recovered exactly, color is not.

References g_boot_ring_count, g_boot_rows, g_ring, g_ring_count, g_scroll_offset, g_shadow, and scrollback_redraw().

Referenced by console_fb_scroll_back().

Here is the call graph for this function:



Here is the caller graph for this function:

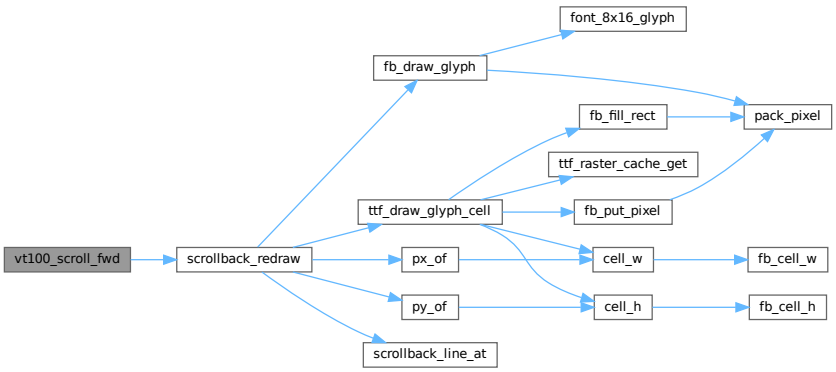
7.92.2.11 vt100_scroll_fwd()

```
void vt100_scroll_fwd (
            uint32_t n )
```

References g_ring, g_scroll_offset, g_shadow, and scrollback_redraw().

Referenced by console_fb_scroll_fwd().

Here is the call graph for this function:



Here is the caller graph for this function:

7.92.2.12 vt100_toggle_graphics()

```
void vt100_toggle_graphics (
            void  )
```

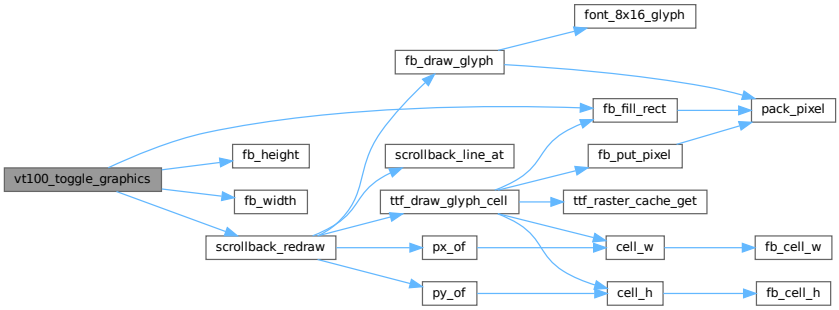
FABRIC-0.md item 4.4y-revised: toggle the full-screen vt100 terminal between visible (normal operation) and hidden (graphics mode – the terminal stops drawing, letting direct framebuffer/TTF-TEXT calls show through undisturbed).

Toggling back to visible does a full redraw of the terminal's current on-screen content, restoring it exactly as it was. No-op before TTF mode is active.

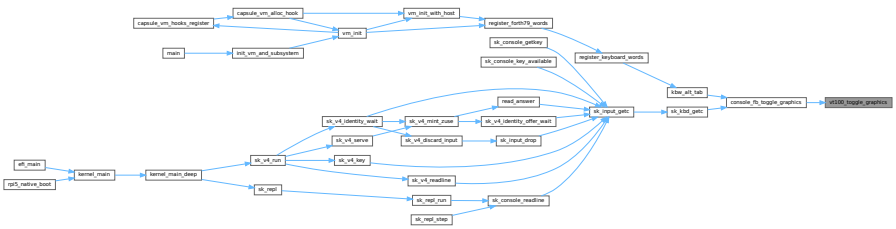
References VT100State::def_bg, fb_fill_rect(), fb_height(), fb_width(), g_glyph_mode, g_terminal_visible, g_ttf_ready, g_vt, scrollback_redraw(), and VT_GLYPH_TTF.

Referenced by console_fb_toggle_graphics().

Here is the call graph for this function:



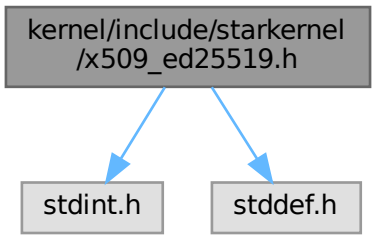
Here is the caller graph for this function:



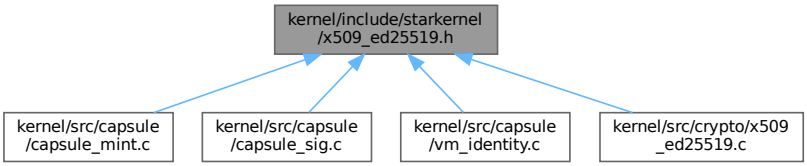
7.93 kernel/include/starkernel/x509_ed25519.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for x509_ed25519.h:



This graph shows which files directly or indirectly include this file:



Functions

- int x509_extract_ed25519_pubkey (const uint8_t *der, size_t der_len, uint8_t pubkey_out[32])
- int x509_verify_signature (const uint8_t *der, size_t der_len, const uint8_t issuer_pubkey[32])
- int x509_extract_serial (const uint8_t *der, size_t der_len, uint8_t *serial_out, size_t serial_out_cap, size_t *serial_len_out)
- size_t x509_build_user_cert (uint8_t *out, size_t out_cap, const uint8_t subject_pubkey[32], const uint8_t serial[16], const uint8_t issuer_seed[32])

7.93.1 Function Documentation

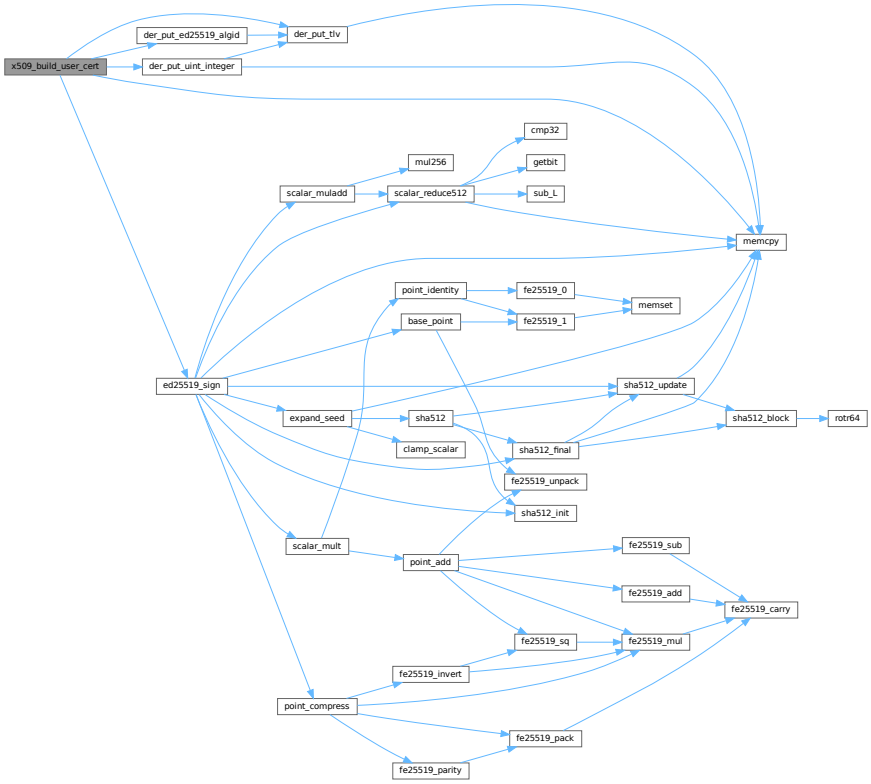
7.93.1.1 x509_build_user_cert()

```
size_t x509_build_user_cert (
            uint8_t * out,
            size_t out_cap,
            const uint8_t subject_pubkey[32],
            const uint8_t serial[16],
            const uint8_t issuer_seed[32] )
```

References der_put_ed25519_algid(), der_put_tlv(), der_put_uint_integer(), ed25519_sign(), and memcpy().

Referenced by capsule_mint_identity().

Here is the call graph for this function:



Here is the caller graph for this function:

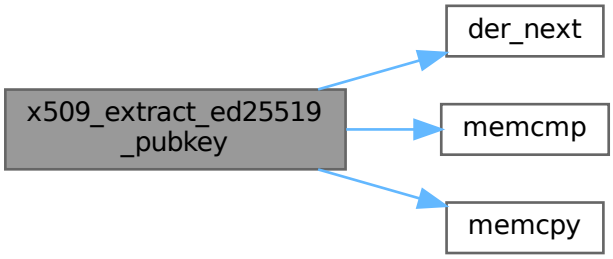
7.93.1.2 x509_extract_ed25519_pubkey()

```
int x509_extract_ed25519_pubkey (
            const uint8_t * der,
            size_t der_len,
            uint8_t pubkey_out[32] )
```

References der_next(), der_span_t::len, memcmp(), memcpy(), and der_span_t::p.

Referenced by find_and_cache_pubkey(), and vm_identity_from_cert().

Here is the call graph for this function:



Here is the caller graph for this function:

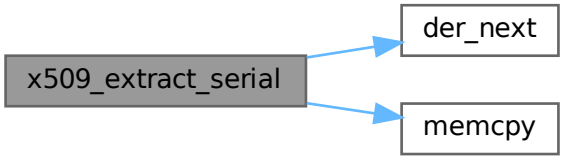
7.93.1.3 x509_extract_serial()

```
int x509_extract_serial (
            const uint8_t * der,
            size_t der_len,
            uint8_t * serial_out,
            size_t serial_out_cap,
            size_t * serial_len_out )
```

References der_next(), der_span_t::len, memcpy(), and der_span_t::p.

Referenced by vm_identity_from_cert().

Here is the call graph for this function:



Here is the caller graph for this function:

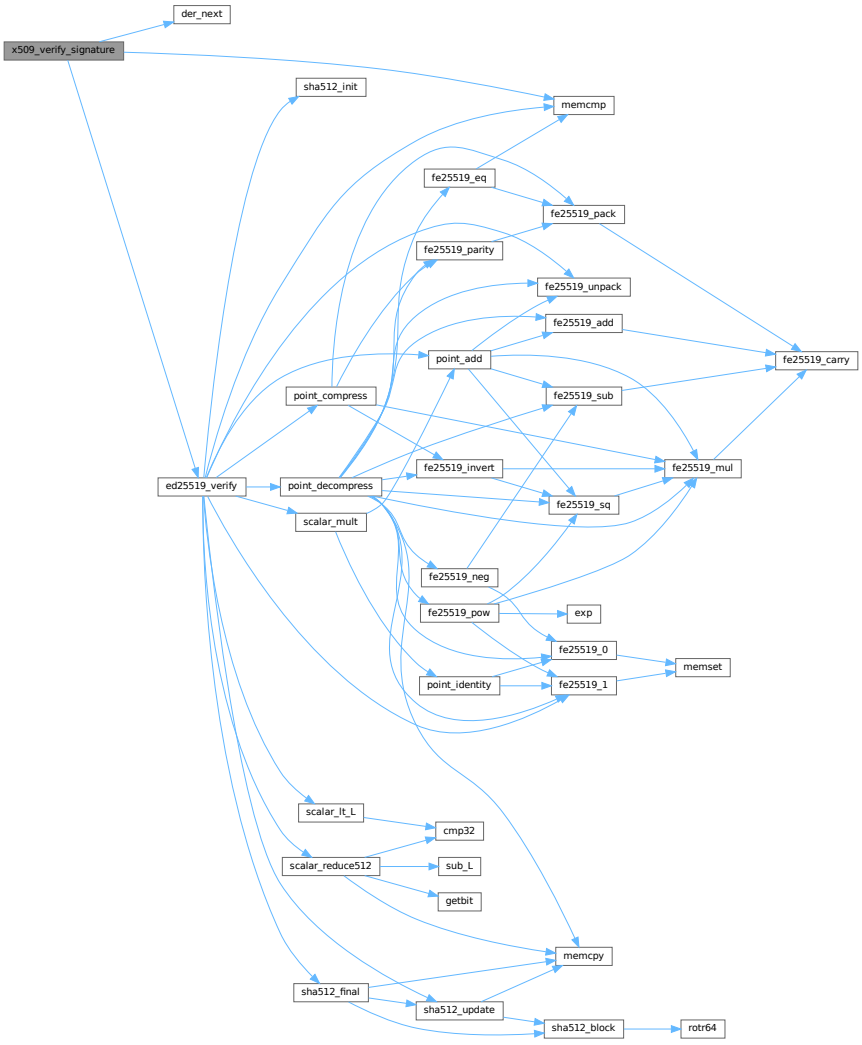
7.93.1.4 x509_verify_signature()

```
int x509_verify_signature (
            const uint8_t * der,
            size_t der_len,
            const uint8_t issuer_pubkey[32] )
```

References der_next(), ed25519_verify(), der_span_t::len, memcmp(), and der_span_t::p.

Referenced by vm_identity_from_cert().

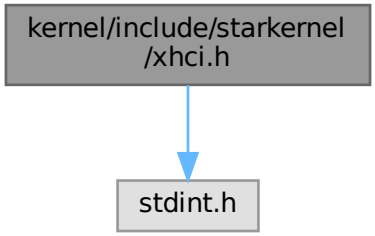
Here is the call graph for this function:



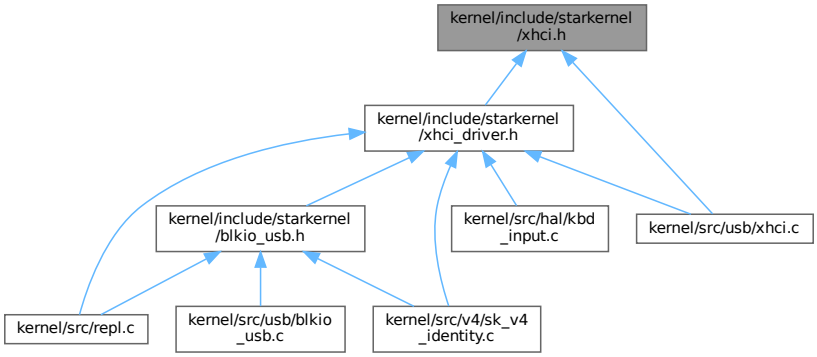
Here is the caller graph for this function:

7.94 kernel/include/starkernel/xhci.h File Reference

`#include <stdint.h>`
Include dependency graph for xhci.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xhci_cap_regs_t
- struct xhci_op_regs_t
- struct xhci_port_regs_t
- struct xhci_intr_regs_t
- struct xhci_runtime_regs_t

- struct xhci_trb_t
- struct usb_setup_packet_t
- struct usb_bot_cbw_t
- struct usb_bot_csw_t
- struct xhci_slot_ctx32_t
- struct xhci_ep_ctx32_t
- struct xhci_input_ctrl_ctx32_t

Macros

- #define XHCI_HCSPARAMS1_MAX_SLOTS(v) ((uint32_t)(v) & 0xFFu)
- #define XHCI_HCSPARAMS1_MAX_INTRS(v) (((uint32_t)(v) >> 8) & 0x7FFu)
- #define XHCI_HCSPARAMS1_MAX_PORTS(v) (((uint32_t)(v) >> 24) & 0xFFu)
- #define XHCI_HCSPARAMS2_MAX_SCRATCHPAD_BUFS(v) ((((uint32_t)(v) >> 21) & 0x1Fu) << 5 | (((uint32_t)(v) >> 27) & 0x1Fu))
- #define XHCI_HCCPARAMS1_CSZ(v) (((uint32_t)(v) >> 2) & 0x1u)
- #define XHCI_USBCMD_RUN (1u << 0) /* Run/Stop: 1 = run */
- #define XHCI_USBCMD_HCRST (1u << 1) /* HC Reset */
- #define XHCI_USBCMD_INTE (1u << 2) /* Interrupter Enable */
- #define XHCI_USBCMD_HSEE (1u << 3) /* Host System Error Enable */
- #define XHCI_USBSTS_HCH (1u << 0) /* HC Halted */
- #define XHCI_USBSTS_HSE (1u << 2) /* Host System Error */
- #define XHCI_USBSTS_EINT (1u << 3) /* Event Interrupt */
- #define XHCI_USBSTS_PCD (1u << 4) /* Port Change Detect */
- #define XHCI_USBSTS_CNR (1u << 11) /* Controller Not Ready */
- #define XHCI_USBSTS_HCE (1u << 12) /* Host Controller Error */
- #define XHCI_CRCR_RCS (1ull << 0) /* Ring Cycle State */
- #define XHCI_CRCR_CS (1ull << 1) /* Command Stop */
- #define XHCI_CRCR_CA (1ull << 2) /* Command Abort */
- #define XHCI_CRCR_CRR (1ull << 3) /* Command Ring Running (read-only) */
- #define XHCI_CRCR_PTR_MASK (~0x3Full) /* pointer must be 64-byte aligned */
- #define XHCI_CONFIG_MAX_SLOTS_EN(n) ((uint32_t)(n) & 0xFFu)
- #define XHCI_PORT_REGS_OFFSET 0x400u
- #define XHCI_PORTSC_CCS (1u << 0) /* Current Connect Status */
- #define XHCI_PORTSC_PED (1u << 1) /* Port Enabled/Disabled */
- #define XHCI_PORTSC_PR (1u << 4) /* Port Reset */
- #define XHCI_PORTSC_PLS_MASK (0xFu << 5) /* Port Link State */
- #define XHCI_PORTSC_PP (1u << 9) /* Port Power */
- #define XHCI_PORTSC_SPEED_MASK (0xFu << 10)
- #define XHCI_PORTSC_SPEED(v) (((uint32_t)(v) >> 10) & 0xFu) /* xHCI 1.2 spec table 7-13 speed IDs */
- #define XHCI_PORTSC_CSC (1u << 17) /* Connect Status Change */
- #define XHCI_PORTSC_PEC (1u << 18) /* Port Enabled/Disabled Change */
- #define XHCI_PORTSC_PRC (1u << 21) /* Port Reset Change */
- #define XHCI_IMAN_IP (1u << 0) /* Interrupt Pending */
- #define XHCI_IMAN_IE (1u << 1) /* Interrupt Enable */
- #define XHCI_ERDP_EHB (1ull << 3) /* Event Handler Busy */
- #define XHCI_ERDP_PTR_MASK (~0xFull) /* pointer occupies bits[63:4] */
- #define XHCI_DB_TARGET(ep) ((uint32_t)(ep) & 0xFFu) /* 0 = command ring */
- #define XHCI_DB_STREAM_ID(sid) (((uint32_t)(sid) & 0xFFFFu) << 16)
- #define XHCI_TRB_CONTROL_CYCLE (1u << 0) /* Cycle bit */
- #define XHCI_TRB_CONTROL_TYPE_SHIFT 10
- #define XHCI_TRB_CONTROL_TYPE_MASK (0x3Fu << XHCI_TRB_CONTROL_TYPE_SHIFT)
- #define XHCI_TRB_TYPE(ctrl) (((ctrl) & XHCI_TRB_CONTROL_TYPE_MASK) >> XHCI_TRB_CONTROL_TYPE_SHIFT)
- #define XHCI_TRB_TYPE_NORMAL 1 /* Transfer Ring, bulk/interrupt/isoch -- not EP0 */

- #define XHCI_TRB_TYPE_LINK 6 /* ring-wraparound marker, Command/Transfer Rings only */
- #define XHCI_TRB_TYPE_ENABLE_SLOT_CMD 9
- #define XHCI_TRB_TYPE_DISABLE_SLOT_CMD 10
- #define XHCI_TRB_TYPE_ADDRESS_DEVICE_CMD 11
- #define XHCI_TRB_TYPE_CONFIGURE_ENDPOINT_CMD 12
- #define XHCI_TRB_TYPE_RESET_ENDPOINT_CMD 14
- #define XHCI_TRB_TYPE_SET_TR_DEQUEUE_POINTER_CMD 16
- #define XHCI_TRB_TYPE_SETUP_STAGE 2 /* Transfer Ring, control transfers only */
- #define XHCI_TRB_TYPE_DATA_STAGE 3
- #define XHCI_TRB_TYPE_STATUS_STAGE 4
- #define XHCI_TRB_TYPE_TRANSFER_EVENT 32
- #define XHCI_TRB_TYPE_COMMAND_COMPLETION_EVT 33
- #define XHCI_TRB_TYPE_PORT_STATUS_CHANGE_EVT 34
- #define XHCI_TRB_CONTROL_TC (1u << 1) /* Toggle Cycle */
- #define XHCI_TRB_CONTROL_IDT (1u << 6)
- #define XHCI_TRB_CONTROL_IOC (1u << 5) /* Interrupt On Completion */
- #define XHCI_TRB_CONTROL_DIR_IN (1u << 16) /* Data/Status Stage: 1=IN, 0=OUT */
- #define XHCI_SETUP_TRT_NO_DATA 0u
- #define XHCI_SETUP_TRT_OUT_DATA 2u
- #define XHCI_SETUP_TRT_IN_DATA 3u
- #define XHCI_TRB_CONTROL_TRT_SHIFT 16 /* Setup Stage TRB only; Data/Status Stage overlays DIR at the same bit */
- #define USB_REQ_GET_DESCRIPTOR 6u
- #define USB_REQ_SET_CONFIGURATION 9u
- #define USB_REQ_CLEAR_FEATURE 1u
- #define USB_DESC_TYPE_DEVICE 1u
- #define USB_DESC_TYPE_CONFIG 2u
- #define USB_DIR_DEVICE_TO_HOST 0x80u
- #define USB_DIR_HOST_TO_DEVICE 0x00u
- #define USB_REQ_TYPE_STANDARD 0u
- #define USB_REQ_TYPE_CLASS 1u
- #define USB_RECIP_DEVICE 0u
- #define USB_RECIP_INTERFACE 1u
- #define USB_RECIP_ENDPOINT 2u
- #define USB_FEATURE_ENDPOINT_HALT 0u
- #define USB_BOT_MASS_STORAGE_RESET 0xFFu
- #define USB_DESC_OFF_LENGTH 0u
- #define USB_DESC_OFF_TYPE 1u
- #define USB_CONFIG_OFF_TOTAL_LENGTH 2u /* wTotalLength, 2 bytes, Configuration descriptor only */
- #define USB_CONFIG_OFF_CONFIG_VALUE 5u /* bConfigurationValue -- the value SET_CONFIGURATION needs in wValue */
- #define USB_DESC_TYPE_INTERFACE 4u
- #define USB_IFACE_OFF_CLASS 5u
- #define USB_IFACE_OFF_SUBCLASS 6u
- #define USB_IFACE_OFF_PROTOCOL 7u
- #define USB_CLASS_MASS_STORAGE 0x08u
- #define USB_SUBCLASS_SCSI 0x06u /* SCSI transparent command set */
- #define USB_PROTOCOL_BOT 0x50u /* Bulk-Only Transport */
- #define USB_IFACE_OFF_NUMBER 2u /* bInterfaceNumber */
- #define USB_CLASS_HID 0x03u
- #define USB_SUBCLASS_HID_BOOT 0x01u
- #define USB_PROTOCOL_HID_KEYBOARD 0x01u
- #define USB_HID_REQ_SET_PROTOCOL 0x0Bu /* class, interface recipient */
- #define USB_HID_PROTOCOL_BOOT 0u
- #define USB_DESC_TYPE_ENDPOINT 5u

- #define USB_EP_OFF_ADDRESS 2u /* bEndpointAddress: bit 7 = direction, bits 3:0 = number */
- #define USB_EP_OFF_ATTRIBUTES 3u /* bmAttributes: bits 1:0 = transfer type */
- #define USB_EP_OFF_MAX_PACKET_SIZE 4u /* wMaxPacketSize, 2 bytes */
- #define USB_EP_ADDR_DIR_MASK 0x80u
- #define USB_EP_ADDR_NUM_MASK 0x0Fu
- #define USB_EP_ATTR_TYPE_MASK 0x03u
- #define USB_EP_TYPE_BULK 0x02u
- #define USB_EP_TYPE_INTERRUPT 0x03u
- #define USB_EP_OFF_INTERVAL 6u /* bInterval */
- #define USB_BOT_CBW_SIGNATURE 0x43425355u /* "USBC", wire byte order U,S,B,C as an LE dword */
- #define USB_BOT_CBW_LENGTH 31u
- #define USB_BOT_CBW_FLAG_DATA_IN 0x80u /* bmCBWFlags: device-to-host data stage */
- #define USB_BOT_CBW_LUN_DEFAULT 0u /* no multi-LUN support -- single-LUN devices only */
- #define SCSI_CMD_READ10 0x28u
- #define SCSI_CDB_LEN_READ10 10u
- #define SCSI_CMD_WRITE10 0x2Au
- #define SCSI_CDB_LEN_WRITE10 10u
- #define SCSI_CMD_TEST_UNIT_READY 0x00u
- #define SCSI_CDB_LEN_TEST_UNIT_READY 6u
- #define XHCI_BOT_TUR_MAX_RETRIES 3u
- #define XHCI_BOT_STALL_MAX_RECOVERIES 2u
- #define SCSI_CMD_READ_CAPACITY10 0x25u
- #define SCSI_CDB_LEN_READ_CAPACITY10 10u
- #define SCSI_READ_CAPACITY10_DATA_LEN 8u
- #define USB_BOT_CSW_SIGNATURE 0x53425355u /* "USBS", wire byte order U,S,B,S as an LE dword */
- #define USB_BOT_CSW_LENGTH 13u
- #define USB_BOT_CSW_STATUS_PASS 0u
- #define USB_BOT_CSW_STATUS_FAILED 1u
- #define USB_BOT_CSW_STATUS_PHASE_ERROR 2u
- #define XHCI_EVT_COMPLETION_CODE(status) (((uint32_t)(status) >> 24) & 0xFFu)
- #define XHCI_EVT_SLOT_ID(control) (((uint32_t)(control) >> 24) & 0xFFu)
- #define XHCI_COMPLETION_CODE_SUCCESS 1u
- #define XHCI_COMPLETION_CODE_STALL_ERROR 6u
- #define XHCI_COMPLETION_CODE_SHORT_PACKET 13u
- #define XHCI_EVT_EP_ID(control) (((uint32_t)(control) >> 16) & 0x1Fu)
- #define XHCI_PSC_EVT_PORT_ID(parameter) (((uint32_t)(parameter) >> 24) & 0xFFu)
- #define XHCI_SLOT_CTX_SPEED_SHIFT 20
- #define XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT 27
- #define XHCI_SLOT_CTX_ROOT_PORT_SHIFT 16
- #define XHCI_SLOT_CTX_INTR_TARGET_SHIFT 22
- #define XHCI_EP_CTX_TYPE_SHIFT 3
- #define XHCI_EP_CTX_TYPE_CONTROL_BIDI 4u /* EP0 */
- #define XHCI_EP_CTX_TYPE_BULK_OUT 2u /* xHCI 1.2 spec table 6-9 */
- #define XHCI_EP_CTX_TYPE_BULK_IN 6u
- #define XHCI_EP_CTX_TYPE_INTERRUPT_IN 7u
- #define XHCI_EP_CTX_INTERVAL_SHIFT 16 /* dword0: period 2^Interval x 125 us */
- #define XHCI_EP_CTX_CERR_SHIFT 1
- #define XHCI_EP_CTX_MAX_PACKET_SHIFT 16
- #define XHCI_EP_ADDR_TO_DCI(addr)
- #define XHCI_INPUT_CTRL_ADD_SLOT (1u << 0)
- #define XHCI_INPUT_CTRL_ADD_EP0 (1u << 1)
- #define XHCI_RING_TRB_COUNT 256u
- #define XHCI_RING_BYTES (XHCI_RING_TRB_COUNT * sizeof(xhci_trb_t))
- #define XHCI_EVT_RING_MAX_DRAIN XHCI_RING_TRB_COUNT
- #define XHCI_MAX_TRACKED_PORTS 32u

Typedefs

- typedef volatile uint32_t xhci_doorbell_t

7.94.1 Macro Definition Documentation

7.94.1.1 SCSI_CDB_LEN_READ10

```
#define SCSI_CDB_LEN_READ10 10u
```

7.94.1.2 SCSI_CDB_LEN_READ_CAPACITY10

```
#define SCSI_CDB_LEN_READ_CAPACITY10 10u
```

7.94.1.3 SCSI_CDB_LEN_TEST_UNIT_READY

```
#define SCSI_CDB_LEN_TEST_UNIT_READY 6u
```

7.94.1.4 SCSI_CDB_LEN_WRITE10

```
#define SCSI_CDB_LEN_WRITE10 10u
```

7.94.1.5 SCSI_CMD_READ10

```
#define SCSI_CMD_READ10 0x28u
```

7.94.1.6 SCSI_CMD_READ_CAPACITY10

```
#define SCSI_CMD_READ_CAPACITY10 0x25u
```

7.94.1.7 SCSI_CMD_TEST_UNIT_READY

```
#define SCSI_CMD_TEST_UNIT_READY 0x00u
```

7.94.1.8 SCSI_CMD_WRITE10

```
#define SCSI_CMD_WRITE10 0x2Au
```

7.94.1.9 SCSI_READ_CAPACITY10_DATA_LEN

```
#define SCSI_READ_CAPACITY10_DATA_LEN 8u
```

7.94.1.10 USB_BOT_CBW_FLAG_DATA_IN

```
#define USB_BOT_CBW_FLAG_DATA_IN 0x80u /* bmCBWFlags:  device-to-host data stage */
```

7.94.1.11 USB_BOT_CBW_LENGTH

```
#define USB_BOT_CBW_LENGTH 31u
```

7.94.1.12 USB_BOT_CBW_LUN_DEFAULT

```
#define USB_BOT_CBW_LUN_DEFAULT 0u /* no multi-LUN support -- single-LUN devices only */
```

7.94.1.13 USB_BOT_CBW_SIGNATURE

```
#define USB_BOT_CBW_SIGNATURE 0x43425355u /* "USBC", wire byte order U,S,B,C as an LE dword */
```

7.94.1.14 USB_BOT_CSW_LENGTH

```
#define USB_BOT_CSW_LENGTH 13u
```

7.94.1.15 USB_BOT_CSW_SIGNATURE

```
#define USB_BOT_CSW_SIGNATURE 0x53425355u /* "USBS", wire byte order U,S,B,S as an LE dword */
```

7.94.1.16 USB_BOT_CSW_STATUS_FAILED

```
#define USB_BOT_CSW_STATUS_FAILED 1u
```

7.94.1.17 USB_BOT_CSW_STATUS_PASS

```
#define USB_BOT_CSW_STATUS_PASS 0u
```

7.94.1.18 USB_BOT_CSW_STATUS_PHASE_ERROR

```
#define USB_BOT_CSW_STATUS_PHASE_ERROR 2u
```

7.94.1.19 USB_BOT_MASS_STORAGE_RESET

```
#define USB_BOT_MASS_STORAGE_RESET 0xFFu
```

7.94.1.20 USB_CLASS_HID

```
#define USB_CLASS_HID 0x03u
```

7.94.1.21 USB_CLASS_MASS_STORAGE

```
#define USB_CLASS_MASS_STORAGE 0x08u
```

7.94.1.22 USB_CONFIG_OFF_CONFIG_VALUE

```
#define USB_CONFIG_OFF_CONFIG_VALUE 5u /* bConfigurationValue -- the value SET_CONFIGURATION
needs in wValue */
```

7.94.1.23 USB_CONFIG_OFF_TOTAL_LENGTH

```
#define USB_CONFIG_OFF_TOTAL_LENGTH 2u /* wTotalLength, 2 bytes, Configuration descriptor only
*/
```

7.94.1.24 USB_DESC_OFF_LENGTH

```
#define USB_DESC_OFF_LENGTH 0u
```

7.94.1.25 USB_DESC_OFF_TYPE

```
#define USB_DESC_OFF_TYPE 1u
```

7.94.1.26 USB_DESC_TYPE_CONFIG

```
#define USB_DESC_TYPE_CONFIG 2u
```

7.94.1.27 USB_DESC_TYPE_DEVICE

```
#define USB_DESC_TYPE_DEVICE 1u
```

7.94.1.28 USB_DESC_TYPE_ENDPOINT

```
#define USB_DESC_TYPE_ENDPOINT 5u
```

7.94.1.29 USB_DESC_TYPE_INTERFACE

```
#define USB_DESC_TYPE_INTERFACE 4u
```

7.94.1.30 USB_DIR_DEVICE_TO_HOST

```
#define USB_DIR_DEVICE_TO_HOST 0x80u
```

7.94.1.31 USB_DIR_HOST_TO_DEVICE

```
#define USB_DIR_HOST_TO_DEVICE 0x00u
```

7.94.1.32 USB_EP_ADDR_DIR_MASK

```
#define USB_EP_ADDR_DIR_MASK 0x80u
```

7.94.1.33 USB_EP_ADDR_NUM_MASK

```
#define USB_EP_ADDR_NUM_MASK 0x0Fu
```

7.94.1.34 USB_EP_ATTR_TYPE_MASK

```
#define USB_EP_ATTR_TYPE_MASK 0x03u
```

7.94.1.35 USB_EP_OFF_ADDRESS

```
#define USB_EP_OFF_ADDRESS 2u /* bEndpointAddress:  bit 7 = direction, bits 3:0 = number */
```

7.94.1.36 USB_EP_OFF_ATTRIBUTES

```
#define USB_EP_OFF_ATTRIBUTES 3u /* bmAttributes:  bits 1:0 = transfer type */
```

7.94.1.37 USB_EP_OFF_INTERVAL

```
#define USB_EP_OFF_INTERVAL 6u /* bInterval */
```

7.94.1.38 USB_EP_OFF_MAX_PACKET_SIZE

```
#define USB_EP_OFF_MAX_PACKET_SIZE 4u /* wMaxPacketSize, 2 bytes */
```

7.94.1.39 USB_EP_TYPE_BULK

```
#define USB_EP_TYPE_BULK 0x02u
```

7.94.1.40 USB_EP_TYPE_INTERRUPT

```
#define USB_EP_TYPE_INTERRUPT 0x03u
```

7.94.1.41 USB_FEATURE_ENDPOINT_HALT

```
#define USB_FEATURE_ENDPOINT_HALT 0u
```

7.94.1.42 USB_HID_PROTOCOL_BOOT

```
#define USB_HID_PROTOCOL_BOOT 0u
```

7.94.1.43 USB_HID_REQ_SET_PROTOCOL

```
#define USB_HID_REQ_SET_PROTOCOL 0x0Bu /* class, interface recipient */
```

7.94.1.44 USB_IFACE_OFF_CLASS

```
#define USB_IFACE_OFF_CLASS 5u
```

7.94.1.45 USB_IFACE_OFF_NUMBER

```
#define USB_IFACE_OFF_NUMBER 2u /* bInterfaceNumber */
```

7.94.1.46 USB_IFACE_OFF_PROTOCOL

```
#define USB_IFACE_OFF_PROTOCOL 7u
```

7.94.1.47 USB_IFACE_OFF_SUBCLASS

```
#define USB_IFACE_OFF_SUBCLASS 6u
```

7.94.1.48 USB_PROTOCOL_BOT

```
#define USB_PROTOCOL_BOT 0x50u /* Bulk-Only Transport */
```

7.94.1.49 USB_PROTOCOL_HID_KEYBOARD

```
#define USB_PROTOCOL_HID_KEYBOARD 0x01u
```

7.94.1.50 USB_RECIP_DEVICE

```
#define USB_RECIP_DEVICE 0u
```

7.94.1.51 USB_RECIP_ENDPOINT

```
#define USB_RECIP_ENDPOINT 2u
```

7.94.1.52 USB_RECIP_INTERFACE

```
#define USB_RECIP_INTERFACE 1u
```

7.94.1.53 USB_REQ_CLEAR_FEATURE

```
#define USB_REQ_CLEAR_FEATURE 1u
```

7.94.1.54 USB_REQ_GET_DESCRIPTOR

```
#define USB_REQ_GET_DESCRIPTOR 6u
```

7.94.1.55 USB_REQ_SET_CONFIGURATION

```
#define USB_REQ_SET_CONFIGURATION 9u
```

7.94.1.56 USB_REQ_TYPE_CLASS

```
#define USB_REQ_TYPE_CLASS 1u
```

7.94.1.57 USB_REQ_TYPE_STANDARD

```
#define USB_REQ_TYPE_STANDARD 0u
```

7.94.1.58 USB_SUBCLASS_HID_BOOT

```
#define USB_SUBCLASS_HID_BOOT 0x01u
```

7.94.1.59 USB_SUBCLASS_SCSI

```
#define USB_SUBCLASS_SCSI 0x06u /* SCSI transparent command set */
```

7.94.1.60 XHCI_BOT_STALL_MAX_RECOVERIES

```
#define XHCI_BOT_STALL_MAX_RECOVERIES 2u
```

7.94.1.61 XHCI_BOT_TUR_MAX_RETRIES

```
#define XHCI_BOT_TUR_MAX_RETRIES 3u
```

7.94.1.62 XHCI_COMPLETION_CODE_SHORT_PACKET

```
#define XHCI_COMPLETION_CODE_SHORT_PACKET 13u
```

7.94.1.63 XHCI_COMPLETION_CODE_STALL_ERROR

```
#define XHCI_COMPLETION_CODE_STALL_ERROR 6u
```

7.94.1.64 XHCI_COMPLETION_CODE_SUCCESS

```
#define XHCI_COMPLETION_CODE_SUCCESS 1u
```

7.94.1.65 XHCI_CONFIG_MAX_SLOTS_EN

```
#define XHCI_CONFIG_MAX_SLOTS_EN(
              n ) ((uint32_t)(n) & 0xFFu)
```

7.94.1.66 XHCI_CRCR_CA

```
#define XHCI_CRCR_CA (1ull << 2) /* Command Abort */
```

7.94.1.67 XHCI_CRCR_CRR

```
#define XHCI_CRCR_CRR (1ull << 3) /* Command Ring Running (read-only) */
```

7.94.1.68 XHCI_CRCR_CS

```
#define XHCI_CRCR_CS (1ull << 1) /* Command Stop */
```

7.94.1.69 XHCI_CRCR_PTR_MASK

```
#define XHCI_CRCR_PTR_MASK (~0x3Full) /* pointer must be 64-byte aligned */
```

7.94.1.70 XHCI_CRCR_RCS

```
#define XHCI_CRCR_RCS (1ull << 0) /* Ring Cycle State */
```

7.94.1.71 XHCI_DB_STREAM_ID

```
#define XHCI_DB_STREAM_ID(
              sid ) (((uint32_t)(sid) & 0xFFFFu) << 16)
```

7.94.1.72 XHCI_DB_TARGET

```
#define XHCI_DB_TARGET(
              ep ) ((uint32_t)(ep) & 0xFFu) /* 0 = command ring */
```

7.94.1.73 XHCI_EP_ADDR_TO_DCI

```
#define XHCI_EP_ADDR_TO_DCI(
              addr )
```

Value:

```
    ((2u * ((uint32_t)(addr) & USB_EP_ADDR_NUM_MASK)) + \
     (((uint32_t)(addr) & USB_EP_ADDR_DIR_MASK) ? 1u : 0u))
```

7.94.1.74 XHCI_EP_CTX_CERR_SHIFT

```
#define XHCI_EP_CTX_CERR_SHIFT 1
```

7.94.1.75 XHCI_EP_CTX_INTERVAL_SHIFT

```
#define XHCI_EP_CTX_INTERVAL_SHIFT 16 /* dword0:  period 2^Interval x 125 us */
```

7.94.1.76 XHCI_EP_CTX_MAX_PACKET_SHIFT

```
#define XHCI_EP_CTX_MAX_PACKET_SHIFT 16
```

7.94.1.77 XHCI_EP_CTX_TYPE_BULK_IN

```
#define XHCI_EP_CTX_TYPE_BULK_IN 6u
```

7.94.1.78 XHCI_EP_CTX_TYPE_BULK_OUT

```
#define XHCI_EP_CTX_TYPE_BULK_OUT 2u /* xHCI 1.2 spec table 6-9 */
```

7.94.1.79 XHCI_EP_CTX_TYPE_CONTROL_BIDI

```
#define XHCI_EP_CTX_TYPE_CONTROL_BIDI 4u /* EP0 */
```

7.94.1.80 XHCI_EP_CTX_TYPE_INTERRUPT_IN

```
#define XHCI_EP_CTX_TYPE_INTERRUPT_IN 7u
```

7.94.1.81 XHCI_EP_CTX_TYPE_SHIFT

```
#define XHCI_EP_CTX_TYPE_SHIFT 3
```

7.94.1.82 XHCI_ERDP_EHB

```
#define XHCI_ERDP_EHB (1ull << 3) /* Event Handler Busy */
```

7.94.1.83 XHCI_ERDP_PTR_MASK

```
#define XHCI_ERDP_PTR_MASK (~0xFull) /* pointer occupies bits[63:4] */
```

7.94.1.84 XHCI_EVT_COMPLETION_CODE

```
#define XHCI_EVT_COMPLETION_CODE(
             status ) (((uint32_t)(status) >> 24) & 0xFFu)
```

7.94.1.85 XHCI_EVT_EP_ID

```
#define XHCI_EVT_EP_ID(
             control ) (((uint32_t)(control) >> 16) & 0x1Fu)
```

7.94.1.86 XHCI_EVT_RING_MAX_DRAIN

```
#define XHCI_EVT_RING_MAX_DRAIN XHCI_RING_TRB_COUNT
```

7.94.1.87 XHCI_EVT_SLOT_ID

```
#define XHCI_EVT_SLOT_ID(
             control ) (((uint32_t)(control) >> 24) & 0xFFu)
```

7.94.1.88 XHCI_HCCPARAMS1_CSZ

```
#define XHCI_HCCPARAMS1_CSZ(
             v ) (((uint32_t)(v) >> 2) & 0x1u)
```

7.94.1.89 XHCI_HCSPARAMS1_MAX_INTRS

```
#define XHCI_HCSPARAMS1_MAX_INTRS(
             v ) (((uint32_t)(v) >> 8) & 0x7FFu)
```

7.94.1.90 XHCI_HCSPARAMS1_MAX_PORTS

```
#define XHCI_HCSPARAMS1_MAX_PORTS(
             v ) (((uint32_t)(v) >> 24) & 0xFFu)
```

7.94.1.91 XHCI_HCSPARAMS1_MAX_SLOTS

```
#define XHCI_HCSPARAMS1_MAX_SLOTS(
             v ) ((uint32_t)(v) & 0xFFu)
```

7.94.1.92 XHCI_HCSPARAMS2_MAX_SCRATCHPAD_BUFS

```
#define XHCI_HCSPARAMS2_MAX_SCRATCHPAD_BUFS(
             v )  ((((uint32_t)(v) >> 21) & 0x1Fu) << 5 | (((uint32_t)(v) >> 27) & 0x1Fu))
```

7.94.1.93 XHCI_IMAN_IE

```
#define XHCI_IMAN_IE (1u << 1) /* Interrupt Enable */
```

7.94.1.94 XHCI_IMAN_IP

```
#define XHCI_IMAN_IP (1u << 0) /* Interrupt Pending */
```

7.94.1.95 XHCI_INPUT_CTRL_ADD_EP0

```
#define XHCI_INPUT_CTRL_ADD_EP0 (1u << 1)
```

7.94.1.96 XHCI_INPUT_CTRL_ADD_SLOT

```
#define XHCI_INPUT_CTRL_ADD_SLOT (1u << 0)
```

7.94.1.97 XHCI_MAX_TRACKED_PORTS

```
#define XHCI_MAX_TRACKED_PORTS 32u
```

7.94.1.98 XHCI_PORT_REGS_OFFSET

```
#define XHCI_PORT_REGS_OFFSET 0x400u
```

7.94.1.99 XHCI_PORTSC_CCS

```
#define XHCI_PORTSC_CCS (1u << 0) /* Current Connect Status */
```

7.94.1.100 XHCI_PORTSC_CSC

```
#define XHCI_PORTSC_CSC (1u << 17) /* Connect Status Change */
```

7.94.1.101 XHCI_PORTSC_PEC

```
#define XHCI_PORTSC_PEC (1u << 18) /* Port Enabled/Disabled Change */
```

7.94.1.102 XHCI_PORTSC_PED

```
#define XHCI_PORTSC_PED (1u << 1) /* Port Enabled/Disabled */
```

7.94.1.103 XHCI_PORTSC_PLS_MASK

```
#define XHCI_PORTSC_PLS_MASK (0xFu << 5) /* Port Link State */
```

7.94.1.104 XHCI_PORTSC_PP

```
#define XHCI_PORTSC_PP (1u << 9) /* Port Power */
```

7.94.1.105 XHCI_PORTSC_PR

```
#define XHCI_PORTSC_PR (1u << 4) /* Port Reset */
```

7.94.1.106 XHCI_PORTSC_PRC

```
#define XHCI_PORTSC_PRC (1u << 21) /* Port Reset Change */
```

7.94.1.107 XHCI_PORTSC_SPEED

```
#define XHCI_PORTSC_SPEED(
             v ) (((uint32_t)(v) >> 10) & 0xFu) /* xHCI 1.2 spec table 7-13 speed IDs */
```

7.94.1.108 XHCI_PORTSC_SPEED_MASK

```
#define XHCI_PORTSC_SPEED_MASK (0xFu << 10)
```

7.94.1.109 XHCI_PSC_EVT_PORT_ID

```
#define XHCI_PSC_EVT_PORT_ID(
             parameter ) (((uint32_t)(parameter) >> 24) & 0xFFu)
```

7.94.1.110 XHCI_RING_BYTES

```
#define XHCI_RING_BYTES (XHCI_RING_TRB_COUNT * sizeof(xhci_trb_t))
```

7.94.1.111 XHCI_RING_TRB_COUNT

```
#define XHCI_RING_TRB_COUNT 256u
```

7.94.1.112 XHCI_SETUP_TRT_IN_DATA

```
#define XHCI_SETUP_TRT_IN_DATA 3u
```

7.94.1.113 XHCI_SETUP_TRT_NO_DATA

```
#define XHCI_SETUP_TRT_NO_DATA 0u
```

7.94.1.114 XHCI_SETUP_TRT_OUT_DATA

```
#define XHCI_SETUP_TRT_OUT_DATA 2u
```

7.94.1.115 XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT

```
#define XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT 27
```

7.94.1.116 XHCI_SLOT_CTX_INTR_TARGET_SHIFT

```
#define XHCI_SLOT_CTX_INTR_TARGET_SHIFT 22
```

7.94.1.117 XHCI_SLOT_CTX_ROOT_PORT_SHIFT

```
#define XHCI_SLOT_CTX_ROOT_PORT_SHIFT 16
```

7.94.1.118 XHCI_SLOT_CTX_SPEED_SHIFT

```
#define XHCI_SLOT_CTX_SPEED_SHIFT 20
```

7.94.1.119 XHCI_TRB_CONTROL_CYCLE

```
#define XHCI_TRB_CONTROL_CYCLE (1u << 0) /* Cycle bit */
```

7.94.1.120 XHCI_TRB_CONTROL_DIR_IN

```
#define XHCI_TRB_CONTROL_DIR_IN (1u << 16) /* Data/Status Stage:  1=IN, 0=OUT */
```

7.94.1.121 XHCI_TRB_CONTROL_IDT

```
#define XHCI_TRB_CONTROL_IDT (1u << 6)
```

7.94.1.122 XHCI_TRB_CONTROL_IOC

```
#define XHCI_TRB_CONTROL_IOC (1u << 5) /* Interrupt On Completion */
```

7.94.1.123 XHCI_TRB_CONTROL_TC

```
#define XHCI_TRB_CONTROL_TC (1u << 1) /* Toggle Cycle */
```

7.94.1.124 XHCI_TRB_CONTROL_TRT_SHIFT

```
#define XHCI_TRB_CONTROL_TRT_SHIFT 16 /* Setup Stage TRB only; Data/Status Stage overlays DIR
at the same bit */
```

7.94.1.125 XHCI_TRB_CONTROL_TYPE_MASK

```
#define XHCI_TRB_CONTROL_TYPE_MASK (0x3Fu << XHCI_TRB_CONTROL_TYPE_SHIFT)
```

7.94.1.126 XHCI_TRB_CONTROL_TYPE_SHIFT

```
#define XHCI_TRB_CONTROL_TYPE_SHIFT 10
```

7.94.1.127 XHCI_TRB_TYPE

```
#define XHCI_TRB_TYPE(
             ctrl ) (((ctrl) & XHCI_TRB_CONTROL_TYPE_MASK) >> XHCI_TRB_CONTROL_TYPE_SHIFT)
```

7.94.1.128 XHCI_TRB_TYPE_ADDRESS_DEVICE_CMD

```
#define XHCI_TRB_TYPE_ADDRESS_DEVICE_CMD 11
```

7.94.1.129 XHCI_TRB_TYPE_COMMAND_COMPLETION_EVT

```
#define XHCI_TRB_TYPE_COMMAND_COMPLETION_EVT 33
```

7.94.1.130 XHCI_TRB_TYPE_CONFIGURE_ENDPOINT_CMD

```
#define XHCI_TRB_TYPE_CONFIGURE_ENDPOINT_CMD 12
```

7.94.1.131 XHCI_TRB_TYPE_DATA_STAGE

```
#define XHCI_TRB_TYPE_DATA_STAGE 3
```

7.94.1.132 XHCI_TRB_TYPE_DISABLE_SLOT_CMD

```
#define XHCI_TRB_TYPE_DISABLE_SLOT_CMD 10
```

7.94.1.133 XHCI_TRB_TYPE_ENABLE_SLOT_CMD

```
#define XHCI_TRB_TYPE_ENABLE_SLOT_CMD 9
```

7.94.1.134 XHCI_TRB_TYPE_LINK

```
#define XHCI_TRB_TYPE_LINK 6 /* ring-wraparound marker, Command/Transfer Rings only */
```

7.94.1.135 XHCI_TRB_TYPE_NORMAL

```
#define XHCI_TRB_TYPE_NORMAL 1 /* Transfer Ring, bulk/interrupt/isoch -- not EP0 */
```

7.94.1.136 XHCI_TRB_TYPE_PORT_STATUS_CHANGE_EVT

```
#define XHCI_TRB_TYPE_PORT_STATUS_CHANGE_EVT 34
```

7.94.1.137 XHCI_TRB_TYPE_RESET_ENDPOINT_CMD

```
#define XHCI_TRB_TYPE_RESET_ENDPOINT_CMD 14
```

7.94.1.138 XHCI_TRB_TYPE_SET_TR_DEQUEUE_POINTER_CMD

```
#define XHCI_TRB_TYPE_SET_TR_DEQUEUE_POINTER_CMD 16
```

7.94.1.139 XHCI_TRB_TYPE_SETUP_STAGE

```
#define XHCI_TRB_TYPE_SETUP_STAGE 2 /* Transfer Ring, control transfers only */
```

7.94.1.140 XHCI_TRB_TYPE_STATUS_STAGE

```
#define XHCI_TRB_TYPE_STATUS_STAGE 4
```

7.94.1.141 XHCI_TRB_TYPE_TRANSFER_EVENT

```
#define XHCI_TRB_TYPE_TRANSFER_EVENT 32
```

7.94.1.142 XHCI_USBCMD_HCRST

```
#define XHCI_USBCMD_HCRST (1u << 1) /* HC Reset */
```

7.94.1.143 XHCI_USBCMD_HSEE

```
#define XHCI_USBCMD_HSEE (1u << 3) /* Host System Error Enable */
```

7.94.1.144 XHCI_USBCMD_INTE

```
#define XHCI_USBCMD_INTE (1u << 2) /* Interrupter Enable */
```

7.94.1.145 XHCI_USBCMD_RUN

```
#define XHCI_USBCMD_RUN (1u << 0) /* Run/Stop:  1 = run */
```

7.94.1.146 XHCI_USBSTS_CNR

```
#define XHCI_USBSTS_CNR (1u << 11) /* Controller Not Ready */
```

7.94.1.147 XHCI_USBSTS_EINT

```
#define XHCI_USBSTS_EINT (1u << 3) /* Event Interrupt */
```

7.94.1.148 XHCI_USBSTS_HCE

```
#define XHCI_USBSTS_HCE (1u << 12) /* Host Controller Error */
```

7.94.1.149 XHCI_USBSTS_HCH

```
#define XHCI_USBSTS_HCH (1u << 0) /* HC Halted */
```

7.94.1.150 XHCI_USBSTS_HSE

```
#define XHCI_USBSTS_HSE (1u << 2) /* Host System Error */
```

7.94.1.151 XHCI_USBSTS_PCD

```
#define XHCI_USBSTS_PCD (1u << 4) /* Port Change Detect */
```

7.94.2 Typedef Documentation

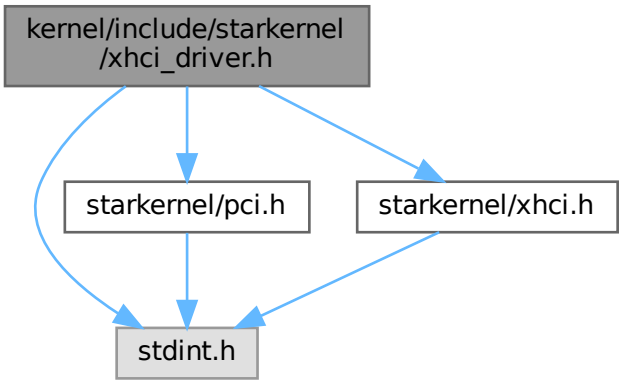
7.94.2.1 xhci_doorbell_t

```
typedef volatile uint32_t xhci_doorbell_t
```

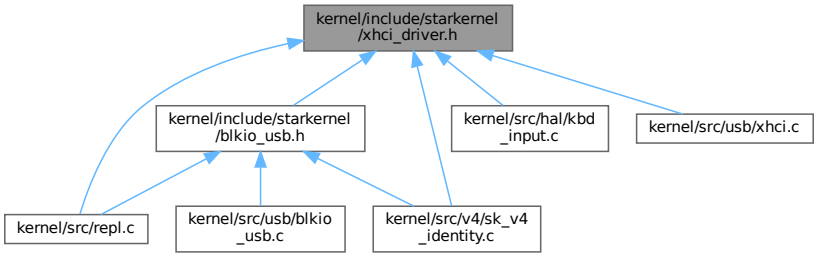
7.95 kernel/include/starkernel/xhci_driver.h File Reference

```
#include <stdint.h>
#include "starkernel/pci.h"
#include "starkernel/xhci.h"
```

Include dependency graph for xhci_driver.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xhci_msc_slot_t
- struct xhci_dev_t

Macros

- #define XHCI_MAX_CONTROLLERS 8u

Enumerations

- enum xhci_transfer_purpose_t {
  XHCI_XFER_NONE = 0 , XHCI_XFER_DEVICE_DESC , XHCI_XFER_CONFIG_DESC_SHORT ,
  XHCI_XFER_CONFIG_DESC_FULL ,
  XHCI_XFER_SET_CONFIG , XHCI_XFER_CBW_SENT , XHCI_XFER_BOT_DATA_IN , XHCI_XFER_BOT_DATA_OUT
  ,
  XHCI_XFER_CSW_RECEIVED , XHCI_XFER_CLEAR_HALT , XHCI_XFER_BOT_RESET , XHCI_XFER_HID_SET_PROTOC
  ,
  XHCI_XFER_HID_SET_LEDS }
- enum xhci_next_action_t {
  XHCI_NEXT_ACTION_NONE = 0 , XHCI_NEXT_ACTION_GET_DEVICE_DESC , XHCI_NEXT_ACTION_GET_CONFIG_DES
  , XHCI_NEXT_ACTION_CONFIGURE_ENDPOINT ,
  XHCI_NEXT_ACTION_SET_CONFIG , XHCI_NEXT_ACTION_BOT_DATA_IN , XHCI_NEXT_ACTION_BOT_DATA_OUT
  , XHCI_NEXT_ACTION_BOT_CSW_RECEIVE ,
  XHCI_NEXT_ACTION_BOT_SEND_TUR , XHCI_NEXT_ACTION_BOT_SEND_READ10 , XHCI_NEXT_ACTION_BOT_SEND
  , XHCI_NEXT_ACTION_BOT_SEND_WRITE10 ,
  XHCI_NEXT_ACTION_CLEAR_HALT , XHCI_NEXT_ACTION_BOT_RESET , XHCI_NEXT_ACTION_HID_SET_PROTOCOL
  , XHCI_NEXT_ACTION_HID_POLL }

Functions

- int xhci_find_and_map (xhci_dev_t *dev, unsigned index)
- int xhci_bringup (xhci_dev_t *dev)
- uint32_t xhci_dev_count (void)
- xhci_dev_t * xhci_dev_at (uint32_t i)
- xhci_msc_slot_t * xhci_msc_slot_for (xhci_dev_t *dev, uint32_t slot_id)
- void xhci_poll_events (void)

- int xhci_cmd_enable_slot (xhci_dev_t *dev)
- int xhci_cmd_disable_slot (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_cmd_address_device (xhci_dev_t *dev, uint32_t slot_id, uint32_t port_id, uint32_t speed)
- int xhci_cmd_configure_endpoint (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_cmd_configure_hid_endpoint (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_ep0_hid_set_protocol (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_keyboard_running (void)
- int xhci_cmd_reset_endpoint (xhci_dev_t *dev, uint32_t slot_id, uint32_t ep_id)
- int xhci_cmd_set_tr_dequeue_pointer (xhci_dev_t *dev, uint32_t slot_id, uint32_t ep_id, uint64_t new_↩ dequeue, uint32_t dcs)
- int xhci_bot_send_read10 (xhci_dev_t *dev, uint32_t slot_id, uint32_t lba, uint16_t num_blocks, uint32_↩ t block_size)
- int xhci_bot_send_write10 (xhci_dev_t *dev, uint32_t slot_id, uint32_t lba, uint16_t num_blocks, uint32_↩ t block_size)
- int xhci_bot_send_test_unit_ready (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_read_block (xhci_dev_t *dev, uint32_t slot_id, uint32_t lba, uint16_t num_blocks, uint32_↩ t block_size)
- int xhci_bot_write_block (xhci_dev_t *dev, uint32_t slot_id, uint32_t lba, uint16_t num_blocks, uint32_↩ t block_size)
- int xhci_bot_send_read_capacity10 (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_get_capacity (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_wait_for_idle (xhci_dev_t *dev, uint32_t max_iters)
- int xhci_bot_read_data_in (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_write_data_out (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_receive_csw (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_ep0_get_device_descriptor (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_ep0_get_config_descriptor (xhci_dev_t *dev, uint32_t slot_id, uint16_t length)
- int xhci_ep0_set_configuration (xhci_dev_t *dev, uint32_t slot_id, uint8_t config_value)
- int xhci_ep0_clear_endpoint_halt (xhci_dev_t *dev, uint32_t slot_id, uint8_t ep_addr)
- int xhci_ep0_bot_mass_storage_reset (xhci_dev_t *dev, uint32_t slot_id)

7.95.1 Macro Definition Documentation

7.95.1.1 XHCI_MAX_CONTROLLERS

```
#define XHCI_MAX_CONTROLLERS 8u
```

7.95.2 Enumeration Type Documentation

7.95.2.1 xhci_next_action_t

```
enum xhci_next_action_t
```

Enumerator

XHCI_NEXT_ACTION_NONE	
XHCI_NEXT_ACTION_GET_DEVICE_DESC	
XHCI_NEXT_ACTION_GET_CONFIG_DESC	
XHCI_NEXT_ACTION_CONFIGURE_ENDPOINT	
XHCI_NEXT_ACTION_SET_CONFIG	
XHCI_NEXT_ACTION_BOT_DATA_IN	

Enumerator

XHCI_NEXT_ACTION_BOT_DATA_OUT	
XHCI_NEXT_ACTION_BOT_CSW_RECEIVE	
XHCI_NEXT_ACTION_BOT_SEND_TUR	
XHCI_NEXT_ACTION_BOT_SEND_READ10	
XHCI_NEXT_ACTION_BOT_SEND_READ_CAPACITY10	
XHCI_NEXT_ACTION_BOT_SEND_WRITE10	
XHCI_NEXT_ACTION_CLEAR_HALT	
XHCI_NEXT_ACTION_BOT_RESET	
XHCI_NEXT_ACTION_HID_SET_PROTOCOL	
XHCI_NEXT_ACTION_HID_POLL	

7.95.2.2 xhci_transfer_purpose_t

```
enum xhci_transfer_purpose_t
```

Enumerator

XHCI_XFER_NONE	
XHCI_XFER_DEVICE_DESC	
XHCI_XFER_CONFIG_DESC_SHORT	
XHCI_XFER_CONFIG_DESC_FULL	
XHCI_XFER_SET_CONFIG	
XHCI_XFER_CBW_SENT	
XHCI_XFER_BOT_DATA_IN	
XHCI_XFER_BOT_DATA_OUT	
XHCI_XFER_CSW_RECEIVED	
XHCI_XFER_CLEAR_HALT	
XHCI_XFER_BOT_RESET	
XHCI_XFER_HID_SET_PROTOCOL	
XHCI_XFER_HID_SET_LEDS	

7.95.3 Function Documentation

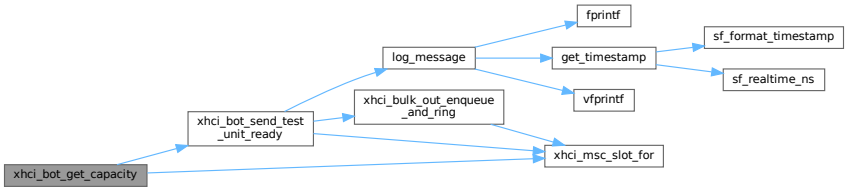
7.95.3.1 xhci_bot_get_capacity()

```
int xhci_bot_get_capacity (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

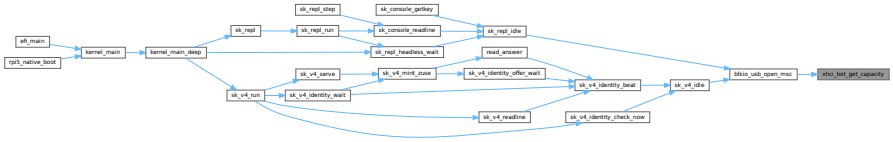
References xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, xhci_bot_send_test_unit_ready(), and xhci_msc_slot_for().

Referenced by blkio_usb_open_msc().

Here is the call graph for this function:



Here is the caller graph for this function:



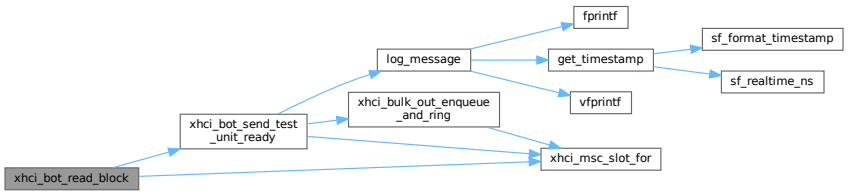
7.95.3.2 xhci_bot_read_block()

```
int xhci_bot_read_block (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t lba,
            uint16_t num_blocks,
            uint32_t block_size )
```

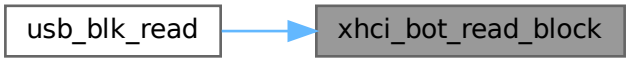
References block_size, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_read10_block_size, xhci_dev_t::bot_read10_lba, xhci_dev_t::bot_read10_num_blocks, xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_msc_slot_t::bulk_in_ep_adc xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, xhci_bot_send_test_unit_ready() and xhci_msc_slot_for().

Referenced by usb_blk_read().

Here is the call graph for this function:



Here is the caller graph for this function:



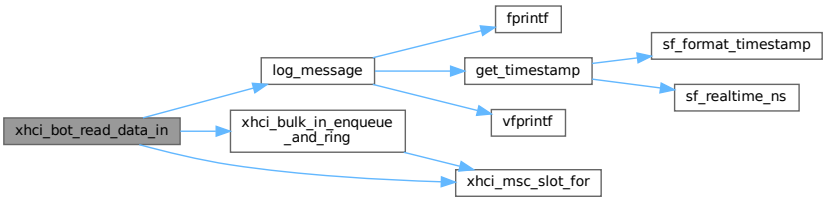
7.95.3.3 xhci_bot_read_data_in()

```
int xhci_bot_read_data_in (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

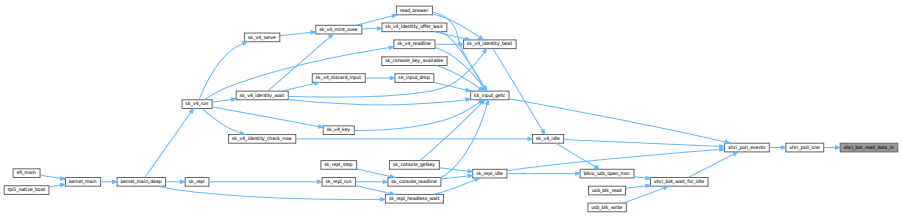
References xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_msc_slot_t::bulk_in_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, xhci_bulk_in_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_BOT_DATA_IN.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



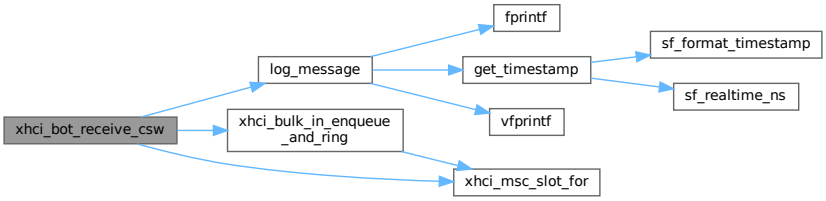
7.95.3.4 xhci_bot_receive_csw()

```
int xhci_bot_receive_csw (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

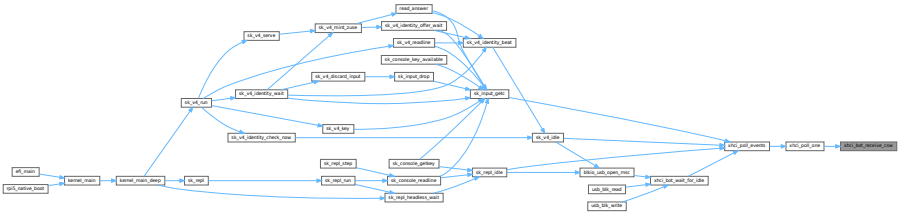
References xhci_dev_t::bot_csw, xhci_msc_slot_t::bulk_in_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_BOT_CSW_LENGTH, xhci_bulk_in_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CSW_RECEIVED.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



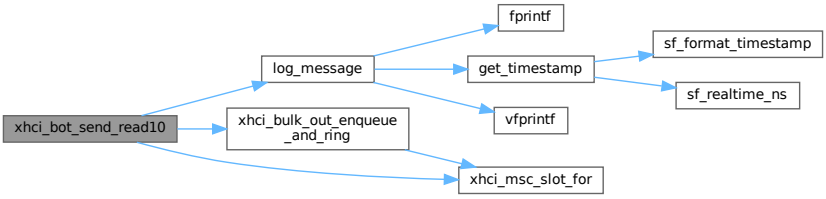
7.95.3.5 xhci_bot_send_read10()

```
int xhci_bot_send_read10 (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t lba,
            uint16_t num_blocks,
            uint32_t block_size )
```

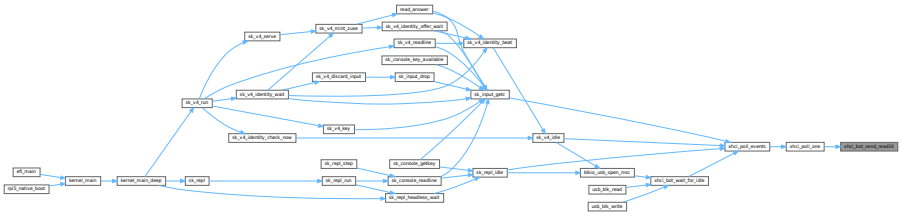
References usb_bot_cbw_t::bCBWCBLength, usb_bot_cbw_t::bCBWLUN, block_size, usb_bot_cbw_t::bmCBWFlags, xhci_dev_t::bot_cbw, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, usb_bot_cbw_t::CBWCB, usb_bot_cbw_t::dCBWDataTransferLe usb_bot_cbw_t::dCBWSignature, usb_bot_cbw_t::dCBWTag, LOG_DEBUG, log_message(), SCSI_CDB_LEN_READ10, SCSI_CMD_READ10, xhci_msc_slot_t::transfer_purpose, USB_BOT_CBW_FLAG_DATA_IN, USB_BOT_CBW_LENGTH, USB_BOT_CBW_LUN_DEFAULT, USB_BOT_CBW_SIGNATURE, xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CBW_SENT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



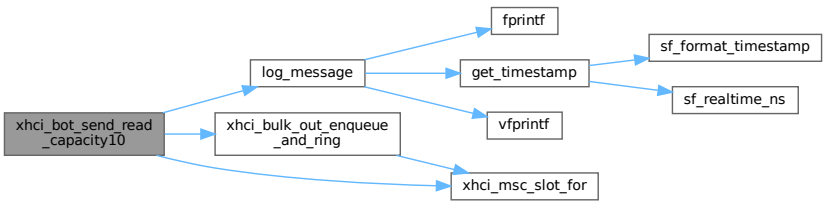
7.95.3.6 xhci_bot_send_read_capacity10()

```
int xhci_bot_send_read_capacity10 (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

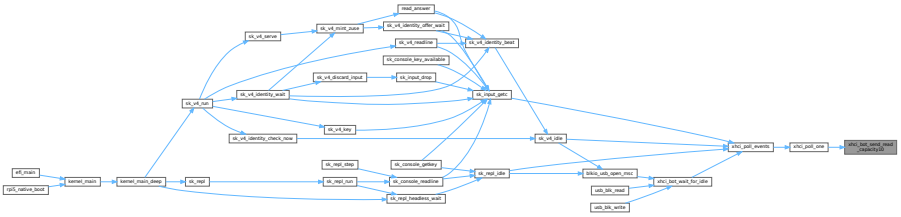
References usb_bot_cbw_t::bCBWCBLength, usb_bot_cbw_t::bCBWLUN, usb_bot_cbw_t::bmCBWFlags, xhci_dev_t::bot_cbw, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, usb_bot_cbw_t::CBWCB, usb_bot_cbw_t::dCBWDataTransferLength, usb_bot_cbw_t::dCBWSignatu usb_bot_cbw_t::dCBWTag, LOG_DEBUG, log_message(), SCSI_CDB_LEN_READ_CAPACITY10, SCSI_CMD_READ_CAPACITY1 SCSI_READ_CAPACITY10_DATA_LEN, xhci_msc_slot_t::transfer_purpose, USB_BOT_CBW_FLAG_DATA_IN, USB_BOT_CBW_LENGTH, USB_BOT_CBW_LUN_DEFAULT, USB_BOT_CBW_SIGNATURE, xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CBW_SENT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



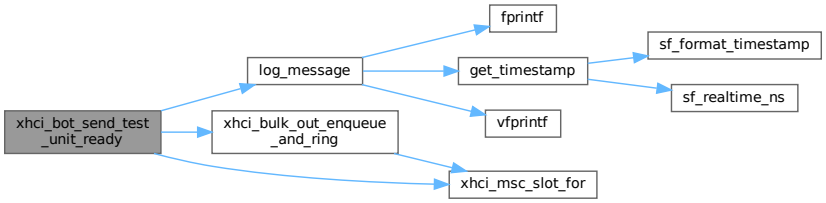
7.95.3.7 xhci_bot_send_test_unit_ready()

```
int xhci_bot_send_test_unit_ready (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

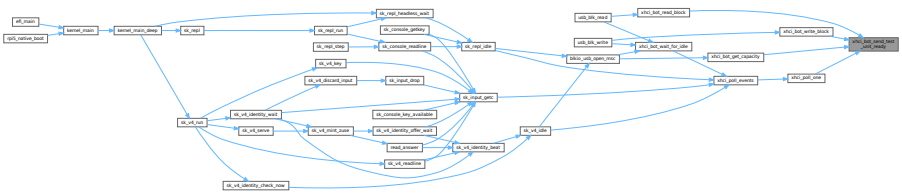
References usb_bot_cbw_t::bCBWCBLength, usb_bot_cbw_t::bCBWLUN, usb_bot_cbw_t::bmCBWFlags, xhci_dev_t::bot_cbw, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, usb_bot_cbw_t::CBWCB, usb_bot_cbw_t::dCBWDataTransferLength, usb_bot_cbw_t::dCBWSignatu usb_bot_cbw_t::dCBWTag, LOG_DEBUG, log_message(), SCSI_CDB_LEN_TEST_UNIT_READY, SCSI_CMD_TEST_UNIT_READ xhci_msc_slot_t::transfer_purpose, USB_BOT_CBW_LENGTH, USB_BOT_CBW_LUN_DEFAULT, USB_BOT_CBW_SIGNATURE, xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CBW_SENT.

Referenced by xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_write_block(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



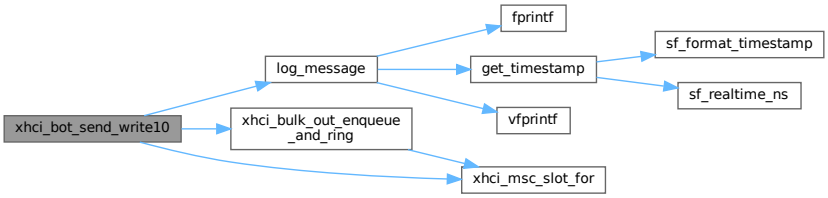
7.95.3.8 xhci_bot_send_write10()

```
int xhci_bot_send_write10 (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t lba,
            uint16_t num_blocks,
            uint32_t block_size )
```

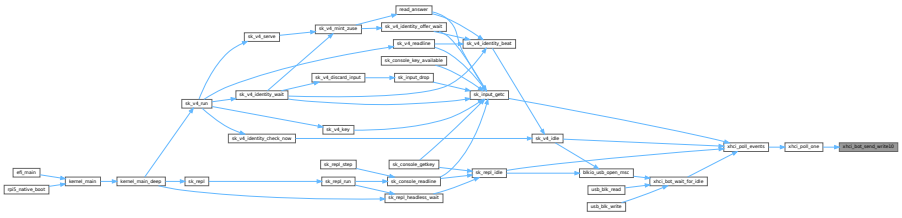
References usb_bot_cbw_t::bCBWCBLength, usb_bot_cbw_t::bCBWLUN, block_size, usb_bot_cbw_t::bmCBWFlags, xhci_dev_t::bot_cbw, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, usb_bot_cbw_t::CBWCB, usb_bot_cbw_t::dCBWDataTransferLe usb_bot_cbw_t::dCBWSignature, usb_bot_cbw_t::dCBWTag, LOG_DEBUG, log_message(), SCSI_CDB_LEN_WRITE10, SCSI_CMD_WRITE10, xhci_msc_slot_t::transfer_purpose, USB_BOT_CBW_LENGTH, USB_BOT_CBW_LUN_DEFAULT, USB_BOT_CBW_SIGNATURE, xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CBW_SENT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



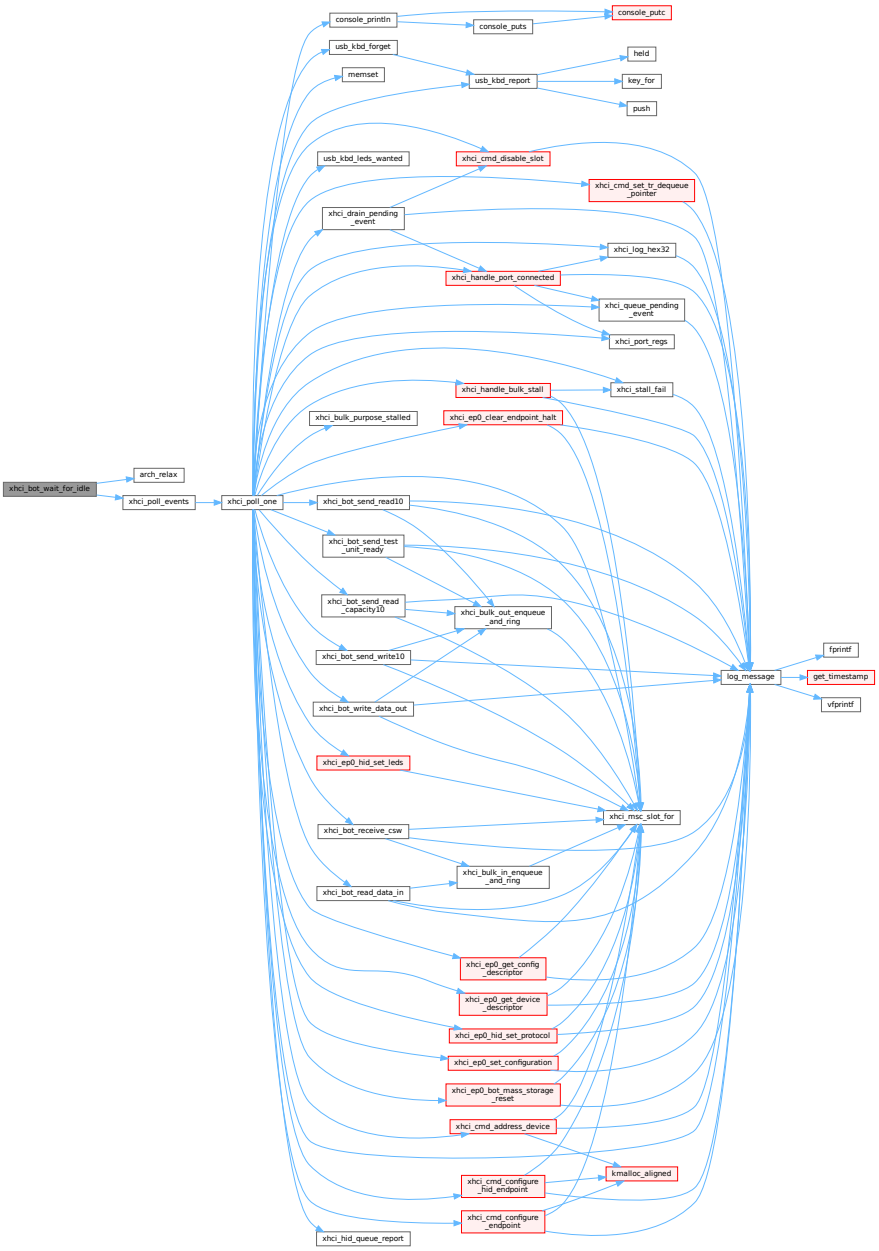
7.95.3.9 xhci_bot_wait_for_idle()

```
int xhci_bot_wait_for_idle (
            xhci_dev_t * dev,
            uint32_t max_iters )
```

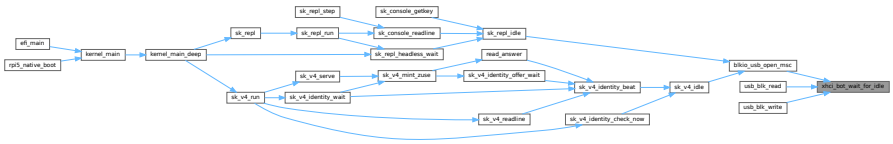
References arch_relax(), xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_last_status, and xhci_poll_events().

Referenced by blkio_usb_open_msc(), usb_blk_read(), and usb_blk_write().

Here is the call graph for this function:



Here is the caller graph for this function:



7.95.3.10 xhci_bot_write_block()

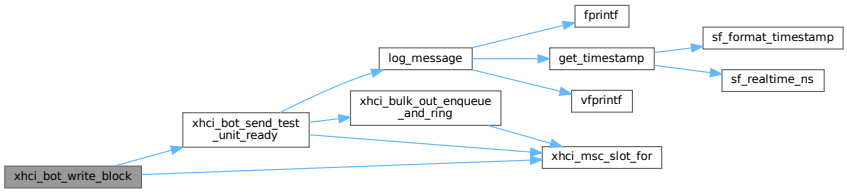
```
int xhci_bot_write_block (
```

```
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t lba,
            uint16_t num_blocks,
            uint32_t block_size )
```

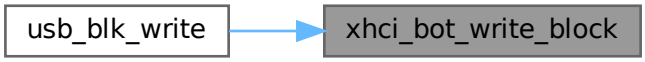
References block_size, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_dev_t::bot_write10_block_size, xhci_dev_t::bot_write10_lba, xhci_dev_t::bot_write10_num_blocks, xhci_msc_slot_t::bulk_in_ep, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, xhci_bot_send_test_unit_ready(), and xhci_msc_slot_for().

Referenced by usb_blk_write().

Here is the call graph for this function:



Here is the caller graph for this function:



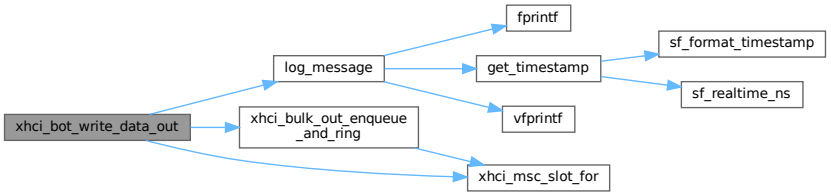
7.95.3.11 xhci_bot_write_data_out()

```
int xhci_bot_write_data_out (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

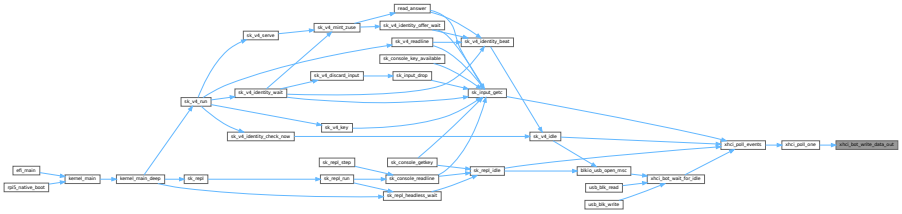
References xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_msc_slot_t::bulk_out_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_BOT_DATA_O

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



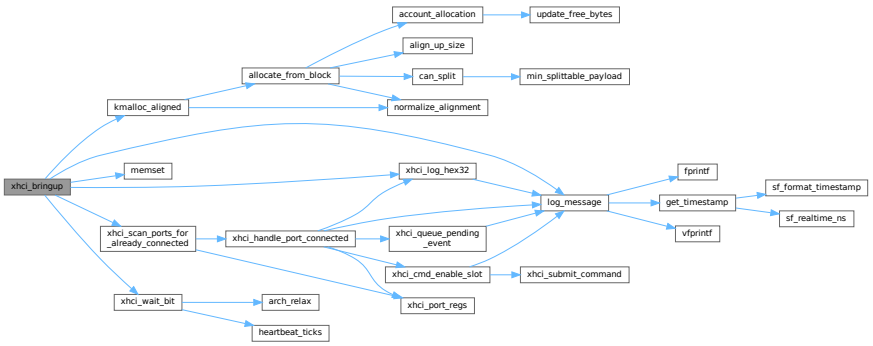
7.95.3.12 xhci_bringup()

```
int xhci_bringup (
            xhci_dev_t * dev )
```

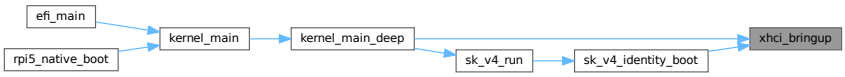
References xhci_dev_t::bot_cap_block_size, xhci_dev_t::bot_cap_last_lba, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_expected_dat xhci_dev_t::bot_last_status, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_dev_t::bot_read10_block_size, xhci_dev_t::bot_read10_lba, xhci_dev_t::bot_read10_num_blocks, xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_dev_t::cap, xhci_dev_t::cmd_ring, xhci_dev_t::cmd_ring_cycle, xhci_dev_t::cmd_ring_enq, xhci_op_regs_t::config, xhci_dev_t::connect_state, xhci_trb_t::control, xhci_op_regs_t::crcr, xhci_dev_t::dcbaa, xhci_op_regs_t::dcbaap, xhci_intr_regs_t::erdp, xhci_intr_regs_t::erstba, xhci_intr_regs_t::erstsz, xhci_dev_t::evt_ring, xhci_dev_t::evt_ring_cycle, xhci_dev_t::evt_ring_deq, xhci_dev_t::evt_ring_seg_table, g_xhci_dev_count, g_xhci_devs, xhci_cap_regs_t::hcc_params1, xhci_dev_t::intr0, kmalloc_aligned(), LOG_DEBUG, LOG_ERROR, log_message(), xhci_dev_t::max_scratchpad_bufs, xhci_dev_t::max_slots, memset(), xhci_dev_t::msc_slots, xhci_dev_t::op, xhci_op_regs_t::page_size, xhci_trb_t::parameter, xhci_dev_t::pending_connect_port_id, xhci_dev_t::pending_connect_slot_id, xhci_dev_t::pending_connect_speed, xhci_dev_t::pending_disable_slot_id, xhci_dev_t::pending_events, xhci_dev_t::port_slot_id, xhci_dev_t::runtime, xhci_dev_t::scratchpad_arr, xhci_trb_t::status, xhci_op_regs_t::usb_cmd, xhci_op_regs_t::usb_sts, xhci_dev_t::valid, XHCI_CONFIG_MAX_SLOTS_EN, XHCI_CRCR_PTR_MASK, XHCI_CRCR_RCS, XHCI_ERDP_PTR_MASK, XHCI_HCCPARAMS1_CSZ, xhci_log_hex32(), XHCI_MAX_CONTROLLERS, XHCI_MAX_TRACKED_PORTS, XHCI_RING_BYTES, XHCI_RING_TRB_COUNT, xhci_scan_ports_for_already_connected(), XHCI_TRB_CONTROL_CYCLE, XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_LINK, XHCI_USBCMD_HCRST, XHCI_USBCMD_RUN, XHCI_USBSTS_CNR, XHCI_USBSTS_HCH, and xhci_wait_bit().

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



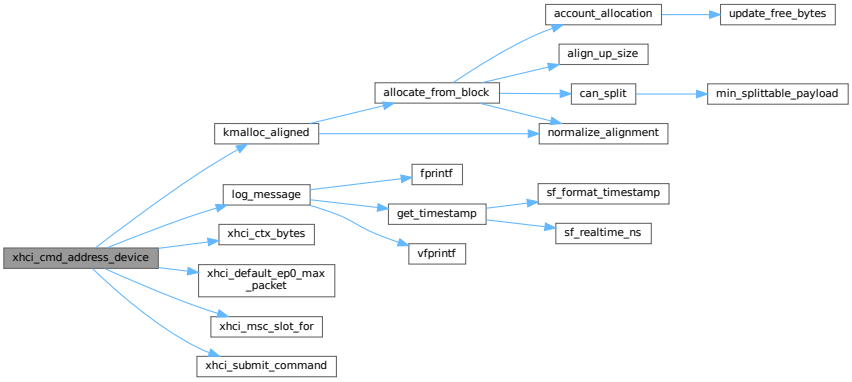
7.95.3.13 xhci_cmd_address_device()

```
int xhci_cmd_address_device (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t port_id,
            uint32_t speed )
```

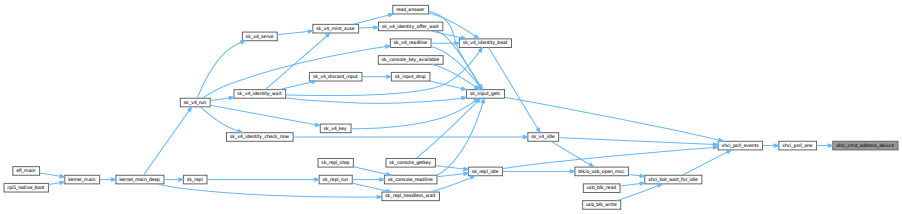
References xhci_input_ctrl_ctx32_t::add_flags, xhci_dev_t::cmd_ring, xhci_trb_t::control, xhci_dev_t::dcbaa, xhci_msc_slot_t::device_ctx, xhci_slot_ctx32_t::dword0, xhci_slot_ctx32_t::dword1, xhci_ep_ctx32_t::dword1, xhci_slot_ctx32_t::dword2, xhci_ep_ctx32_t::dword4, xhci_msc_slot_t::ep0_ring, xhci_msc_slot_t::ep0_ring_cycle, xhci_msc_slot_t::ep0_ring_enq, xhci_msc_slot_t::hid_ep_addr, xhci_msc_slot_t::hid_kbd, xhci_msc_slot_t::hid_running, xhci_msc_slot_t::in_use, xhci_msc_slot_t::input_ctx, kmalloc_aligned(), LOG_DEBUG, log_message(), xhci_trb_t::parameter, xhci_trb_t::status, xhci_ep_ctx32_t::tr_dequeue_ptr, xhci_ctx_bytes(), xhci_default_ep0_max_packet(), XHCI_EP_CTX_CERR_SHIF XHCI_EP_CTX_MAX_PACKET_SHIFT, XHCI_EP_CTX_TYPE_CONTROL_BIDI, XHCI_EP_CTX_TYPE_SHIFT, XHCI_INPUT_CTRL_ADD_EP0, XHCI_INPUT_CTRL_ADD_SLOT, xhci_msc_slot_for(), XHCI_RING_BYTES, XHCI_RING_TRB_COUNT, XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT, XHCI_SLOT_CTX_INTR_TARGET_SHIFT, XHCI_SLOT_CTX_ROOT_PORT_SHIFT, XHCI_SLOT_CTX_SPEED_SHIFT, xhci_submit_command(), XHCI_TRB_CONTROL_CY XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_ADDRESS_DEVICE_CMD, and XHCI_TRB_TYPE_LINK.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



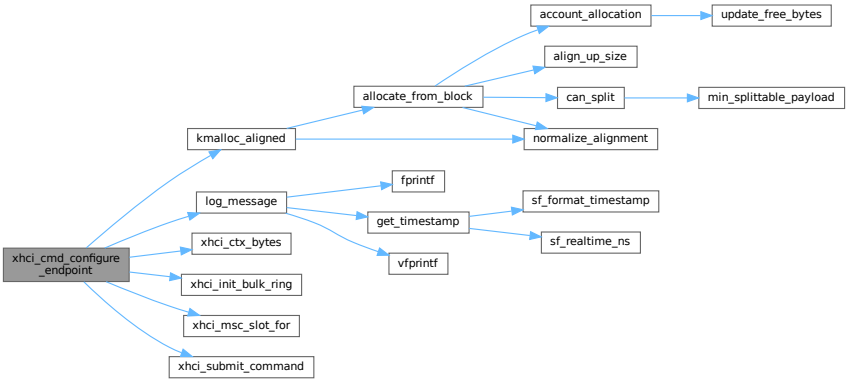
7.95.3.14 xhci_cmd_configure_endpoint()

```
int xhci_cmd_configure_endpoint (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

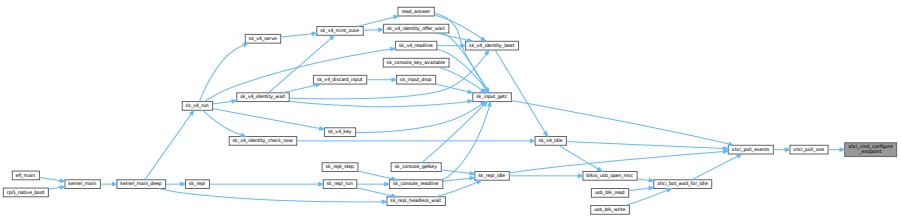
References xhci_input_ctrl_ctx32_t::add_flags, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_max_packet, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_in_ring_cycle, xhci_msc_slot_t::bulk_in_ring_enq, xhci_msc_slot_t::bulk_out_ep, xhci_msc_slot_t::bulk_out_max_packet, xhci_msc_slot_t::bulk_out_ring, xhci_msc_slot_t::bulk_out_ring_cycle, xhci_msc_slot_t::bulk_out_ring_enq, xhci_dev_t::cmd_ring, xhci_dev_t::dcbaa, xhci_msc_slot_t::device_ctx, xhci_slot_ctx32_t::dword0, xhci_slot_ctx32_t::dword1, xhci_ep_ctx32_t::dword1, xhci_slot_ctx32_t::dword2, xhci_ep_ctx32_t::dword4, xhci_msc_slot_t::input_ctx, kmalloc_aligned(), LOG_DEBUG, log_message(), xhci_ep_ctx32_t::tr_dequeue_ptr, xhci_ctx_bytes(), XHCI_EP_ADDR_TO_DCI, XHCI_EP_CTX_CERR_SHIFT, XHCI_EP_CTX_MAX_PACKET_SHIFT, XHCI_EP_CTX_TYPE_BULK_IN, XHCI_EP_CTX_TYPE_BULK_OUT, XHCI_EP_CTX_TYPE_SHIFT, xhci_init_bulk_ring(), XHCI_INPUT_CTRL_ADD_SLOT, xhci_msc_slot_for(), XHCI_RING_BYTES, XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT, xhci_submit_command(), and XHCI_TRB_TYPE_CONFIGU

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



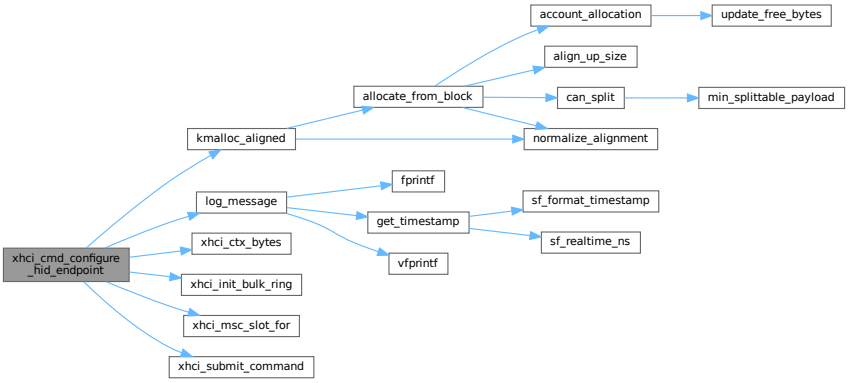
7.95.3.15 xhci_cmd_configure_hid_endpoint()

```
int xhci_cmd_configure_hid_endpoint (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

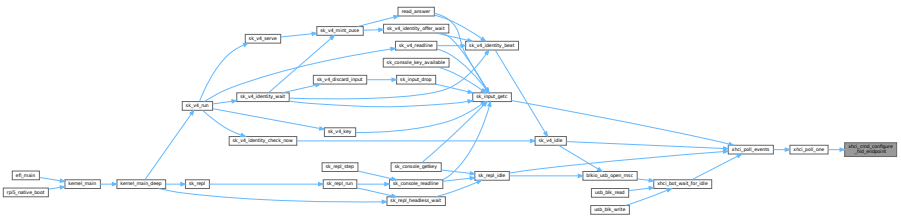
References xhci_input_ctrl_ctx32_t::add_flags, xhci_dev_t::cmd_ring, xhci_dev_t::dcbaa, xhci_msc_slot_t::device_ctx, xhci_slot_ctx32_t::dword0, xhci_ep_ctx32_t::dword0, xhci_slot_ctx32_t::dword1, xhci_ep_ctx32_t::dword1, xhci_slot_ctx32_t::dword2, xhci_ep_ctx32_t::dword4, xhci_msc_slot_t::hid_buf, xhci_msc_slot_t::hid_ep_addr, xhci_msc_slot_t::hid_interval, xhci_msc_slot_t::hid_max_packet, xhci_msc_slot_t::hid_ring, xhci_msc_slot_t::hid_ring_cycle, xhci_msc_slot_t::hid_ring_enq, xhci_msc_slot_t::input_ctx, kmalloc_aligned(), LOG_DEBUG, log_message(), xhci_ep_ctx32_t::tr_dequeue_ptr, xhci_ctx_bytes(), XHCI_EP_ADDR_TO_DCI, XHCI_EP_CTX_CERR_SHIFT, XHCI_EP_CTX_INTERVAL_SHIFT, XHCI_EP_CTX_MAX_PACKET_SHIFT, XHCI_EP_CTX_TYPE_INTERRUPT_IN, XHCI_EP_CTX_TYPE_SHIFT, xhci_init_bulk_ring(), XHCI_INPUT_CTRL_ADD_SLOT, xhci_msc_slot_for(), XHCI_RING_BYTES, XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT, XHCI_SLOT_CTX_SPEED_SHIFT, xhci_submit_command(), and XHCI_TRB_TYPE_CONFIGURE_ENDPOINT_CMD.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



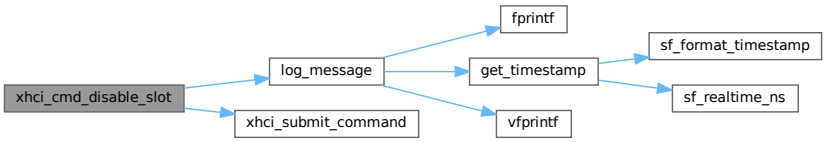
7.95.3.16 xhci_cmd_disable_slot()

```
int xhci_cmd_disable_slot (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

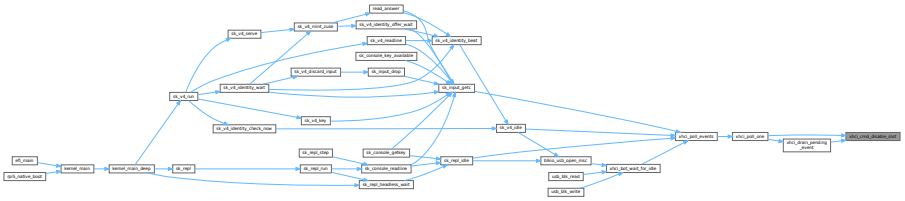
References xhci_dev_t::cmd_ring, LOG_DEBUG, log_message(), xhci_submit_command(), and XHCI_TRB_TYPE_DISABLE_SLOT

Referenced by xhci_drain_pending_event(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



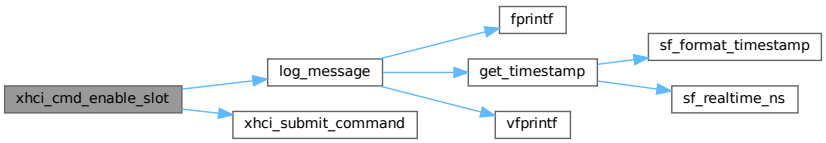
7.95.3.17 xhci_cmd_enable_slot()

```
int xhci_cmd_enable_slot (
            xhci_dev_t * dev )
```

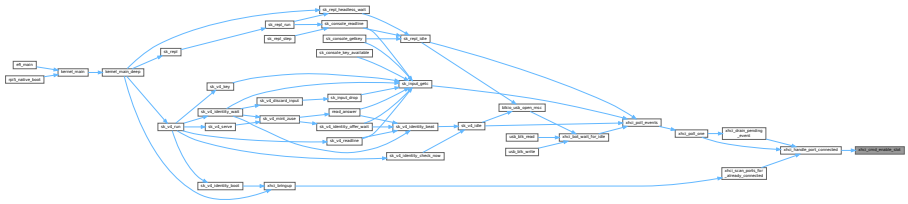
References xhci_dev_t::cmd_ring, LOG_DEBUG, log_message(), xhci_submit_command(), and XHCI_TRB_TYPE_ENABLE_SLOT_

Referenced by xhci_handle_port_connected().

Here is the call graph for this function:



Here is the caller graph for this function:



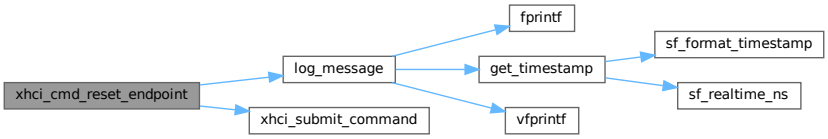
7.95.3.18 xhci_cmd_reset_endpoint()

```
int xhci_cmd_reset_endpoint (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t ep_id )
```

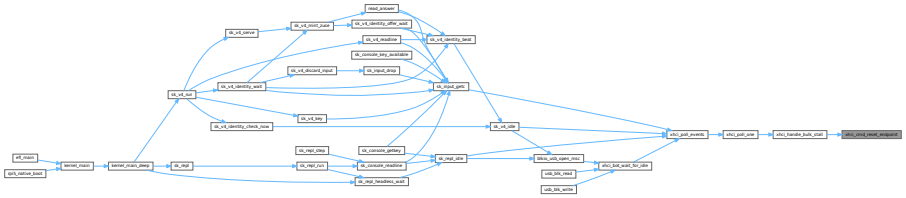
References xhci_dev_t::cmd_ring, LOG_DEBUG, log_message(), XHCI_EP_ADDR_TO_DCI, xhci_submit_command(), and XHCI_TRB_TYPE_RESET_ENDPOINT_CMD.

Referenced by xhci_handle_bulk_stall().

Here is the call graph for this function:



Here is the caller graph for this function:



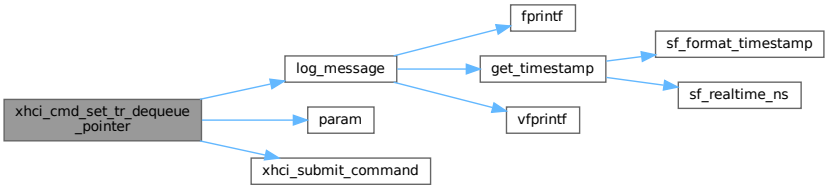
7.95.3.19 xhci_cmd_set_tr_dequeue_pointer()

```
int xhci_cmd_set_tr_dequeue_pointer (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t ep_id,
            uint64_t new_dequeue,
            uint32_t dcs )
```

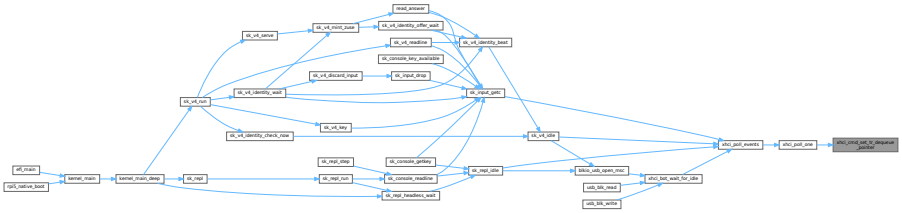
References xhci_dev_t::cmd_ring, LOG_DEBUG, log_message(), param(), xhci_submit_command(), and XHCI_TRB_TYPE_SET_TR_DEQUEUE_POINTER_CMD.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



7.95.3.20 xhci_dev_at()

```
xhci_dev_t * xhci_dev_at (
            uint32_t i )
```

References g_xhci_dev_count, and g_xhci_devs.

Referenced by sk_repl_idle(), sk_v4_idle(), and sk_word_blk_attach_ack().

Here is the caller graph for this function:

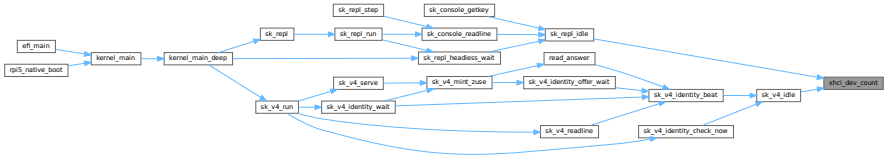
7.95.3.21 xhci_dev_count()

```
uint32_t xhci_dev_count (
            void  )
```

References g_xhci_dev_count.

Referenced by sk_repl_idle(), and sk_v4_idle().

Here is the caller graph for this function:



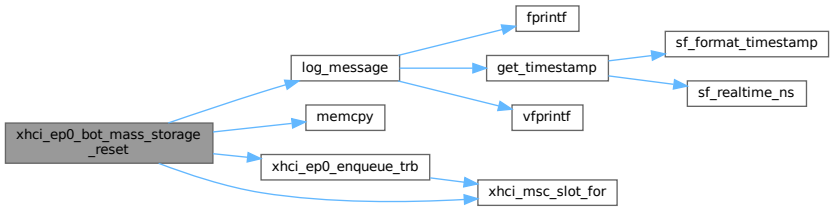
7.95.3.22 xhci_ep0_bot_mass_storage_reset()

```
int xhci_ep0_bot_mass_storage_reset (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

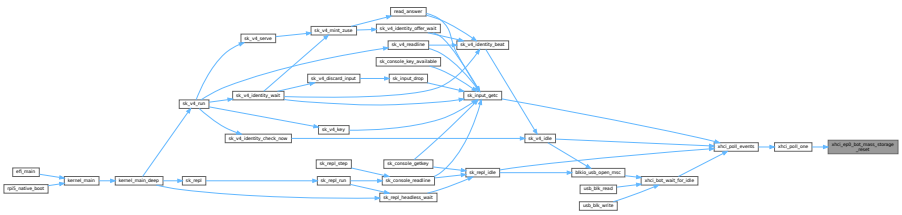
References usb_setup_packet_t::bmRequestType, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), memcpy(), xhci_msc_slot_t::transfer_purpose, USB_BOT_MASS_STORAGE_RESET, USB_RECIP_INTERFACE, USB_REQ_TYPE_CLASS, XHCI_DB_TARGET, xhci_ep0_enqueue_trb(), xhci_msc_slot_for(), XHCI_SETUP_TRT_NO_DATA, XHCI_TRB_CONTROL_DIR_IN, XHCI_TRB_CONTROL_IDT, XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TRT_SHIFT, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_SETUP_STAGE, XHCI_TRB_TYPE_STATUS_STAGE, and XHCI_XFER_BOT_RESET.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



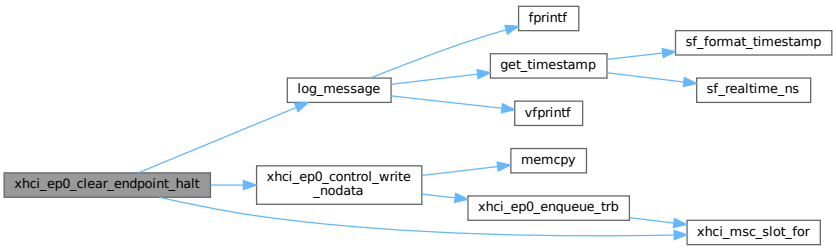
7.95.3.23 xhci_ep0_clear_endpoint_halt()

```
int xhci_ep0_clear_endpoint_halt (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint8_t ep_addr )
```

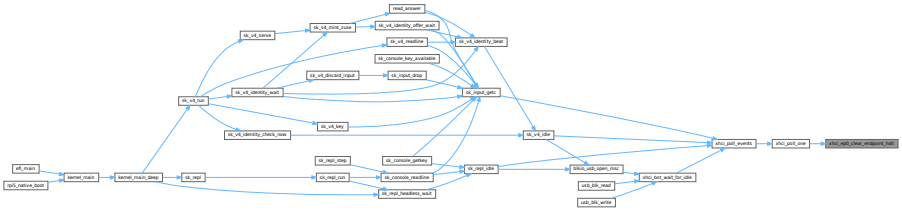
References xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_FEATURE_ENDPOINT_HALT, USB_REQ_CLEAR_FEATURE, XHCI_DB_TARGET, xhci_ep0_control_write_nodata(), xhci_msc_slot_for(), and XHCI_XFER_CLEAR_HALT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



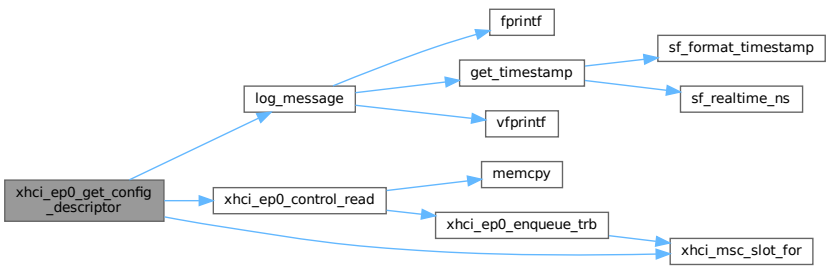
7.95.3.24 xhci_ep0_get_config_descriptor()

```
int xhci_ep0_get_config_descriptor (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint16_t length )
```

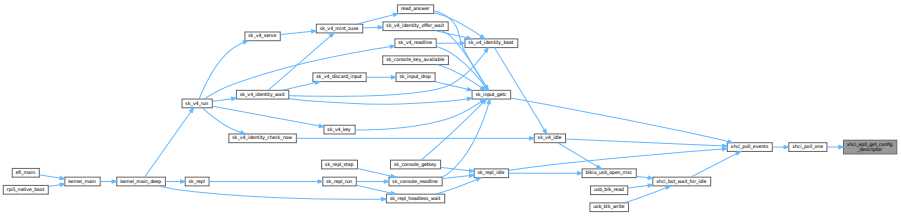
References xhci_msc_slot_t::config_descriptor, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_DESC_TYPE_CONFIG, USB_REQ_GET_DESCRIPTOR, XHCI_DB_TARGET, xhci_ep0_control_read(), xhci_msc_slot_for(), XHCI_XFER_CONFIG_DESC_FULL, and XHCI_XFER_CONFIG_DESC_SHORT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



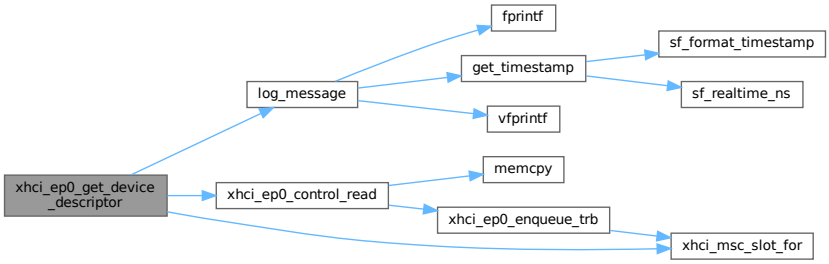
7.95.3.25 xhci_ep0_get_device_descriptor()

```
int xhci_ep0_get_device_descriptor (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

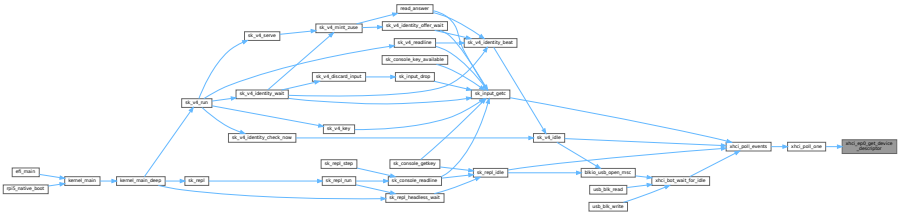
References xhci_msc_slot_t::device_descriptor, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_DESC_TYPE_DEVICE, USB_REQ_GET_DESCRIPTOR, XHCI_DB_TARGET, xhci_ep0_control_read(), xhci_msc_slot_for(), and XHCI_XFER_DEVICE_DESC.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



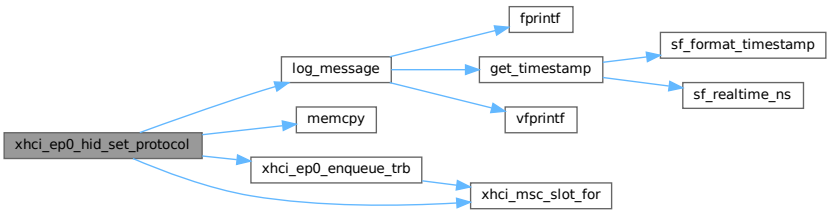
7.95.3.26 xhci_ep0_hid_set_protocol()

```
int xhci_ep0_hid_set_protocol (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

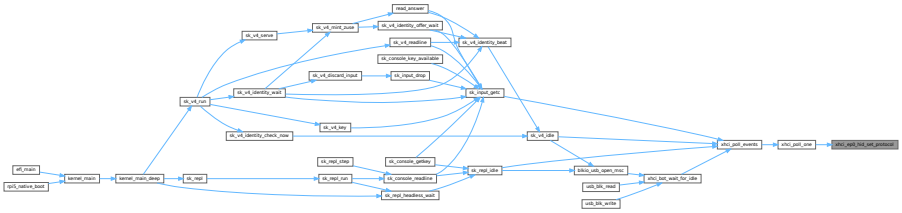
References usb_setup_packet_t::bmRequestType, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, xhci_msc_slot_t::hid_iface, LOG_DEBUG, log_message(), memcpy(), xhci_msc_slot_t::transfer_purpose, USB_HID_PROTOCOL_BOOT, USB_HID_REQ_SET_PROTOCOL, USB_RECIP_INTERFACE, USB_REQ_TYPE_CLASS, XHCI_DB_TARGET, xhci_ep0_enqueue_trb(), xhci_msc_slot_for(), XHCI_SETUP_TRT_NO_DATA, XHCI_TRB_CONTROL_DIR_IN, XHCI_TRB_CONTROL_IDT, XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TRT_SHIFT, XHCI_TRB_CONTROL_TYPE_SH XHCI_TRB_TYPE_SETUP_STAGE, XHCI_TRB_TYPE_STATUS_STAGE, and XHCI_XFER_HID_SET_PROTOCOL.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



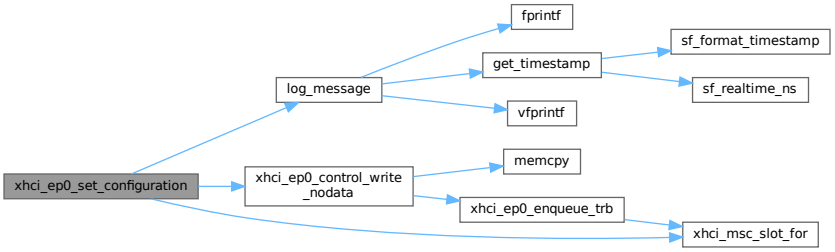
7.95.3.27 xhci_ep0_set_configuration()

```
int xhci_ep0_set_configuration (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint8_t config_value )
```

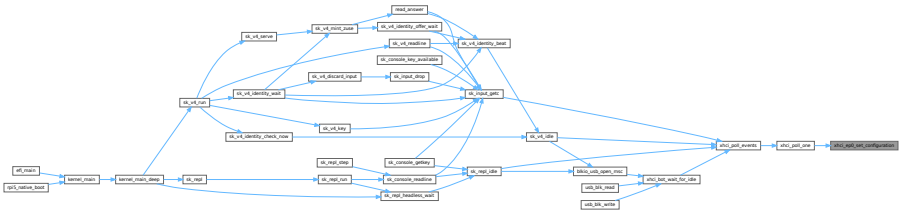
References xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_REQ_SET_CONFIGURATION, XHCI_DB_TARGET, xhci_ep0_control_write_nodata(), xhci_msc_slot_for(), and XHCI_XFER_SET_CONFIG.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



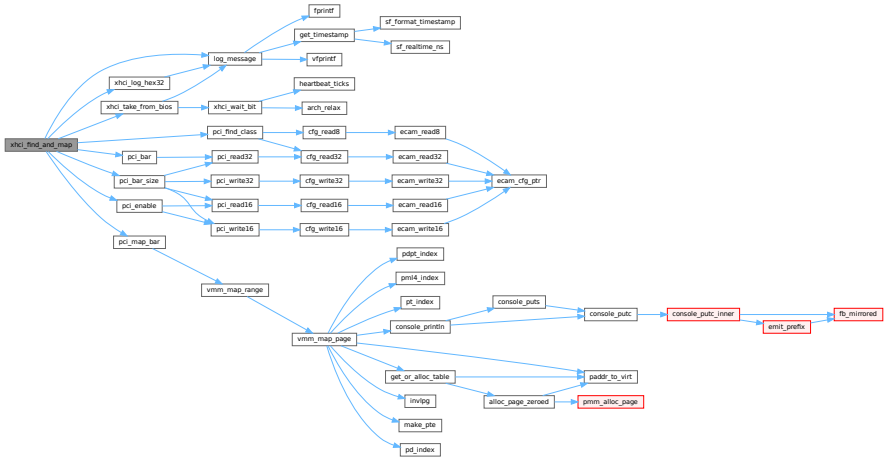
7.95.3.28 xhci_find_and_map()

```
int xhci_find_and_map (
            xhci_dev_t * dev,
            unsigned index )
```

References xhci_dev_t::bar0_phys, xhci_dev_t::cap, xhci_cap_regs_t::cap_length, xhci_cap_regs_t::db_off, PciDevice::device_id, xhci_dev_t::doorbell, xhci_cap_regs_t::hcs_params1, xhci_cap_regs_t::hcs_params2, LOG_DEBUG, LOG_ERROR, log_message(), xhci_dev_t::max_intrs, xhci_dev_t::max_ports, xhci_dev_t::max_scratchpad_bufs, xhci_dev_t::max_slots, xhci_dev_t::op, xhci_dev_t::pci, pci_bar(), pci_bar_size(), pci_enable(), pci_find_class(), pci_map_bar(), xhci_cap_regs_t::rts_off, xhci_dev_t::runtime, PciDevice::vendor_id, XHCI_BAR0_MIN_MAP_SIZE, XHCI_HCSPARAMS1_MAX_INTRS, XHCI_HCSPARAMS1_MAX_PORTS, XHCI_HCSPARAMS1_MAX_SLOTS, XHCI_HCSPARAMS2_MAX_SCRATCHPAD_BUFS, xhci_log_hex32(), XHCI_PCI_CLASS, XHCI_PCI_PROG_IF, XHCI_PCI_SUBCLASS, and xhci_take_from_bios().

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:

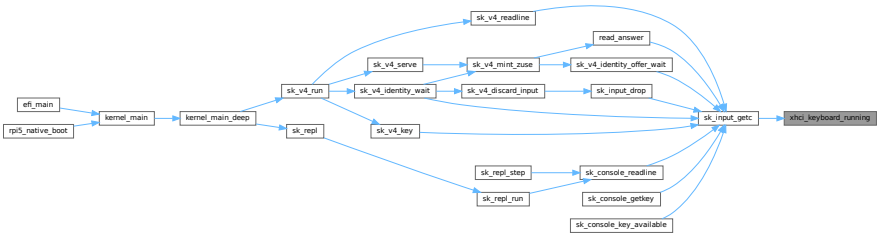
7.95.3.29 xhci_keyboard_running()

```
int xhci_keyboard_running (
            void  )
```

References g_xhci_keyboards.

Referenced by sk_input_getc().

Here is the caller graph for this function:



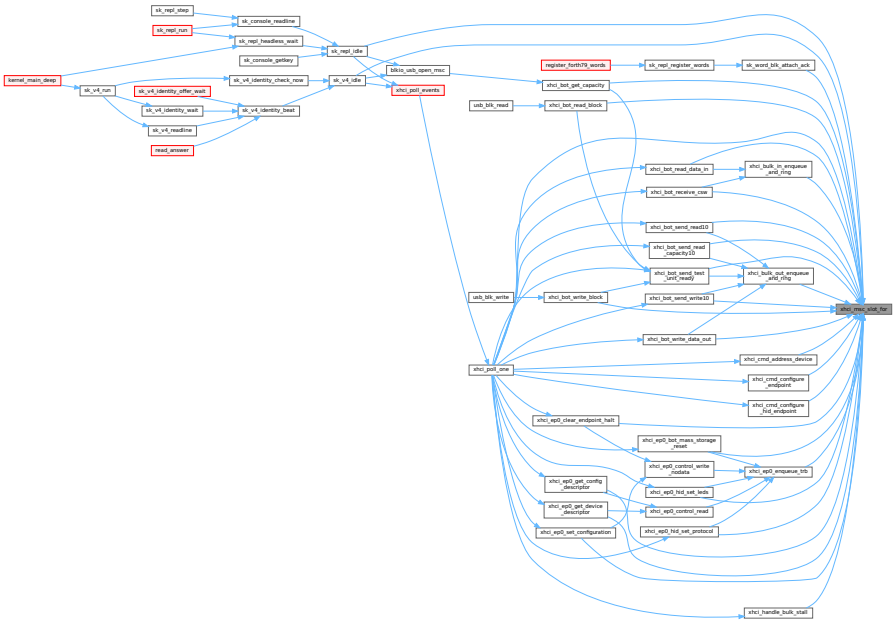
7.95.3.30 xhci_msc_slot_for()

```
xhci_msc_slot_t * xhci_msc_slot_for (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

References xhci_dev_t::max_slots, and xhci_dev_t::msc_slots.

Referenced by sk_repl_idle(), sk_v4_idle(), sk_word_blk_attach_ack(), xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_read_data_in(), xhci_bot_receive_csw(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_block(), xhci_bot_write_data_out(), xhci_bulk_in_enqueue_and_ring(), xhci_bulk_out_enqueue_and_ring(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), xhci_cmd_configure_hid_endpoint(), xhci_ep0_bot_mass_storage_reset(), xhci_ep0_clear_endpoint_halt(), xhci_ep0_enqueue_trb(), xhci_ep0_get_config_descriptor(), xhci_ep0_get_device_descriptor(), xhci_ep0_hid_set_leds(), xhci_ep0_hid_set_protocol(), xhci_ep0_set_configuration(), xhci_handle_bulk_stall(), and xhci_poll_one().

Here is the caller graph for this function:



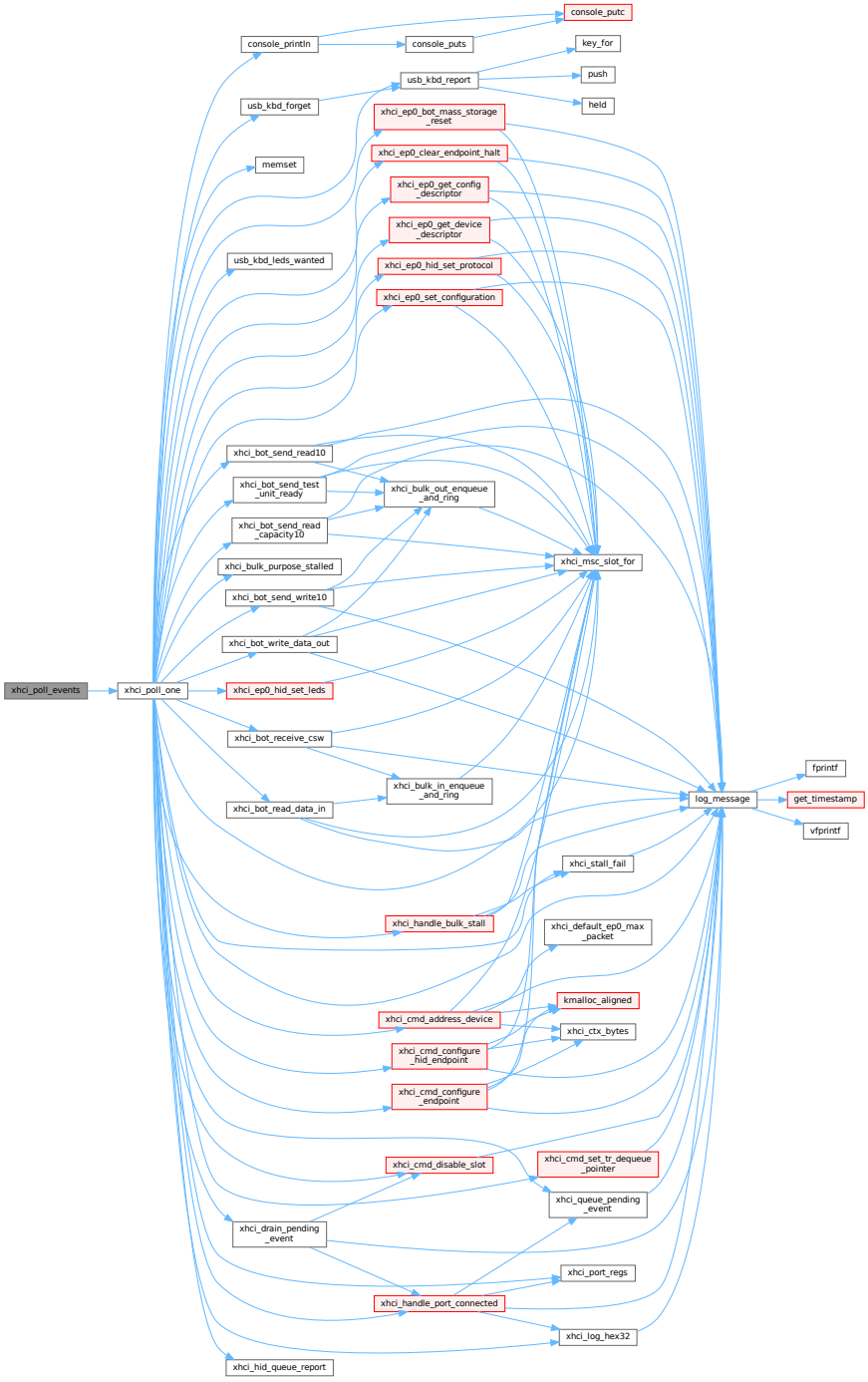
7.95.3.31 xhci_poll_events()

```
void xhci_poll_events (
            void  )
```

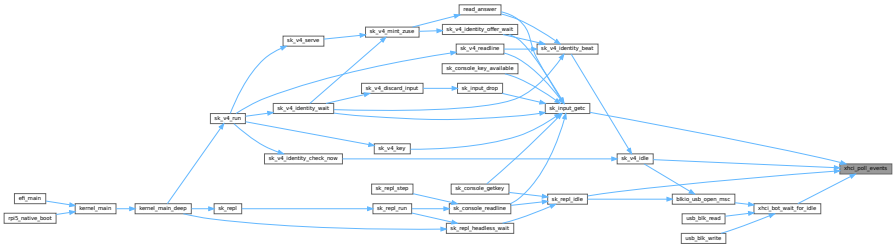
References g_xhci_dev_count, g_xhci_devs, and xhci_poll_one().

Referenced by sk_input_getc(), sk_repl_idle(), sk_v4_idle(), and xhci_bot_wait_for_idle().

Here is the call graph for this function:



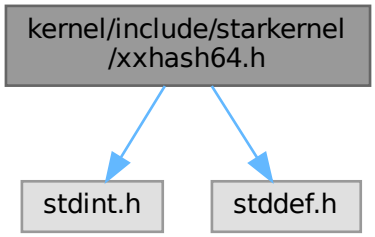
Here is the caller graph for this function:



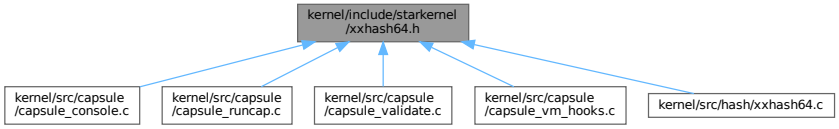
7.96 kernel/include/starkernel/xxhash64.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for xxhash64.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct XXHash64State

  Streaming state structure.

Macros

- #define XXHASH64_CAPSULE_SEED 0ULL

  xxhash64.h - xxHash64 Implementation (Freestanding)

Functions

- uint64_t xxhash64 (const void *data, size_t len, uint64_t seed)

  xxhash64 - Compute xxHash64 of a buffer
- static uint64_t xxhash64_capsule (const void *data, size_t len)

  xxhash64_capsule - Hash capsule payload with standard seed
- void xxhash64_reset (XXHash64State *state, uint64_t seed)

  xxhash64_reset - Initialize streaming state
- void xxhash64_update (XXHash64State *state, const void *data, size_t len)

  xxhash64_update - Feed data into streaming hash
- uint64_t xxhash64_digest (const XXHash64State *state)

  xxhash64_digest - Finalize and get hash value

7.96.1 Macro Definition Documentation

7.96.1.1 XXHASH64_CAPSULE_SEED

```
#define XXHASH64_CAPSULE_SEED 0ULL
```

xxhash64.h - xxHash64 Implementation (Freestanding)

Fast, deterministic 64-bit hash function for capsule content addressing. No libc dependency - works in kernel context.

Based on xxHash by Yann Collet (BSD-2-Clause license). https://github.com/Cyan4973/xxHash
Default seed for capsule hashing (deterministic)

7.96.2 Function Documentation

7.96.2.1 xxhash64()

```
uint64_t xxhash64 (
            const void * data,
            size_t len,
            uint64_t seed )
```

xxhash64 - Compute xxHash64 of a buffer

Parameters

data	Input data
len	Length in bytes
seed	Hash seed (use XXHASH64_CAPSULE_SEED for capsules)

Returns

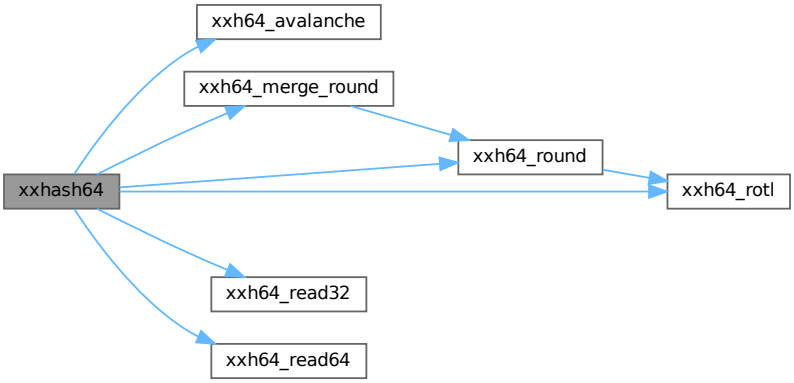
64-bit hash value

References limit, v1, v2, xxh64_avalanche(), xxh64_merge_round(), xxh64_read32(), xxh64_read64(), xxh64_rotl(),

xxh64_round(), XXHASH64_PRIME1, XXHASH64_PRIME2, XXHASH64_PRIME3, XXHASH64_PRIME4, and XXHASH64_PRIME5.

Referenced by xxhash64_capsule().

Here is the call graph for this function:



Here is the caller graph for this function:

7.96.2.2 xxhash64_capsule()

```
static uint64_t xxhash64_capsule (
            const void * data,
            size_t len )  [inline], [static]
```

xxhash64_capsule - Hash capsule payload with standard seed

Convenience wrapper using XXHASH64_CAPSULE_SEED.

Parameters

data	Capsule payload bytes
len	Payload length

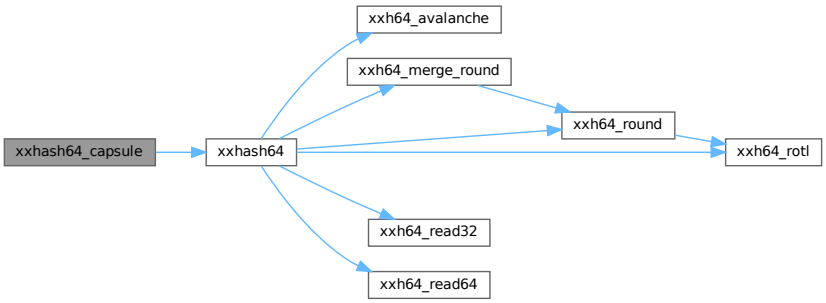
Returns

Content hash (capsule_id)

References xxhash64(), and XXHASH64_CAPSULE_SEED.

Referenced by capsule_console_birth(), capsule_runcap_birth(), and capsule_validate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.96.2.3 xxhash64_digest()

```
uint64_t xxhash64_digest (
            const XXHash64State * state )
```

xxhash64_digest - Finalize and get hash value

Parameters

state	Streaming state

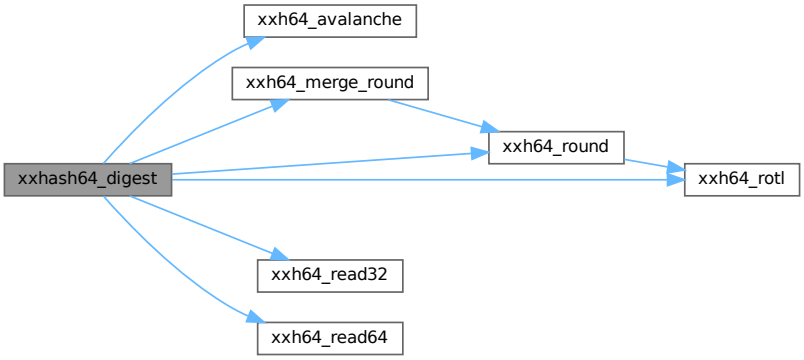
Returns

64-bit hash value

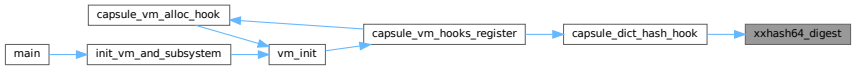
References XXHash64State::buffer, XXHash64State::buffer_size, XXHash64State::seed, XXHash64State::total_len, XXHash64State::v1, XXHash64State::v2, XXHash64State::v3, XXHash64State::v4, xxh64_avalanche(), xxh64_merge_round(), xxh64_read32(), xxh64_read64(), xxh64_rotl(), xxh64_round(), XXHASH64_PRIME1, XXHASH64_PRIME2, XXHASH64_PRIME3, XXHASH64_PRIME4, and XXHASH64_PRIME5.

Referenced by capsule_dict_hash_hook().

Here is the call graph for this function:



Here is the caller graph for this function:



7.96.2.4 xxhash64_reset()

```
void xxhash64_reset (
            XXHash64State * state,
            uint64_t seed )
```

xxhash64_reset - Initialize streaming state

Parameters

state	State to initialize
seed	Hash seed

References XXHash64State::buffer_size, XXHash64State::seed, XXHash64State::total_len, XXHash64State::v1, XXHash64State::v2, XXHash64State::v3, XXHash64State::v4, XXHASH64_PRIME1, and XXHASH64_PRIME2.

Referenced by capsule_dict_hash_hook().

Here is the caller graph for this function:

7.96.2.5 xxhash64_update()

```
void xxhash64_update (
            XXHash64State * state,
            const void * data,
            size_t len )
```

xxhash64_update - Feed data into streaming hash

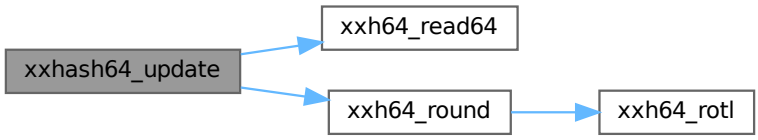
Parameters

state	Streaming state
data	Input data
len	Length in bytes

References XXHash64State::buffer, XXHash64State::buffer_size, limit, XXHash64State::total_len, XXHash64State::v1, XXHash64State::v2, XXHash64State::v3, XXHash64State::v4, xxh64_read64(), and xxh64_round().

Referenced by capsule_dict_hash_hook().

Here is the call graph for this function:

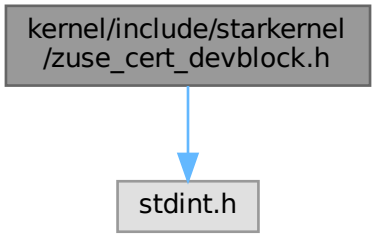


Here is the caller graph for this function:

7.97 kernel/include/starkernel/zuse_cert_devblock.h File Reference

```
#include <stdint.h>
```

Include dependency graph for zuse_cert_devblock.h:



Data Structures

- struct zuse_cert_devblock_t

Macros

- #define ZUSE_CERT_DEVBLOCK_MAGIC ((uint32_t)'Z' | ((uint32_t)'U' << 8) | ((uint32_t)'S' << 16) | ((uint32_t)'E' << 24))
- #define ZUSE_CERT_DEVBLOCK_VERSION 1u

7.97.1 Macro Definition Documentation

7.97.1.1 ZUSE_CERT_DEVBLOCK_MAGIC

```
#define ZUSE_CERT_DEVBLOCK_MAGIC  ((uint32_t)'Z' | ((uint32_t)'U' << 8) | ((uint32_t)'S' << 16) | ((uint32_t)'E' << 24))
```

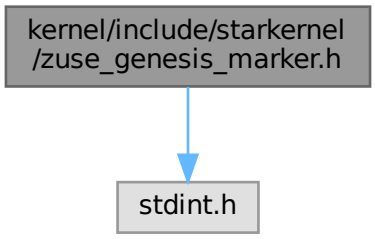
7.97.1.2 ZUSE_CERT_DEVBLOCK_VERSION

```
#define ZUSE_CERT_DEVBLOCK_VERSION 1u
```

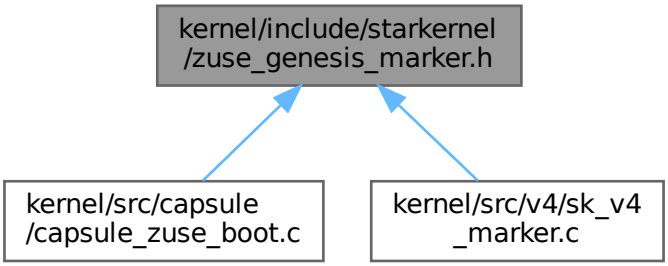
7.98 kernel/include/starkernel/zuse_genesis_marker.h File Reference

```
#include <stdint.h>
```

Include dependency graph for zuse_genesis_marker.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct zuse_genesis_marker_t
- struct zuse_genesis_marker_v2_t

Macros

- #define ZUSE_GENESIS_MARKER_MAGIC ((uint32_t)'Z' | ((uint32_t)'G' << 8) | ((uint32_t)'E' << 16) | ((uint32_t)'N' << 24))
- #define ZUSE_GENESIS_MARKER_VERSION 1u
- #define ZUSE_GENESIS_MARKER_VERSION_2 2u

7.98.1 Macro Definition Documentation

7.98.1.1 ZUSE_GENESIS_MARKER_MAGIC

```
#define ZUSE_GENESIS_MARKER_MAGIC  ((uint32_t)'Z' | ((uint32_t)'G' << 8) | ((uint32_t)'E' <<
16) | ((uint32_t)'N' << 24))
```

7.98.1.2 ZUSE_GENESIS_MARKER_VERSION

```
#define ZUSE_GENESIS_MARKER_VERSION 1u
```

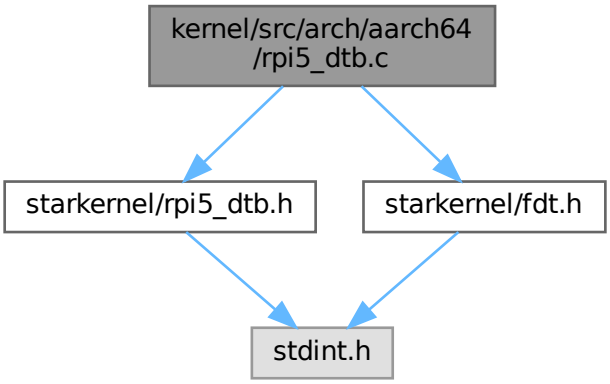
7.98.1.3 ZUSE_GENESIS_MARKER_VERSION_2

```
#define ZUSE_GENESIS_MARKER_VERSION_2 2u
```

7.99 kernel/src/arch/aarch64/rpi5_dtb.c File Reference

```
#include "starkernel/rpi5_dtb.h"
#include "starkernel/fdt.h"
```

Include dependency graph for rpi5_dtb.c:



Macros

- #define BCM2712_SOC_RANGES_OFFSET 0x1000000000ULL

  rpi5_dtb.c - Raspberry Pi 5 (BCM2712) devicetree-based peripheral discovery (FABRIC-3.md ?IV.3, 2026-09-04).

Functions

- static uint32_t reg_first_cell_be32 (const unsigned char *b)

  Read a devicetree `reg` property's first cell as big-endian.
- static uint64_t rpi5_peripheral_base (const void *dtb, const char *compatible)

  Find `compatible's` node, read its `reg` base address, and apply the one fixed BCM2712 `soc`-node translation offset.
- uint64_t rpi5_uart_base (const void *dtb)

  rpi5_dtb.h - Raspberry Pi 5 (BCM2712) devicetree-based peripheral discovery, for the native (non-UEFI) boot path (FABRIC-3.md ?IV.3).
- uint64_t rpi5_mailbox_base (const void *dtb)

  Find the VideoCore mailbox property interface's CPU-physical base address from the DTB.

7.99.1 Macro Definition Documentation

7.99.1.1 BCM2712_SOC_RANGES_OFFSET

```
#define BCM2712_SOC_RANGES_OFFSET 0x1000000000ULL
```

rpi5_dtb.c - Raspberry Pi 5 (BCM2712) devicetree-based peripheral discovery (FABRIC-3.md ?IV.3, 2026-09-04).

Confirmed directly against `bcm2712.dtsi` (raspberrypi/linux, rpi-6.12.y) before writing this, not assumed:

soc: soc@107c000000 { compatible = "simple-bus"; ranges = <0x0 0x10 0x0 0x80000000>; #address-cells = <1>; #size-cells = <1>; ... uart10: serial@7d001000 { compatible = "arm,pl011", "arm,primecell"; reg = <0x7d001000 0x200>; }; mailbox: mailbox@7c013880 { compatible = "brcm,bcm2835-mbox"; reg = <0x7c013880 0x40>; }; };

Every peripheral this file cares about lives under this one "soc" node, which applies exactly one translation to every child `reg` value: child address 0 maps to parent (CPU-physical) address 0x10_0000_0000 (`ranges`'s own `<0x0 0x10 0x0 ...>` — child cell 0, parent cells `0x10 0x0`, i.e. `0x10 << 32`). The child side's own `#address-cells = <1>` confirms each `reg` value's first cell is the whole child address (32-bit, no cell-splitting needed) before adding the offset. `fdt.c`'s reader deliberately does not parse `ranges` generally (it is "not a general devicetree library," per its own header comment) — this one, fixed, SoC-wide offset is applied here by name instead, since it is a single known hardware fact, not a general mechanism this codebase needs elsewhere yet.

7.99.2 Function Documentation

7.99.2.1 reg_first_cell_be32()

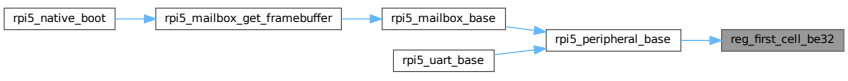
```
static uint32_t reg_first_cell_be32 (
            const unsigned char * b )  [static]
```

Read a devicetree `reg` property's first cell as big-endian.

`fdt.c` keeps its own `be32()` helper file-local (freestanding, no shared byte-swap utility to reuse) – duplicated here rather than exposing it, same "a few lines is simpler than a new shared dependency" precedent `tools/pkcs8_ed25519.c` already set in this codebase.

Referenced by rpi5_peripheral_base().

Here is the caller graph for this function:



7.99.2.2 rpi5_mailbox_base()

```
uint64_t rpi5_mailbox_base (
            const void * dtb )
```

Find the VideoCore mailbox property interface's CPU-physical base address from the DTB.

Parameters

dtb	Devicetree blob; NULL is safe.

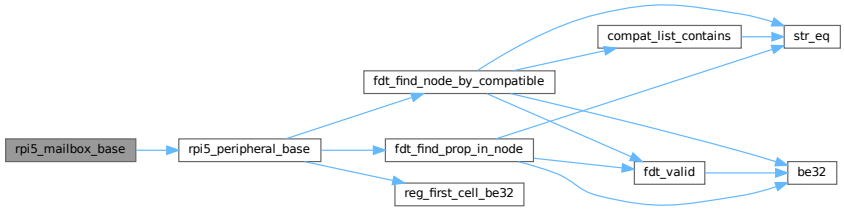
Returns

Final CPU-physical MMIO base address, or 0 if the node is absent, malformed, or `dtb` is invalid.

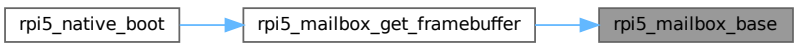
References rpi5_peripheral_base().

Referenced by rpi5_mailbox_get_framebuffer().

Here is the call graph for this function:



Here is the caller graph for this function:



7.99.2.3 rpi5_peripheral_base()

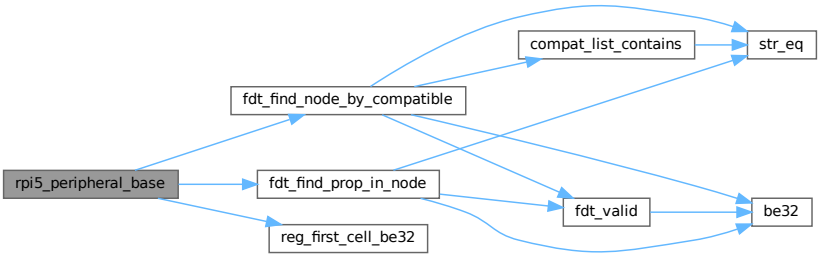
```
static uint64_t rpi5_peripheral_base (
            const void * dtb,
            const char * compatible )  [static]
```

Find `compatible`'s node, read its `reg` base address, and apply the one fixed BCM2712 `soc`-node translation offset.

References BCM2712_SOC_RANGES_OFFSET, fdt_find_node_by_compatible(), fdt_find_prop_in_node(), and reg_first_cell_be32().

Referenced by rpi5_mailbox_base(), and rpi5_uart_base().

Here is the call graph for this function:



Here is the caller graph for this function:

7.99.2.4 rpi5_uart_base()

```
uint64_t rpi5_uart_base (
            const void * dtb )
```

rpi5_dtb.h - Raspberry Pi 5 (BCM2712) devicetree-based peripheral discovery, for the native (non-UEFI) boot path (FABRIC-3.md ?IV.3).

Two lookups: the PL011 UART (early console) and the VideoCore mailbox property interface (framebuffer setup, ?IV.3 item 3). Both peripherals live under BCM2712's own devicetree "soc" simple-bus node, which applies one fixed address translation to every child `reg` value — see `rpi5_dtb.c`'s own doc comment for the confirmed offset and where it was verified. `fdt.c`'s reader deliberately does not do general `ranges`-property translation (it is "not a general devicetree library"); this file applies the one, fixed, SoC-wide offset by name instead of teaching `fdt.c` a general mechanism for a single known hardware fact.

Find the PL011 UART's CPU-physical base address from the DTB.

Parameters

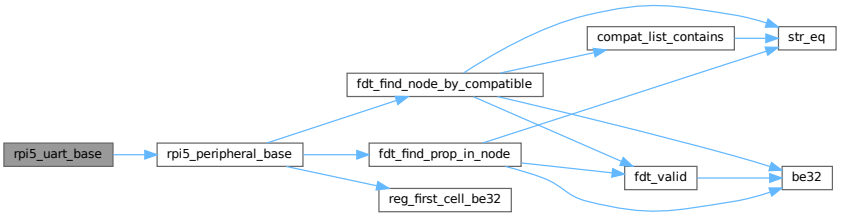
dtb	Devicetree blob, as passed to the native boot entry (or `BootInfo->dtb`); NULL is safe.

Returns

Final CPU-physical MMIO base address, or 0 if the node is absent, malformed, or `dtb` is invalid.

References rpi5_peripheral_base().

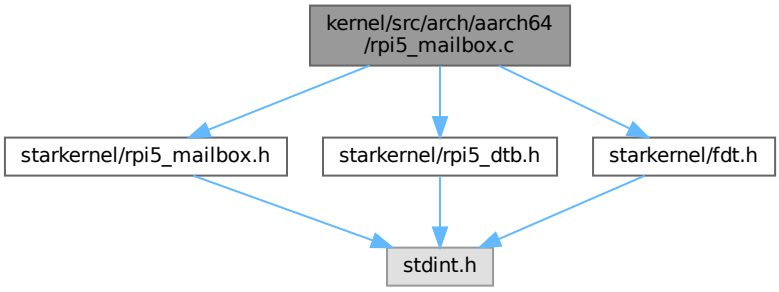
Here is the call graph for this function:



7.100 kernel/src/arch/aarch64/rpi5_mailbox.c File Reference

```
#include "starkernel/rpi5_mailbox.h"
#include "starkernel/rpi5_dtb.h"
#include "starkernel/fdt.h"
```

Include dependency graph for rpi5_mailbox.c:



Macros

- #define MBOX_OFF_READ 0x00u

  rpi5_mailbox.c - VideoCore mailbox property-interface framebuffer setup (FABRIC-3.md ?IV.3 item 3, 2026-09-04).
- #define MBOX_OFF_STATUS0 0x18u
- #define MBOX_OFF_WRITE 0x20u
- #define MBOX_OFF_STATUS1 0x38u
- #define MBOX_FULL 0x80000000u
- #define MBOX_EMPTY 0x40000000u
- #define MBOX_CH_PROPERTY 8u
- #define MBOX_CODE_REQUEST 0x00000000u
- #define MBOX_CODE_RESPONSE_OK 0x80000000u
- #define TAG_SET_PHYS_SIZE 0x00048003u
- #define TAG_SET_VIRT_SIZE 0x00048004u
- #define TAG_SET_DEPTH 0x00048005u
- #define TAG_SET_PIXEL_ORDER 0x00048006u
- #define TAG_SET_VIRTUAL_OFFS 0x00048009u
- #define TAG_ALLOCATE_BUFFER 0x00040001u
- #define TAG_GET_PITCH 0x00040008u
- #define TAG_END 0x00000000u
- #define PIXEL_ORDER_RGB 1u /∗ per the wiki's own tag doc: 0=BGR, 1=RGB ∗/

Functions

- static uint32_t g_mbox_buf[40] __attribute__ ((aligned(16)))
- static void mmio_write32 (uint64_t addr, uint32_t val)
- static uint32_t mmio_read32 (uint64_t addr)
- static uint64_t counter_now (void)
- static uint64_t counter_freq (void)
- static void buffer_to_ram (void)
- static uint32_t mb_be32 (const unsigned char *p)
- static uint64_t mb_cells (const unsigned char *p, uint32_t n)
- static uint32_t mb_cell_prop (const void *dtb, const void *node, const char *name, uint32_t otherwise)
- static int mb_compatible (const void *dtb, const void *node, const char *want)
- static int mb_find_path (const void *dtb, const void *node, const char *want, const void **path, int depth, int max)
- static int bus_to_phys (const void *dtb, uint64_t bus, uint64_t *phys)
- static void append_tag (uint32_t *idx, uint32_t tag_id, uint32_t value_bytes, const uint32_t *values, uint32_t nvalues)
- int rpi5_mailbox_get_framebuffer (const void *dtb, uint32_t width, uint32_t height, uint32_t bpp, Rpi5FramebufferInfo *out)

  Request a framebuffer from the VideoCore firmware via the mailbox property interface, at the requested resolution/depth.

7.100.1 Macro Definition Documentation

7.100.1.1 MBOX_CH_PROPERTY

```
#define MBOX_CH_PROPERTY 8u
```

7.100.1.2 MBOX_CODE_REQUEST

```
#define MBOX_CODE_REQUEST 0x00000000u
```

7.100.1.3 MBOX_CODE_RESPONSE_OK

```
#define MBOX_CODE_RESPONSE_OK 0x80000000u
```

7.100.1.4 MBOX_EMPTY

```
#define MBOX_EMPTY 0x40000000u
```

7.100.1.5 MBOX_FULL

```
#define MBOX_FULL 0x80000000u
```

7.100.1.6 MBOX_OFF_READ

```
#define MBOX_OFF_READ 0x00u
```

rpi5_mailbox.c - VideoCore mailbox property-interface framebuffer setup (FABRIC-3.md ?IV.3 item 3, 2026-09-04).

Never run on real hardware — the Pi 5 isn't in hand until 2026-09-17 (FABRIC-3.md ?II). Everything below is built from real, cited sources rather than assumed, but is compile-only-verified.

Register layout (offsets from `rpi5_mailbox_base()`, i.e. the `mailbox` devicetree node's own translated base address — `rpi5_dtb.c` already confirmed `0x107C013880` on real Pi 5 hardware, matching a Pi-5-specific bare-metal reference (main.lv/writeup/raspberry5_baremetal_framebuffer.md) independently confirming the same `0x107C000000` SoC base + `0x13880` mailbox offset this codebase's own devicetree-driven lookup already produces — two independent sources agreeing is why this offset is trusted, not just cited once):

+0x00 MBOX0_READ (receive: VC -> ARM) +0x18 MBOX0_STATUS (bit30 MBOX_EMPTY: nothing to read) +0x20 MBOX1_WRITE (send: ARM -> VC) +0x38 MBOX1_STATUS (bit31 MBOX_FULL: cannot write yet)

(`+0x20/+0x38` per the same Pi-5-specific reference above, which treats MBOX0/MBOX1 as two structurally identical 0x20-byte blocks back to back — a real difference from older-chip tutorials, which often describe a single shared status register at `+0x18` for both directions. Trusted here because it's the one source actually confirmed against Pi 5 hardware, not an older SoC.)

Message encoding (official wiki: github.com/raspberrypi/firmware/wiki/Mailbox-property-interface, channel and buffer format; property tag IDs/lengths from the same page): write `(buffer_addr & ~0xF) | channel` to MBOX1_WRITE; read back the same encoded value from MBOX0_READ once available, confirming it before trusting the buffer (the response overwrites the request in place — same shared buffer, not a separate response buffer). No bus-address translation applied to the buffer pointer itself (confirmed against the same Pi-5-specific reference: it writes the plain masked pointer, no added GPU-bus-alias bit, unlike the classic 32-bit-Pi convention some older tutorials describe) — genuinely different per-SoC-generation behavior, not an oversight.

Buffer/tag format (official wiki, same page): u32 total size, u32 request(0)/response(0x80000000|len) code, tags..., u32 end-tag(0). Each tag: u32 id, u32 value-buffer size, u32 req-size/resp-flag, value bytes padded to 4-byte alignment. Buffer itself must be 16-byte aligned (only the upper 28 bits of its address travel through the mailbox).

7.100.1.7 MBOX_OFF_STATUS0

```
#define MBOX_OFF_STATUS0 0x18u
```

7.100.1.8 MBOX_OFF_STATUS1

```
#define MBOX_OFF_STATUS1 0x38u
```

7.100.1.9 MBOX_OFF_WRITE

```
#define MBOX_OFF_WRITE 0x20u
```

7.100.1.10 PIXEL_ORDER_RGB

```
#define PIXEL_ORDER_RGB 1u /* per the wiki's own tag doc:  0=BGR, 1=RGB */
```

7.100.1.11 TAG_ALLOCATE_BUFFER

```
#define TAG_ALLOCATE_BUFFER 0x00040001u
```

7.100.1.12 TAG_END

```
#define TAG_END 0x00000000u
```

7.100.1.13 TAG_GET_PITCH

```
#define TAG_GET_PITCH 0x00040008u
```

7.100.1.14 TAG_SET_DEPTH

```
#define TAG_SET_DEPTH 0x00048005u
```

7.100.1.15 TAG_SET_PHYS_SIZE

```
#define TAG_SET_PHYS_SIZE 0x00048003u
```

7.100.1.16 TAG_SET_PIXEL_ORDER

```
#define TAG_SET_PIXEL_ORDER 0x00048006u
```

7.100.1.17 TAG_SET_VIRT_SIZE

```
#define TAG_SET_VIRT_SIZE 0x00048004u
```

7.100.1.18 TAG_SET_VIRTUAL_OFFS

```
#define TAG_SET_VIRTUAL_OFFS 0x00048009u
```

7.100.2 Function Documentation

7.100.2.1 __attribute__()

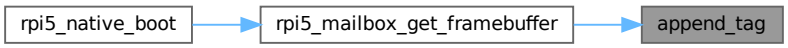
```
static uint32_t g_mbox_buf[40] __attribute__ (
            (aligned(16))  )  [static]
```

7.100.2.2 append_tag()

```
static void append_tag (
             uint32_t * idx,
             uint32_t tag_id,
             uint32_t value_bytes,
             const uint32_t * values,
             uint32_t nvalues )  [static]
```

Referenced by rpi5_mailbox_get_framebuffer().

Here is the caller graph for this function:

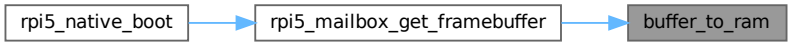


7.100.2.3 buffer_to_ram()

```
static void buffer_to_ram (
             void  )  [static]
```

Referenced by rpi5_mailbox_get_framebuffer().

Here is the caller graph for this function:



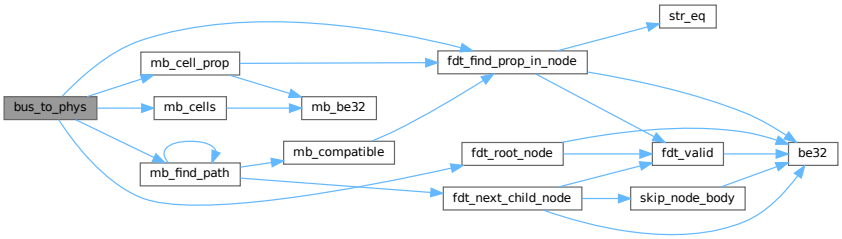
7.100.2.4 bus_to_phys()

```
static int bus_to_phys (
             const void * dtb,
             uint64_t bus,
             uint64_t * phys )  [static]
```

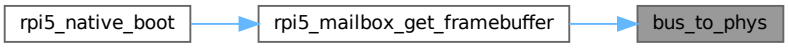
References entry, fdt_find_prop_in_node(), fdt_root_node(), mb_cell_prop(), mb_cells(), and mb_find_path().

Referenced by rpi5_mailbox_get_framebuffer().

Here is the call graph for this function:



Here is the caller graph for this function:

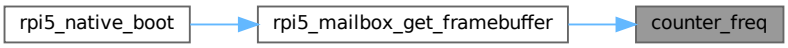


7.100.2.5 counter_freq()

```
static uint64_t counter_freq (
            void  )  [static]
```

Referenced by rpi5_mailbox_get_framebuffer().

Here is the caller graph for this function:

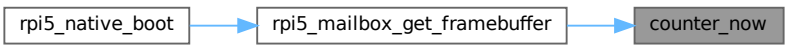


7.100.2.6 counter_now()

```
static uint64_t counter_now (
            void  )  [static]
```

Referenced by rpi5_mailbox_get_framebuffer().

Here is the caller graph for this function:

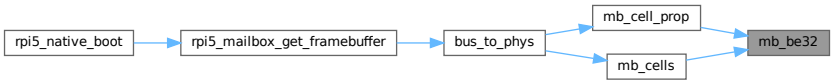


7.100.2.7 mb_be32()

```
static uint32_t mb_be32 (
            const unsigned char * p )  [static]
```

Referenced by mb_cell_prop(), and mb_cells().

Here is the caller graph for this function:



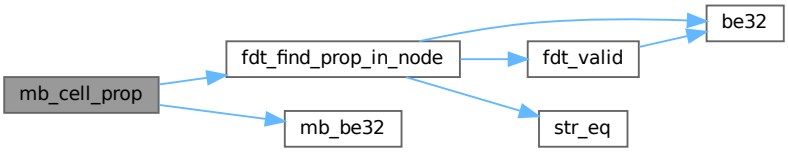
7.100.2.8 mb_cell_prop()

```
static uint32_t mb_cell_prop (
            const void * dtb,
            const void * node,
            const char * name,
            uint32_t otherwise )  [static]
```

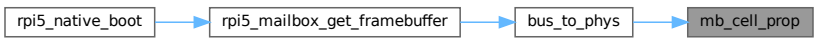
References fdt_find_prop_in_node(), mb_be32(), and name.

Referenced by bus_to_phys().

Here is the call graph for this function:



Here is the caller graph for this function:



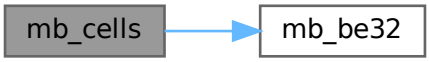
7.100.2.9 mb_cells()

```
static uint64_t mb_cells (
            const unsigned char * p,
            uint32_t n )  [static]
```

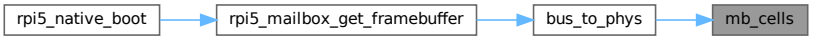
References mb_be32().

Referenced by bus_to_phys().

Here is the call graph for this function:



Here is the caller graph for this function:



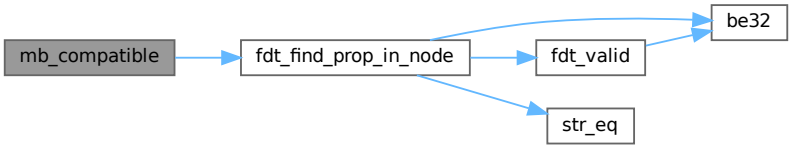
7.100.2.10 mb_compatible()

```
static int mb_compatible (
            const void * dtb,
            const void * node,
            const char * want )  [static]
```

References fdt_find_prop_in_node().

Referenced by mb_find_path().

Here is the call graph for this function:



Here is the caller graph for this function:

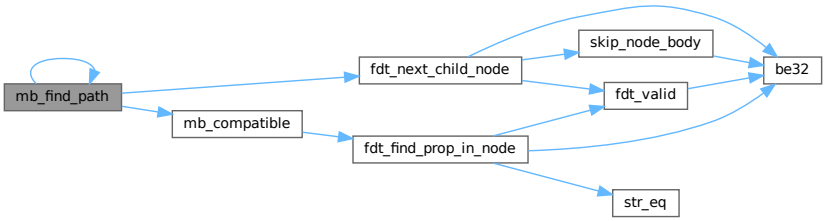
7.100.2.11 mb_find_path()

```
static int mb_find_path (
            const void * dtb,
            const void * node,
            const char * want,
            const void ** path,
            int depth,
            int max )  [static]
```

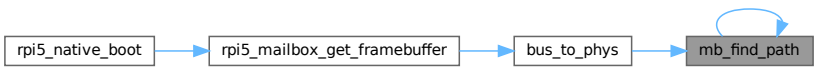
References fdt_next_child_node(), mb_compatible(), and mb_find_path().

Referenced by bus_to_phys(), and mb_find_path().

Here is the call graph for this function:



Here is the caller graph for this function:

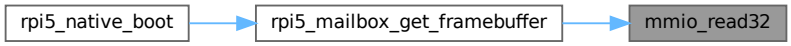


7.100.2.12 mmio_read32()

```
static uint32_t mmio_read32 (
            uint64_t addr )  [inline], [static]
```

Referenced by rpi5_mailbox_get_framebuffer().

Here is the caller graph for this function:

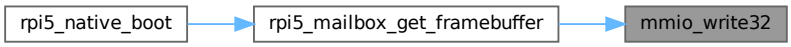


7.100.2.13 mmio_write32()

```
static void mmio_write32 (
            uint64_t addr,
            uint32_t val )  [inline], [static]
```

Referenced by rpi5_mailbox_get_framebuffer().

Here is the caller graph for this function:



7.100.2.14 rpi5_mailbox_get_framebuffer()

```
int rpi5_mailbox_get_framebuffer (
            const void * dtb,
            uint32_t width,
            uint32_t height,
            uint32_t bpp,
            Rpi5FramebufferInfo * out )
```

Request a framebuffer from the VideoCore firmware via the mailbox property interface, at the requested resolution/depth.

Sends one buffer with all six setup tags (physical size, virtual size, depth, pixel order, virtual offset, allocate) plus a get-pitch tag, in one request/response round trip.

Parameters

dtb	Devicetree blob (passed to `rpi5_mailbox_base()`).
width	Requested physical+virtual width, pixels.
height	Requested physical+virtual height, pixels.
bpp	Requested bits per pixel (32 is the only depth this driver has been designed against; others are not refused outright but are unverified).
out	Populated on success; untouched on failure.

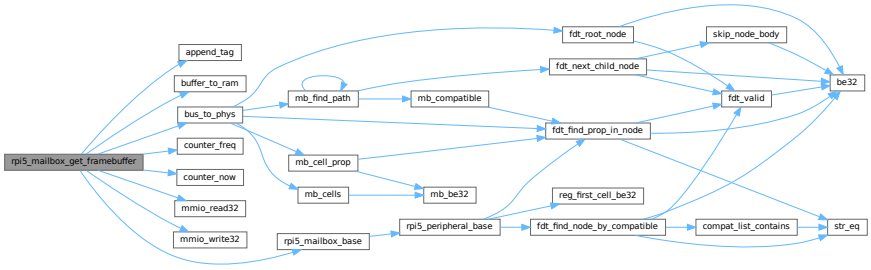
Returns

0 on success, negative on failure: -2 no mailbox node in the DTB, -3 the VC firmware rejected the request, -4 a response came back empty, -5 no answer within one second (every wait is bounded), -6 the devicetree's dma-ranges cannot place the framebuffer's bus address.

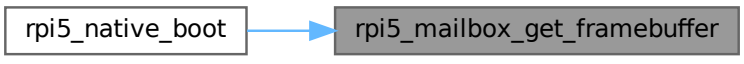
References append_tag(), Rpi5FramebufferInfo::base, base, buffer_to_ram(), bus_to_phys(), counter_freq(), counter_now(), Rpi5FramebufferInfo::height, limit, MBOX_CH_PROPERTY, MBOX_CODE_REQUEST, MBOX_CODE_RESPONSE_, MBOX_EMPTY, MBOX_FULL, MBOX_OFF_READ, MBOX_OFF_STATUS0, MBOX_OFF_STATUS1, MBOX_OFF_WRITE, mmio_read32(), mmio_write32(), Rpi5FramebufferInfo::pixel_format, PIXEL_ORDER_RGB, Rpi5FramebufferInfo::pixels_per_scanline, rpi5_mailbox_base(), Rpi5FramebufferInfo::size, TAG_ALLOCATE_BUFFER, TAG_END, TAG_GET_PITCH, TAG_SET_DEPTH, TAG_SET_PHYS_SIZE, TAG_SET_PIXEL_ORDER, TAG_SET_VIRT_SIZE, TAG_SET_VIRTUAL_OFFS, v2, and Rpi5FramebufferInfo::width.

Referenced by rpi5_native_boot().

Here is the call graph for this function:



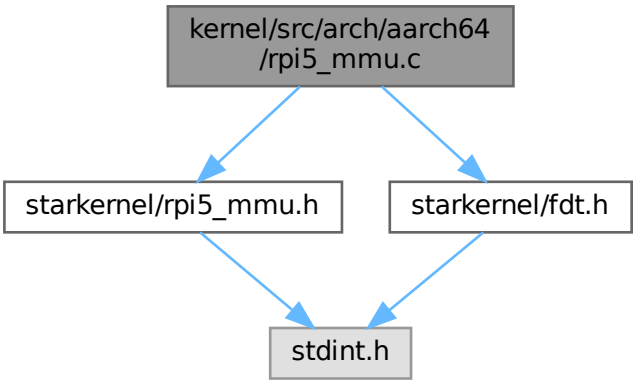
Here is the caller graph for this function:



7.101 kernel/src/arch/aarch64/rpi5_mmu.c File Reference

```
#include "starkernel/rpi5_mmu.h"
#include "starkernel/fdt.h"
```

Include dependency graph for rpi5_mmu.c:



Macros

- #define GIB (1ull << 30)
- #define MIB2 (1ull << 21)
- #define BLOCK_NORMAL_WB 0x701ull /* AF, SH inner, AttrIndx 0 */
- #define BLOCK_NORMAL_NC 0x709ull /* AF, SH inner, AttrIndx 2 */

Functions

- static uint64_t l2_gib (void)
- static uint32_t current_el (void)
- static void tlb_flush_all (void)
- static void replace_entry (volatile uint64_t *entry, uint64_t value)
- static uint32_t be32_at (const unsigned char *p)
- static uint64_t cells_at (const unsigned char *p, uint32_t n)
- int rpi5_mmu_map_ram (const void *dtb)

  rpi5_mmu.h - the Pi 5 native path's identity map, once C runs (rpi5_mmu.c; native_rpi5_entry.S turns the MMU on).
- int rpi5_mmu_uncached (uint64_t addr, uint64_t size)

Variables

- uint64_t g_rpi5_l1 [512]

  rpi5_mmu.c - the Pi 5 native path's identity map, once C runs (2026-10-14; boards/BRINGUP.md R2).
- uint64_t g_rpi5_l2 [512]
- char __kernel_start [ ]
- char __kernel_end [ ]

7.101.1 Macro Definition Documentation

7.101.1.1 BLOCK_NORMAL_NC

```
#define BLOCK_NORMAL_NC 0x709ull /* AF, SH inner, AttrIndx 2 */
```

7.101.1.2 BLOCK_NORMAL_WB

```
#define BLOCK_NORMAL_WB 0x701ull /* AF, SH inner, AttrIndx 0 */
```

7.101.1.3 GIB

```
#define GIB (1ull << 30)
```

7.101.1.4 MIB2

```
#define MIB2 (1ull << 21)
```

7.101.2 Function Documentation

7.101.2.1 be32_at()

```
static uint32_t be32_at (
            const unsigned char * p )  [static]
```

Referenced by cells_at().

Here is the caller graph for this function:

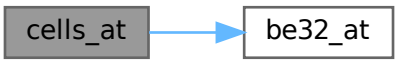
7.101.2.2 cells_at()

```
static uint64_t cells_at (
            const unsigned char * p,
            uint32_t n )  [static]
```

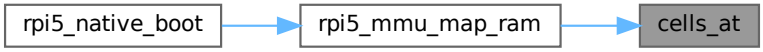
References be32_at().

Referenced by rpi5_mmu_map_ram().

Here is the call graph for this function:



Here is the caller graph for this function:

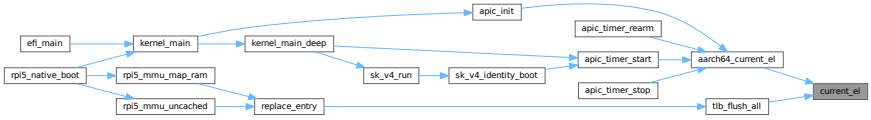


7.101.2.3 current_el()

```
static uint32_t current_el (
            void  )  [static]
```

Referenced by aarch64_current_el(), and tlb_flush_all().

Here is the caller graph for this function:



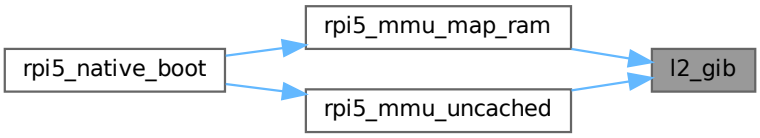
7.101.2.4 l2_gib()

```
static uint64_t l2_gib (
            void  )  [static]
```

References __kernel_start.

Referenced by rpi5_mmu_map_ram(), and rpi5_mmu_uncached().

Here is the caller graph for this function:



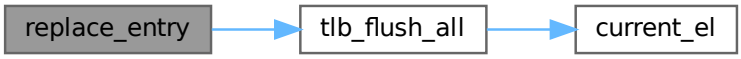
7.101.2.5 replace_entry()

```
static void replace_entry (
            volatile uint64_t * entry,
            uint64_t value )  [static]
```

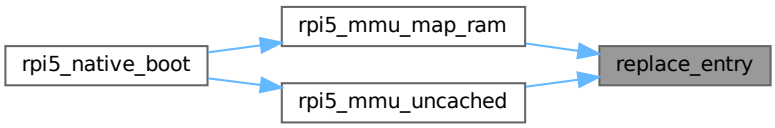
References entry, and tlb_flush_all().

Referenced by rpi5_mmu_map_ram(), and rpi5_mmu_uncached().

Here is the call graph for this function:



Here is the caller graph for this function:



7.101.2.6 rpi5_mmu_map_ram()

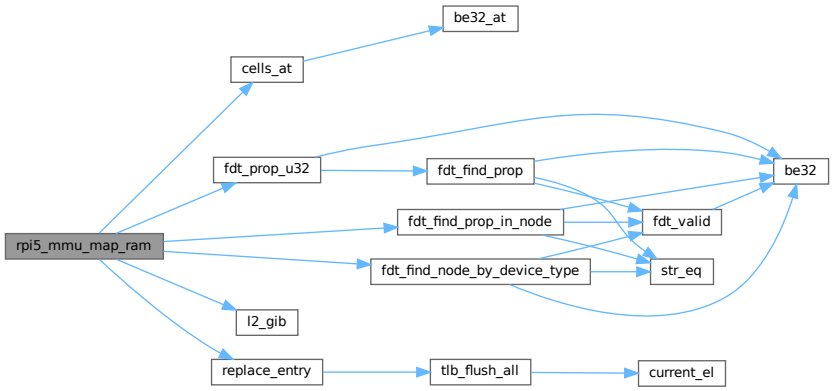
```
int rpi5_mmu_map_ram (
            const void * dtb )
```

rpi5_mmu.h - the Pi 5 native path's identity map, once C runs (rpi5_mmu.c; native_rpi5_entry.S turns the MMU on).

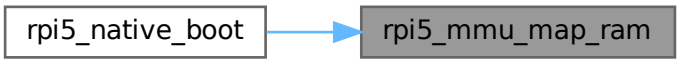
References BLOCK_NORMAL_WB, cells_at(), entry, fdt_find_node_by_device_type(), fdt_find_prop_in_node(), fdt_prop_u32(), g_rpi5_l1, GIB, l2_gib(), and replace_entry().

Referenced by rpi5_native_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



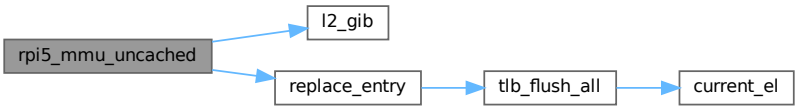
7.101.2.7 rpi5_mmu_uncached()

```
int rpi5_mmu_uncached (
            uint64_t addr,
            uint64_t size )
```

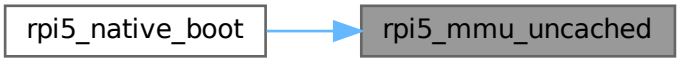
References __kernel_end, __kernel_start, BLOCK_NORMAL_NC, g_rpi5_l1, g_rpi5_l2, l2_gib(), MIB2, and replace_entry().

Referenced by rpi5_native_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.101.2.8 tlb_flush_all()

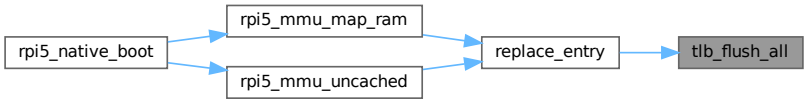
```
static void tlb_flush_all (
            void  )  [static]
```

References current_el().

Referenced by replace_entry().

Here is the call graph for this function:

Here is the caller graph for this function:



7.101.3 Variable Documentation

7.101.3.1 __kernel_end

```
char __kernel_end[]
```

Referenced by pmm_init(), and rpi5_mmu_uncached().

7.101.3.2 __kernel_start

```
char __kernel_start[]  [extern]
```

Referenced by l2_gib(), and rpi5_mmu_uncached().

7.101.3.3 g_rpi5_l1

```
uint64_t g_rpi5_l1[512]  [extern]
```

rpi5_mmu.c - the Pi 5 native path's identity map, once C runs (2026-10-14; boards/BRINGUP.md R2).

native_rpi5_entry.S turns the MMU and caches on before any C: the GiB the image is in Normal write-back in 2 MiB blocks (g_rpi5_l2) – GiB 0 on the Pi 5 – everything else Device, 1 GiB blocks (g_rpi5_l1). From here:

- rpi5_mmu_map_ram() makes every further GiB the devicetree's /memory covers entirely Normal write-back, so RAM past the first GiB is cached and can hold unaligned data – a GiB only partly RAM stays Device rather than let the CPU speculate into what is not RAM;
- rpi5_mmu_uncached() makes a range Normal non-cacheable, for the framebuffer, which the display reads straight from RAM. Each change breaks before it makes (invalid entry, TLB flush, new entry), as the architecture requires for a change of memory type.

Never run on real hardware yet. Rehearsed on QEMU's virt machine at EL2 (FABRIC-3.md ?XXXIX).

Referenced by rpi5_mmu_map_ram(), and rpi5_mmu_uncached().

7.101.3.4 g_rpi5_l2

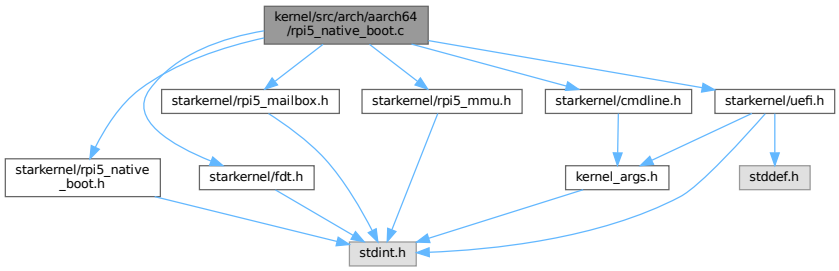
```
uint64_t g_rpi5_l2[512]  [extern]
```

Referenced by rpi5_mmu_uncached().

7.102 kernel/src/arch/aarch64/rpi5_native_boot.c File Reference

```
#include "starkernel/rpi5_native_boot.h"
#include "starkernel/fdt.h"
#include "starkernel/rpi5_mailbox.h"
#include "starkernel/rpi5_mmu.h"
#include "starkernel/cmdline.h"
#include "starkernel/uefi.h"
```

Include dependency graph for rpi5_native_boot.c:



Data Structures

- struct rpi5_range_t

Macros

- #define RPI5_FB_DEFAULT_WIDTH 1920u

  rpi5_native_boot.c - Raspberry Pi 5 DTB->BootInfo constructor (FABRIC-3.md ?IV.3 item 2, 2026-09-04).
- #define RPI5_FB_DEFAULT_HEIGHT 1080u
- #define RPI5_FB_DEFAULT_BPP 32u
- #define RPI5_MAX_RAM_REGIONS 4
- #define RPI5_MAX_RESERVED_RANGES 16
- #define RPI5_MAX_MEMMAP_ENTRIES (RPI5_MAX_RAM_REGIONS ∗ (2 ∗ RPI5_MAX_RESERVED_RANGES + 1))

Functions

- void kernel_main (BootInfo ∗boot_info)

  Main kernel entry point after UEFI handoff — executes milestones M0–M6.
- static uint32_t be32_at (const unsigned char ∗p)
- static uint64_t read_cells (const unsigned char ∗p, uint32_t ncells)

  Read one big-endian cell group (1 or 2 u32 cells) as a uint64_t.
- static void rpi5_native_halt (void)

  Halt this core.
- static uint32_t collect_ram_regions (const void ∗dtb, uint32_t acells, uint32_t scells, rpi5_range_t ∗out, uint32_t max)

  Parse `/memory`'s `reg` into raw (addr, size) ranges, honoring the root's own `#address-cells`/`#size-cells` (NOT the `soc` node's – see this file's own doc comment; `/memory` is a direct child of root).

- static uint32_t node_cells_or_fallback (const void *dtb, const void *node, const char *prop_name, uint32_t fallback)

  Read one node-scoped `#address-cells/#size-cells`-style u32 property, falling back to `fallback` if the node doesn't declare its own (DT spec ?3.5.4's own inheritance rule for `/reserved-memory`).
- static uint32_t collect_reserved_ranges (const void *dtb, uint32_t root_acells, uint32_t root_scells, rpi5_range_t *out, uint32_t max)

  Collect every `/reserved-memory` child's static `reg` range(s).
- static void sort_ranges_by_addr (rpi5_range_t *arr, uint32_t n)

  Insertion sort – small, fixed N; no libc qsort in freestanding.
- static void add_memmap_entry (uint32_t *idx, uint32_t max_entries, uint32_t type, uint64_t addr, uint64_t size)
- static void emit_region_with_carveouts (uint64_t ram_addr, uint64_t ram_size, const rpi5_range_t *reserved, uint32_t reserved_count, uint32_t *idx, uint32_t max_entries)

  Emit one RAM region as alternating conventional/reserved descriptors, clipping `reserved` (already sorted by address) to the region's own bounds.
- static uint32_t build_memory_map (const void *dtb)

  Build `g_rpi5_memmap` from `/memory` with `/reserved-memory`'s static children carved out.
- void rpi5_native_boot (uint64_t dtb)

  rpi5_native_boot.h - Raspberry Pi 5 DTB->BootInfo constructor (FABRIC-3.md ?IV.3 item 2).

Variables

- static BootInfo g_rpi5_boot_info = {0}
- static EFI_MEMORY_DESCRIPTOR g_rpi5_memmap [(4 *(2 *16+1))]

7.102.1 Macro Definition Documentation

7.102.1.1 RPI5_FB_DEFAULT_BPP

```
#define RPI5_FB_DEFAULT_BPP 32u
```

7.102.1.2 RPI5_FB_DEFAULT_HEIGHT

```
#define RPI5_FB_DEFAULT_HEIGHT 1080u
```

7.102.1.3 RPI5_FB_DEFAULT_WIDTH

```
#define RPI5_FB_DEFAULT_WIDTH 1920u
```

rpi5_native_boot.c - Raspberry Pi 5 DTB->BootInfo constructor (FABRIC-3.md ?IV.3 item 2, 2026-09-04).

Never run on real hardware – the Pi 5 isn't in hand until 2026-09-17 (FABRIC-3.md ?II). Compile-only-verified; 3-arch QEMU boot confirms this compiles and links, and does not disturb the existing UEFI/ACPI path – it cannot exercise a single line of this function, since nothing in that path calls it (same caveat as `rpi5_dtb.c`/`rpi5_mailbox.c`/`native_rpi5_entry.S` before it).

`rpi5_native_start` (`native_rpi5_entry.S`) tail-calls this function with the masked DTB pointer it captured. This builds the *existing*, unmodified `BootInfo` struct from the devicetree instead of UEFI protocols, then calls the *existing*, unmodified `kernel_main()` – this is the crux of why most of M1-M9 stays shared between the UEFI and native boot paths.

/memory's own reg (root #address-cells/#size-cells, confirmed against bcm2712.dtsi's actual root node – <2>/<2> – not assumed) is turned into EfiConventionalMemory descriptors, with /reserved-memory children that carry a static reg (brcm,bcm2712-ds.dtsi's real reserved-memory node has one, atf@0, confirmed by fetching that file rather than assumed) carved out as EfiReservedMemoryType descriptors instead of being silently left inside a conventional range. Children with no reg (linux,cma, which uses size/alloc-ranges – a dynamic reservation with no fixed address, confirmed against the same file) have nothing fixed to carve and are skipped – there is no allocator this early to service a dynamic reservation against anyway. no-map/reusable flags are not distinguished: every static reservation is excluded from EfiConventionalMemory regardless of which flag it carries, since this pass has no different treatment for either. /reserved-memory itself may declare its own #address-cells/#size-cells different from the root's (confirmed: bcm2712-ds.dtsi's does – <2>/<1>, not <2>/<2>) – read from the node itself, falling back to root's only if absent, per DT spec ?3.5.4's own inheritance rule.

What this deliberately does NOT do yet, named honestly rather than silently skipped:

- **The mailbox framebuffer request uses a fixed 1920x1080x32 default**, not real display negotiation (no EDID query exists in this codebase). Flagged, not guessed past this comment – revisit once real hardware and a real attached display are in hand.
- **Root and /reserved-memory #address-cells/#size-cells must each be exactly 1 or 2** – other widths halt rather than guess.

7.102.1.4 RPI5_MAX_MEMMAP_ENTRIES

```
#define RPI5_MAX_MEMMAP_ENTRIES (RPI5_MAX_RAM_REGIONS * (2 * RPI5_MAX_RESERVED_RANGES + 1))
```

7.102.1.5 RPI5_MAX_RAM_REGIONS

```
#define RPI5_MAX_RAM_REGIONS 4
```

7.102.1.6 RPI5_MAX_RESERVED_RANGES

```
#define RPI5_MAX_RESERVED_RANGES 16
```

7.102.2 Function Documentation

7.102.2.1 add_memmap_entry()

```
static void add_memmap_entry (
            uint32_t * idx,
            uint32_t max_entries,
            uint32_t type,
            uint64_t addr,
            uint64_t size )  [static]
```

References EFI_MEMORY_DESCRIPTOR::Attribute, g_rpi5_memmap, EFI_MEMORY_DESCRIPTOR::NumberOfPages, EFI_MEMORY_DESCRIPTOR::PhysicalStart, EFI_MEMORY_DESCRIPTOR::Type, and EFI_MEMORY_DESCRIPTOR::VirtualStart

Referenced by emit_region_with_carveouts().

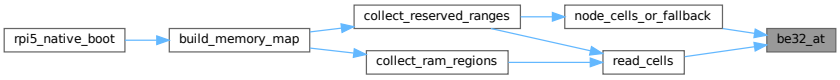
Here is the caller graph for this function:

7.102.2.2 be32_at()

```
static uint32_t be32_at (
            const unsigned char * p )  [static]
```

Referenced by node_cells_or_fallback(), and read_cells().

Here is the caller graph for this function:



7.102.2.3 build_memory_map()

```
static uint32_t build_memory_map (
            const void * dtb )  [static]
```

Build `g_rpi5_memmap` from `/memory` with `/reserved-memory`'s static children carved out.

See this file's own top doc comment for exactly what is and is not handled.

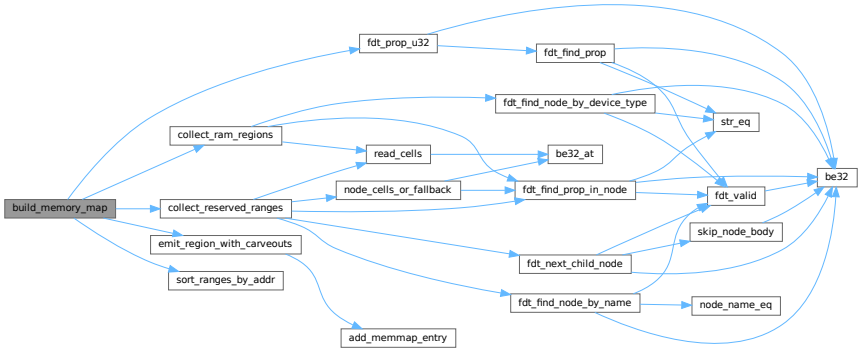
Returns

Number of entries written (0 on any failure – absence of `/reserved-memory` itself is not a failure, just zero carve-outs).

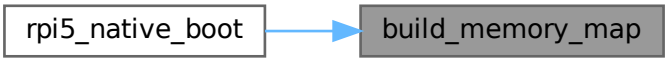
References collect_ram_regions(), collect_reserved_ranges(), emit_region_with_carveouts(), fdt_prop_u32(), RPI5_MAX_MEMMAP_ENTRIES, RPI5_MAX_RAM_REGIONS, RPI5_MAX_RESERVED_RANGES, and sort_ranges_by_addr().

Referenced by rpi5_native_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.102.2.4 collect_ram_regions()

```
static uint32_t collect_ram_regions (
            const void * dtb,
            uint32_t acells,
            uint32_t scells,
            rpi5_range_t * out,
            uint32_t max )  [static]
```

Parse `/memory`'s `reg` into raw (addr, size) ranges, honoring the root's own `#address-cells`/`#size-cells` (NOT the `soc` node's – see this file's own doc comment; `/memory` is a direct child of root).

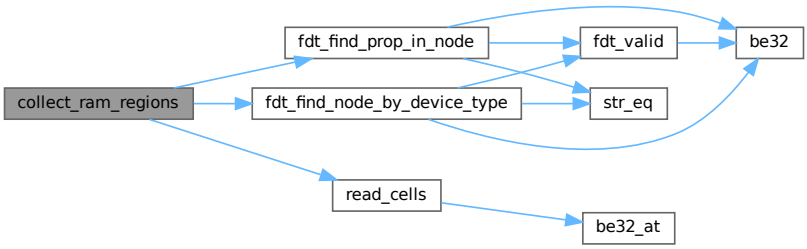
Returns

Number of ranges written (0 on any failure).

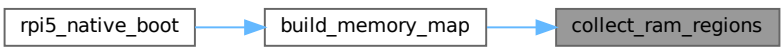
References rpi5_range_t::addr, count, fdt_find_node_by_device_type(), fdt_find_prop_in_node(), read_cells(), and rpi5_range_t::size.

Referenced by build_memory_map().

Here is the call graph for this function:



Here is the caller graph for this function:



7.102.2.5 collect_reserved_ranges()

```
static uint32_t collect_reserved_ranges (
            const void * dtb,
            uint32_t root_acells,
            uint32_t root_scells,
            rpi5_range_t * out,
            uint32_t max )  [static]
```

Collect every `/reserved-memory` child's static `reg` range(s).

Children with no `reg` (dynamic reservations – `size/alloc-ranges` only) are skipped: there is no allocator this early to service a dynamic reservation against, and nothing fixed to carve either way.

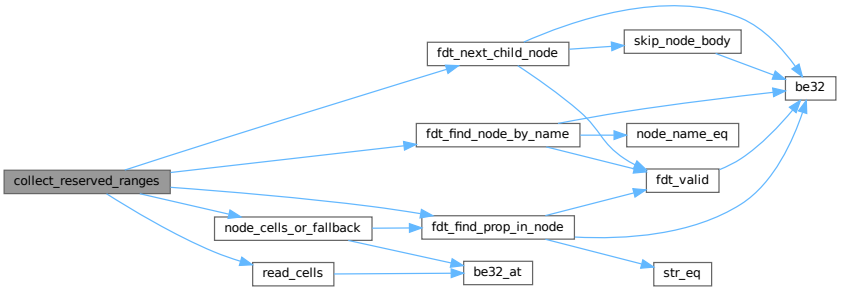
Returns

Number of ranges written (0 if `/reserved-memory` is absent or empty of static reservations – not an error, just nothing to carve).

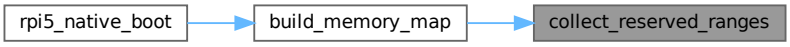
References rpi5_range_t::addr, count, fdt_find_node_by_name(), fdt_find_prop_in_node(), fdt_next_child_node(), node_cells_or_fallback(), read_cells(), and rpi5_range_t::size.

Referenced by build_memory_map().

Here is the call graph for this function:



Here is the caller graph for this function:



7.102.2.6 emit_region_with_carveouts()

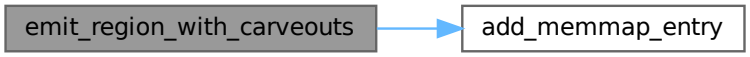
```
static void emit_region_with_carveouts (
            uint64_t ram_addr,
            uint64_t ram_size,
            const rpi5_range_t * reserved,
            uint32_t reserved_count,
            uint32_t * idx,
            uint32_t max_entries )  [static]
```

Emit one RAM region as alternating conventional/reserved descriptors, clipping `reserved` (already sorted by address) to the region's own bounds.

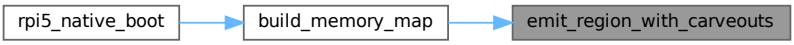
References add_memmap_entry(), rpi5_range_t::addr, EfiConventionalMemory, EfiReservedMemoryType, ram_size, and rpi5_range_t::size.

Referenced by build_memory_map().

Here is the call graph for this function:



Here is the caller graph for this function:



7.102.2.7 kernel_main()

```
void kernel_main (
            BootInfo * boot_info )  [extern]
```

Main kernel entry point after UEFI handoff — executes milestones M0–M6.

On amd64 and riscv64, `kernel_entry.S` switches the stack from UEFI's default to a 2 MiB zero-initialised BSS stack and tail-calls this function as `kernel_main_impl`. On aarch64 the assembly trampoline is not yet implemented and the UEFI loader calls `kernel_main` directly.

Milestone sequence:

- **M0 — Architecture early init** (`arch_early_init()`): On amd64, installs a minimal GDT with a proper 64-bit code segment at selector 0x08 and reloads CS via `lretq`. Without this, UEFI's 64-bit segment at 0x38 is in scope and the ISR's `INT` gate (which expects CS 0x08) would fault silently.

- **M1 — Console** (`console_init()`): brings up UART 16550 at 115200 8N1 and the framebuffer VT100 terminal. Then prints banner and memory map.
- **M2 — PMM** (`pmm_init()`): initialises the physical memory manager's 4 KiB page bitmap from the EFI memory map.
- **M3 — VMM** (`vmm_init()`): builds 4-level x86-64 page tables, maps all conventional RAM at the kernel virtual base, and loads CR3.
- **M4 — IDT + APIC** (`arch_interrupts_init()` + `apic_init()`): programs the 64-entry IDT, masks the legacy 8259A PIC, and initialises the Local APIC in xAPIC MMIO mode at 0xFEE00000.
- **M5 — Timer** (`timer_init()`): calibrates the TSC and HPET.
- **M6 — Heap** (`kmalloc_init()`): initialises the kernel slab allocator with `heap_size` from the boot args or `KARGS_DEFAULT_HEAP_SIZE` (2 GiB).

Stashes `boot_info->runtime_services` in `g_sk_runtime_services` for later use by kernel FORTH words (e.g. `REBOOT`). Then tail-calls `kernel_main_deep()` for M7 and the REPL.

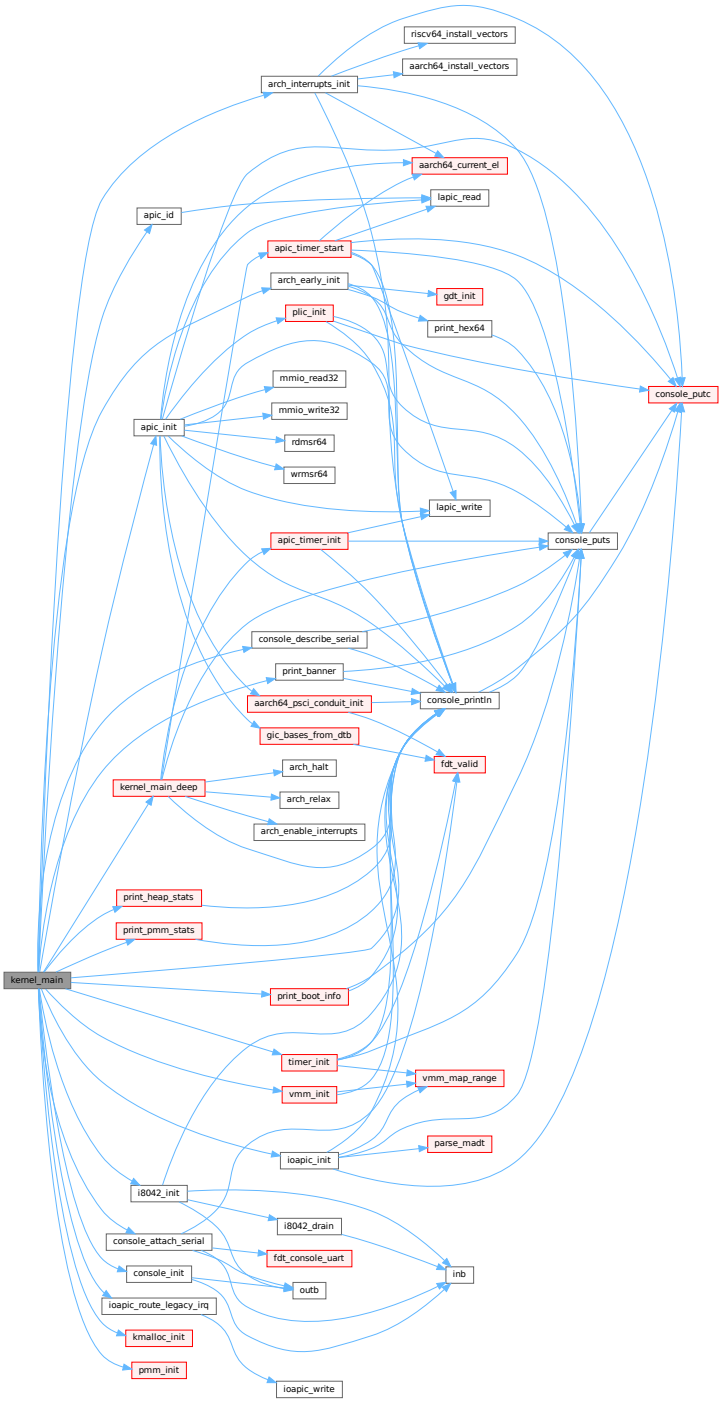
Parameters

boot_info	`BootInfo` populated by `uefi_loader.c` before `ExitBootServices()`; provides the memory map, ACPI pointer, framebuffer descriptor, runtime services pointer, and parsed kernel command-line arguments.

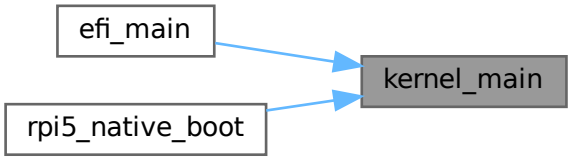
References BootInfo::acpi_table, apic_id(), apic_init(), arch_early_init(), arch_interrupts_init(), BootInfo::args, console_attach_serial(), console_describe_serial(), console_init(), console_println(), BootInfo::dtb, g_sk_runtime_services, KernelArgs::heap_size, i8042_init(), I8042_KEYBOARD_VECTOR, ioapic_init(), ioapic_route_legacy_irq(), KARGS_DEFAULT_HEAP_SIZE, kernel_main_deep(), kmalloc_init(), pmm_init(), print_banner(), print_boot_info(), print_heap_stats(), print_pmm_stats(), BootInfo::runtime_services, timer_init(), and vmm_init().

Referenced by efi_main(), and rpi5_native_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.102.2.8 node_cells_or_fallback()

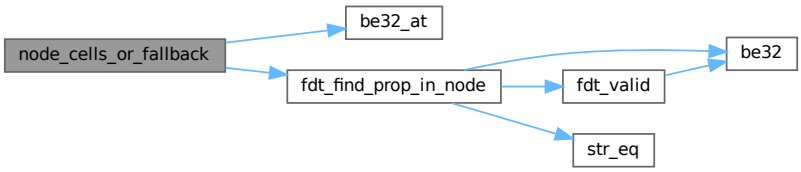
```
static uint32_t node_cells_or_fallback (
            const void * dtb,
            const void * node,
            const char * prop_name,
            uint32_t fallback )  [static]
```

Read one node-scoped `#address-cells`/`#size-cells`-style u32 property, falling back to `fallback` if the node doesn't declare its own (DT spec ?3.5.4's own inheritance rule for `/reserved-memory`).

References be32_at(), and fdt_find_prop_in_node().

Referenced by collect_reserved_ranges().

Here is the call graph for this function:



Here is the caller graph for this function:

7.102.2.9 read_cells()

```
static uint64_t read_cells (
            const unsigned char * p,
            uint32_t ncells )  [static]
```

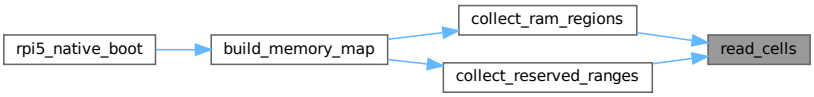
Read one big-endian cell group (1 or 2 u32 cells) as a uint64_t.

References be32_at().

Referenced by collect_ram_regions(), and collect_reserved_ranges().

Here is the call graph for this function:

Here is the caller graph for this function:



7.102.2.10 rpi5_native_boot()

```
void rpi5_native_boot (
            uint64_t dtb )
```

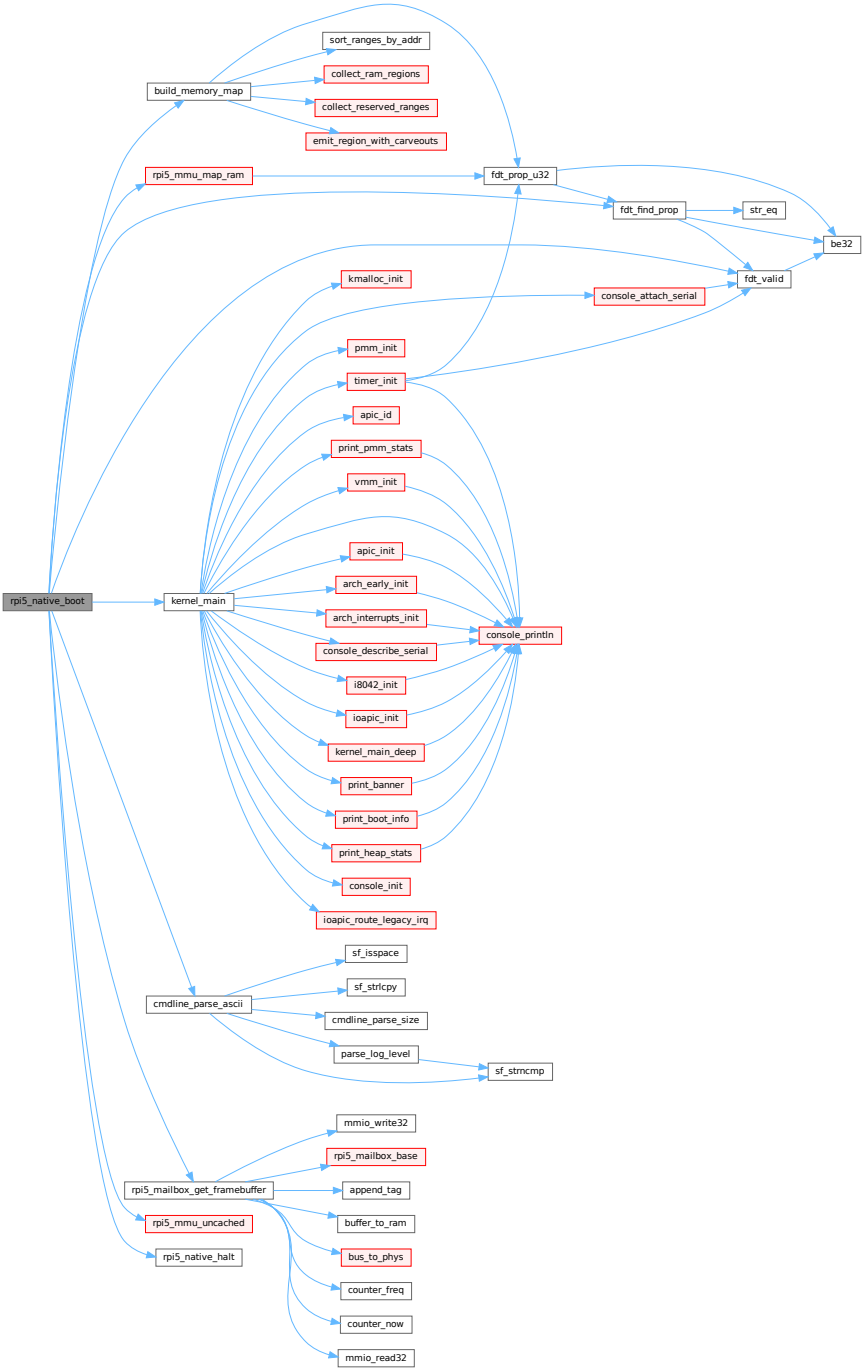
rpi5_native_boot.h - Raspberry Pi 5 DTB->BootInfo constructor (FABRIC-3.md ?IV.3 item 2).

`native_rpi5_entry.S`'s `rpi5_native_start` tail-calls `rpi5_native_boot()` with the masked DTB pointer it captured. This function builds the *existing*, unmodified `BootInfo` struct (`include/starkernel/uefi.h`) from the devicetree instead of UEFI protocols, then calls the *existing*, unmodified `kernel_main()` — see this file's own `.c` for exactly what is and is not populated, and why.

References BootInfo::acpi_table, BootInfo::args, Rpi5FramebufferInfo::base, FramebufferInfo::base, build_memory_map(), cmdline_parse_ascii(), BootInfo::dtb, fdt_find_prop(), fdt_valid(), BootInfo::framebuffer, g_rpi5_boot_info, g_rpi5_memmap, Rpi5FramebufferInfo::height, FramebufferInfo::height, kernel_main(), BootInfo::kernel_stack_base, BootInfo::kernel_stack_size, BootInfo::memory_map, BootInfo::memory_map_descriptor_size, BootInfo::memory_map_size, Rpi5FramebufferInfo::pixel_format, FramebufferInfo::pixel_format, Rpi5FramebufferInfo::pixels_per_scanline, FramebufferInfo::pixels_per_scanline, RPI5_FB_DEFAULT_BPP, RPI5_FB_DEFAULT_HEIGHT, RPI5_FB_DEFAULT_WIDTH, rpi5_mailbox_get_framebuffer(), rpi5_mmu_map_ram(), rpi5_mmu_uncached(), rpi5_native_halt(), BootInfo::runtime_services,

Rpi5FramebufferInfo::size, FramebufferInfo::size, BootInfo::uefi_boot_services_exited, Rpi5FramebufferInfo::width, and FramebufferInfo::width.

Here is the call graph for this function:



7.102.2.11 rpi5_native_halt()

```
static void rpi5_native_halt (
            void  )  [static]
```

Halt this core.

No console exists yet (M1 hasn't run) and there is nowhere safe to report failure – this is the same halt shape `native_rpi5_entry.S` itself uses before this function existed.

Referenced by rpi5_native_boot().

Here is the caller graph for this function:

7.102.2.12 sort_ranges_by_addr()

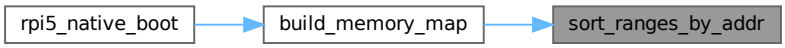
```
static void sort_ranges_by_addr (
            rpi5_range_t * arr,
            uint32_t n )  [static]
```

Insertion sort – small, fixed N; no libc qsort in freestanding.

References rpi5_range_t::addr.

Referenced by build_memory_map().

Here is the caller graph for this function:



7.102.3 Variable Documentation

7.102.3.1 g_rpi5_boot_info

```
BootInfo g_rpi5_boot_info = {0}  [static]
```

Referenced by rpi5_native_boot().

7.102.3.2 g_rpi5_memmap

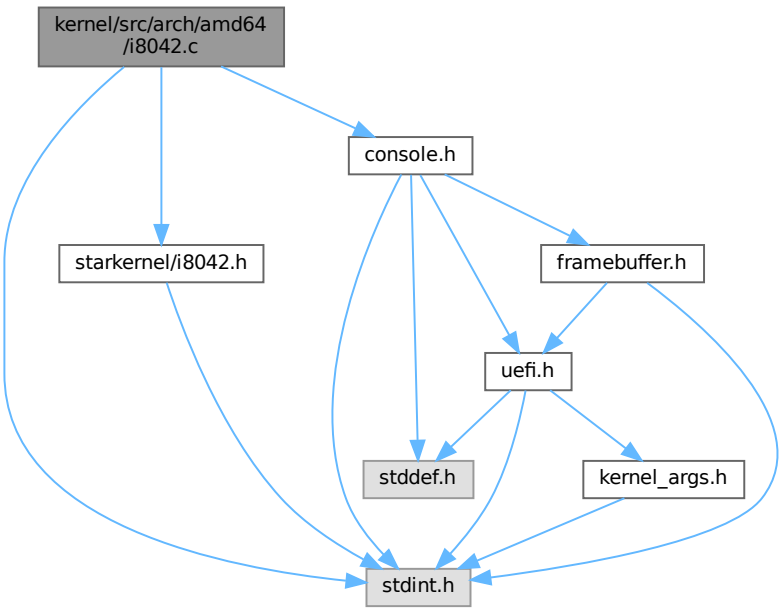
```
EFI_MEMORY_DESCRIPTOR g_rpi5_memmap[(4 *(2 * 16+1))]  [static]
```

Referenced by add_memmap_entry(), and rpi5_native_boot().

7.103 kernel/src/arch/amd64/i8042.c File Reference

```
#include <stdint.h>
#include "starkernel/i8042.h"
#include "console.h"
```

Include dependency graph for i8042.c:



Macros

- #define I8042_DATA_PORT 0x60

  i8042.c - PS/2 keyboard controller driver (amd64)
- #define I8042_STATUS_PORT 0x64
- #define I8042_CMD_PORT 0x64
- #define I8042_STATUS_OUTPUT_FULL (1u << 0)
- #define I8042_STATUS_INPUT_FULL (1u << 1)
- #define I8042_CMD_READ_CONFIG 0x20
- #define I8042_CMD_WRITE_CONFIG 0x60
- #define I8042_CONFIG_IRQ1_ENABLE (1u << 0)
- #define I8042_DEV_CMD_ENABLE_SCANNING 0xF4
- #define I8042_DEV_ACK 0xFA
- #define I8042_INIT_WAIT_ITERS 100000
- #define RING_SIZE 32

Functions

- static void outb (uint16_t port, uint8_t val)
- static uint8_t inb (uint16_t port)
- static void i8042_drain (void)

- void i8042_init (void)

  Enable IRQ1 delivery in the i8042 controller's command byte.
- void i8042_drain_stale (void)

  Drain any byte sitting in the output buffer right before unmasking IRQ1.
- void i8042_handle_irq (void)

  Called from the keyboard IRQ dispatch path (interrupts.c) on every I8042_KEYBOARD_VECTOR interrupt.
- int i8042_pop_scancode (uint8_t *out)

  Pop one raw scancode off the ring buffer, from non-interrupt context.

Variables

- static int g_i8042_present
- static volatile uint8_t ring [32]
- static volatile uint8_t ring_head = 0
- static volatile uint8_t ring_tail = 0
- volatile uint32_t g_i8042_isr_count = 0

7.103.1 Macro Definition Documentation

7.103.1.1 I8042_CMD_PORT

```
#define I8042_CMD_PORT 0x64
```

7.103.1.2 I8042_CMD_READ_CONFIG

```
#define I8042_CMD_READ_CONFIG 0x20
```

7.103.1.3 I8042_CMD_WRITE_CONFIG

```
#define I8042_CMD_WRITE_CONFIG 0x60
```

7.103.1.4 I8042_CONFIG_IRQ1_ENABLE

```
#define I8042_CONFIG_IRQ1_ENABLE (1u << 0)
```

7.103.1.5 I8042_DATA_PORT

```
#define I8042_DATA_PORT 0x60
```

i8042.c - PS/2 keyboard controller driver (amd64)

Item 4.3.5 (FABRIC-0.md ?27.5).

7.103.1.6 I8042_DEV_ACK

```
#define I8042_DEV_ACK 0xFA
```

7.103.1.7 I8042_DEV_CMD_ENABLE_SCANNING

```
#define I8042_DEV_CMD_ENABLE_SCANNING 0xF4
```

7.103.1.8 I8042_INIT_WAIT_ITERS

```
#define I8042_INIT_WAIT_ITERS 100000
```

7.103.1.9 I8042_STATUS_INPUT_FULL

```
#define I8042_STATUS_INPUT_FULL (1u << 1)
```

7.103.1.10 I8042_STATUS_OUTPUT_FULL

```
#define I8042_STATUS_OUTPUT_FULL (1u << 0)
```

7.103.1.11 I8042_STATUS_PORT

```
#define I8042_STATUS_PORT 0x64
```

7.103.1.12 RING_SIZE

```
#define RING_SIZE 32
```

7.103.2 Function Documentation

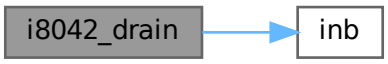
7.103.2.1 i8042_drain()

```
static void i8042_drain (
            void  )  [static]
```

References I8042_DATA_PORT, I8042_STATUS_OUTPUT_FULL, I8042_STATUS_PORT, and inb().

Referenced by i8042_drain_stale(), and i8042_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.103.2.2 i8042_drain_stale()

```
void i8042_drain_stale (
            void  )
```

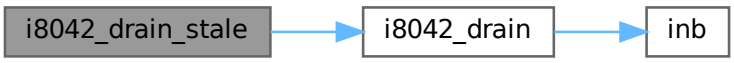
Drain any byte sitting in the output buffer right before unmasking IRQ1.

Edge-triggered lines only assert on a rising edge; if OBF is already set (from real wall-clock time elapsing between i8042_init() and unmask while OBF was already high), there is no edge left to fire on, ever.

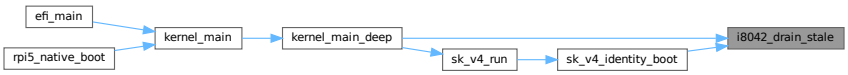
References g_i8042_present, and i8042_drain().

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.103.2.3 i8042_handle_irq()

```
void i8042_handle_irq (
            void  )
```

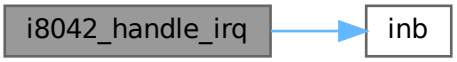
Called from the keyboard IRQ dispatch path (interrupts.c) on every I8042_KEYBOARD_VECTOR interrupt.

Trivial by design: reads the scancode from port 0x60 (must always be read, or the controller never clears OBF and stops delivering further interrupts) and pushes it onto a small ring buffer. No printing, no translation, no other work — that all happens outside interrupt context via i8042_pop_scancode(). Does not issue apic_eoi() — the caller does.

References g_i8042_isr_count, I8042_DATA_PORT, inb(), ring, ring_head, RING_SIZE, and ring_tail.

Referenced by isr_common_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

7.103.2.4 i8042_init()

```
void i8042_init (
            void  )
```

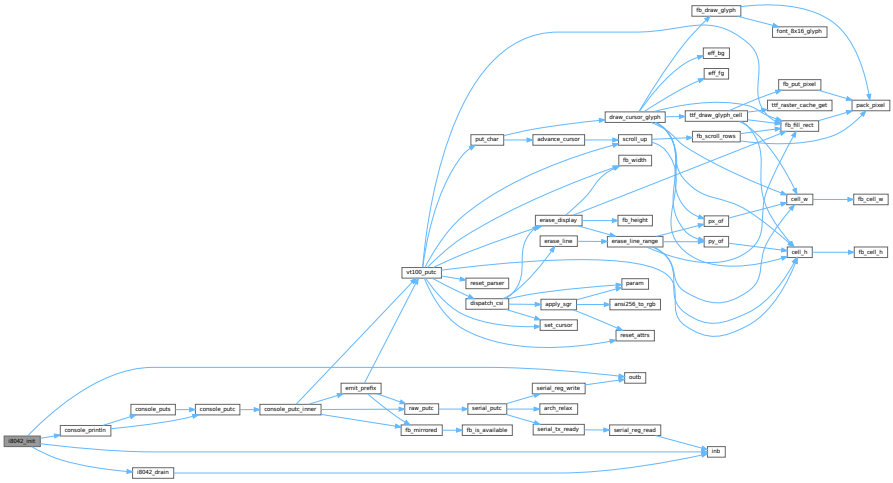
Enable IRQ1 delivery in the i8042 controller's command byte.

Assumes the controller was already brought up by firmware (OVMF) — does not run the 0xAA self-test or the full two-port init sequence, since that is more than this item's scope requires.

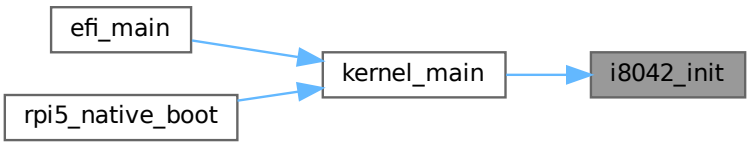
References console_println(), g_i8042_present, I8042_CMD_PORT, I8042_CMD_READ_CONFIG, I8042_CMD_WRITE_CONFIG, I8042_CONFIG_IRQ1_ENABLE, I8042_DATA_PORT, I8042_DEV_ACK, I8042_DEV_CMD_ENABLE_SCANNING, i8042_drain(), I8042_INIT_WAIT_ITERS, I8042_STATUS_INPUT_FULL, I8042_STATUS_OUTPUT_FULL, I8042_STATUS_PORT, inb(), and outb().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.103.2.5 i8042_pop_scancode()

```
int i8042_pop_scancode (
            uint8_t * out )
```

Pop one raw scancode off the ring buffer, from non-interrupt context.

Single producer (the ISR), single reader (this function) — same shape already established safe on one hart elsewhere in this tree (?21.1).

Parameters

out	Written with the popped scancode on success.

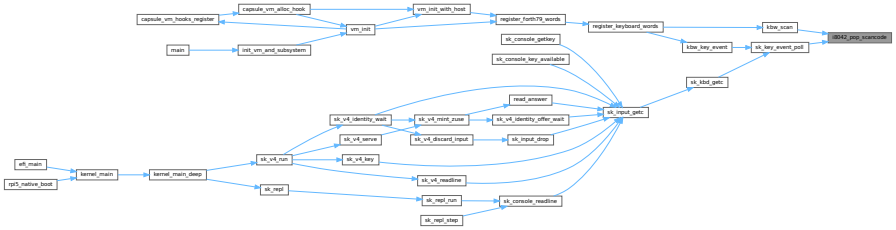
Returns

1 if a scancode was popped, 0 if the buffer was empty.

References ring, ring_head, RING_SIZE, and ring_tail.

Referenced by kbw_scan(), and sk_key_event_poll().

Here is the caller graph for this function:

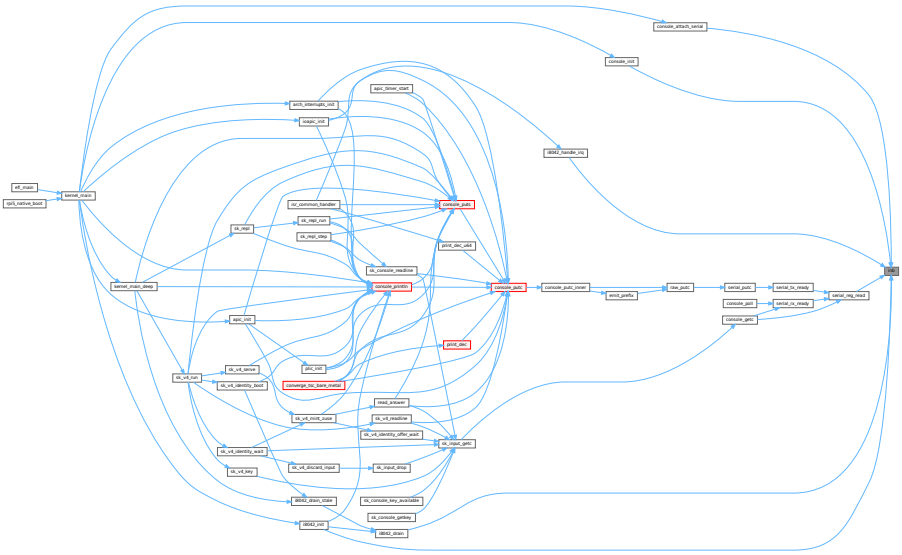


7.103.2.6 inb()

```
static uint8_t inb (
            uint16_t port )  [inline], [static]
```

Referenced by console_attach_serial(), console_init(), i8042_drain(), i8042_handle_irq(), i8042_init(), and serial_reg_read().

Here is the caller graph for this function:

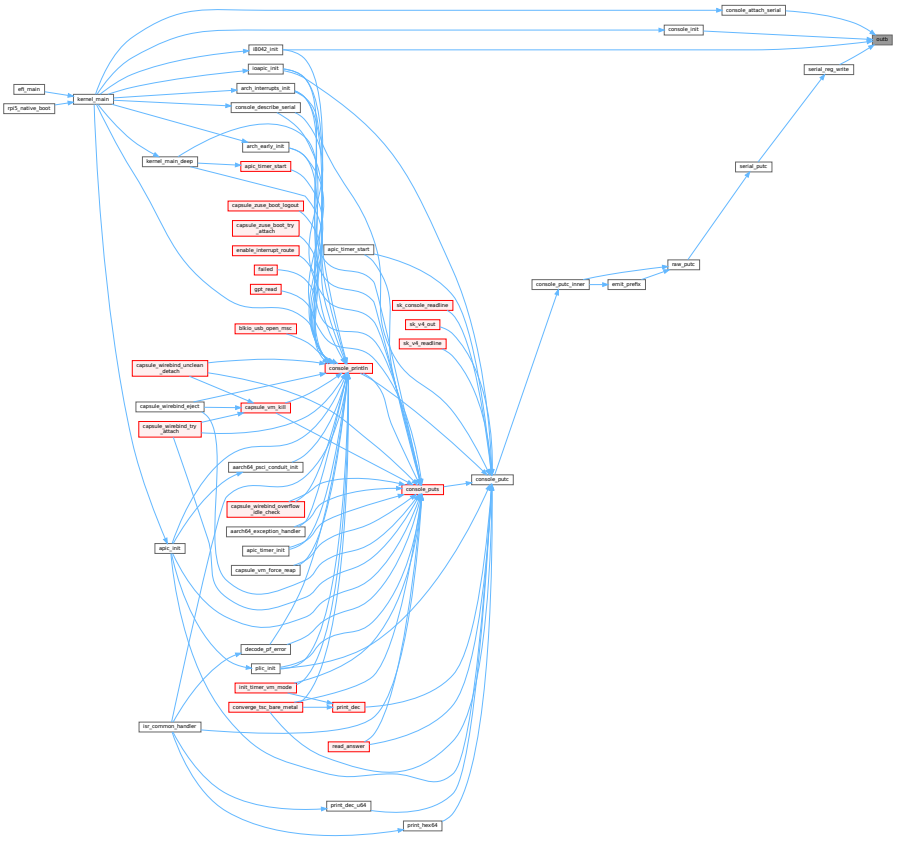


7.103.2.7 outb()

```
static void outb (
            uint16_t port,
            uint8_t val )  [inline], [static]
```

Referenced by console_attach_serial(), console_init(), i8042_init(), and serial_reg_write().

Here is the caller graph for this function:



7.103.3 Variable Documentation

7.103.3.1 g_i8042_isr_count

```
volatile uint32_t g_i8042_isr_count = 0
```

Referenced by i8042_handle_irq(), and kbw_debug().

7.103.3.2 g_i8042_present

```
int g_i8042_present  [static]
```

Referenced by i8042_drain_stale(), and i8042_init().

7.103.3.3 ring

```
volatile uint8_t ring[32]  [static]
```

Referenced by i8042_handle_irq(), i8042_pop_scancode(), physics_analytics_publish_event(), and xhci_init_bulk_ring().

7.103.3.4 ring_head

```
volatile uint8_t ring_head = 0  [static]
```

Referenced by i8042_handle_irq(), and i8042_pop_scancode().

7.103.3.5 ring_tail

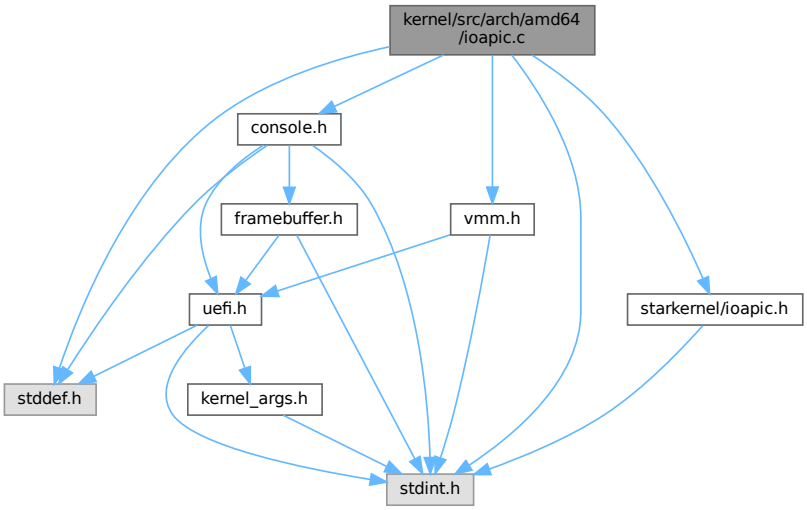
```
volatile uint8_t ring_tail = 0  [static]
```

Referenced by i8042_handle_irq(), and i8042_pop_scancode().

7.104 kernel/src/arch/amd64/ioapic.c File Reference

```
#include <stddef.h>
#include <stdint.h>
#include "starkernel/ioapic.h"
#include "console.h"
#include "vmm.h"
```

Include dependency graph for ioapic.c:



Data Structures

- struct __attribute__

  ioapic.c - I/O APIC driver (amd64)

Macros

- #define MADT_TYPE_IOAPIC 1
- #define MADT_TYPE_INT_SRC 2
- #define ISO_POLARITY_MASK 0x3u
- #define ISO_POLARITY_HIGH 0x1u
- #define ISO_POLARITY_LOW 0x3u
- #define ISO_TRIGGER_MASK 0xCu
- #define ISO_TRIGGER_EDGE 0x4u
- #define ISO_TRIGGER_LEVEL 0xCu
- #define MAX_OVERRIDES 16
- #define MAX_REDIR_ENTRIES 24

Functions

- static void ioapic_write (uint8_t reg, uint32_t val)
- static int sig4_eq (const uint8_t ∗p, const char ∗s)
- static int sig8_eq (const uint8_t ∗p, const char ∗s)
- static int parse_madt (void ∗rsdp)
- int ioapic_init (void ∗acpi_rsdp)

  ioapic.h - I/O APIC interface (amd64 only)
- int ioapic_route_legacy_irq (uint8_t isa_irq, uint8_t vector, uint8_t dest_apic_id)

  Program a redirection entry for a legacy ISA IRQ, initially masked.
- void ioapic_unmask_legacy_irq (uint8_t isa_irq)

  Clear the mask bit on a previously routed legacy IRQ's redirection entry.

Variables

- static uint64_t g_ioapic_base = 0
- static uint32_t g_ioapic_gsi_base = 0
- static int g_ioapic_ready = 0
- static MadtIntSrcOverride g_overrides [16]
- static int g_override_count = 0
- static uint32_t g_redir_low [24]
- static int g_redir_valid [24]

7.104.1 Macro Definition Documentation

7.104.1.1 ISO_POLARITY_HIGH

```
#define ISO_POLARITY_HIGH 0x1u
```

7.104.1.2 ISO_POLARITY_LOW

```
#define ISO_POLARITY_LOW 0x3u
```

7.104.1.3 ISO_POLARITY_MASK

```
#define ISO_POLARITY_MASK 0x3u
```

7.104.1.4 ISO_TRIGGER_EDGE

```
#define ISO_TRIGGER_EDGE 0x4u
```

7.104.1.5 ISO_TRIGGER_LEVEL

```
#define ISO_TRIGGER_LEVEL 0xCu
```

7.104.1.6 ISO_TRIGGER_MASK

```
#define ISO_TRIGGER_MASK 0xCu
```

7.104.1.7 MADT_TYPE_INT_SRC

```
#define MADT_TYPE_INT_SRC 2
```

7.104.1.8 MADT_TYPE_IOAPIC

```
#define MADT_TYPE_IOAPIC 1
```

7.104.1.9 MAX_OVERRIDES

```
#define MAX_OVERRIDES 16
```

7.104.1.10 MAX_REDIR_ENTRIES

```
#define MAX_REDIR_ENTRIES 24
```

7.104.2 Function Documentation

7.104.2.1 ioapic_init()

```
int ioapic_init (
            void * acpi_rsdp )
```

ioapic.h - I/O APIC interface (amd64 only)

Item 4.3.5 (FABRIC-0.md ?27.5): no I/O APIC driver existed anywhere in this tree before this item. The Local APIC (apic.h) self-interrupts for the timer and needs no routing; any *legacy* IRQ (i8042 keyboard's IRQ1 included) requires the I/O APIC to redirect it to a Local APIC vector. Locate and map the I/O APIC via the ACPI MADT (parsed from `acpi_rsdp`). Also captures any Interrupt Source Override entries so legacy ISA IRQs with a non-default GSI/polarity/trigger mapping route correctly.

Parameters

acpi_rsdp	BootInfo->acpi_table (RSDP), or NULL.

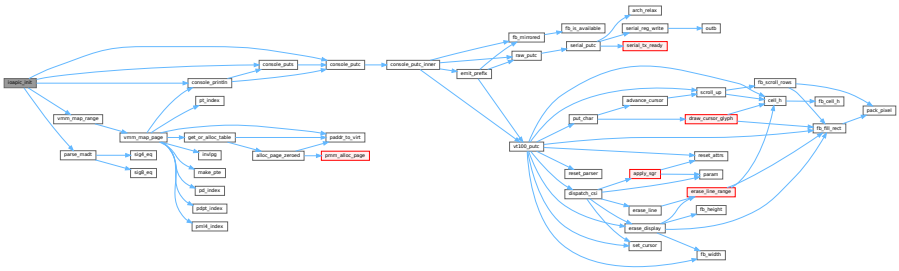
Returns

0 on success, -1 if no MADT/I/O APIC entry was found.

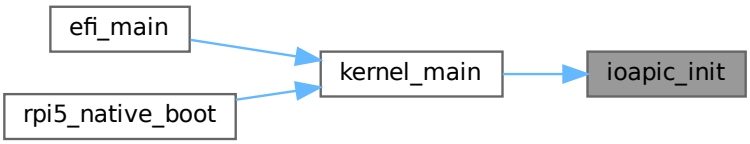
References console_println(), console_putc(), console_puts(), flags, g_ioapic_base, g_ioapic_gsi_base, g_ioapic_ready, g_override_count, g_overrides, parse_madt(), VMM_FLAG_CACHE_DISABLE, VMM_FLAG_WRITABLE, and vmm_map_range().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.104.2.2 ioapic_route_legacy_irq()

```
int ioapic_route_legacy_irq (
            uint8_t isa_irq,
            uint8_t vector,
            uint8_t dest_apic_id )
```

Program a redirection entry for a legacy ISA IRQ, initially masked.

Honors an Interrupt Source Override for `isa_irq` if the MADT had one (different GSI, polarity, or trigger mode); otherwise uses the ISA default (GSI == isa_irq, active-high, edge-triggered) per the ACPI specification's "conforms to bus" default.

Parameters

isa_irq	Legacy IRQ number (0-15).
vector	IDT vector to deliver (e.g. 0x21).
dest_apic↩ _id	Destination Local APIC ID (see apic_id()).

Returns

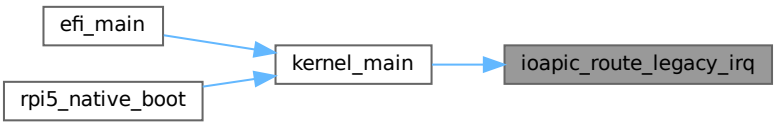
0 on success, -1 if ioapic_init() has not succeeded.

References flags, g_ioapic_gsi_base, g_ioapic_ready, g_override_count, g_overrides, g_redir_low, g_redir_valid, ioapic_write(), ISO_POLARITY_LOW, ISO_POLARITY_MASK, ISO_TRIGGER_LEVEL, ISO_TRIGGER_MASK, and MAX_REDIR_ENTRIES.

Referenced by kernel_main().

Here is the call graph for this function:

Here is the caller graph for this function:



7.104.2.3 ioapic_unmask_legacy_irq()

```
void ioapic_unmask_legacy_irq (
            uint8_t isa_irq )
```

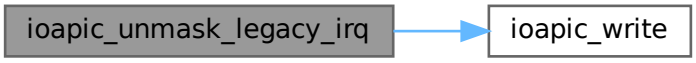
Clear the mask bit on a previously routed legacy IRQ's redirection entry.

Call only after ioapic_route_legacy_irq() has programmed it.

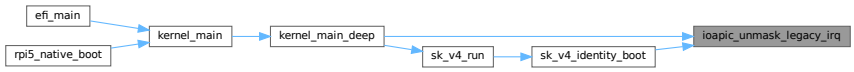
References g_ioapic_gsi_base, g_ioapic_ready, g_override_count, g_overrides, g_redir_low, g_redir_valid, ioapic_write(), and MAX_REDIR_ENTRIES.

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.104.2.4 ioapic_write()

```
static void ioapic_write (
            uint8_t reg,
            uint32_t val )  [static]
```

References g_ioapic_base.

Referenced by ioapic_route_legacy_irq(), and ioapic_unmask_legacy_irq().

Here is the caller graph for this function:

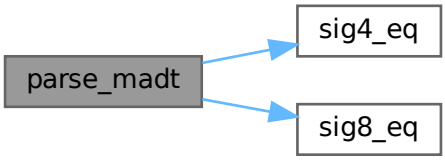
7.104.2.5 parse_madt()

```
static int parse_madt (
            void * rsdp )  [static]
```

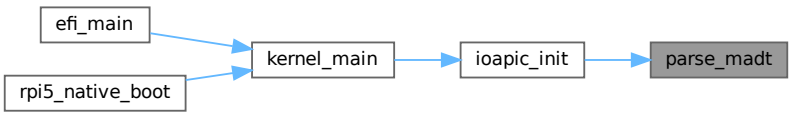
References g_ioapic_base, g_ioapic_gsi_base, g_override_count, g_overrides, MADT_TYPE_INT_SRC, MADT_TYPE_IOAPIC, MAX_OVERRIDES, sig4_eq(), sig8_eq(), and __attribute__::xsdt_address.

Referenced by ioapic_init().

Here is the call graph for this function:



Here is the caller graph for this function:

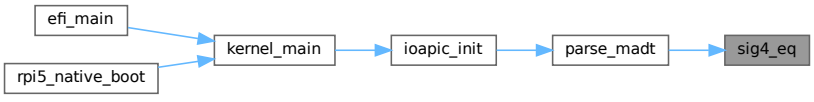


7.104.2.6 sig4_eq()

```
static int sig4_eq (
            const uint8_t * p,
            const char * s )  [static]
```

Referenced by parse_madt().

Here is the caller graph for this function:

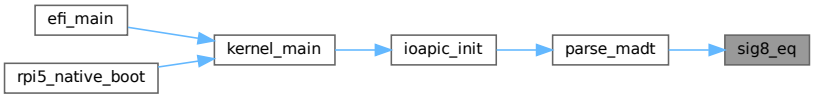


7.104.2.7 sig8_eq()

```
static int sig8_eq (
            const uint8_t * p,
            const char * s )  [static]
```

Referenced by parse_madt().

Here is the caller graph for this function:



7.104.3 Variable Documentation

7.104.3.1 g_ioapic_base

```
uint64_t g_ioapic_base = 0  [static]
```

Referenced by ioapic_init(), ioapic_write(), and parse_madt().

7.104.3.2 g_ioapic_gsi_base

```
uint32_t g_ioapic_gsi_base = 0  [static]
```

Referenced by ioapic_init(), ioapic_route_legacy_irq(), ioapic_unmask_legacy_irq(), and parse_madt().

7.104.3.3 g_ioapic_ready

```
int g_ioapic_ready = 0  [static]
```

Referenced by ioapic_init(), ioapic_route_legacy_irq(), and ioapic_unmask_legacy_irq().

7.104.3.4 g_override_count

```
int g_override_count = 0  [static]
```

Referenced by ioapic_init(), ioapic_route_legacy_irq(), ioapic_unmask_legacy_irq(), and parse_madt().

7.104.3.5 g_overrides

```
MadtIntSrcOverride g_overrides[16]  [static]
```

Referenced by ioapic_init(), ioapic_route_legacy_irq(), ioapic_unmask_legacy_irq(), and parse_madt().

7.104.3.6 g_redir_low

```
uint32_t g_redir_low[24]  [static]
```

Referenced by ioapic_route_legacy_irq(), and ioapic_unmask_legacy_irq().

7.104.3.7 g_redir_valid

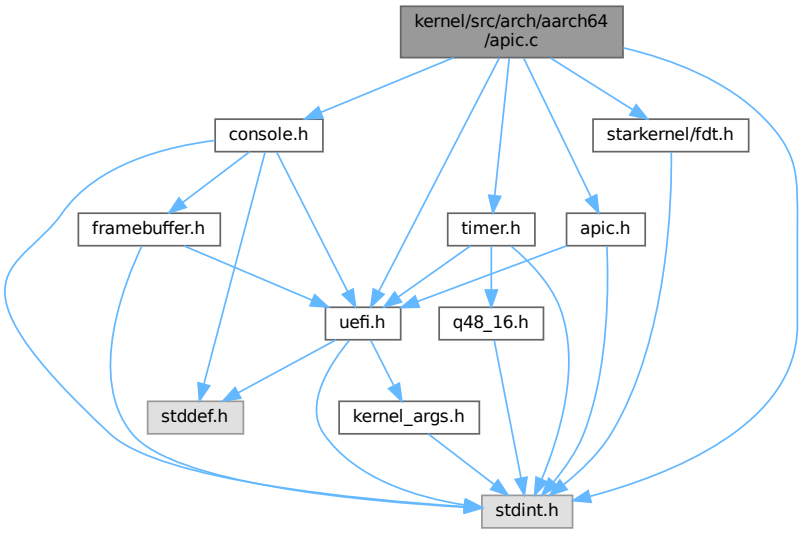
```
int g_redir_valid[24]  [static]
```

Referenced by ioapic_route_legacy_irq(), and ioapic_unmask_legacy_irq().

7.105 kernel/src/arch/aarch64/apic.c File Reference

```
#include "apic.h"
#include "uefi.h"
#include "console.h"
#include "timer.h"
#include "starkernel/fdt.h"
#include <stdint.h>
```

Include dependency graph for apic.c:



Macros

- #define GICD_BASE_PA_QEMU_DEFAULT 0x08000000UL
- #define GICC_BASE_PA_QEMU_DEFAULT 0x08010000UL
- #define TIMER_PPI_EL1 30u
- #define TIMER_PPI_EL2 26u
- #define GICD_CTLR 0x000

- #define GICD_ISENABLER0 0x100
- #define GICD_IPRIORITYR 0x400
- #define GICD_ITARGETSR 0x800 /∗ item 4.3.5d -- SPI target-CPU byte array ∗/
- #define GICD_ICFGR 0xC00
- #define GICD_ISPENDR 0x200 /∗ item 4.3.5d -- software set-pending, SPIs only ∗/
- #define GICC_CTLR 0x00
- #define GICC_PMR 0x04
- #define GICC_IAR 0x0C
- #define GICC_EOIR 0x10
- #define GICD_CTLR_ENABLE_GRP0 (1u << 0)
- #define GICC_CTLR_ENABLE_GRP0 (1u << 0)
- #define TIMER_PRIORITY 0x80u
- #define PMR_ALLOW_ALL 0xFFu

Functions

- int aarch64_current_el (void)

  apic.c (aarch64) - Minimal GICv2 driver + ARM Generic Timer (items 0.6/0.7/ 4.3.5d).
- void aarch64_psci_conduit_init (const void ∗dtb)

  Discover the real PSCI conduit ("smc" vs "hvc") from the DTB.
- static void mmio_write32 (uintptr_t base, uint32_t offset, uint32_t val)
- static uint32_t mmio_read32 (uintptr_t base, uint32_t offset)
- static int gic_bases_from_dtb (const void ∗dtb, uintptr_t ∗gicd_out, uintptr_t ∗gicc_out)

  Try to discover the real GICD/GICC base addresses from the DTB.
- int apic_init (BootInfo ∗boot_info)

  Initialise the GICv2 distributor and CPU interface (M4 milestone).
- void apic_spi_enable (uint32_t intid)

  Enable and route one SPI (item 4.3.5d).
- uint32_t apic_read_iar (void)

  Read the GIC CPU interface's Interrupt Acknowledge Register.
- uint32_t apic_timer_ppi (void)

  Return the timer PPI this build is watching for.
- void apic_eoi_intid (uint32_t intid)

  Signal End of Interrupt to the GIC CPU interface.
- void apic_eoi (void)

  Signal End of Interrupt (common cross-ISA signature — see below).
- int apic_timer_init (uint64_t tsc_hz, uint32_t tick_hz)

  Compute the expected system-counter ticks per heartbeat period.
- void apic_timer_start (void)

  Arm the ARM Generic Timer and enable its interrupt (item 0.7).
- void apic_timer_rearm (void)

  Re-arm the ARM Generic Timer for the next period (item 0.7).
- void apic_timer_stop (void)

  Stop timer delivery by clearing CTL.ENABLE (item 0.7).
- uint64_t apic_timer_period_tsc (void)

  Return the expected system-counter ticks per heartbeat period.

Variables

- static uintptr_t s_gicd_base = 0x08000000UL
- static uintptr_t s_gicc_base = 0x08010000UL
- static uint32_t s_timer_ppi = 30u
- static uint64_t s_timer_period_tsc = 0
- static uint64_t s_counter_hz_apic = 0

7.105.1 Macro Definition Documentation

7.105.1.1 GICC_BASE_PA_QEMU_DEFAULT

```
#define GICC_BASE_PA_QEMU_DEFAULT 0x08010000UL
```

7.105.1.2 GICC_CTLR

```
#define GICC_CTLR 0x00
```

7.105.1.3 GICC_CTLR_ENABLE_GRP0

```
#define GICC_CTLR_ENABLE_GRP0 (1u << 0)
```

7.105.1.4 GICC_EOIR

```
#define GICC_EOIR 0x10
```

7.105.1.5 GICC_IAR

```
#define GICC_IAR 0x0C
```

7.105.1.6 GICC_PMR

```
#define GICC_PMR 0x04
```

7.105.1.7 GICD_BASE_PA_QEMU_DEFAULT

```
#define GICD_BASE_PA_QEMU_DEFAULT 0x08000000UL
```

7.105.1.8 GICD_CTLR

```
#define GICD_CTLR 0x000
```

7.105.1.9 GICD_CTLR_ENABLE_GRP0

```
#define GICD_CTLR_ENABLE_GRP0 (1u << 0)
```

7.105.1.10 GICD_ICFGR

```
#define GICD_ICFGR 0xC00
```

7.105.1.11 GICD_IPRIORITYR

```
#define GICD_IPRIORITYR 0x400
```

7.105.1.12 GICD_ISENABLER0

```
#define GICD_ISENABLER0 0x100
```

7.105.1.13 GICD_ISPENDR

```
#define GICD_ISPENDR 0x200 /* item 4.3.5d -- software set-pending, SPIs only */
```

7.105.1.14 GICD_ITARGETSR

```
#define GICD_ITARGETSR 0x800 /* item 4.3.5d -- SPI target-CPU byte array */
```

7.105.1.15 PMR_ALLOW_ALL

```
#define PMR_ALLOW_ALL 0xFFu
```

7.105.1.16 TIMER_PPI_EL1

```
#define TIMER_PPI_EL1 30u
```

7.105.1.17 TIMER_PPI_EL2

```
#define TIMER_PPI_EL2 26u
```

7.105.1.18 TIMER_PRIORITY

```
#define TIMER_PRIORITY 0x80u
```

7.105.2 Function Documentation

7.105.2.1 aarch64_current_el()

```
int aarch64_current_el (
            void  )  [extern]
```

apic.c (aarch64) - Minimal GICv2 driver + ARM Generic Timer (items 0.6/0.7/ 4.3.5d).

"Minimal" is deliberate and stays deliberate: item 0.6 enabled exactly one PPI (the timer); item 4.3.5d adds the ability to enable exactly one SPI (for virtio-keyboard-pci, item 4.3.5e), still by explicit per-INTID call, not a general GIC driver – no SGI support, no GICv3/ITS, no SPI beyond the one 4.3.5e will actually enable.

GICD/GICC base addresses – DTB-discovered when a DTB is present, QEMU virt-machine constants otherwise (FABRIC-3.md ?IV.3 item 7, 2026-09-04 fix; originally recorded as a deliberate hardcode, FABRIC-0.md item 0.6 GAP-B1 follow-up, because `fdt_valid(boot_info->dtb)` fails on this system's QEMU aarch64 firmware – qemu-efi-aarch64 2025.11-3ubuntu7 does not forward a devicetree to the guest, confirmed live rather than assumed). That reasoning covered the QEMU/UEFI path correctly but never covered the native Pi 5 boot path (`rpi5_native_boot.c`), which receives a real DTB directly and calls this same, unmodified `apic_init()` – audited 2026-09-04, found still hardcoding the QEMU addresses unconditionally, which blocks reaching `ok>` on real BCM2712 hardware (its real GIC-400 sits at `0x10_7fff9000`, nowhere near QEMU's `0x08000000`). `gic_bases_from_dtb()` below closes that gap: tries DTB discovery first, falls back to the QEMU constants only when no DTB is present or the GIC node isn't found there – so the QEMU/UEFI path's existing, working behaviour is unchanged.

QEMU's own values were not recalled from memory either – they were read out of QEMU 10.2.1's own internal devicetree (`qemu-system-aarch64 -machine virt,dumpdtb=...`, decoded with this tree's own fdt.c reader) and are therefore correct for this exact QEMU version, though not guaranteed stable across others. Register *offsets* within each block (GICD_CTLR, GICC_IAR, etc.) are fixed by the GICv2 architecture, not the board, and were cross-checked against Linux's own driver header (linux/irqchip/arm-gic.h, from the linux-headers package installed on the build host) rather than recalled either.

apic.c (aarch64) - Minimal GICv2 driver + ARM Generic Timer (items 0.6/0.7/ 4.3.5d).

Reads `CurrentEL` (bits [3:2] hold the EL number, [1:0] are RES0) on first call and caches the result; every later call returns the cached value. Safe to call at any point in boot, including before `console_init()` — the read itself has no dependency, only the printing of it does (see `arch_interrupts_init()` in `interrupts.c`, the earliest point at which both this and a working console are available).

EDK2 on QEMU's aarch64 `virt` machine has been observed to leave the kernel at either EL1 or EL2 depending on firmware build; nothing in this tree may assume one over the other (FABRIC-0.md ?25.7.1 GAP-B3). Every EL-dependent choice — `VBAR_EL1` vs `VBAR_EL2`, the `ELR_ELx` / `SPSR_ELx` saved-state pair, and `CNTP_*_EL0` vs `CNTHP_*_EL2` — must read this accessor rather than hardcode a level.

Returns

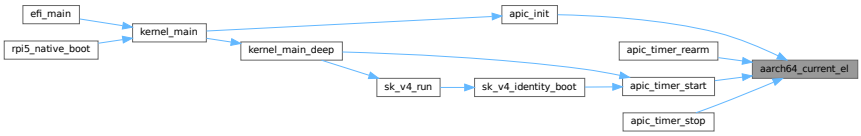
1 or 2. Never 0 or 3: a UEFI application that survived far enough to reach this call cannot be running at EL0 (unprivileged) or EL3 (secure monitor — QEMU's virt firmware runs the kernel at EL1/EL2).

References current_el(), and s_current_el.

Referenced by apic_init(), apic_timer_rearm(), apic_timer_start(), and apic_timer_stop().

Here is the call graph for this function:

Here is the caller graph for this function:



7.105.2.2 aarch64_psci_conduit_init()

```
void aarch64_psci_conduit_init (
            const void * dtb )  [extern]
```

Discover the real PSCI conduit ("smc" vs "hvc") from the DTB.

Looks up the devicetree's PSCI firmware node and reads its `method` property. BCM2712's own `/psci` node declares `compatible = "arm,psci-1.0", "arm,psci-0.2", "arm,psci"` and `method = "smc"` – confirmed directly against `bcm2712.dtsi`, not assumed. `fdt_find_node_by_compatible()` matches any entry in a node's compatible list, so searching for just the first (most specific) string is sufficient; the other two exist for older PSCI-spec-version consumers, not because this node might be tagged differently.

Leaves `s_psci_use_smc` at its default (0, HVC) if `dtb` is not a valid FDT, no PSCI node is found, or `method` is anything other than exactly `"smc"` – including the explicit `"hvc"` case, and the "property absent" case, both of which should keep the existing HVC behaviour rather than guess.

Called once from `apic_init()` (`arch/aarch64/apic.c`), the one point in boot that already has `boot_info->dtb` in hand at the right time for this kind of discovery (same rationale as that file's own GIC-base lookup) – collocated here in `arch.c`, not there, because `arch_cold_reset()` is this function's actual consumer and has no `boot_info` of its own to probe with by the time it runs (called from deep in VM execution, via `BYE`). Same extern-in-place convention this file and apic.c already use for `aarch64_current_el()`.

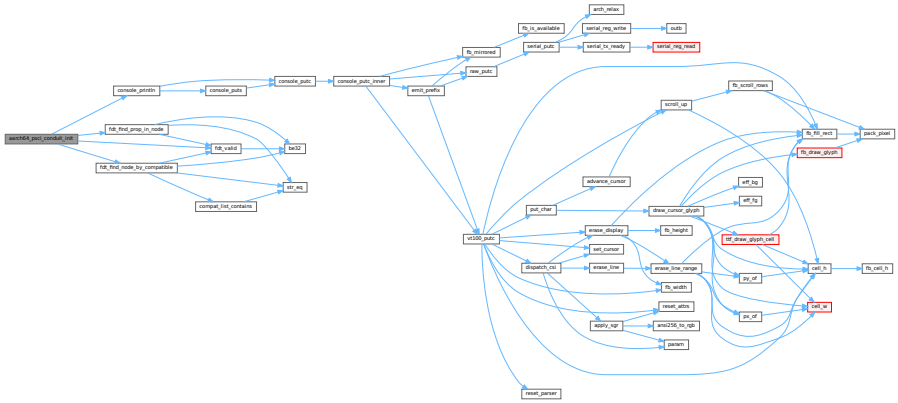
Parameters

dtb	Candidate devicetree blob (`BootInfo->dtb`); NULL-safe.

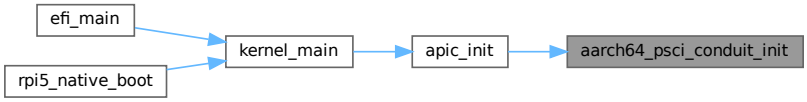
References console_println(), fdt_find_node_by_compatible(), fdt_find_prop_in_node(), fdt_valid(), and s_psci_use_smc.

Referenced by apic_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.105.2.3 apic_eoi()

```
void apic_eoi (
            void  )
```

Signal End of Interrupt (common cross-ISA signature — see below).

The shared `apic.h` interface has no way to pass which interrupt is being completed, but GICv2's `GICC_EOIR` needs the exact INTID `GICC_IAR` returned, not just "an" EOI. `aarch64_irq_handler()` calls `apic_eoi_intid()` directly with that value instead; this form exists only to satisfy the common call site and is not used on the interrupt path GIC brings up.

7.105.2.4 apic_eoi_intid()

```
void apic_eoi_intid (
            uint32_t intid )
```

Signal End of Interrupt to the GIC CPU interface.

Writes `intid` back to `GICC_EOIR`, matching the value `GICC_IAR` returned in `aarch64_irq_handler()`. Required after every acknowledged interrupt or the GIC considers it permanently active and will not re-signal it.

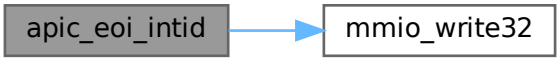
Parameters

intid	INTID as read from `GICC_IAR` (10 bits used by GICv2).

References GICC_EOIR, mmio_write32(), and s_gicc_base.

Referenced by aarch64_irq_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

7.105.2.5 apic_init()

```
int apic_init (
            BootInfo * boot_info )
```

Initialise the GICv2 distributor and CPU interface (M4 milestone).

Enables exactly one interrupt source: the physical timer PPI for the exception level detected in item 0.4 (30 at EL1, 26 at EL2 — see `TIMER_PPI_EL1` / `TIMER_PPI_EL2` above). Sequence:

0. Try `gic_bases_from_dtb()` against `boot_info->dtb`; on success, `s_gicd_base` / `s_gicc_base` are overridden from the real hardware addresses. On failure (no DTB, e.g. this system's QEMU/UEFI firmware — see file header) they keep the QEMU constant defaults, exactly the behaviour this function had before this step existed.

1. Set this interrupt's priority (`GICD_IPRIORITYR`) below the CPU interface's priority mask, so it is never itself masked out.
2. Enable it in `GICD_ISENABLER0` (PPIs 16–31 live in the first word).
3. Enable the distributor, Group 0 only (`GICD_CTLR`) — this deployment has no GICv2 security-extension split in use.
4. Set the CPU interface priority mask to allow everything through (`GICC_PMR` = 0xFF) and enable the CPU interface (`GICC_CTLR`).

Deliberately not configured: `GICD_ICFGR` (trigger sensitivity). The ARM generic timer's PPI is architecturally level-sensitive; this is not a general GIC driver and item 0.6's own scope does not call for reconfiguring it.

Parameters

boot_info	Kernel `BootInfo`; `dtb` is consulted for real-hardware GIC base discovery as of the 2026-09-04 fix (previously unused — see file header), and, as of the same fix, for `aarch64_psci_conduit_init()`'s PSCI conduit discovery (arch.c) — bundled into this call for the same reason GIC discovery is: this is the one point in boot with `boot_info->dtb` already in hand.

Returns

0 always.

7.105.2.6 apic_read_iar()

```
uint32_t apic_read_iar (
            void  )
```

Read the GIC CPU interface's Interrupt Acknowledge Register.

Reading `GICC_IAR` returns the INTID of the highest-priority pending interrupt and simultaneously moves it to the active state — this is the GICv2 acknowledgement step. Called from `aarch64_irq_handler()` in `interrupts.c` to identify which interrupt fired; the matching `apic_eoi_intid()` call below completes it.

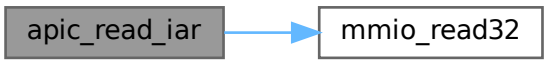
Returns

INTID (10 bits used by GICv2).

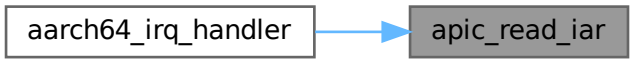
References GICC_IAR, mmio_read32(), and s_gicc_base.

Referenced by aarch64_irq_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.105.2.7 apic_spi_enable()

```
void apic_spi_enable (
            uint32_t intid )
```

Enable and route one SPI (item 4.3.5d).

Extends item 0.6's PPI-only init just far enough to bring up one Shared Peripheral Interrupt – still not a general GIC driver, still one explicit INTID per call, same bounded-scope posture 0.6 used.

Programs, for `intid` (expected range: SPIs, INTID >= 32):

1. `GICD_IPRIORITYR` (byte-indexed, same pattern `apic_init()` already uses for the timer PPI) – `TIMER_PRIORITY`, below `PMR_ALLOW_ALL`.
2. `GICD_ISENABLER` at `0x100` + 4*(intid/32) – PPIs/SGIs (0-31) live in word 0 (`GICD_ISENABLER0`); SPIs beyond use word `intid/32`, same architecture layout, not board-specific.
3. `GICD_ITARGETSR` (byte-indexed like IPRIORITYR) – routes to CPU 0 only; this target has no `-smp`, so no other bit is ever valid.
4. `GICD_ICFGR` is read back, not written, unless the readback disagrees with the expected level-triggered configuration – FABRIC-0.md ?27.5.1's decoded QEMU `interrupt-map` says PCI legacy INTx on this board already is level-triggered by default, so this keeps item 0.6's "don't touch ICFGR unless forced to" posture rather than writing it unconditionally.

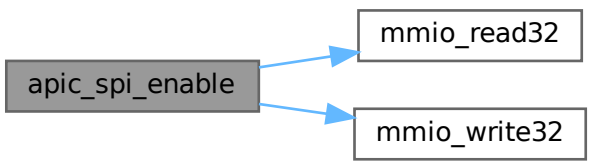
Does **not** touch `GICD_CTLR` or `GICC_CTLR/PMR` – `apic_init()` already enabled the distributor and CPU interface for the timer PPI, and those are global, not per-INTID; nothing here needs to repeat that.

Parameters

intid	Target SPI's INTID (32 + GIC SPI number).

References GICD_ICFGR, GICD_IPRIORITYR, GICD_ISENABLER0, GICD_ITARGETSR, mmio_read32(), mmio_write32(), s_gicd_base, and TIMER_PRIORITY.

Here is the call graph for this function:



7.105.2.8 apic_timer_init()

```
int apic_timer_init (
```

```
            uint64_t tsc_hz,
            uint32_t tick_hz )
```

Compute the expected system-counter ticks per heartbeat period.

Computes s_timer_period_tsc (the expected number of CNTPCT_EL0 ticks between heartbeats), used both by heartbeat_init() to seed the rolling-window expected_delta and by apic_timer_start() / apic_timer_rearm() (item 0.7) as the TVAL step written on every arm and re-arm.

If either tsc_hz or tick_hz is zero, a fallback of 10,000,000 ticks is stored (approximately 160 ms at 62.5 MHz, a safe non-zero sentinel that prevents division-by-zero in the heartbeat path).

Parameters

tsc_hz	System counter frequency from timer_init() (Hz).
tick_hz	Desired heartbeat rate (Hz); 0 → uses fallback.

Returns

0 always.

7.105.2.9 apic_timer_period_tsc()

```
uint64_t apic_timer_period_tsc (
            void  )
```

Return the expected system-counter ticks per heartbeat period.

Get the configured timer period in TSC ticks.

Returns s_timer_period_tsc, set by apic_timer_init() as tsc_hz / tick_hz. This value is consumed by heartbeat_init() to seed the rolling-window expected_delta so that inter-tick deviation measurements are relative to the correct nominal interval.

Returns

Expected CNTPCT_EL0 ticks per heartbeat period; 10,000,000 if frequencies were not available at init time.

7.105.2.10 apic_timer_ppi()

```
uint32_t apic_timer_ppi (
            void  )
```

Return the timer PPI this build is watching for.

Set once in apic_init() from aarch64_current_el() (30 at EL1, 26 at EL2). aarch64_irq_handler() compares apic_read_iar() 's result against this rather than a hardcoded constant.

References s_timer_ppi.

Referenced by aarch64_irq_handler().

Here is the caller graph for this function:

7.105.2.11 apic_timer_rearm()

```
void apic_timer_rearm (
            void  )
```

Re-arm the ARM Generic Timer for the next period (item 0.7).

Re-arm the APIC timer at the current adaptive period (item 0.8, ?26).

Writes a fresh `TVAL` — the entirety of a periodic re-arm, since `TVAL` is relative to "now" rather than an absolute deadline. Unlike riscv64's SBI re-arm (item 0.3), there is no drift-correction bookkeeping: each write means "@c s_timer_period_tsc ticks from this instant," so time spent servicing the interrupt before this call simply lengthens that one period rather than accumulating error across ticks.

This is the one call that must never be skipped. The ARM Generic Timer does not auto-reload: once `TVAL` crosses zero, `CTL.ISTATUS` latches and stays asserted — a level condition — until software writes a new `TVAL` or clears `CTL.ENABLE`. A return path that skips this leaves the interrupt permanently pending, which the GIC would deliver again the instant it is EOI'd: a real interrupt storm, not a hypothetical one (this is exactly the failure this driver was checked against during item 0.6's own verification, and confirmed absent there only because nothing armed the timer yet).

Called from `aarch64_irq_handler()` (interrupts.c) before `heartbeat_tick()` — same ordering discipline as `riscv64_timer_rearm()` (item 0.3): re-arm first, so a fault in the rest of the handler cannot also cost the next tick.

The step written is `heartbeat_next_period_ns()` (item 0.8, ?26) converted to `CNTPCT_EL0` ticks via `s_counter_hz_apic`, not the fixed `s_timer_period_tsc` `apic_timer_start()` used for the very first arm – that value remains `apic_timer_period_tsc()'s` return for `heartbeat_init()'s` initial `expected_delta`, unchanged.

7.105.2.12 apic_timer_start()

```
void apic_timer_start (
            void  )
```

Arm the ARM Generic Timer and enable its interrupt (item 0.7).

Start the APIC timer (enables periodic interrupts).

Writes `CNTP_TVAL_EL0` (or `CNTHP_TVAL_EL2` at EL2 — `aarch64_current_el()`, same EL-aware discipline as items 0.4–0.6) to `s_timer_period_tsc`, then sets `CTL.ENABLE=1`, `CTL.IMASK=0`.

TVAL is a signed 32-bit *relative* countdown — writing it sets "fire @c s_timer_period_tsc ticks from this instant," unlike riscv64's SBI interface (item 0.3), which takes an absolute deadline the caller must compute. No "now" reading is needed here at all.

The ISB after the CTL write is required, not decorative: per the architecture, and confirmed against Linux's own arch_timer_reg_write_cp15() (arch/arm64/include/asm/arch_timer.h), only the *control* register write needs synchronising before the enable/mask state is guaranteed visible to the interrupt pipeline — TVAL/CVAL writes do not carry the same requirement, which is why apic_timer_rearm() below omits it.

s_timer_period_tsc must already be set by apic_timer_init(). If it is still 0, TVAL=0 fires immediately and continuously until apic_timer_rearm() runs — a correctly-functioning if oddly-paced timer, not a hang.

7.105.2.13 apic_timer_stop()

```
void apic_timer_stop (
            void  )
```

Stop timer delivery by clearing CTL.ENABLE (item 0.7).

Stop the APIC timer (disables periodic interrupts).

Per the architecture, ENABLE=0 unconditionally stops the count and de-asserts the interrupt condition regardless of IMASK — a single write is sufficient, no separate mask step needed.

7.105.2.14 gic_bases_from_dtb()

```
static int gic_bases_from_dtb (
            const void * dtb,
            uintptr_t * gicd_out,
            uintptr_t * gicc_out )  [static]
```

Try to discover the real GICD/GICC base addresses from the DTB.

Looks up the "arm,gic-400" node and reads its reg property's first two entries. Returns 0 (leaving gicd_out / gicc_out untouched) if dtb is not a valid FDT, no matching node exists, or the reg property is shorter than two entries – callers must keep their own QEMU-constant default in that case, exactly as before this function existed (FABRIC-3.md ?IV.3 item 7 fix, 2026-09-04; see file header for why this is needed on the native Pi 5 path specifically).

Parameters

dtb	Candidate devicetree blob (BootInfo->dtb); NULL-safe.
gicd_out	Receives the distributor base address on success.
gicc_out	Receives the CPU interface 0 base address on success.

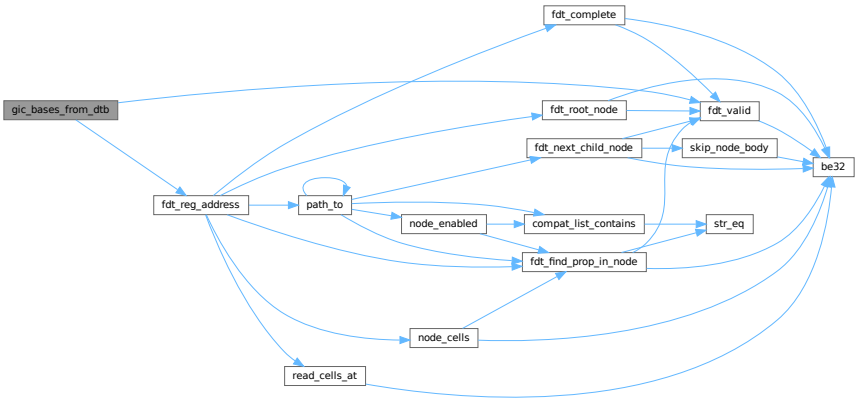
Returns

1 on success, 0 if discovery failed for any reason.

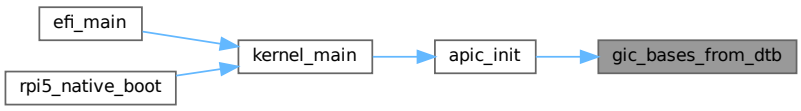
References fdt_reg_address(), and fdt_valid().

Referenced by apic_init().

Here is the call graph for this function:



Here is the caller graph for this function:



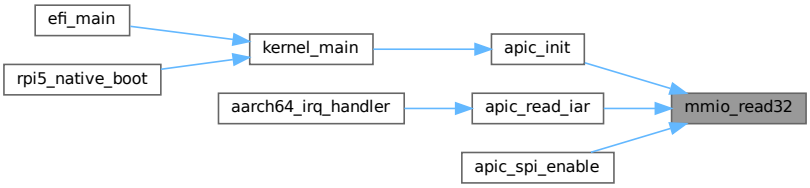
7.105.2.15 mmio_read32()

```
static uint32_t mmio_read32 (
            uintptr_t base,
            uint32_t offset )  [inline], [static]
```

References base.

Referenced by apic_init(), apic_read_iar(), and apic_spi_enable().

Here is the caller graph for this function:



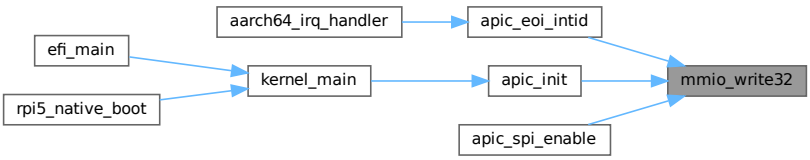
7.105.2.16 mmio_write32()

```
static void mmio_write32 (
            uintptr_t base,
            uint32_t offset,
            uint32_t val )  [inline], [static]
```

References base.

Referenced by apic_eoi_intid(), apic_init(), and apic_spi_enable().

Here is the caller graph for this function:



7.105.3 Variable Documentation

7.105.3.1 s_counter_hz_apic

```
uint64_t s_counter_hz_apic = 0  [static]
```

Referenced by apic_timer_init(), and apic_timer_rearm().

7.105.3.2 s_gicc_base

```
uintptr_t s_gicc_base = 0x08010000UL  [static]
```

Referenced by apic_eoi_intid(), apic_init(), and apic_read_iar().

7.105.3.3 s_gicd_base

```
uintptr_t s_gicd_base = 0x08000000UL  [static]
```

Referenced by apic_init(), and apic_spi_enable().

7.105.3.4 s_timer_period_tsc

```
uint64_t s_timer_period_tsc = 0  [static]
```

Referenced by apic_timer_init(), apic_timer_period_tsc(), apic_timer_rearm(), apic_timer_start(), and riscv64_timer_rearm().

7.105.3.5 s_timer_ppi

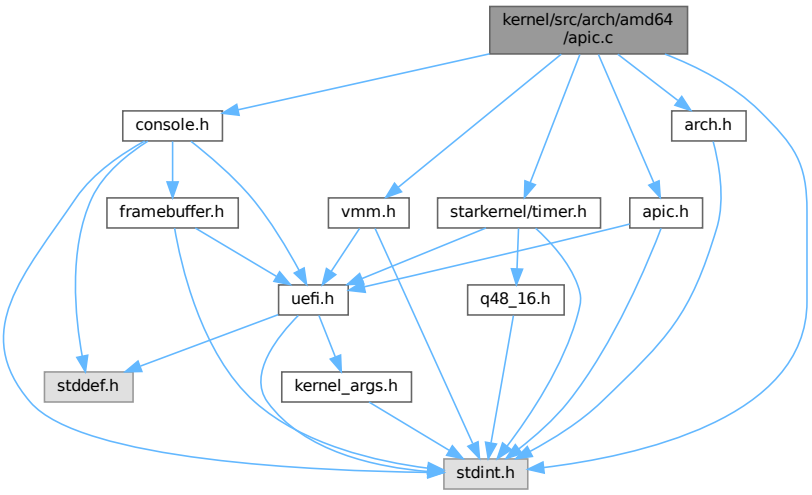
```
uint32_t s_timer_ppi = 30u  [static]
```

Referenced by apic_init(), and apic_timer_ppi().

7.106 kernel/src/arch/amd64/apic.c File Reference

```
#include <stdint.h>
#include "apic.h"
#include "console.h"
#include "vmm.h"
#include "arch.h"
#include "starkernel/timer.h"
```

Include dependency graph for apic.c:



Macros

- #define LAPIC_DEFAULT_PHYS 0xFEE00000u

  apic.c - Local APIC and Timer (amd64)
- #define LAPIC_VIRT_BASE 0xFEE00000u /* identity-mapped */
- #define IA32_APIC_BASE_MSR 0x1Bu
- #define IA32_APIC_BASE_GLOBAL_EN (1ull << 11) /* Global APIC hardware enable */
- #define IA32_APIC_BASE_X2APIC_EN (1ull << 10) /* x2APIC mode enable */
- #define APIC_REG_ID 0x020 /* APIC ID */
- #define APIC_REG_TPR 0x080 /* Task Priority Register */
- #define APIC_REG_EOI 0x0B0 /* End of Interrupt */
- #define APIC_REG_SIVR 0x0F0 /* Spurious Interrupt Vector Register */
- #define APIC_REG_LVT_TIMER 0x320 /* LVT Timer Register */
- #define APIC_REG_TIMER_ICR 0x380 /* Timer Initial Count Register */
- #define APIC_REG_TIMER_CCR 0x390 /* Timer Current Count Register */
- #define APIC_REG_TIMER_DCR 0x3E0 /* Timer Divide Configuration Register */

- #define LVT_MASKED (1u << 16) /* Interrupt masked */
- #define LVT_MODE_PERIODIC (1u << 17) /* Periodic mode */
- #define LVT_MODE_ONESHOT 0 /* One-shot mode (default) */
- #define TIMER_DIV_1 0x0B /* Divide by 1 */
- #define TIMER_DIV_2 0x00 /* Divide by 2 */
- #define TIMER_DIV_4 0x01 /* Divide by 4 */
- #define TIMER_DIV_8 0x02 /* Divide by 8 */
- #define TIMER_DIV_16 0x03 /* Divide by 16 */
- #define TIMER_DIV_32 0x08 /* Divide by 32 */
- #define TIMER_DIV_64 0x09 /* Divide by 64 */
- #define TIMER_DIV_128 0x0A /* Divide by 128 */
- #define APIC_REG_ISR_BASE 0x100

Functions

- static uint64_t rdmsr64 (uint32_t msr)

  Read a 64-bit Model Specific Register via `RDMSR`.
- static void wrmsr64 (uint32_t msr, uint64_t val)

  Write a 64-bit value to a Model Specific Register via `WRMSR`.
- static void lapic_write (uint32_t reg, uint32_t val)

  Write a 32-bit value to a Local APIC MMIO register.
- static uint32_t lapic_read (uint32_t reg)

  Read a 32-bit value from a Local APIC MMIO register.
- int apic_init (BootInfo *boot_info)

  Initialise the Local APIC in xAPIC MMIO mode.
- void apic_eoi (void)

  Signal End of Interrupt to the Local APIC.
- uint8_t apic_id (void)

  Return this CPU's xAPIC ID.
- static uint32_t calibrate_apic_timer (uint64_t tsc_hz)

  Calibrate the APIC timer against the TSC to determine its frequency.
- int apic_timer_init (uint64_t tsc_hz, uint32_t tick_hz)

  Initialise and configure the APIC timer for periodic heartbeat delivery.
- void apic_timer_start (void)

  Unmask and re-arm the APIC timer to begin periodic interrupt delivery.
- void apic_timer_stop (void)

  Mask the APIC timer to suppress further periodic interrupts.
- void apic_timer_rearm (void)

  Re-arm the APIC timer at the current adaptive period (item 0.8, ?26).
- uint64_t apic_timer_period_tsc (void)

  Return the expected TSC-tick count per APIC heartbeat period.

Variables

- static volatile uint32_t * lapic_base = (volatile uint32_t *)(uintptr_t) 0xFEE00000u
- static uint64_t lapic_phys_base = 0xFEE00000u
- static uint32_t timer_initial_count = 0
- static uint64_t timer_period_tsc_ticks = 0
- static uint32_t timer_tick_hz = 0
- static uint32_t s_apic_hz = 0

7.106.1 Macro Definition Documentation

7.106.1.1 APIC_REG_EOI

```
#define APIC_REG_EOI 0x0B0 /* End of Interrupt */
```

7.106.1.2 APIC_REG_ID

```
#define APIC_REG_ID 0x020 /* APIC ID */
```

7.106.1.3 APIC_REG_ISR_BASE

```
#define APIC_REG_ISR_BASE 0x100
```

7.106.1.4 APIC_REG_LVT_TIMER

```
#define APIC_REG_LVT_TIMER 0x320 /* LVT Timer Register */
```

7.106.1.5 APIC_REG_SIVR

```
#define APIC_REG_SIVR 0x0F0 /* Spurious Interrupt Vector Register */
```

7.106.1.6 APIC_REG_TIMER_CCR

```
#define APIC_REG_TIMER_CCR 0x390 /* Timer Current Count Register */
```

7.106.1.7 APIC_REG_TIMER_DCR

```
#define APIC_REG_TIMER_DCR 0x3E0 /* Timer Divide Configuration Register */
```

7.106.1.8 APIC_REG_TIMER_ICR

```
#define APIC_REG_TIMER_ICR 0x380 /* Timer Initial Count Register */
```

7.106.1.9 APIC_REG_TPR

```
#define APIC_REG_TPR 0x080 /* Task Priority Register */
```

7.106.1.10 IA32_APIC_BASE_GLOBAL_EN

```
#define IA32_APIC_BASE_GLOBAL_EN (1ull << 11) /* Global APIC hardware enable */
```

7.106.1.11 IA32_APIC_BASE_MSR

```
#define IA32_APIC_BASE_MSR 0x1Bu
```

7.106.1.12 IA32_APIC_BASE_X2APIC_EN

```
#define IA32_APIC_BASE_X2APIC_EN (1ull << 10) /* x2APIC mode enable */
```

7.106.1.13 LAPIC_DEFAULT_PHYS

```
#define LAPIC_DEFAULT_PHYS 0xFEE00000u
```

apic.c - Local APIC and Timer (amd64)

Provides:

- APIC initialization
- APIC timer for periodic heartbeat interrupts

7.106.1.14 LAPIC_VIRT_BASE

```
#define LAPIC_VIRT_BASE 0xFEE00000u /* identity-mapped */
```

7.106.1.15 LVT_MASKED

```
#define LVT_MASKED (1u << 16) /* Interrupt masked */
```

7.106.1.16 LVT_MODE_ONESHOT

```
#define LVT_MODE_ONESHOT 0 /* One-shot mode (default) */
```

7.106.1.17 LVT_MODE_PERIODIC

```
#define LVT_MODE_PERIODIC (1u << 17) /* Periodic mode */
```

7.106.1.18 TIMER_DIV_1

```
#define TIMER_DIV_1 0x0B /* Divide by 1 */
```

7.106.1.19 TIMER_DIV_128

```
#define TIMER_DIV_128 0x0A /* Divide by 128 */
```

7.106.1.20 TIMER_DIV_16

```
#define TIMER_DIV_16 0x03 /* Divide by 16 */
```

7.106.1.21 TIMER_DIV_2

```
#define TIMER_DIV_2 0x00 /* Divide by 2 */
```

7.106.1.22 TIMER_DIV_32

```
#define TIMER_DIV_32 0x08 /* Divide by 32 */
```

7.106.1.23 TIMER_DIV_4

```
#define TIMER_DIV_4 0x01 /* Divide by 4 */
```

7.106.1.24 TIMER_DIV_64

```
#define TIMER_DIV_64 0x09 /* Divide by 64 */
```

7.106.1.25 TIMER_DIV_8

```
#define TIMER_DIV_8 0x02 /* Divide by 8 */
```

7.106.2 Function Documentation

7.106.2.1 apic_eoi()

```
void apic_eoi (
            void  )
```

Signal End of Interrupt to the Local APIC.

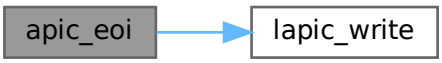
Signal End of Interrupt (common cross-ISA signature — see below).

Writes zero to `APIC_REG_EOI` (offset 0x0B0). This informs the APIC that the currently serviced interrupt has been handled so that the next interrupt in the vector's priority class can be delivered. Must be called at the end of every interrupt service routine before `IRET`; omitting the EOI stalls all future interrupt delivery at the same or lower priority.

For the APIC timer heartbeat path this is called inside `isr_common_handler()` immediately after `heartbeat_tick()`.

References APIC_REG_EOI, and lapic_write().

Here is the call graph for this function:



7.106.2.2 apic_id()

```
uint8_t apic_id (
            void  )
```

Return this CPU's xAPIC ID.

Reads `APIC_REG_ID` and extracts bits 31:24, the xAPIC ID field. Used by the I/O APIC driver (item 4.3.5) as the destination field when programming a redirection entry — interrupts routed to this ID are delivered to the current CPU's Local APIC.

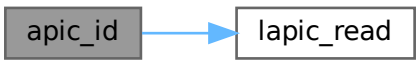
Returns

This CPU's 8-bit xAPIC ID.

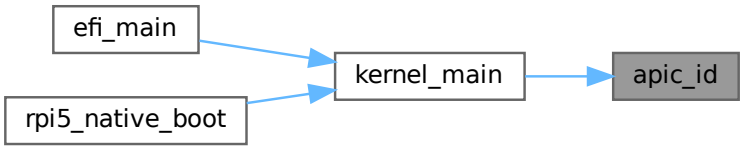
References APIC_REG_ID, and lapic_read().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.106.2.3 apic_init()

```
int apic_init (
            BootInfo * boot_info )
```

Initialise the Local APIC in xAPIC MMIO mode.

Initialise the GICv2 distributor and CPU interface (M4 milestone).

Called during M4 (interrupt controller init) before the IDT is loaded. Performs the following steps:

1. Reads IA32_APIC_BASE_MSR (0x1B) to determine the current APIC mode.
2. If x2APIC mode (bit 10) is active, clears it to switch to xAPIC MMIO mode while keeping the global-enable bit (bit 11) set, as required by Intel SDM Vol.3 ?10.12.5 (never clear both bits simultaneously).
3. If the global-enable bit (bit 11) was clear, sets it.
4. Software-enables the APIC via APIC_REG_SIVR (Spurious Interrupt Vector Register) and assigns spurious vector 0xFF.
5. Clears the Task Priority Register (TPR=0) so all interrupt priorities are delivered; a non-zero TPR left by UEFI would silently suppress all APIC interrupts at or below that priority level.
6. Issues a spurious EOI write to clear any stale in-service interrupt that UEFI may have left pending (specifically the APIC timer vector that fires just before ExitBootServices).

The boot_info parameter is reserved for future use (e.g., reading the ACPI MADT for the physical LAPIC base address); it is currently unused and the default identity-mapped address LAPIC_DEFAULT_PHYS = 0xFEE00000 is always used.

Parameters

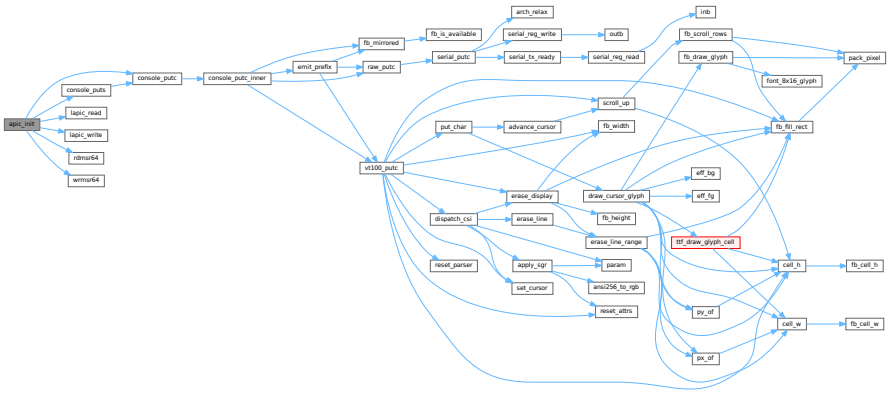
boot_info	Pointer to kernel BootInfo (ACPI/memory-map data); currently unused, reserved for MADT-based LAPIC relocation.

Returns

0 on success (always; the APIC is assumed to be present on x86-64).

References APIC_REG_EOI, APIC_REG_ISR_BASE, APIC_REG_SIVR, APIC_REG_TPR, console_putc(), console_puts(), IA32_APIC_BASE_GLOBAL_EN, IA32_APIC_BASE_MSR, IA32_APIC_BASE_X2APIC_EN, lapic_base, LAPIC_DEFAULT_PHYS, lapic_phys_base, lapic_read(), LAPIC_VIRT_BASE, lapic_write(), rdmsr64(), and wrmsr64().

Here is the call graph for this function:



7.106.2.4 apic_timer_init()

```
int apic_timer_init (
            uint64_t tsc_hz,
            uint32_t tick_hz )
```

Initialise and configure the APIC timer for periodic heartbeat delivery.

Compute the expected system-counter ticks per heartbeat period.

Calibrates the APIC timer frequency via `calibrate_apic_timer()` then programs the LVT Timer Register and Initial Count Register to deliver `APIC_TIMER_VECTOR` periodically at `tick_hz` interrupts per second. The timer is left **masked** after initialisation; call `apic_timer_start()` after the IDT is loaded and interrupts are enabled to begin delivery.

Derived quantities stored for later use:

- `timer_initial_count` — APIC ticks per heartbeat period (`apic_hz` / `tick_hz`), written to `APIC_REG_TIMER_ICR`.
- `timer_tick_hz` — requested tick rate (100 Hz default).
- `timer_period_tsc_ticks` — expected TSC ticks per heartbeat (`tsc_hz` / `tick_hz`), used by `heartbeat_tick()` to compute inter-tick variance for the TIME-TRUST Q48.16 metric.

If `tick_hz` is 0 it defaults to 100 Hz (10 ms period). If calibration returns 0 (no APIC timer detected or counter stuck) the function prints a diagnostic and returns -1 without programming the timer.

Parameters

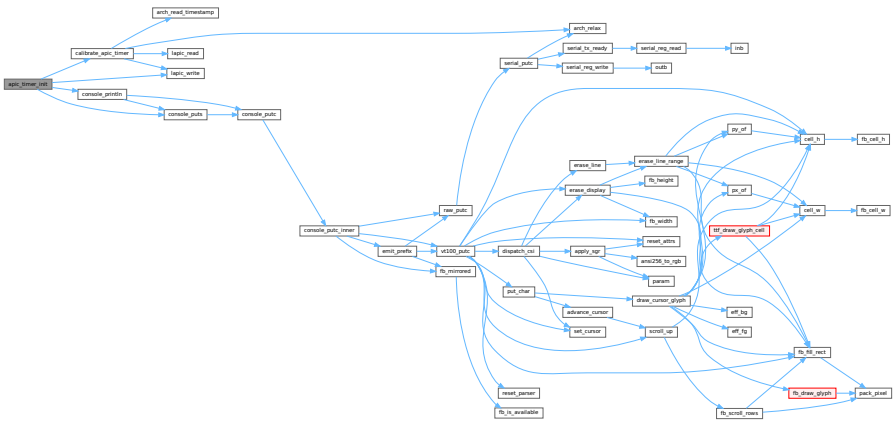
tsc_hz	TSC frequency in Hz (from `timer_init()`); used to derive the 10 ms calibration window and `timer_period_tsc_ticks`. Pass 0 if TSC is not yet calibrated.
tick_hz	Desired interrupt rate in Hz; 0 → defaults to 100 Hz.

Returns

0 on success, -1 if APIC timer calibration failed.

References APIC_REG_LVT_TIMER, APIC_REG_TIMER_DCR, APIC_REG_TIMER_ICR, APIC_TIMER_VECTOR, calibrate_apic_timer(), console_println(), console_puts(), lapic_write(), LVT_MASKED, LVT_MODE_PERIODIC, s_apic_hz, TIMER_DIV_1, timer_initial_count, timer_period_tsc_ticks, and timer_tick_hz.

Here is the call graph for this function:



7.106.2.5 apic_timer_period_tsc()

```
uint64_t apic_timer_period_tsc (
            void  )
```

Return the expected TSC-tick count per APIC heartbeat period.

Get the configured timer period in TSC ticks.

Returns `timer_period_tsc_ticks`, computed by `apic_timer_init()` as `tsc_hz` / `tick_hz`. The value represents the ideal number of TSC ticks that should elapse between consecutive `APIC_TIMER_VECTOR` interrupts when the TSC runs at `tsc_hz` Hz and the heartbeat runs at `tick_hz` Hz.

Used by `timer.c` to seed the heartbeat rolling window with the nominal inter-tick interval so that the first few ticks produce meaningful variance estimates rather than comparing against zero. Returns 0 if `tsc_hz` was 0 at initialisation time (TSC unknown).

Returns

Expected TSC ticks per heartbeat period, or 0 if unavailable.

References timer_period_tsc_ticks.

7.106.2.6 apic_timer_rearm()

```
void apic_timer_rearm (
            void  )
```

Re-arm the APIC timer at the current adaptive period (item 0.8, ?26).

Re-arm the ARM Generic Timer for the next period (item 0.7).

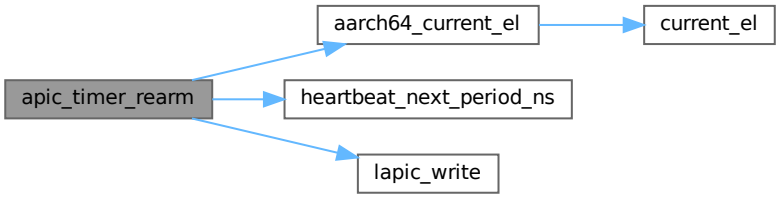
amd64's APIC timer runs in periodic mode: hardware auto-reloads `APIC_REG_TIMER_ICR` on every expiry with no software intervention, unlike riscv64's one-shot SBI deadline or aarch64's one-shot `CNTP_TVAL_EL0` – neither of which needed a rearm function before this item for the same reason this one now exists. To make the period adaptive, this function recomputes `timer_initial_count` from `heartbeat_next_period_ns()` (?26) and `s_apic_hz`, then writes it to `APIC_REG_TIMER_ICR`. A periodic-mode ICR write takes effect immediately and restarts the countdown at the new value – the same mechanism `apic_timer_start()` already relies on to force QEMU TCG's emulated APIC to begin counting.

Called from `isr_common_handler()` on every `APIC_TIMER_VECTOR` interrupt, before `heartbeat_tick()` – same ordering discipline as riscv64/aarch64's rearm-before-heartbeat_tick(), so a fault in `heartbeat_tick()` cannot also cost the next tick.

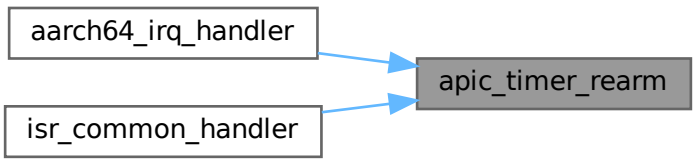
References aarch64_current_el(), APIC_REG_TIMER_ICR, heartbeat_next_period_ns(), lapic_write(), s_apic_hz, s_counter_hz_apic, s_timer_period_tsc, and timer_initial_count.

Referenced by aarch64_irq_handler(), and isr_common_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.106.2.7 apic_timer_start()

```
void apic_timer_start (
            void  )
```

Unmask and re-arm the APIC timer to begin periodic interrupt delivery.

Start the APIC timer (enables periodic interrupts).

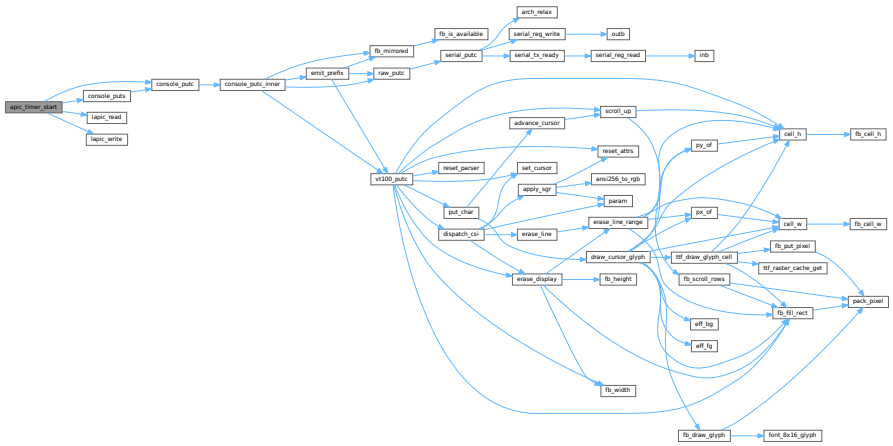
Called after `arch_interrupts_init()` and `arch_enable_interrupts()` to start the 100 Hz heartbeat that drives `heartbeat_tick()`. The sequence is:

1. Read the current LVT Timer Register.
2. Clear the `LVT_MASKED` bit (bit 16) to unmask the interrupt.
3. Write the LVT register back.
4. Write `timer_initial_count` to `APIC_REG_TIMER_ICR` to (re-)arm the countdown. QEMU TCG requires an explicit ICR write after unmask — it does not auto-restart a previously masked periodic timer. On real hardware this is harmless and equally correct.
5. Reads back LVT and ICR values and prints them for boot-log verification.

Must be called exactly once per boot after the IDT is loaded. Calling it again restarts the countdown from the initial value without otherwise reconfiguring the timer.

References APIC_REG_LVT_TIMER, APIC_REG_TIMER_ICR, console_putc(), console_puts(), lapic_read(), lapic_write(), and timer_initial_count.

Here is the call graph for this function:



7.106.2.8 apic_timer_stop()

```
void apic_timer_stop (
            void  )
```

Mask the APIC timer to suppress further periodic interrupts.

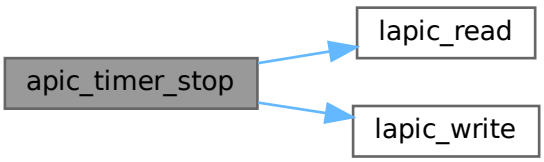
Stop the APIC timer (disables periodic interrupts).

Sets the LVT_MASKED bit (bit 16) in the LVT Timer Register by performing a read-modify-write on APIC↩ _REG_LVT_TIMER. The existing periodic-mode and vector configuration is preserved; only the mask bit changes. The APIC timer's internal countdown continues running but interrupt delivery is suppressed. Calling apic_timer_start() again after this function will unmask and re-arm the timer.

Used in the kernel panic path and during any critical section where heartbeat interrupts would corrupt shared state.

References APIC_REG_LVT_TIMER, lapic_read(), lapic_write(), and LVT_MASKED.

Here is the call graph for this function:



7.106.2.9 calibrate_apic_timer()

```
static uint32_t calibrate_apic_timer (
            uint64_t tsc_hz ) [static]
```

Calibrate the APIC timer against the TSC to determine its frequency.

The APIC timer runs from an internal bus-clock-derived source that has no fixed architectural relationship to the TSC. This function measures the APIC timer frequency empirically using a TSC-timed spin loop:

1. Configures the APIC timer DCR (Divide Configuration Register) to divisor 1 for maximum resolution.
2. Masks the LVT timer entry so no interrupt fires during calibration.
3. Loads the maximum initial count (0xFFFFFFFF) so the counter does not wrap during the measurement window.
4. Spins for a calibration window of `tsc_hz/100` TSC ticks (≈ 10 ms) using `arch_read_timestamp()`. Falls back to 10,000,000 cycles if `tsc_hz` is not yet known.
5. Reads `APIC_REG_TIMER_CCR` and subtracts from 0xFFFFFFFF to get the number of APIC ticks that elapsed in the window.
6. Scales by `tsc_hz` / calibration_tsc_ticks to obtain APIC Hz.

If `tsc_hz` is 0 (TSC not yet calibrated), the APIC count is multiplied by 100 as a rough fallback (assumes the 10M-cycle window was ≈ 10 ms on a 1 GHz core — good enough for initial bringup on QEMU TCG).

Parameters

tsc_hz	TSC frequency in Hz from `timer_init()`; 0 if not yet known.

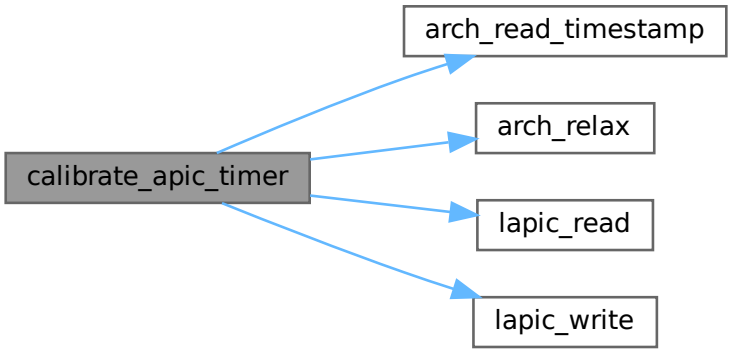
Returns

Measured APIC timer frequency in ticks per second (APIC Hz), or 0 if calibration could not produce a meaningful result.

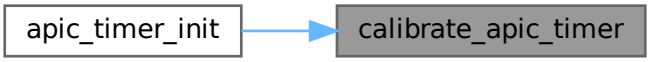
References APIC_REG_LVT_TIMER, APIC_REG_TIMER_CCR, APIC_REG_TIMER_DCR, APIC_REG_TIMER_ICR, arch_read_timestamp(), arch_relax(), lapic_read(), lapic_write(), LVT_MASKED, and TIMER_DIV_1.

Referenced by apic_timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.106.2.10 lapic_read()

```
static uint32_t lapic_read (
            uint32_t reg )  [static]
```

Read a 32-bit value from a Local APIC MMIO register.

Performs a `volatile` 32-bit MMIO load from the register at byte offset `reg` within the LAPIC page. The `volatile` qualifier on `lapic_base` prevents the compiler from caching the value across reads, which is essential for registers that change asynchronously (e.g., `APIC_REG_TIMER_CCR` which counts down independently of the CPU instruction stream).

Parameters

reg	Byte offset of the APIC register (e.g., `APIC_REG_TIMER_CCR`).

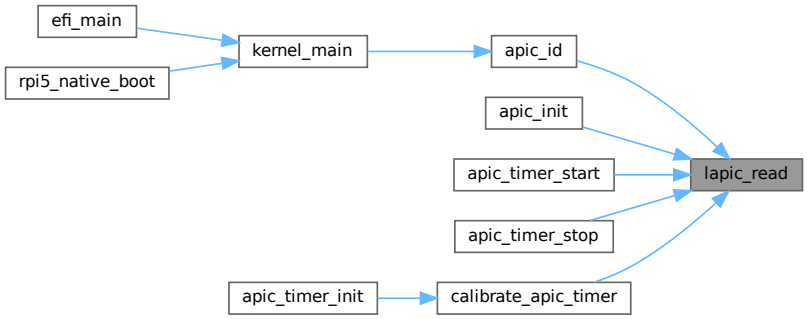
Returns

Current 32-bit value of the register.

References lapic_base.

Referenced by apic_id(), apic_init(), apic_timer_start(), apic_timer_stop(), and calibrate_apic_timer().

Here is the caller graph for this function:



7.106.2.11 lapic_write()

```
static void lapic_write (
            uint32_t reg,
            uint32_t val )  [static]
```

Write a 32-bit value to a Local APIC MMIO register.

All xAPIC registers are 32-bit wide and 16-byte aligned within the 4 KB MMIO page at `lapic_base`. The register offset `reg` is a byte offset (e.g., `APIC_REG_EOI` = 0x0B0); dividing by 4 converts it to a `uint32_t` array index. Writes must be 32-bit aligned and must not be merged or reordered by the compiler, hence the `volatile` qualifier on `lapic_base`.

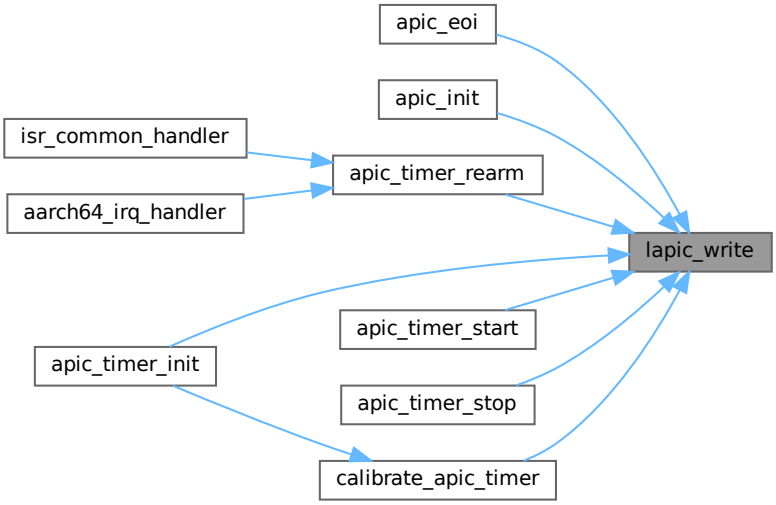
Parameters

reg	Byte offset of the APIC register (e.g., `APIC_REG_EOI`).
val	32-bit value to write.

References lapic_base.

Referenced by apic_eoi(), apic_init(), apic_timer_init(), apic_timer_rearm(), apic_timer_start(), apic_timer_stop(), and calibrate_apic_timer().

Here is the caller graph for this function:



7.106.2.12 rdmsr64()

```
static uint64_t rdmsr64 (
            uint32_t msr )  [static]
```

Read a 64-bit Model Specific Register via `RDMSR`.

Executes the `RDMSR` instruction with `msr` in `ECX`. The processor returns the high 32 bits in `EDX` and the low 32 bits in `EAX`; this function combines them into a single 64-bit value. Must run at CPL 0 (ring 0); a #GP fault results if executed in user mode or if `msr` denotes a reserved or non-existent register.

Parameters

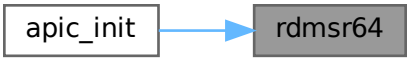
msr	32-bit MSR index (e.g., `IA32_APIC_BASE_MSR` = 0x1B).

Returns

Current 64-bit value of the specified MSR.

Referenced by apic_init().

Here is the caller graph for this function:



7.106.2.13 wrmsr64()

```
static void wrmsr64 (
            uint32_t msr,
            uint64_t val )  [static]
```

Write a 64-bit value to a Model Specific Register via `WRMSR`.

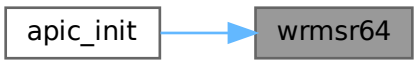
Executes the `WRMSR` instruction with `msr` in `ECX`, the high 32 bits of `val` in `EDX`, and the low 32 bits in `EAX`. Must run at CPL 0; a #GP results for reserved, non-existent, or read-only MSRs, or for writes that set reserved bits. Used here exclusively to manipulate `IA32_APIC_BASE_MSR` to switch from x2APIC to xAPIC mode and to assert the global hardware-enable bit.

Parameters

msr	32-bit MSR index.
val	64-bit value to write.

Referenced by apic_init().

Here is the caller graph for this function:



7.106.3 Variable Documentation

7.106.3.1 lapic_base

```
volatile uint32_t* lapic_base = (volatile uint32_t *)(uintptr_t) 0xFEE00000u  [static]
```

Referenced by apic_init(), lapic_read(), and lapic_write().

7.106.3.2 lapic_phys_base

```
uint64_t lapic_phys_base = 0xFEE00000u  [static]
```

Referenced by apic_init().

7.106.3.3 s_apic_hz

```
uint32_t s_apic_hz = 0  [static]
```

Referenced by apic_timer_init(), and apic_timer_rearm().

7.106.3.4 timer_initial_count

```
uint32_t timer_initial_count = 0  [static]
```

Referenced by apic_timer_init(), apic_timer_rearm(), and apic_timer_start().

7.106.3.5 timer_period_tsc_ticks

```
uint64_t timer_period_tsc_ticks = 0  [static]
```

Referenced by apic_timer_init(), and apic_timer_period_tsc().

7.106.3.6 timer_tick_hz

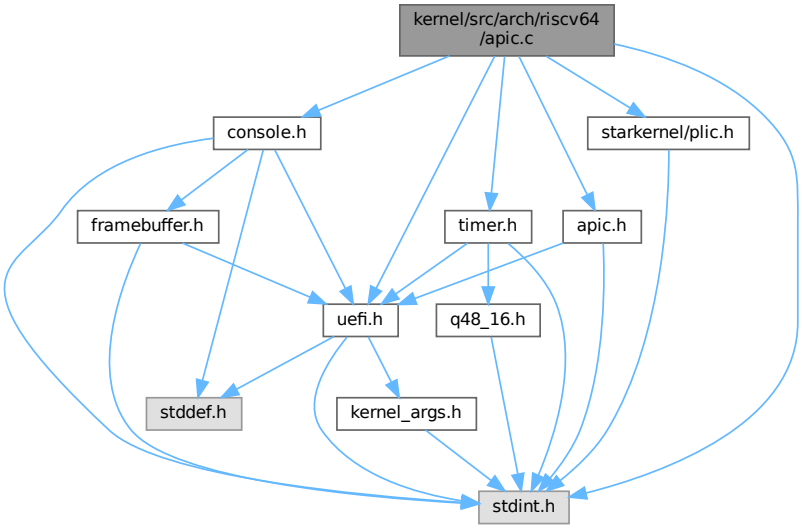
```
uint32_t timer_tick_hz = 0  [static]
```

Referenced by apic_timer_init().

7.107 kernel/src/arch/riscv64/apic.c File Reference

```
#include "apic.h"
#include "uefi.h"
#include "console.h"
#include "timer.h"
#include "starkernel/plic.h"
#include <stdint.h>
```

Include dependency graph for apic.c:



Data Structures

- struct sbiret_t

Macros

- #define SBI_EXT_BASE 0x10UL
- #define SBI_BASE_FID_PROBE_EXT 3UL
- #define SBI_EXT_TIME 0x54494D45UL /* "TIME" */
- #define SBI_TIME_FID_SET_TIMER 0UL
- #define SBI_SUCCESS 0L
- #define SIE_STIE (1UL << 5)

Functions

- static sbiret_t sbi_call1 (unsigned long eid, unsigned long fid, unsigned long arg0)

  Issue an SBI ECALL with one argument.
- static uint64_t rdtime (void)

  Read the RISC-V `time` CSR.
- static int sbi_set_timer (uint64_t deadline)

Arm the SBI timer for `deadline`.

- void riscv64_timer_rearm (void)

  Re-arm the one-shot SBI timer and account the tick.

- int apic_init (BootInfo *boot_info)

  Initialise the GICv2 distributor and CPU interface (M4 milestone).

- void apic_eoi (void)

  Signal End of Interrupt to the interrupt controller (RISC-V stub).

- int apic_timer_init (uint64_t tsc_hz, uint32_t tick_hz)

  Compute the expected system-counter ticks per heartbeat period.

- void apic_timer_start (void)

  Start timer delivery via the SBI TIME extension.

- void apic_timer_stop (void)

  Stop timer delivery by masking `sie.STIE`.

- uint64_t apic_timer_period_tsc (void)

  Return the expected `time`-counter ticks per heartbeat period.

Variables

- static uint64_t s_timer_period_tsc = 0

  apic.c (riscv64) - APIC stub.

- static uint64_t s_time_hz = 0
- static int s_sbi_time_ok = 0
- static uint64_t s_next_deadline = 0

7.107.1 Macro Definition Documentation

7.107.1.1 SBI_BASE_FID_PROBE_EXT

```
#define SBI_BASE_FID_PROBE_EXT 3UL
```

7.107.1.2 SBI_EXT_BASE

```
#define SBI_EXT_BASE 0x10UL
```

7.107.1.3 SBI_EXT_TIME

```
#define SBI_EXT_TIME 0x54494D45UL /* "TIME" */
```

7.107.1.4 SBI_SUCCESS

```
#define SBI_SUCCESS 0L
```

7.107.1.5 SBI_TIME_FID_SET_TIMER

```
#define SBI_TIME_FID_SET_TIMER 0UL
```

7.107.1.6 SIE_STIE

```
#define SIE_STIE (1UL << 5)
```

7.107.2 Function Documentation

7.107.2.1 apic_eoi()

```
void apic_eoi (
            void  )
```

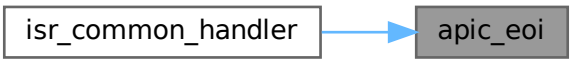
Signal End of Interrupt to the interrupt controller (RISC-V stub).

Signal End of Interrupt (common cross-ISA signature — see below).

On x86-64 this writes to the Local APIC EOI register. On RISC-V EOI is handled by reading the PLIC claim/complete register; that path is not yet wired. This stub is a no-op that satisfies the common interface.

Referenced by isr_common_handler().

Here is the caller graph for this function:



7.107.2.2 apic_init()

```
int apic_init (
            BootInfo * boot_info )
```

Initialise the GICv2 distributor and CPU interface (M4 milestone).

Enables exactly one interrupt source: the physical timer PPI for the exception level detected in item 0.4 (30 at EL1, 26 at EL2 — see `TIMER_PPI_EL1` / `TIMER_PPI_EL2` above). Sequence:

0. Try `gic_bases_from_dtb()` against `boot_info->dtb`; on success, `s_gicd_base` / `s_gicc_base` are overridden from the real hardware addresses. On failure (no DTB, e.g. this system's QEMU/UEFI firmware — see file header) they keep the QEMU constant defaults, exactly the behaviour this function had before this step existed.

1. Set this interrupt's priority (`GICD_IPRIORITYR`) below the CPU interface's priority mask, so it is never itself masked out.
2. Enable it in `GICD_ISENABLER0` (PPIs 16–31 live in the first word).
3. Enable the distributor, Group 0 only (`GICD_CTLR`) — this deployment has no GICv2 security-extension split in use.
4. Set the CPU interface priority mask to allow everything through (`GICC_PMR` = 0xFF) and enable the CPU interface (`GICC_CTLR`).

Deliberately not configured: `GICD_ICFGR` (trigger sensitivity). The ARM generic timer's PPI is architecturally level-sensitive; this is not a general GIC driver and item 0.6's own scope does not call for reconfiguring it.

Parameters

boot_info	Kernel `BootInfo`; `dtb` is consulted for real-hardware GIC base discovery as of the 2026-09-04 fix (previously unused — see file header), and, as of the same fix, for `aarch64_psci_conduit_init()`'s PSCI conduit discovery (arch.c) — bundled into this call for the same reason GIC discovery is: this is the one point in boot with `boot_info->dtb` already in hand.

Returns

0 always.

Initialise the GICv2 distributor and CPU interface (M4 milestone).

Called during M4 (interrupt controller init) before the IDT is loaded. Performs the following steps:

1. Reads `IA32_APIC_BASE_MSR` (0x1B) to determine the current APIC mode.
2. If x2APIC mode (bit 10) is active, clears it to switch to xAPIC MMIO mode while keeping the global-enable bit (bit 11) set, as required by Intel SDM Vol.3 ?10.12.5 (never clear both bits simultaneously).
3. If the global-enable bit (bit 11) was clear, sets it.
4. Software-enables the APIC via `APIC_REG_SIVR` (Spurious Interrupt Vector Register) and assigns spurious vector 0xFF.
5. Clears the Task Priority Register (TPR=0) so all interrupt priorities are delivered; a non-zero TPR left by UEFI would silently suppress all APIC interrupts at or below that priority level.
6. Issues a spurious EOI write to clear any stale in-service interrupt that UEFI may have left pending (specifically the APIC timer vector that fires just before `ExitBootServices`).

The `boot_info` parameter is reserved for future use (e.g., reading the ACPI MADT for the physical LAPIC base address); it is currently unused and the default identity-mapped address `LAPIC_DEFAULT_PHYS` = 0xFEE00000 is always used.

Parameters

boot_info	Pointer to kernel `BootInfo` (ACPI/memory-map data); currently unused, reserved for MADT-based LAPIC relocation.

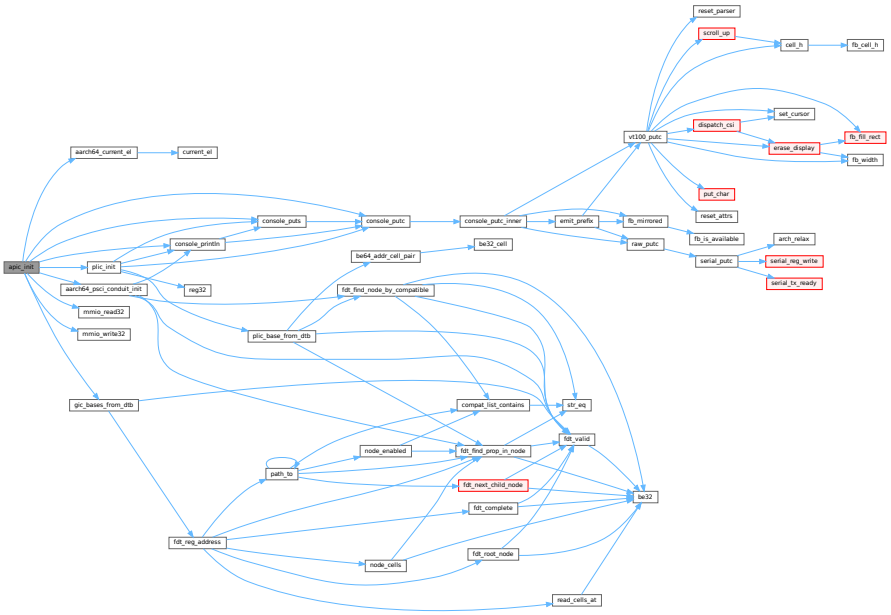
Returns

0 on success (always; the APIC is assumed to be present on x86-64).

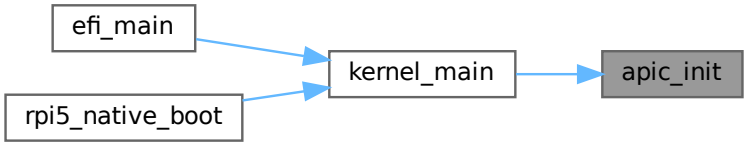
References aarch64_current_el(), aarch64_psci_conduit_init(), console_println(), console_putc(), console_puts(), BootInfo::dtb, gic_bases_from_dtb(), GICC_CTLR, GICC_CTLR_ENABLE_GRP0, GICC_PMR, GICD_CTLR, GICD_CTLR_ENABLE_GRP0, GICD_IPRIORITYR, GICD_ISENABLER0, mmio_read32(), mmio_write32(), plic_init(), PMR_ALLOW_ALL, s_gicc_base, s_gicd_base, s_timer_ppi, TIMER_PPI_EL1, TIMER_PPI_EL2, and TIMER_PRIORITY.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.107.2.3 apic_timer_init()

```
int apic_timer_init (
            uint64_t tsc_hz,
            uint32_t tick_hz )
```

Compute the expected system-counter ticks per heartbeat period.

Computes `s_timer_period_tsc` (the expected number of `CNTPCT_EL0` ticks between heartbeats), used both by `heartbeat_init()` to seed the rolling-window `expected_delta` and by `apic_timer_start()` / `apic_timer_rearm()` (item 0.7) as the `TVAL` step written on every arm and re-arm.

If either `tsc_hz` or `tick_hz` is zero, a fallback of 10,000,000 ticks is stored (approximately 160 ms at 62.5 MHz, a safe non-zero sentinel that prevents division-by-zero in the heartbeat path).

Parameters

tsc_hz	System counter frequency from `timer_init()` (Hz).
tick_hz	Desired heartbeat rate (Hz); 0 → uses fallback.

Returns

0 always.

Compute the expected system-counter ticks per heartbeat period.

Calibrates the APIC timer frequency via `calibrate_apic_timer()` then programs the LVT Timer Register and Initial Count Register to deliver `APIC_TIMER_VECTOR` periodically at `tick_hz` interrupts per second. The timer is left **masked** after initialisation; call `apic_timer_start()` after the IDT is loaded and interrupts are enabled to begin delivery.

Derived quantities stored for later use:

- `timer_initial_count` — APIC ticks per heartbeat period (`apic_hz` / `tick_hz`), written to `APIC_REG_TIMER_ICR`.
- `timer_tick_hz` — requested tick rate (100 Hz default).
- `timer_period_tsc_ticks` — expected TSC ticks per heartbeat (`tsc_hz` / `tick_hz`), used by `heartbeat_tick()` to compute inter-tick variance for the TIME-TRUST Q48.16 metric.

If `tick_hz` is 0 it defaults to 100 Hz (10 ms period). If calibration returns 0 (no APIC timer detected or counter stuck) the function prints a diagnostic and returns -1 without programming the timer.

Parameters

tsc_hz	TSC frequency in Hz (from `timer_init()`); used to derive the 10 ms calibration window and `timer_period_tsc_ticks`. Pass 0 if TSC is not yet calibrated.
tick_hz	Desired interrupt rate in Hz; 0 → defaults to 100 Hz.

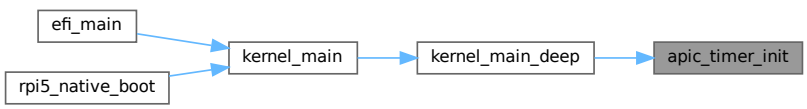
Returns

0 on success, -1 if APIC timer calibration failed.

References s_counter_hz_apic, s_time_hz, and s_timer_period_tsc.

Referenced by kernel_main_deep().

Here is the caller graph for this function:



7.107.2.4 apic_timer_period_tsc()

```
uint64_t apic_timer_period_tsc (
            void  )
```

Return the expected `time`-counter ticks per heartbeat period.

Get the configured timer period in TSC ticks.

Returns `s_timer_period_tsc`, set by `apic_timer_init()` as `tsc_hz` / `tick_hz`. Consumed by `heartbeat_init()` to seed the rolling-window `expected_delta`, and by `apic_timer_start()` / `riscv64_timer_rearm()` as the SBI deadline step.

Returns

Expected `time`-counter ticks per heartbeat period; 10,000,000 if frequencies were not available at init time.

References s_timer_period_tsc.

7.107.2.5 apic_timer_start()

```
void apic_timer_start (
            void  )
```

Start timer delivery via the SBI TIME extension.

Start the APIC timer (enables periodic interrupts).

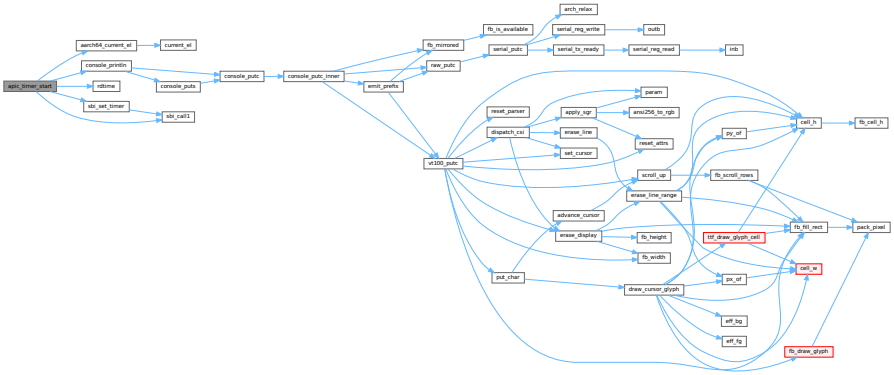
Probes for the TIME extension first. If the SEE does not provide it the timer is **not** armed and the condition is reported loudly rather than papered over with the legacy EID 0x00 call: a heartbeat that silently never ticks is far worse to diagnose than one that says why at boot.

On success: computes the first absolute deadline, arms it, and sets `sie.STIE`. Global delivery is gated separately by `sstatus.SIE`, which `arch_enable_interrupts()` sets.

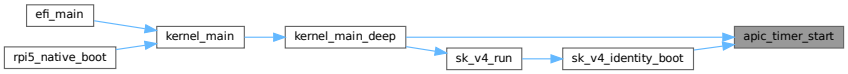
References aarch64_current_el(), console_println(), sbiret_t::error, rdtime(), s_next_deadline, s_sbi_time_ok, s_timer_period_tsc, SBI_BASE_FID_PROBE_EXT, sbi_call1(), SBI_EXT_BASE, SBI_EXT_TIME, sbi_set_timer(), SBI_SUCCESS, SIE_STIE, and sbiret_t::value.

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.107.2.6 apic_timer_stop()

```
void apic_timer_stop (
            void  )
```

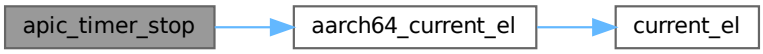
Stop timer delivery by masking `sie.STIE`.

Stop the APIC timer (disables periodic interrupts).

The SBI timer cannot be cancelled outright — masking the enable bit is the supported way to stop delivery. Any deadline already programmed simply goes unserviced.

References aarch64_current_el(), s_sbi_time_ok, and SIE_STIE.

Here is the call graph for this function:



7.107.2.7 rdtime()

```
static uint64_t rdtime (
            void  )  [inline], [static]
```

Read the RISC-V `time` CSR.

Deadlines handed to `sbi_set_timer()` are absolute values on this counter. `arch/riscv64/timer.c` keeps its own copy of this accessor; duplicating four instructions is preferable to widening `timer.h` with an architecture-specific accessor that only these two files can use.

Referenced by apic_timer_start(), and riscv64_timer_rearm().

Here is the caller graph for this function:

7.107.2.8 riscv64_timer_rearm()

```
void riscv64_timer_rearm (
            void  )
```

Re-arm the one-shot SBI timer and account the tick.

The SBI timer is one-shot by nature. Servicing a timer interrupt without programming the next deadline leaves the heartbeat stopped permanently, with no error anywhere — the single most likely silent failure of this driver. Every path out of a timer interrupt must reach this function.

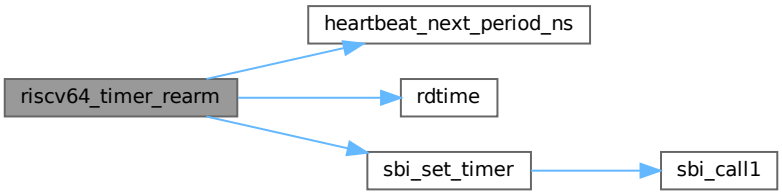
The step is `heartbeat_next_period_ns()` (item 0.8, ?26) converted to `time`-counter ticks via `s_time_hz`, not the fixed `s_timer_period_tsc` used to seed the very first deadline in `apic_timer_start()` – the latter remains `apic_timer_period_tsc()'s` return value for `heartbeat_init()'s` initial `expected_delta`, unchanged.

Called from `riscv64_interrupt_handler()` in `interrupts.c` on `scause` cause 5.

References heartbeat_next_period_ns(), rdtime(), s_next_deadline, s_sbi_time_ok, s_time_hz, s_timer_period_tsc, and sbi_set_timer().

Referenced by riscv64_interrupt_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

7.107.2.9 sbi_call1()

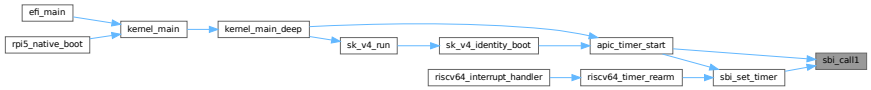
```
static sbiret_t sbi_call1 (
            unsigned long eid,
            unsigned long fid,
            unsigned long arg0 )  [static]
```

Issue an SBI ECALL with one argument.

References sbiret_t::error, and sbiret_t::value.

Referenced by apic_timer_start(), and sbi_set_timer().

Here is the caller graph for this function:



7.107.2.10 sbi_set_timer()

```
static int sbi_set_timer (
            uint64_t deadline )  [static]
```

Arm the SBI timer for `deadline`.

Returns

1 on success, 0 if the SEE rejected the call.

References sbiret_t::error, sbi_call1(), SBI_EXT_TIME, SBI_SUCCESS, and SBI_TIME_FID_SET_TIMER.

Referenced by apic_timer_start(), and riscv64_timer_rearm().

Here is the call graph for this function:

Here is the caller graph for this function:

7.107.3 Variable Documentation

7.107.3.1 s_next_deadline

```
uint64_t s_next_deadline = 0  [static]
```

Referenced by apic_timer_start(), and riscv64_timer_rearm().

7.107.3.2 s_sbi_time_ok

```
int s_sbi_time_ok = 0  [static]
```

Referenced by apic_timer_start(), apic_timer_stop(), and riscv64_timer_rearm().

7.107.3.3 s_time_hz

```
uint64_t s_time_hz = 0  [static]
```

Referenced by apic_timer_init(), and riscv64_timer_rearm().

7.107.3.4 s_timer_period_tsc

```
uint64_t s_timer_period_tsc = 0  [static]
```

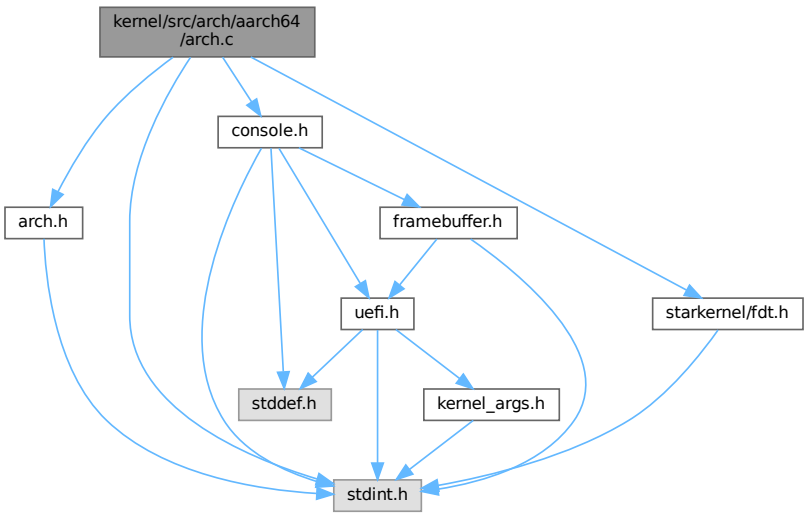
apic.c (riscv64) - APIC stub.

No Local APIC on RISC-V; interrupt controller is PLIC (Platform-Level Interrupt Controller). apic_init() calls into plic.c for the real PLIC bring-up (item 4.3.5b); apic_eoi() stays a no-op since PLIC completion happens per-source in the trap handler (claim/complete), not via a single shared EOI register.

7.108 kernel/src/arch/aarch64/arch.c File Reference

```
#include "arch.h"
#include "console.h"
#include "starkernel/fdt.h"
#include <stdint.h>
```

Include dependency graph for arch.c:



Functions

- void aarch64_install_vectors (void)

  arch.c (aarch64) - Architecture abstractions for StarKernel on AArch64
- int aarch64_current_el (void)

  Return the AArch64 exception level the kernel is running at.
- void arch_early_init (void)

  Perform the earliest AArch64 architecture initialisation (no-op).
- void arch_enable_interrupts (void)

  Enable IRQ delivery on the current core (AArch64 DAIFCLR).
- void arch_disable_interrupts (void)

  Disable IRQ delivery on the current core (AArch64 DAIFSET).
- void arch_halt (void)

  Halt the core until the next event (AArch64 WFI).
- void aarch64_psci_conduit_init (const void *dtb)

  Discover the real PSCI conduit ("smc" vs "hvc") from the DTB.
- void arch_cold_reset (void)
- uint64_t arch_read_timestamp (void)

  Read the AArch64 system counter (CNTPCT_EL0).
- void arch_mmu_init (void)

  Initialise the AArch64 MMU (stub — deferred to a later milestone).

Variables

- static int s_current_el = -1
- static int s_psci_use_smc = 0

7.108.1 Function Documentation

7.108.1.1 aarch64_current_el()

```
int aarch64_current_el (
            void  )
```

Return the AArch64 exception level the kernel is running at.

apic.c (aarch64) - Minimal GICv2 driver + ARM Generic Timer (items 0.6/0.7/ 4.3.5d).

Reads `CurrentEL` (bits [3:2] hold the EL number, [1:0] are RES0) on first call and caches the result; every later call returns the cached value. Safe to call at any point in boot, including before `console_init()` — the read itself has no dependency, only the printing of it does (see `arch_interrupts_init()` in `interrupts.c`, the earliest point at which both this and a working console are available).

EDK2 on QEMU's aarch64 `virt` machine has been observed to leave the kernel at either EL1 or EL2 depending on firmware build; nothing in this tree may assume one over the other (FABRIC-0.md ?25.7.1 GAP-B3). Every EL-dependent choice — `VBAR_EL1` vs `VBAR_EL2`, the `ELR_ELx` / `SPSR_ELx` saved-state pair, and `CNTP_*_EL0` vs `CNTHP_*_EL2` — must read this accessor rather than hardcode a level.

Returns

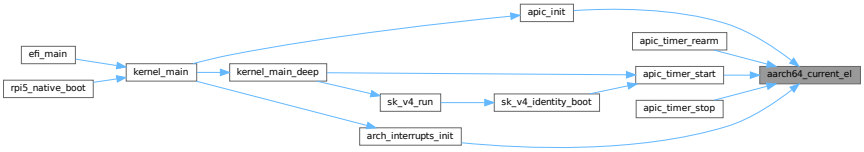
1 or 2. Never 0 or 3: a UEFI application that survived far enough to reach this call cannot be running at EL0 (unprivileged) or EL3 (secure monitor — QEMU's virt firmware runs the kernel at EL1/EL2).

References current_el(), and s_current_el.

Referenced by apic_init(), apic_timer_rearm(), apic_timer_start(), apic_timer_stop(), and arch_interrupts_init().

Here is the call graph for this function:

Here is the caller graph for this function:



7.108.1.2 aarch64_install_vectors()

```
void aarch64_install_vectors (
            void  )  [extern]
```

arch.c (aarch64) - Architecture abstractions for StarKernel on AArch64

7.108.1.3 aarch64_psci_conduit_init()

```
void aarch64_psci_conduit_init (
            const void * dtb )
```

Discover the real PSCI conduit ("smc" vs "hvc") from the DTB.

Looks up the devicetree's PSCI firmware node and reads its `method` property. BCM2712's own `/psci` node declares `compatible = "arm,psci-1.0", "arm,psci-0.2", "arm,psci"` and `method = "smc"` – confirmed directly against `bcm2712.dtsi`, not assumed. `fdt_find_node_by_compatible()` matches any entry in a node's compatible list, so searching for just the first (most specific) string is sufficient; the other two exist for older PSCI-spec-version consumers, not because this node might be tagged differently.

Leaves `s_psci_use_smc` at its default (0, HVC) if `dtb` is not a valid FDT, no PSCI node is found, or `method` is anything other than exactly `"smc"` – including the explicit `"hvc"` case, and the "property absent" case, both of which should keep the existing HVC behaviour rather than guess.

Called once from `apic_init()` (`arch/aarch64/apic.c`), the one point in boot that already has `boot_info->dtb` in hand at the right time for this kind of discovery (same rationale as that file's own GIC-base lookup) – collocated here in `arch.c`, not there, because `arch_cold_reset()` is this function's actual consumer and has no `boot_info` of its own to probe with by the time it runs (called from deep in VM execution, via `BYE`). Same extern-in-place convention this file and apic.c already use for `aarch64_current_el()`.

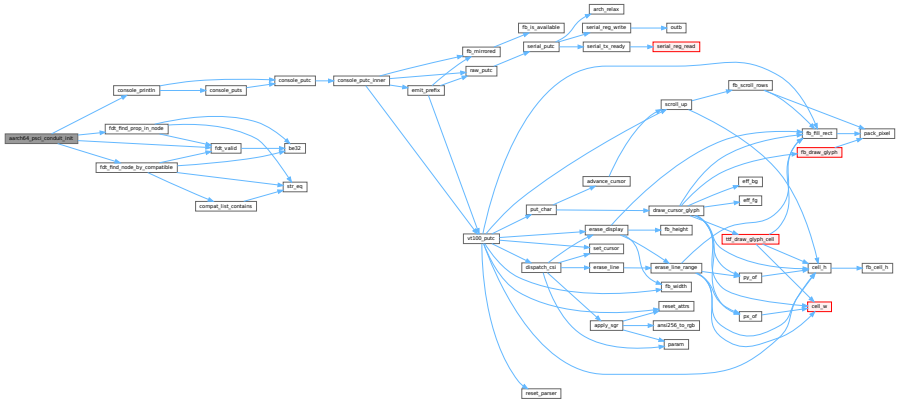
Parameters

dtb	Candidate devicetree blob (`BootInfo->dtb`); NULL-safe.

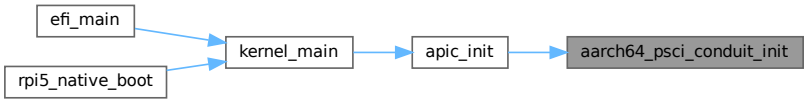
References console_println(), fdt_find_node_by_compatible(), fdt_find_prop_in_node(), fdt_valid(), and s_psci_use_smc.

Referenced by apic_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.108.1.4 arch_cold_reset()

```
void arch_cold_reset (
            void  )
```

7.108.1.5 arch_disable_interrupts()

```
void arch_disable_interrupts (
            void  )
```

Disable IRQ delivery on the current core (AArch64 DAIFSET).

Issues `MSR` DAIFSET, #2 which sets the `I` bit (IRQ mask) in the `DAIF` flags, masking IRQ exceptions. FIQ, SError, and Debug exceptions are unaffected. Used around critical sections in the single-threaded kernel where interrupt delivery would corrupt shared state.

The memory clobber acts as a compiler-level barrier; for a full hardware memory barrier use DSB before calling this function.

Referenced by arch_cold_reset(), and arch_cold_reset().

Here is the caller graph for this function:

7.108.1.6 arch_early_init()

```
void arch_early_init (
            void  )
```

Perform the earliest AArch64 architecture initialisation (no-op).

arch.h - Architecture abstraction layer for StarKernel

On x86-64 this function loads the GDT; AArch64 has no equivalent because privilege separation is provided by the exception-level architecture (EL0/EL1/EL2/EL3) rather than segment descriptors. The exception vector table is installed later in arch_interrupts_init() by calling aarch64_install_vectors() which writes VBAR_EL1.

This stub exists to satisfy the common arch_early_init() call site in kernel_main() so that the same milestone sequence compiles across all three supported ISAs (amd64 / aarch64 / riscv64).

7.108.1.7 arch_enable_interrupts()

```
void arch_enable_interrupts (
            void  )
```

Enable IRQ delivery on the current core (AArch64 DAIFCLR).

Issues MSR DAIFCLR, #2 which clears the I bit (IRQ mask) in the DAIF (Debug / Abort / IRQ / FIQ) flags of the current exception level. After this instruction the core will take IRQ exceptions from the GIC. The memory clobber prevents the compiler from reordering stores across the barrier.

Must be called after arch_interrupts_init() has installed the vector table in VBAR_EL1 and the GIC timer driver has been configured.

7.108.1.8 arch_halt()

```
void arch_halt (
            void  )
```

Halt the core until the next event (AArch64 WFI).

Issues the WFI (Wait For Interrupt) instruction. The core enters a low-power standby state and resumes execution when an IRQ, FIQ, SError, debug event, or reset is detected. In the kernel panic path this is called inside an infinite loop with interrupts disabled (DAIFSET) to stop the processor permanently. In the idle path it reduces power consumption between heartbeat ticks.

The memory clobber prevents the compiler from sinking stores across the WFI.

7.108.1.9 arch_mmu_init()

```
void arch_mmu_init (
            void  )
```

Initialise the AArch64 MMU (stub — deferred to a later milestone).

Full page-table setup (TTBR0/TTBR1, MAIR, TCR, SCTLR.M) is deferred to a later milestone. This stub exists to satisfy the common `arch_mmu_init()` call site shared across all three ISAs. The VMM subsystem (`vmm.c`) handles page-table management independently of this hook.

7.108.1.10 arch_read_timestamp()

```
uint64_t arch_read_timestamp (
            void  )
```

Read the AArch64 system counter (CNTPCT_EL0).

Issues `MRS %0, cntpct_el0` to read the 64-bit physical system counter. On AArch64 the system counter is the architectural equivalent of the x86-64 TSC: a monotonically incrementing clock whose frequency is reported by `CNTFRQ_EL0` and set by firmware at boot.

Unlike `RDTSC` on x86-64, `CNTPCT_EL0` is fully synchronised across cores (no per-core skew), invariant across power-state changes, and does not require serialisation for most measurement use cases. It is used as the time base for `timer_now_ns()` and `heartbeat_tick()`.

Returns

Current 64-bit system counter value in counter ticks (frequency given by `CNTFRQ_EL0`, typically 10–62.5 MHz).

7.108.2 Variable Documentation

7.108.2.1 s_current_el

```
int s_current_el = -1  [static]
```

Referenced by aarch64_current_el().

7.108.2.2 s_psci_use_smc

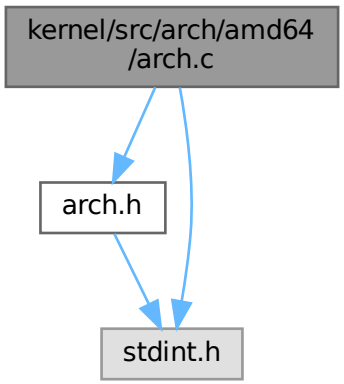
```
int s_psci_use_smc = 0  [static]
```

Referenced by aarch64_psci_conduit_init(), and arch_cold_reset().

7.109 kernel/src/arch/amd64/arch.c File Reference

```
#include "arch.h"
#include <stdint.h>
```

Include dependency graph for arch.c:



Data Structures

- struct gdt_entry

  arch.c (amd64) - Architecture abstractions for StarKernel on x86_64
- struct gdtr

Functions

- struct gdt_entry __attribute__ ((packed))
- static struct gdt_entry gdt[3] __attribute__ ((aligned(16)))
- static void gdt_set_entry (int index, uint8_t access, uint8_t granularity)

  Populate a GDT entry with limit 0xFFFF and a zero base.
- static void gdt_init (void)

  Initialise and load the minimal 3-entry GDT required for long mode.
- void arch_early_init (void)

  Perform the earliest x86-64 architecture initialisation.
- void arch_enable_interrupts (void)

  Enable hardware interrupts on the current CPU core (x86-64 STI).
- void arch_disable_interrupts (void)

  Disable hardware interrupts on the current CPU core (x86-64 CLI).
- void arch_halt (void)

  Halt the CPU until the next interrupt (x86-64 HLT).
- void arch_cold_reset (void)
- uint64_t arch_read_timestamp (void)

  Read the x86-64 Time Stamp Counter (RDTSC).
- void arch_mmu_init (void)

  Initialise the x86-64 MMU (stub — deferred to a later milestone).

Variables

- uint16_t limit_low
- uint16_t base_low
- uint8_t base_mid
- uint8_t access
- uint8_t granularity
- uint8_t base_high
- uint16_t limit
- uint64_t base

7.109.1 Function Documentation

7.109.1.1 __attribute__() [1/2]

```
static struct gdt_entry gdt[3] __attribute__ (
             (aligned(16))  )  [static]
```

7.109.1.2 __attribute__() [2/2]

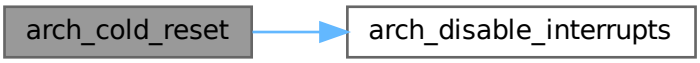
```
struct gdt_entry __attribute__ (
             (packed)  )
```

7.109.1.3 arch_cold_reset()

```
void arch_cold_reset (
             void  )
```

References arch_disable_interrupts().

Here is the call graph for this function:



7.109.1.4 arch_disable_interrupts()

```
void arch_disable_interrupts (
             void  )
```

Disable hardware interrupts on the current CPU core (x86-64 CLI).

Disable IRQ delivery on the current core (AArch64 DAIFSET).

Issues the `CLI` instruction to clear the Interrupt Flag. Used around critical sections in the single-threaded kernel where interrupt delivery would corrupt shared state.

7.109.1.5 arch_early_init()

```
void arch_early_init (
            void  )
```

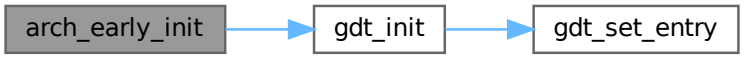
Perform the earliest x86-64 architecture initialisation.

arch.h - Architecture abstraction layer for StarKernel

Called immediately after the UEFI boot stub transfers control to the kernel, before the console or interrupt controller are available. Loads the minimal three-entry GDT via `gdt_init()` so that subsequent code has valid segment selectors. IDT setup is deferred until after the console is live (called separately via `arch_interrupts_init()`).

References gdt_init().

Here is the call graph for this function:



7.109.1.6 arch_enable_interrupts()

```
void arch_enable_interrupts (
            void  )
```

Enable hardware interrupts on the current CPU core (x86-64 STI).

Enable IRQ delivery on the current core (AArch64 DAIFCLR).

Issues the `STI` instruction to set the Interrupt Flag. Must be called after the IDT is loaded and the APIC is configured, otherwise the first interrupt will triple-fault.

7.109.1.7 arch_halt()

```
void arch_halt (
            void  )
```

Halt the CPU until the next interrupt (x86-64 HLT).

Halt the core until the next event (AArch64 WFI).

Issues the `HLT` instruction. In the kernel panic path this is called in an infinite loop (interrupts disabled) to stop the processor permanently. In normal use it is called once per idle iteration to reduce power consumption.

7.109.1.8 arch_mmu_init()

```
void arch_mmu_init (
            void  )
```

Initialise the x86-64 MMU (stub — deferred to a later milestone).

Initialise the AArch64 MMU (stub — deferred to a later milestone).

Full four-level page table setup is deferred; the VMM subsystem (vmm.c) handles page table management independently. This function exists as an architectural hook for future per-CPU MMU configuration that must run after GDT/IDT setup but before the first memory access through a non-identity-mapped virtual address.

7.109.1.9 arch_read_timestamp()

```
uint64_t arch_read_timestamp (
            void  )
```

Read the x86-64 Time Stamp Counter (RDTSC).

Read the AArch64 system counter (CNTPCT_EL0).

Executes RDTSC and combines EDX:EAX into a 64-bit value. Does not serialise instruction execution; use RDTSCP or a surrounding CPUID where ordering guarantees are required.

Returns

Current TSC value in CPU cycles.

7.109.1.10 gdt_init()

```
static void gdt_init (
            void  )  [static]
```

Initialise and load the minimal 3-entry GDT required for long mode.

Builds three GDT entries in the gdt[] array and loads the descriptor via LGDT followed by a far-return sequence to reload CS to selector 0x08 and all other segment registers to 0x10:

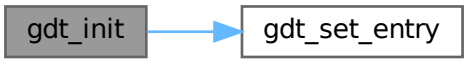
- Slot 0 (0x00): Null descriptor — all fields zero.
- Slot 1 (0x08): 64-bit kernel code (access=0x9A, granularity=0x20).
- Slot 2 (0x10): Kernel data (access=0x92, granularity=0x00).

Must be called before `arch_interrupts_init()` because the IDT uses selector 0x08 for all gate descriptors.

References gdtr::base, gdt_set_entry(), and gdtr::limit.

Referenced by arch_early_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.109.1.11 gdt_set_entry()

```
static void gdt_set_entry (
            int index,
            uint8_t access,
            uint8_t granularity )  [static]
```

Populate a GDT entry with limit 0xFFFF and a zero base.

In 64-bit long mode the base and limit fields of code/data segment descriptors are ignored by the processor; only the `access` and `granularity` flag bytes are architecturally significant. This helper writes the canonical long-mode descriptor layout: limit = 0xFFFF, all base bytes = 0, then stores the provided access and granularity values.

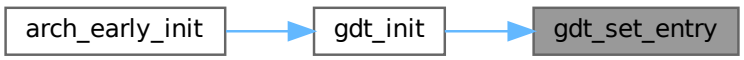
Parameters

index	Index into the `gdt[]` array (0 = null, 1 = code, 2 = data).
access	Access byte (e.g., 0x9A for kernel code, 0x92 for data).
granularity	Granularity/flags byte (e.g., 0x20 sets the L-bit for 64-bit code).

References access, and granularity.

Referenced by gdt_init().

Here is the caller graph for this function:



7.109.2 Variable Documentation

7.109.2.1 access

```
uint8_t access
```

Referenced by gdt_set_entry().

7.109.2.2 base

```
uint64_t base
```

Referenced by analytics_heap_allocate(), blk_vm_assign(), blk_vm_evict(), blk_vm_flush_all(), blk_vm_load(), blkio_ram_init_state(), block_word_update(), dev_rw4k(), div_ud_by_base(), erf_q48(), fdt_find_node_by_compatible(), fdt_find_node_by_device_type(), fdt_find_node_by_name(), fdt_find_prop(), format_word_hash(), itoa_simple(), mmio_read32(), mmio_write32(), print_number_formatted(), read_devblock(), read_devblock(), read_devblock(), read_devblock_4k(), rpi5_mailbox_get_framebuffer(), vm_addr_from_ptr(), vm_create_word(), vm_debug_dump_state(), vm_dictionary_get_data_field(), vm_parse_number(), vm_parse_number(), vocab_sync_vm_vars(), write_devblock(), and write_devblock_4k().

7.109.2.3 base_high

```
uint8_t base_high
```

7.109.2.4 base_low

```
uint16_t base_low
```

7.109.2.5 base_mid

```
uint8_t base_mid
```

7.109.2.6 granularity

```
uint8_t granularity
```

Referenced by gdt_set_entry().

7.109.2.7 limit

```
uint16_t limit
```

Referenced by blk_allocate(), control_forth_runtime_do(), control_forth_runtime_qdo(), der_next(), der_put_ed25519_algid(), der_put_tlv(), der_put_uint_integer(), profiler_generate_report(), profiler_print_hotspots(), rpi5_mailbox_get_framebuffer(), string_word_search(), walk_virtio_caps(), walk_virtio_caps(), walk_virtio_caps(), xxhash64(), and xxhash64_update().

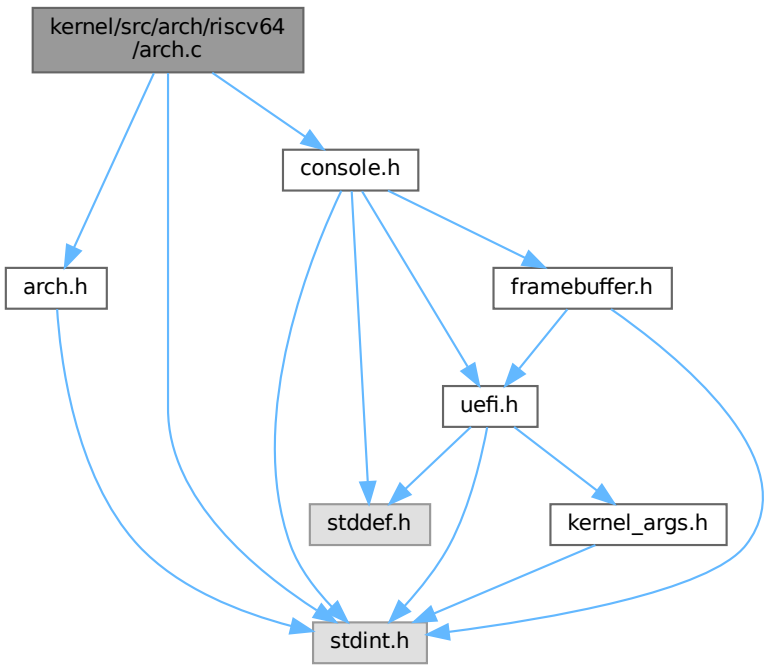
7.109.2.8 limit_low

```
uint16_t limit_low
```

7.110 kernel/src/arch/riscv64/arch.c File Reference

```
#include "arch.h"
#include "console.h"
#include <stdint.h>
```

Include dependency graph for arch.c:



Functions

- void riscv64_install_vectors (void)

  arch.c (riscv64) - Architecture abstractions for StarKernel on RISC-V 64-bit
- static void print_hex64 (uint64_t val)
- void arch_early_init (void)

  Perform the earliest RISC-V architecture initialisation (no-op).
- void arch_enable_interrupts (void)

  Enable supervisor-mode interrupts (RISC-V CSRSI sstatus).
- void arch_disable_interrupts (void)

  Disable supervisor-mode interrupts (RISC-V CSRCI sstatus).
- void arch_halt (void)

  Halt the core until the next interrupt (RISC-V WFI).
- void arch_cold_reset (void)
- uint64_t arch_read_timestamp (void)

  Read the RISC-V CPU cycle counter (`rdcycle`).
- void arch_mmu_init (void)

  Initialise the RISC-V MMU (stub — deferred to a later milestone).

Variables

- char __kernel_start [ ]

7.110.1 Function Documentation

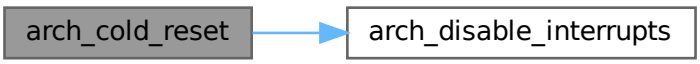
7.110.1.1 arch_cold_reset()

```
void arch_cold_reset (
            void  )
```

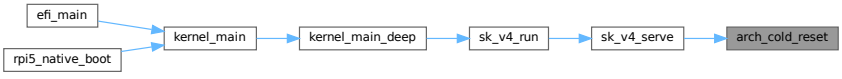
References arch_disable_interrupts(), and s_psci_use_smc.

Referenced by sk_v4_serve().

Here is the call graph for this function:



Here is the caller graph for this function:



7.110.1.2 arch_disable_interrupts()

```
void arch_disable_interrupts (
            void  )
```

Disable supervisor-mode interrupts (RISC-V CSRCI sstatus).

Disable IRQ delivery on the current core (AArch64 DAIFSET).

Issues `CSRCI sstatus, 0x2` which clears bit 1 (`SIE`) in `sstatus`, masking all supervisor-mode interrupts. Machine-mode interrupts are unaffected. Used around critical sections in the single-threaded kernel. The `memory` clobber acts as a compiler barrier; pair with a `FENCE` instruction for a hardware barrier.

7.110.1.3 arch_early_init()

```
void arch_early_init (
            void  )
```

Perform the earliest RISC-V architecture initialisation (no-op).

arch.h - Architecture abstraction layer for StarKernel

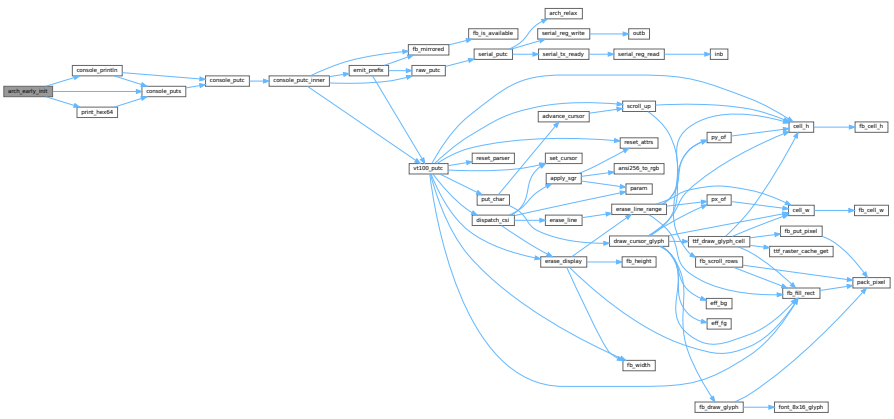
RISC-V has no GDT or IDT; privilege separation is handled by the machine/supervisor/user execution modes (M/S/U), and the trap vector table is installed by writing `stvec` in `arch_interrupts_init()` via `riscv64_install_vectors()`. This stub satisfies the common `arch_early_init()` call site in `kernel_main()` without any RISC-V-specific action.

Correction, 2026-09-04 (FABRIC-3.md ?V.3 item 7's code audit). The unconditional Bare-mode switch this function used to perform was justified entirely by behaviour observed under this system's QEMU/EDK2 RISC-V firmware (`satp.MODE=10/Sv57`, kernel identity-mapped within it — see the comment on `print_hex64()` above). That observation cannot apply to the native boot path (U-Boot+OpenSBI, no EDK2 at all, FABRIC-3.md ?V.1) this kernel now also has, and OpenSBI's own S-mode handoff convention leaves `satp` already at 0 (Bare) in the common case — meaning the switch below was very likely running as a harmless no-op there, but on an unverified assumption rather than a checked one. Rather than build a general Sv39/Sv48/Sv57 page-table walker just to *prove* the running address is identity-mapped before switching (out of proportion to this item's real scope), the fix narrows what actually needs the QEMU-specific safety argument: `**skip the switch entirely when satp is already 0.**` That is the expected case on both a from-Bare native boot and any firmware that already left translation off, and it needs no safety argument at all — there is nothing to switch away from. The unconditional-switch path still exists for the confirmed QEMU/EDK2 case (`mode != 0`), where the identity-mapping argument above remains the actual justification, unchanged.

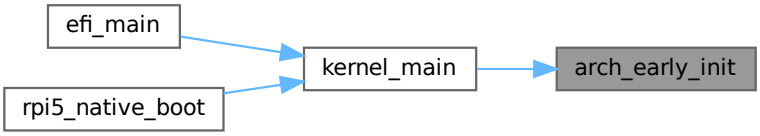
References __kernel_start, console_println(), console_puts(), and print_hex64().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.110.1.4 arch_enable_interrupts()

```
void arch_enable_interrupts (
            void  )
```

Enable supervisor-mode interrupts (RISC-V CSRSI sstatus).

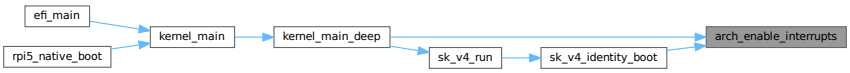
Enable IRQ delivery on the current core (AArch64 DAIFCLR).

Issues `CSRSI sstatus, 0x2` which sets bit 1 (`SIE` — Supervisor Interrupt Enable) in the `sstatus` CSR. While `SIE` = 1, pending supervisor-mode interrupts are taken. The `memory` clobber prevents the compiler from reordering stores across the barrier.

Must be called after `arch_interrupts_init()` has installed the `stvec` trap handler, and after the PLIC driver has been configured to forward the desired interrupt sources.

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the caller graph for this function:



7.110.1.5 arch_halt()

```
void arch_halt (
            void  )
```

Halt the core until the next interrupt (RISC-V WFI).

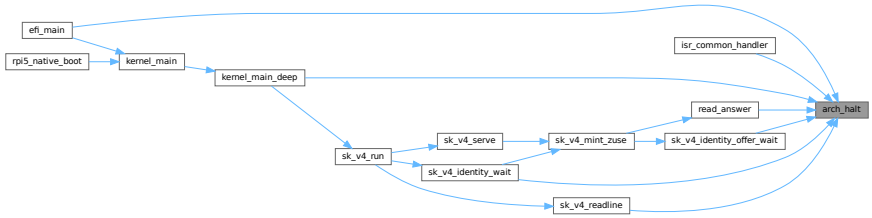
Halt the core until the next event (AArch64 WFI).

Issues the `WFI` (Wait For Interrupt) instruction in supervisor mode. The core enters a low-power state and resumes when a pending interrupt is detected (or after a platform-specific timeout). In the kernel panic path this is called inside an infinite loop with `SIE` cleared to stop the processor permanently. In the idle path it reduces power consumption between heartbeat ticks.

The `memory` clobber prevents store-sinking across the `WFI`.

Referenced by efi_main(), isr_common_handler(), kernel_main_deep(), read_answer(), sk_v4_identity_offer_wait(), sk_v4_identity_wait(), and sk_v4_readline().

Here is the caller graph for this function:



7.110.1.6 arch_mmu_init()

```
void arch_mmu_init (
            void  )
```

Initialise the RISC-V MMU (stub — deferred to a later milestone).

Initialise the AArch64 MMU (stub — deferred to a later milestone).

Full Sv39/Sv48 page-table setup (SATP, PMP, etc.) is deferred to a later milestone. This stub satisfies the common `arch_mmu_init()` call site shared across all three supported ISAs. The VMM subsystem (`vmm.c`) handles page-table management independently of this hook.

7.110.1.7 arch_read_timestamp()

```
uint64_t arch_read_timestamp (
            void  )
```

Read the RISC-V CPU cycle counter (`rdcycle`).

Read the AArch64 system counter (CNTPCT_EL0).

Issues the `RDCYCLE` pseudo-instruction (a `CSRRS` on `cycle`, CSR 0xC00) to read the 64-bit hardware performance counter that counts elapsed clock cycles since reset. On RISC-V the cycle counter is the architectural equivalent of the x86-64 TSC.

Unlike the AArch64 `CNTPCT_EL0`, the RISC-V `cycle` register is a per-hart counter and is not architecturally guaranteed to be synchronised across harts. For single-core LithosAnanke this is not a concern. Frequency is not architecturally specified; `timer_init()` assumes 1 GHz for QEMU `virt` board compatibility.

Returns

Current 64-bit cycle counter value; frequency platform-dependent.

Referenced by calibrate_apic_timer().

Here is the caller graph for this function:

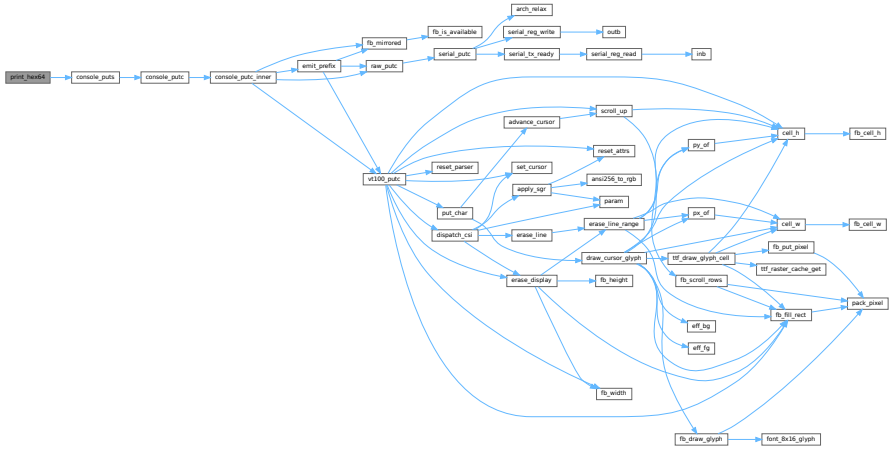
7.110.1.8 print_hex64()

```
static void print_hex64 (
            uint64_t val )  [static]
```

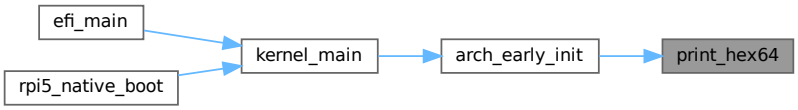
References console_puts().

Referenced by arch_early_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.110.1.9 riscv64_install_vectors()

```
void riscv64_install_vectors (
            void  )  [extern]
```

arch.c (riscv64) - Architecture abstractions for StarKernel on RISC-V 64-bit

7.110.2 Variable Documentation

7.110.2.1 __kernel_start

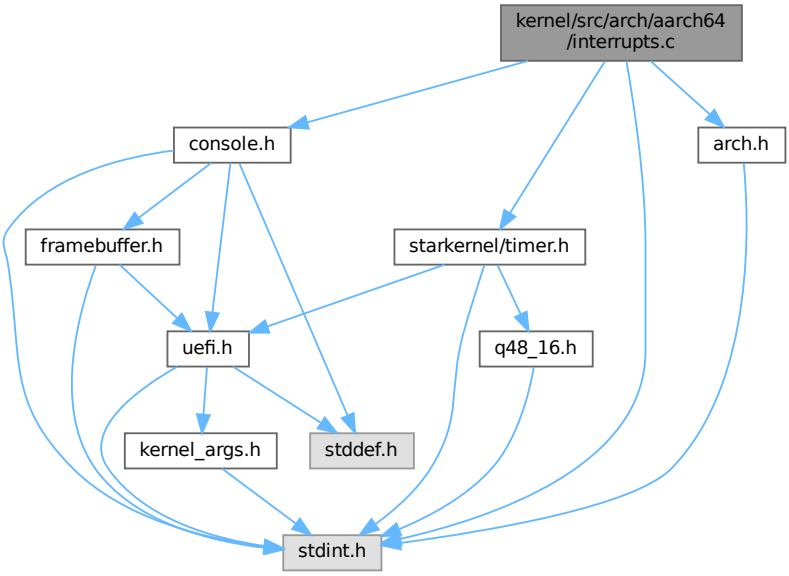
```
char __kernel_start[]  [extern]
```

Referenced by arch_early_init().

7.111 kernel/src/arch/aarch64/interrupts.c File Reference

```
#include <stdint.h>
#include "arch.h"
#include "console.h"
#include "starkernel/timer.h"
```

Include dependency graph for interrupts.c:



Macros

- #define GIC_INTID_SPURIOUS 1023u

Functions

- void aarch64_install_vectors (void)
- int aarch64_current_el (void)

  Return the AArch64 exception level the kernel is running at.
- uint32_t apic_read_iar (void)

  Read the GIC CPU interface's Interrupt Acknowledge Register.
- uint32_t apic_timer_ppi (void)

  Return the timer PPI this build is watching for.
- void apic_eoi_intid (uint32_t intid)

  Signal End of Interrupt to the GIC CPU interface.
- void apic_timer_rearm (void)

  Re-arm the ARM Generic Timer for the next period (item 0.7).
- void virtio_input_isr (void)
- void aarch64_irq_handler (void)

  Dispatch an IRQ taken at current-EL with SPx (punch-list item 0.7).
- static void print_hex (uint64_t val)

Print a 64-bit value as "0xNNNNNNNNNNNNNNNN" to the kernel console.

- void aarch64_exception_handler (void)

  AArch64 common exception handler — print diagnostic and spin.

- void arch_interrupts_init (void)

  Install the AArch64 exception vector table (M4 milestone).

Variables

- volatile const char * g_sk_fault_word = (void *)0

  interrupts.c (aarch64) - Exception vector setup and common handler

- uint32_t g_virtio_input_gic_intid

7.111.1 Macro Definition Documentation

7.111.1.1 GIC_INTID_SPURIOUS

```
#define GIC_INTID_SPURIOUS 1023u
```

7.111.2 Function Documentation

7.111.2.1 aarch64_current_el()

```
int aarch64_current_el (
            void  )  [extern]
```

Return the AArch64 exception level the kernel is running at.

apic.c (aarch64) - Minimal GICv2 driver + ARM Generic Timer (items 0.6/0.7/ 4.3.5d).

Reads `CurrentEL` (bits [3:2] hold the EL number, [1:0] are RES0) on first call and caches the result; every later call returns the cached value. Safe to call at any point in boot, including before `console_init()` — the read itself has no dependency, only the printing of it does (see `arch_interrupts_init()` in `interrupts.c`, the earliest point at which both this and a working console are available).

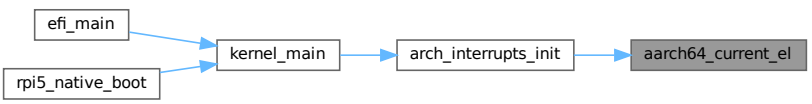
EDK2 on QEMU's aarch64 `virt` machine has been observed to leave the kernel at either EL1 or EL2 depending on firmware build; nothing in this tree may assume one over the other (FABRIC-0.md ?25.7.1 GAP-B3). Every EL-dependent choice — `VBAR_EL1` vs `VBAR_EL2`, the `ELR_ELx` / `SPSR_ELx` saved-state pair, and `CNTP_*_EL0` vs `CNTHP_*_EL2` — must read this accessor rather than hardcode a level.

Returns

1 or 2. Never 0 or 3: a UEFI application that survived far enough to reach this call cannot be running at EL0 (unprivileged) or EL3 (secure monitor — QEMU's virt firmware runs the kernel at EL1/EL2).

Referenced by arch_interrupts_init().

Here is the caller graph for this function:



7.111.2.2 aarch64_exception_handler()

```
void aarch64_exception_handler (
            void  )
```

AArch64 common exception handler — print diagnostic and spin.

Called from the `isr.S` exception vector table stubs for all exception types (synchronous, IRQ, FIQ, SError) that are not handled by a specific driver. Reads the four key EL1 exception system registers directly:

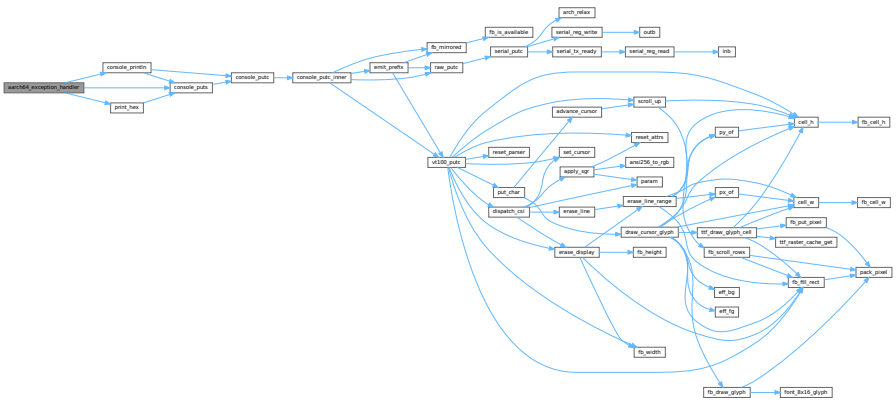
- `ESR_EL1` — Exception Syndrome Register: encodes the exception class (EC, bits [31:26]) and instruction-specific syndrome (ISS, bits [24:0]). The EC identifies the cause (e.g., 0x21 = data abort from EL1, 0x25 = data abort from EL0, 0x15 = SVC64).
- `ELR_EL1` — Exception Link Register: the PC of the instruction that caused the exception (or the instruction after, for IRQ/FIQ/SError).
- `FAR_EL1` — Fault Address Register: the virtual address that triggered a Data Abort or Instruction Abort; undefined for other exception types.
- `SPSR_EL1` — Saved Program Status Register: the PSTATE at the time of exception, including the mode field and interrupt-mask bits.

Prints all four values to the kernel console then enters an infinite `WFE` loop to halt the affected core permanently. All exceptions that reach this handler are fatal — there is no recovery path.

The global `g_sk_fault_word` is set by the VM FORTH dispatcher just before calling each word's function pointer; it names the word executing at the time of the fault, aiding post-mortem analysis from the serial log. (Currently logged only on the amd64 #GP path; future AArch64 diagnostic expansion can include it here.)

References console_println(), console_puts(), and print_hex().

Here is the call graph for this function:

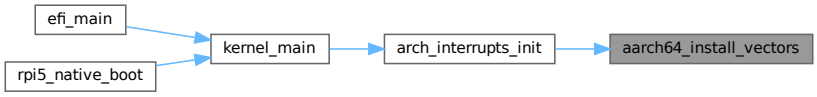


7.111.2.3 aarch64_install_vectors()

```
void aarch64_install_vectors (
            void  )  [extern]
```

Referenced by arch_interrupts_init().

Here is the caller graph for this function:



7.111.2.4 aarch64_irq_handler()

```
void aarch64_irq_handler (
            void  )
```

Dispatch an IRQ taken at current-EL with SPx (punch-list item 0.7).

Called from `irq_spx_trampoline` in `isr.S`, which has already saved the full caller-saved integer and FP/SIMD register sets plus `ELR_ELx` / `SPSR_ELx`, and will restore all of it and `ERET` after this function returns — unlike `aarch64_exception_handler()`, this path is designed to return normally.

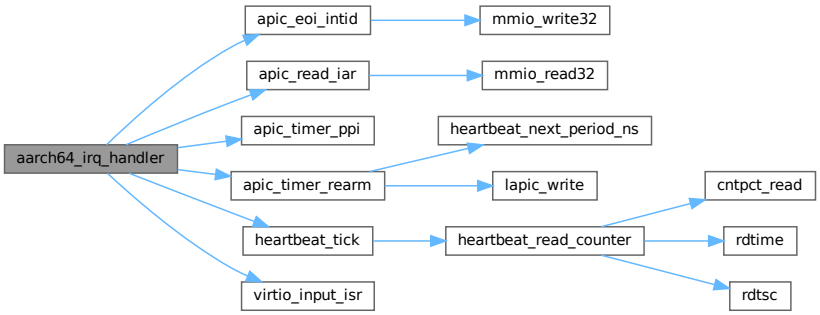
Reads `GICC_IAR` once (the GICv2 acknowledgement step — this also moves the interrupt to the active state), dispatches on the returned INTID, and always completes with the matching `GICC_EOIR` write, spurious or not: a spurious read must still not be treated as "nothing to clean up," and an unrecognised real INTID still needs the CPU interface unblocked for the next one.

For the timer PPI (`apic_timer_ppi()` — 30 at EL1, 26 at EL2, decided once in `apic_init()`: `apic_timer_rearm()` runs **first**, before `heartbeat_tick()` — the ARM Generic Timer does not auto-reload, so skipping this leaves the interrupt condition permanently latched (see `apic_timer_rearm()`'s own doc comment). Ordering the re-arm ahead of the rest of the dispatch means a fault in `heartbeat_tick()` cannot also cost the next tick — same discipline as `riscv64_timer_rearm()` (item 0.3).

virtio-input (item 4.3.5e, `g_virtio_input_gic_intid`): dispatched **before** the `GICC_EOIR` write below, not after — the ISR itself is what deasserts the device's (level-triggered) interrupt line by reading its ISR-status capability; EOI'ing first would let the GIC immediately re-signal a still-asserted line. Same claim-dispatch-complete ordering `riscv64_interrupt_handler()` uses for its own PLIC-routed devices.

References apic_eoi_intid(), apic_read_iar(), apic_timer_ppi(), apic_timer_rearm(), g_virtio_input_gic_intid, GIC_INTID_SPURIOUS, heartbeat_tick(), and virtio_input_isr().

Here is the call graph for this function:



7.111.2.5 apic_eoi_intid()

```
void apic_eoi_intid (
            uint32_t intid )  [extern]
```

Signal End of Interrupt to the GIC CPU interface.

Writes `intid` back to `GICC_EOIR`, matching the value `GICC_IAR` returned in `aarch64_irq_handler()`. Required after every acknowledged interrupt or the GIC considers it permanently active and will not re-signal it.

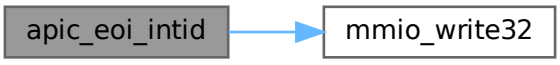
Parameters

intid	INTID as read from `GICC_IAR` (10 bits used by GICv2).

References GICC_EOIR, mmio_write32(), and s_gicc_base.

Referenced by aarch64_irq_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

7.111.2.6 apic_read_iar()

```
uint32_t apic_read_iar (
            void  )  [extern]
```

Read the GIC CPU interface's Interrupt Acknowledge Register.

Reading `GICC_IAR` returns the INTID of the highest-priority pending interrupt and simultaneously moves it to the active state — this is the GICv2 acknowledgement step. Called from `aarch64_irq_handler()` in `interrupts.c` to identify which interrupt fired; the matching `apic_eoi_intid()` call below completes it.

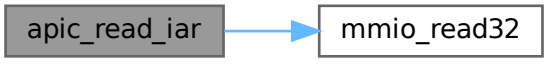
Returns

INTID (10 bits used by GICv2).

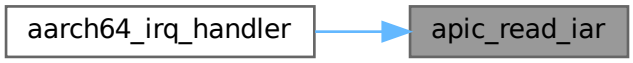
References GICC_IAR, mmio_read32(), and s_gicc_base.

Referenced by aarch64_irq_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.111.2.7 apic_timer_ppi()

```
uint32_t apic_timer_ppi (
            void  )  [extern]
```

Return the timer PPI this build is watching for.

Set once in `apic_init()` from `aarch64_current_el()` (30 at EL1, 26 at EL2). `aarch64_irq_handler()` compares `apic_read_iar()`'s result against this rather than a hardcoded constant.

References s_timer_ppi.

Referenced by aarch64_irq_handler().

Here is the caller graph for this function:

7.111.2.8 apic_timer_rearm()

```
void apic_timer_rearm (
            void  )  [extern]
```

Re-arm the ARM Generic Timer for the next period (item 0.7).

Re-arm the APIC timer at the current adaptive period (item 0.8, ?26).

Writes a fresh `TVAL` — the entirety of a periodic re-arm, since `TVAL` is relative to "now" rather than an absolute deadline. Unlike riscv64's SBI re-arm (item 0.3), there is no drift-correction bookkeeping: each write means "@c s_timer_period_tsc ticks from this instant," so time spent servicing the interrupt before this call simply lengthens that one period rather than accumulating error across ticks.

This is the one call that must never be skipped. The ARM Generic Timer does not auto-reload: once `TVAL` crosses zero, `CTL.ISTATUS` latches and stays asserted — a level condition — until software writes a new `TVAL` or clears `CTL.ENABLE`. A return path that skips this leaves the interrupt permanently pending, which the GIC would deliver again the instant it is EOI'd: a real interrupt storm, not a hypothetical one (this is exactly the failure this driver was checked against during item 0.6's own verification, and confirmed absent there only because nothing armed the timer yet).

Called from `aarch64_irq_handler()` (interrupts.c) before `heartbeat_tick()` — same ordering discipline as `riscv64_timer_rearm()` (item 0.3): re-arm first, so a fault in the rest of the handler cannot also cost the next tick.

The step written is `heartbeat_next_period_ns()` (item 0.8, ?26) converted to `CNTPCT_EL0` ticks via `s_counter_hz_apic`, not the fixed `s_timer_period_tsc` `apic_timer_start()` used for the very first arm – that value remains `apic_timer_period_tsc()`'s return for `heartbeat_init()`'s initial `expected_delta`, unchanged.

Re-arm the ARM Generic Timer for the next period (item 0.7).

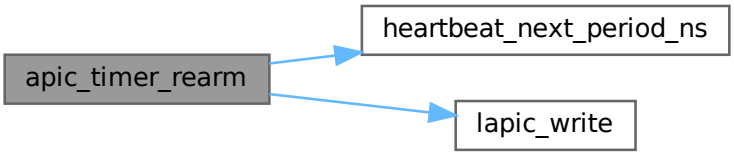
amd64's APIC timer runs in periodic mode: hardware auto-reloads `APIC_REG_TIMER_ICR` on every expiry with no software intervention, unlike riscv64's one-shot SBI deadline or aarch64's one-shot `CNTP_TVAL_EL0` – neither of which needed a rearm function before this item for the same reason this one now exists. To make the period adaptive, this function recomputes `timer_initial_count` from `heartbeat_next_period_ns()` (?26) and `s_apic_hz`, then writes it to `APIC_REG_TIMER_ICR`. A periodic-mode ICR write takes effect immediately and restarts the countdown at the new value – the same mechanism `apic_timer_start()` already relies on to force QEMU TCG's emulated APIC to begin counting.

Called from `isr_common_handler()` on every `APIC_TIMER_VECTOR` interrupt, before `heartbeat_tick()` – same ordering discipline as riscv64/aarch64's rearm-before-heartbeat_tick(), so a fault in `heartbeat_tick()` cannot also cost the next tick.

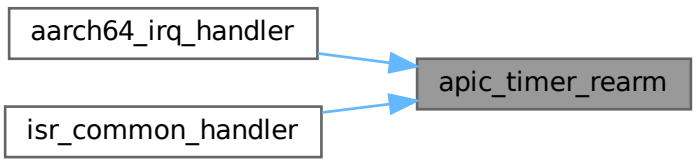
References APIC_REG_TIMER_ICR, heartbeat_next_period_ns(), lapic_write(), s_apic_hz, and timer_initial_count.

Referenced by aarch64_irq_handler(), and isr_common_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.111.2.9 arch_interrupts_init()

```
void arch_interrupts_init (
            void  )
```

Install the AArch64 exception vector table (M4 milestone).

First reads and reports the exception level via aarch64_current_el() (punch-list item 0.4) — this is the earliest point in boot with both a working console (up since M1) and a genuine need for the answer (vector installation is the first EL-dependent operation). The read itself is cheap and safe far earlier than this; only the console print is gated on this call site.

Then calls aarch64_install_vectors() (defined in isr.S) which writes the address of the vector table base into VBAR_EL1 or VBAR_EL2 (Vector Base Address Register), whichever matches the exception level aarch64_current_el() just reported. After this instruction all exceptions at that level (synchronous, IRQ, FIQ, SError) dispatch through the 512-byte-aligned vector table assembled in isr.S.

Correction, 2026-09-04 (FABRIC-3.md ?IV.3 item 7's code audit): this doc comment and the runtime console message below previously claimed EL2 was "a known gap... @c VBAR_EL1 is written unconditionally regardless of the detected level," citing punch-list item 0.5 as the still-open work to fix it. Checked against isr.S's own aarch64_install_vectors() (the actual implementation, not this comment) and found that claim stale↩ : the assembly already branches on aarch64_current_el() and writes vbar_el2 at EL2 / vbar_el1 at EL1, threading the same answer through el2_mode_flag for the IRQ trampoline's ELR_ELx/SPSR_ELx selection too (see isr.S's own "EL selection" header comment). There never was a gap here to close — item 0.5's EL2 vector work was already done by the time this comment was written describing it as open; only the comment was wrong, not the code.

On AArch64 there is no IDT and no PIC to disable; the GIC replaces both. The arch_interrupts_init() name is kept identical across all ISAs so that kernel_main() can call the same symbol regardless of the target architecture.

Must be called after the kernel stack is established (so that the correct-EL SP is valid) and before arch_enable_interrupts().

7.111.2.10 print_hex()

```
static void print_hex (
             uint64_t val )  [static]
```

Print a 64-bit value as "0xNNNNNNNNNNNNNNNN" to the kernel console.

Formats val as a 16-nibble lowercase hex string preceded by "0x" into a 19-byte stack buffer (including null terminator) then emits it via console_puts(). Used by aarch64_exception_handler() to print ESR_EL1, ELR_EL1, FAR_EL1, and SPSR_EL1 in the exception diagnostic dump. No heap allocation or libc dependency.

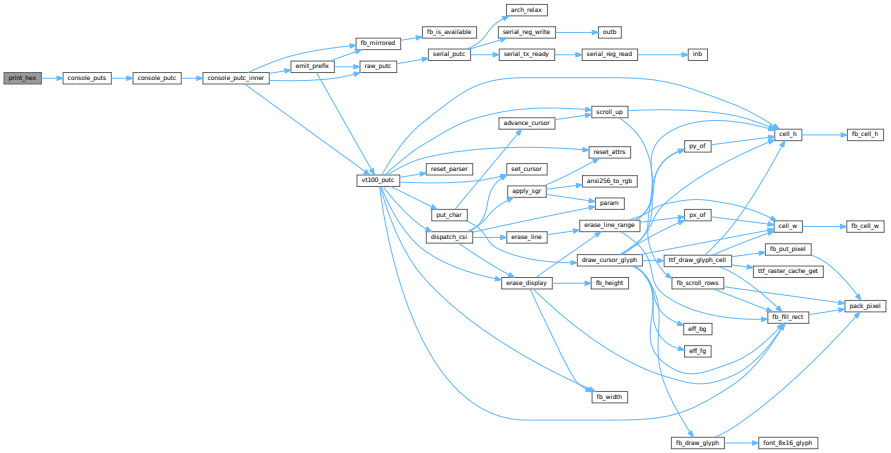
Parameters

val	64-bit value to format.

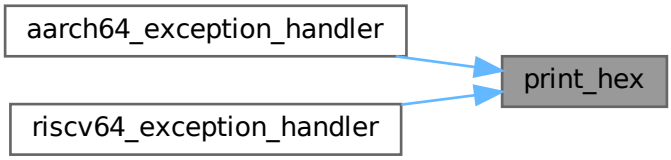
References console_puts().

Referenced by aarch64_exception_handler(), and riscv64_exception_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

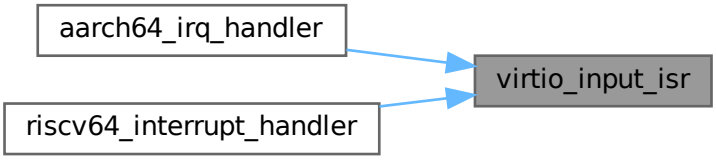


7.111.2.11 virtio_input_isr()

```
void virtio_input_isr (
            void  )  [extern]
```

Referenced by aarch64_irq_handler(), and riscv64_interrupt_handler().

Here is the caller graph for this function:



7.111.3 Variable Documentation

7.111.3.1 g_sk_fault_word

```
volatile const char* g_sk_fault_word = (void *)0
```

interrupts.c (aarch64) - Exception vector setup and common handler

Referenced by isr_common_handler().

7.111.3.2 g_virtio_input_gic_intid

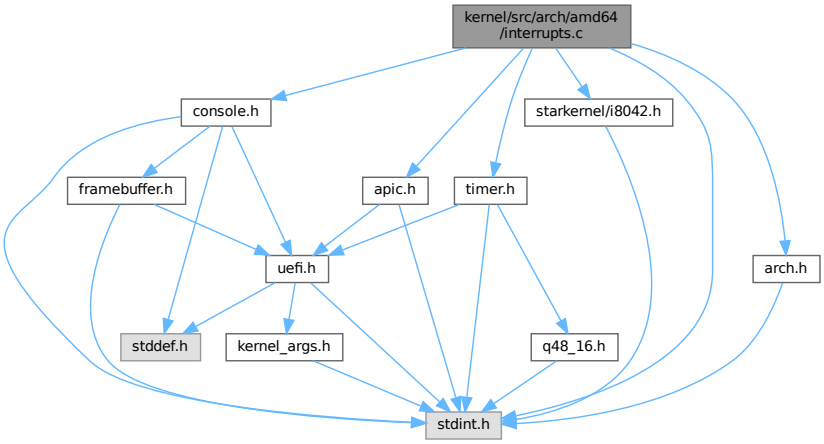
```
uint32_t g_virtio_input_gic_intid  [extern]
```

Referenced by aarch64_irq_handler().

7.112 kernel/src/arch/amd64/interrupts.c File Reference

```
#include <stdint.h>
#include "arch.h"
#include "console.h"
#include "apic.h"
#include "timer.h"
#include "starkernel/i8042.h"
```

Include dependency graph for interrupts.c:



Data Structures

- struct idt_entry
- struct idtr

Macros

- #define IDT_ENTRIES 256
- #define INTERRUPT_GATE 0x8E

Functions

- static void outb (uint16_t port, uint8_t val)

  Write a single byte to an x86 I/O port via `OUT`.
- struct idt_entry __attribute__ ((packed))
- void ∗isr_stub_table[256] __attribute__ ((visibility("hidden")))
- void isr_common_handler (uint64_t vector, uint64_t error_code, uint64_t rip, uint64_t cs, uint64_t rflags, uint64_t cr2, uint64_t stack_frame_ptr)

  Unified C-level interrupt and exception handler.
- static void set_idt_entry (int vector, void ∗isr)

  Write a single 64-bit interrupt gate descriptor into the IDT.
- static void lidt (struct idtr ∗idtr_desc)

  Load the Interrupt Descriptor Table register (`LIDT`).
- static void print_hex64 (uint64_t value)

  Print a 64-bit value as 16 lowercase hex digits to the kernel console.
- static void print_dec_u64 (uint64_t v)

  Print a 64-bit unsigned integer in decimal to the kernel console.
- static void pic_disable (void)

  Fully mask the legacy 8259A PIC to suppress all legacy IRQs.
- static void decode_pf_error (uint64_t ec)

  Decode and print the x86-64 Page Fault error code to the kernel console.
- void isr_stub0 (void)

  Initialise the x86-64 IDT and disable the legacy 8259 PIC (M4).

Variables

- volatile const char ∗ g_sk_fault_word = (void ∗)0

  interrupts.c (amd64) - Interrupt setup and exception/IRQ handling
- volatile uint32_t g_spurious_count = 0
- uint16_t offset_low
- uint16_t selector
- uint8_t ist
- uint8_t type_attr
- uint16_t offset_mid
- uint32_t offset_high
- uint32_t zero
- uint16_t limit
- uint64_t base
- static struct idt_entry idt [256]

7.112.1 Macro Definition Documentation

7.112.1.1 IDT_ENTRIES

```
#define IDT_ENTRIES 256
```

7.112.1.2 INTERRUPT_GATE

```
#define INTERRUPT_GATE 0x8E
```

7.112.2 Function Documentation

7.112.2.1 __attribute__() [1/2]

```
struct idt_entry __attribute__ (
             (packed)  )
```

7.112.2.2 __attribute__() [2/2]

```
void *isr_stub_table[256] __attribute__ (
             (visibility("hidden"))  )  [extern]
```

7.112.2.3 decode_pf_error()

```
static void decode_pf_error (
             uint64_t ec )  [static]
```

Decode and print the x86-64 Page Fault error code to the kernel console.

The processor pushes a 32-bit error code onto the stack for a #PF (vector 14). Each bit encodes information about the faulting access:

- bit 0 (P): 0 = page not present; 1 = protection violation.
- bit 1 (W/R): 0 = read access; 1 = write access.
- bit 2 (U/S): 0 = supervisor mode access; 1 = user mode access.
- bit 3 (RSVD): 1 = fault caused by a reserved bit set in a PTE.
- bit 4 (I/D): 1 = instruction fetch (no-execute violation).
- bit 5 (PK): 1 = protection-key violation (if PKRU is active).
- bit 6 (SS): 1 = shadow-stack access violation (CET).
- bit 15 (SGX): 1 = SGX enclave access violation.

Prints each active flag as a short tag to the kernel console followed by a newline. The CR2 register (captured by the ISR stub) holds the faulting linear address and is printed separately by `isr_common_handler()`.

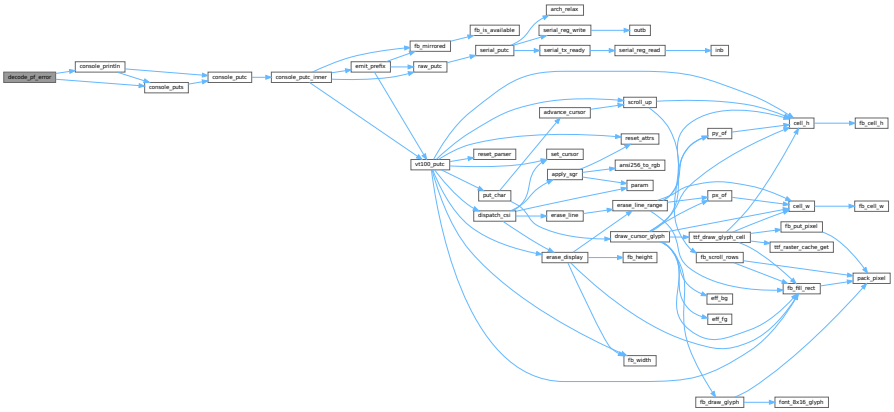
Parameters

ec	64-bit page fault error code (upper 32 bits are always zero on current x86-64 hardware; tested up to bit 15 per SDM).

References console_println(), and console_puts().

Referenced by isr_common_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

7.112.2.4 isr_common_handler()

```
void isr_common_handler (
            uint64_t vector,
            uint64_t error_code,
            uint64_t rip,
            uint64_t cs,
            uint64_t rflags,
            uint64_t cr2,
            uint64_t stack_frame_ptr )  [extern]
```

Unified C-level interrupt and exception handler.

All 256 IDT entries funnel through a common assembly trampoline in `isr.S` which captures caller-saved registers, reads `CR2` (the page-fault linear address), normalises the error code to 0 for vectors that do not push one, and then calls this function with six arguments reconstructed from the interrupt stack frame.

APIC timer path (vector == `APIC_TIMER_VECTOR`): Calls `heartbeat_tick()` to update the rolling window of inter-tick TSC deviations, then issues `apic_eoi()` and returns. This is the hot path — it executes at 100 Hz during normal kernel operation and must be kept brief.

Exception path (all other vectors): Prints a structured diagnostic to the kernel console:

- Vector number (decimal and hex)
- Error code (hex)
- RIP (hex) — faulting instruction pointer
- CR2 (hex) — page-fault linear address (meaningful only for vector 14)
- RSP (derived from `stack_frame_ptr` + 160, covering the ISR stub's pushed register block)
- RBP (read from the saved-register block at `stack_frame_ptr` + 64)
- Return address (first quadword at the derived RSP)
- Vector-specific decoding: `#DE` (0), `#GP` (13) with the active FORTH word name from `g_sk_fault_word`, `#PF` (14) with `decode_pf_error()`. Then halts the CPU in an infinite `HLT` loop — exceptions are fatal.

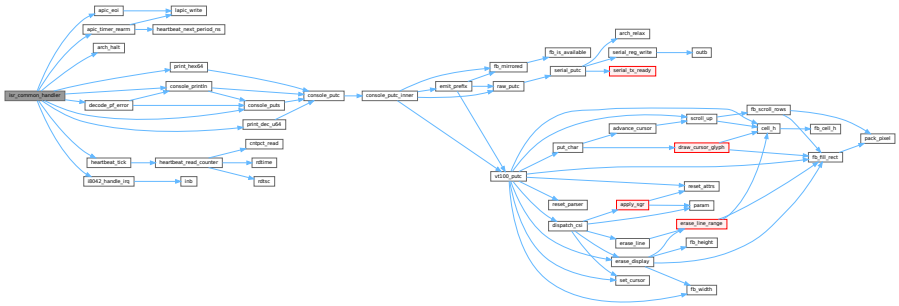
The `cs` and `rflags` parameters are currently unused (suppressed with (void) casts) but retained in the signature for future use (e.g., printing privilege level on #GP).

Parameters

vector	IDT vector that fired [0, 255].
error_code	Error code from the CPU stack (0 if not applicable).
rip	Faulting or interrupted instruction pointer.
cs	Code segment selector at the time of the interrupt.
rflags	RFLAGS at the time of the interrupt.
cr2	Value of `CR2` (page-fault linear address; only architecturally meaningful for vector 14).
stack_frame_ptr	Address of the ISR stub's saved-register block on the stack; used to reconstruct RSP and RBP for the diagnostic dump.

References apic_eoi(), APIC_SPURIOUS_VECTOR, apic_timer_rearm(), APIC_TIMER_VECTOR, arch_halt(), console_println(), console_puts(), decode_pf_error(), g_sk_fault_word, g_spurious_count, heartbeat_tick(), i8042_handle_irq(), I8042_KEYBOARD_VECTOR, print_dec_u64(), and print_hex64().

Here is the call graph for this function:



7.112.2.5 isr_stub0()

```
void isr_stub0 (
            void  )  [extern]
```

Initialise the x86-64 IDT and disable the legacy 8259 PIC (M4).

This is the M4 milestone function called from kernel_main() after the GDT is live (arch_early_init()) and before arch_enable_interrupts(). It performs three operations:

1. **Relocation fixup for the ISR stub table.** isr.S exports isr_stub_table[], a 256-entry table of link-time function pointers assembled at their expected virtual addresses. In a UEFI PE32+ image the UEFI loader places the image at an arbitrary base; pointer-sized relocations in .data are not applied by the PE loader (only code-relative relocations in .text are). The fixup computes the runtime delta between isr_stub0's runtime address and its link-time address stored in isr_stub_table[0], then applies that delta to every entry before installing it in the IDT.
2. **Fill all 256 IDT entries.** Calls set_idt_entry() for each vector with the corrected stub address, selector 0x08, IST 0, and type_attr INTERRUPT_GATE (0x8E).
3. **Load the IDT and mask the PIC.** Calls lidt() with the fully populated idt[] array, then calls pic_disable() to mask both PIC1 and PIC2 before STI is issued. This prevents spurious legacy PIC IRQs from reaching the IDT before the Local APIC takes over.

Must be called after gdt_init() (for selector 0x08) and before apic_init() / apic_timer_init() / arch_enable_interrupts().

7.112.2.6 lidt()

```
static void lidt (
            struct idtr * idtr_desc )  [static]
```

Load the Interrupt Descriptor Table register (LIDT).

Executes the LIDT instruction with the descriptor at idtr_desc, which contains the 16-bit limit (byte size of the IDT minus 1) and the 64-bit linear base address of the IDT array. After this instruction the CPU uses the new IDT for all exception and interrupt dispatch.

Must be called after all 256 entries in idt[] have been filled by set_idt_entry(). Called at the end of arch_interrupts_init().

Parameters

idtr_desc	Pointer to an idtr struct containing limit and base.

7.112.2.7 outb()

```
static void outb (
            uint16_t port,
            uint8_t val )  [inline], [static]
```

Write a single byte to an x86 I/O port via OUT.

Issues the OUT instruction to port port with data byte val. Used exclusively to program the legacy 8259 PIC's interrupt-mask registers during pic_disable(). In a kernel that uses the Local APIC for all interrupt delivery the PIC must be fully masked before enabling CPU interrupts; otherwise spurious IRQ0 from the PIC timer can corrupt the APIC timer ISR path.

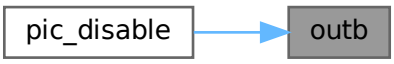
The "Nd" constraint allows the assembler to emit the short-form OUT imm8 encoding for constants in [0, 255] and the register form OUT DX otherwise.

Parameters

port	16-bit I/O port number (e.g., 0x21 = PIC1 data, 0xA1 = PIC2 data).
val	Byte value to write to the port.

Referenced by pic_disable().

Here is the caller graph for this function:



7.112.2.8 pic_disable()

```
static void pic_disable (
            void  )  [static]
```

Fully mask the legacy 8259A PIC to suppress all legacy IRQs.

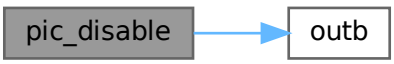
Writes 0xFF to PIC1's interrupt-mask register (port 0x21) and PIC2's interrupt-mask register (port 0xA1). When all bits are set every IRQ line (IRQ0–IRQ15) is masked and the PIC will not assert INTR to the CPU.

This is necessary because UEFI firmware initialises the 8259 PIC in compatibility mode with IRQ0 (timer) running at 18.2 Hz. After `ExitBootServices()` the firmware no longer handles these IRQs; if the PIC is not masked before `STI`, spurious PIC interrupts are delivered to the kernel's IDT at legacy vectors 0x20–0x2F, most of which have not been set up as ISRs and will triple-fault.

Called from `arch_interrupts_init()` after the IDT is loaded but before `arch_enable_interrupts()`.

References outb().

Here is the call graph for this function:



7.112.2.9 print_dec_u64()

```
static void print_dec_u64 (
            uint64_t v )  [static]
```

Print a 64-bit unsigned integer in decimal to the kernel console.

Converts `v` to a decimal ASCII string using a 32-byte stack buffer (digit reversal technique) and emits each character via `console_putc()`. Prints "0" for a zero value. Used by `isr_common_handler()` to print the exception vector number in decimal alongside its hex representation. No allocation or libc dependency.

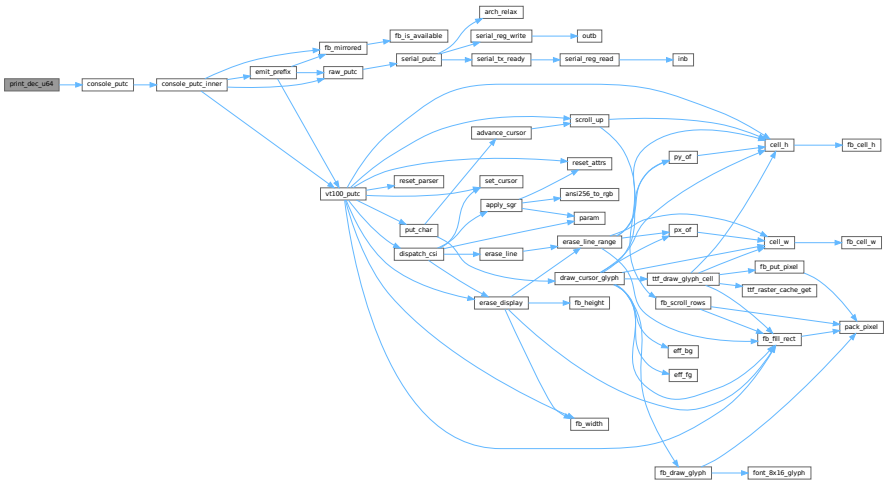
Parameters

v	64-bit unsigned integer to print.

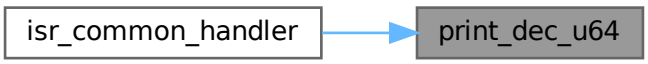
References console_putc().

Referenced by isr_common_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.112.2.10 print_hex64()

```
static void print_hex64 (
            uint64_t value )  [static]
```

Print a 64-bit value as 16 lowercase hex digits to the kernel console.

Emits exactly 16 nibbles (no `0x` prefix, no separators) by iterating from bit 60 down to bit 0 in steps of 4. Used exclusively by `isr_common_handler()` to print register values (RIP, CR2, RSP, RBP, return address) in the exception diagnostic dump. No allocation or libc dependency.

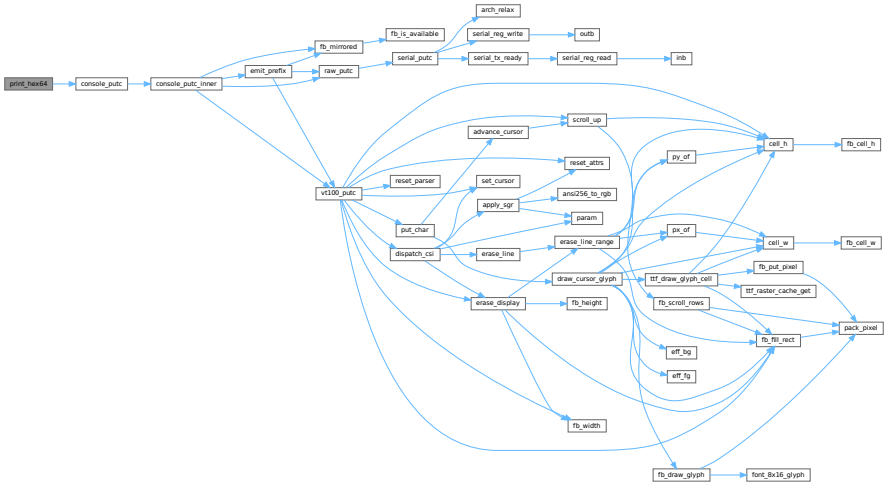
Parameters

value	64-bit value to format as lowercase hex.

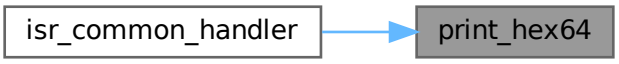
References console_putc().

Referenced by isr_common_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.112.2.11 set_idt_entry()

```
static void set_idt_entry (
            int vector,
            void * isr )  [static]
```

Write a single 64-bit interrupt gate descriptor into the IDT.

Fills `idt[vector]` with an x86-64 interrupt gate (type 0x8E):

- `offset_low` / `offset_mid` / `offset_high` — three parts of the ISR handler address `isr` split across the 16-byte descriptor layout.

- `selector` = 0x08 — kernel code segment (GDT slot 1).
- `ist` = 0 — no interrupt stack table; uses the current RSP.
- `type_attr` = `INTERRUPT_GATE` (0x8E) — P=1, DPL=0, 64-bit interrupt gate; the CPU automatically clears RFLAGS.IF on entry.
- `zero` = 0 — upper reserved dword must be zero.

Called once per vector inside `arch_interrupts_init()`'s loop after the relocation offset has been applied to the stub address from `isr_stub_table`.

Parameters

vector	IDT vector index [0, 255].
isr	Runtime address of the per-vector assembly stub in `isr.S`.

References idt, INTERRUPT_GATE, idt_entry::ist, idt_entry::offset_high, idt_entry::offset_low, idt_entry::offset_mid, idt_entry::selector, idt_entry::type_attr, and idt_entry::zero.

7.112.3 Variable Documentation

7.112.3.1 base

```
uint64_t base
```

7.112.3.2 g_sk_fault_word

```
volatile const char* g_sk_fault_word = (void *)0
```

interrupts.c (amd64) - Interrupt setup and exception/IRQ handling

interrupts.c (aarch64) - Exception vector setup and common handler

7.112.3.3 g_spurious_count

```
volatile uint32_t g_spurious_count = 0
```

Referenced by isr_common_handler(), and kbw_debug().

7.112.3.4 idt

```
struct idt_entry idt[256]  [static]
```

Referenced by set_idt_entry().

7.112.3.5 ist

```
uint8_t ist
```

7.112.3.6 limit

```
uint16_t limit
```

7.112.3.7 offset_high

```
uint32_t offset_high
```

7.112.3.8 offset_low

```
uint16_t offset_low
```

7.112.3.9 offset_mid

```
uint16_t offset_mid
```

7.112.3.10 selector

```
uint16_t selector
```

7.112.3.11 type_attr

```
uint8_t type_attr
```

7.112.3.12 zero

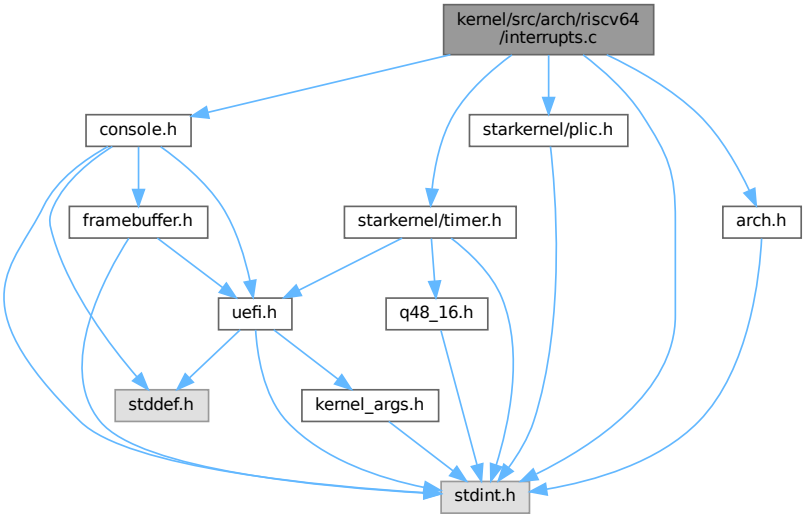
```
uint32_t zero
```

Referenced by bucket_posterior(), cache_posterior(), fe25519_neg(), gpt_grow_last(), gpt_read(), gpt_write_one(), point_decompress(), q48_from_double(), sha512_final(), and sk_v4_mint_zuse().

7.113 kernel/src/arch/riscv64/interrupts.c File Reference

```
#include <stdint.h>
#include "arch.h"
#include "console.h"
#include "starkernel/timer.h"
#include "starkernel/plic.h"
```

Include dependency graph for interrupts.c:



Macros

- #define SCAUSE_INTERRUPT_BIT (1ULL << 63)
- #define SCAUSE_S_TIMER 5ULL
- #define SCAUSE_S_EXTERNAL 9ULL

Functions

- void riscv64_install_vectors (void)
- void riscv64_timer_rearm (void)

  Re-arm the one-shot SBI timer and account the tick.
- void virtio_input_isr (void)
- void riscv64_interrupt_handler (uint64_t scause)

  Dispatch an asynchronous supervisor interrupt.
- static void print_hex (uint64_t val)

  Print a 64-bit value as "0xNNNNNNNNNNNNNNNN" to the kernel console.
- void riscv64_exception_handler (void)

  RISC-V common trap handler — print diagnostic and spin.
- void arch_interrupts_init (void)

  Install the RISC-V supervisor trap vector table (M4 milestone).

Variables

- volatile const char * g_sk_fault_word = (void *)0

  interrupts.c (riscv64) - Trap vector setup and common exception handler
- uint32_t g_virtio_input_plic_source
- volatile uint32_t g_plic_claim_count = 0

7.113.1 Macro Definition Documentation

7.113.1.1 SCAUSE_INTERRUPT_BIT

```
#define SCAUSE_INTERRUPT_BIT (1ULL << 63)
```

7.113.1.2 SCAUSE_S_EXTERNAL

```
#define SCAUSE_S_EXTERNAL 9ULL
```

7.113.1.3 SCAUSE_S_TIMER

```
#define SCAUSE_S_TIMER 5ULL
```

7.113.2 Function Documentation

7.113.2.1 arch_interrupts_init()

```
void arch_interrupts_init (
            void  )
```

Install the RISC-V supervisor trap vector table (M4 milestone).

Install the AArch64 exception vector table (M4 milestone).

Calls `riscv64_install_vectors()` (defined in `isr.S`) which writes the trap vector base address into `stvec` (Supervisor Trap Vector Base Address Register, CSR 0x105). The vector mode field (bits [1:0]) is set to `0` (Direct mode) so all traps dispatch to the single base address, or `1` (Vectored mode) so async interrupts dispatch to `stvec` + 4*cause.

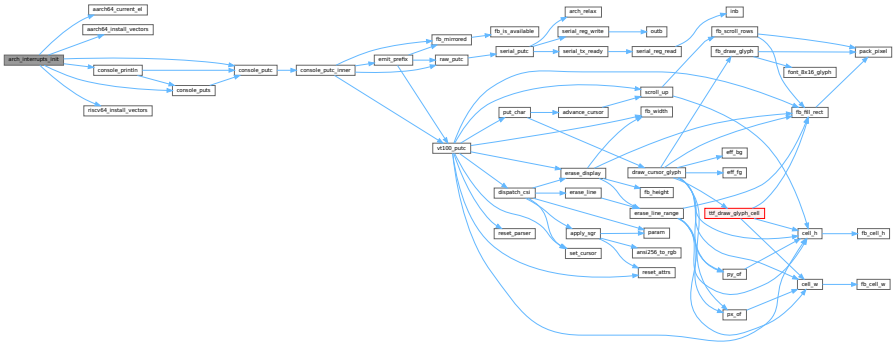
On RISC-V there is no IDT and no legacy PIC; the PLIC replaces both. The `arch_interrupts_init()` name is shared across all three ISAs so that `kernel_main()` calls the same symbol regardless of target.

Must be called after the supervisor stack (`sp`) is established and before `arch_enable_interrupts()`.

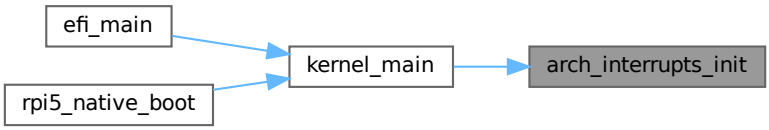
References aarch64_current_el(), aarch64_install_vectors(), console_println(), console_putc(), console_puts(), and riscv64_install_vectors().

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.113.2.2 print_hex()

```
static void print_hex (
            uint64_t val )  [static]
```

Print a 64-bit value as "0xNNNNNNNNNNNNNNNN" to the kernel console.

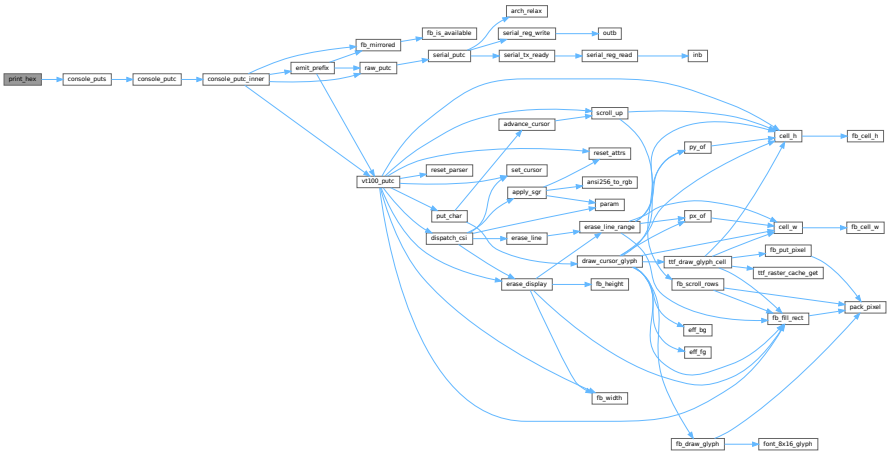
Formats `val` as a 16-nibble lowercase hex string preceded by "0x" into a 19-byte stack buffer (including null terminator) and emits it via `console_puts()`. Used by `riscv64_exception_handler()` to print `scause`, `sepc`, and `stval` in the exception diagnostic dump. No heap allocation or libc dependency.

Parameters

val	64-bit value to format.

References console_puts().

Here is the call graph for this function:



7.113.2.3 riscv64_exception_handler()

```
void riscv64_exception_handler (
            void  )
```

RISC-V common trap handler — print diagnostic and spin.

Called from the `isr.S` trap vector for all synchronous exceptions and asynchronous interrupts that are not handled by a specific driver. Reads three supervisor-mode CSRs directly:

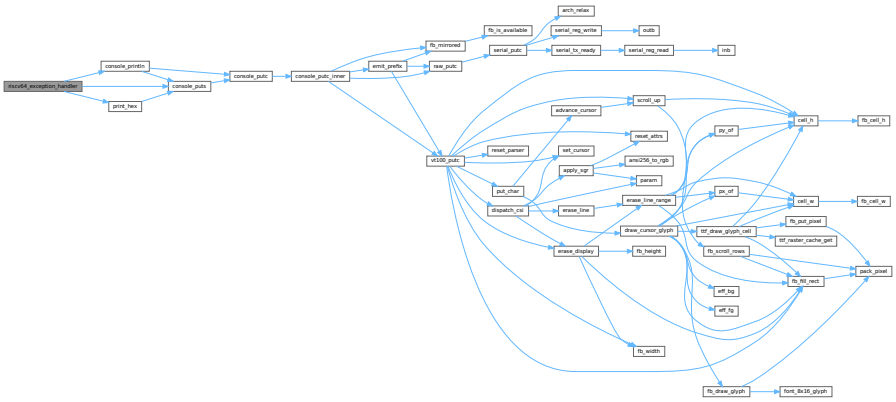
- `scause` — Supervisor Cause Register. Bit 63 distinguishes interrupts (1) from exceptions (0); bits [62:0] encode the cause code (e.g., 0 = instruction address misaligned, 2 = illegal instruction, 5 = load access fault, 13 = store page fault, 8 = ecall from U-mode). See RISC-V Privileged Spec ?4.1.9.
- `sepc` — Supervisor Exception Program Counter. The PC of the instruction that caused the trap (or the return address for interrupt handlers); execution resumes from `sepc` on SRET.
- `stval` — Supervisor Trap Value. For load/store page faults this holds the faulting virtual address; for illegal instruction faults it holds the faulting instruction encoding; zero for other causes.

Prints all three values to the kernel console then enters an infinite `WFI` loop to halt the core permanently. All exceptions that reach this handler are fatal in the current single-hart kernel.

`g_sk_fault_word` is set by the VM FORTH dispatcher and names the word executing at the time of the trap, aiding post-mortem serial log analysis.

References console_println(), console_puts(), and print_hex().

Here is the call graph for this function:

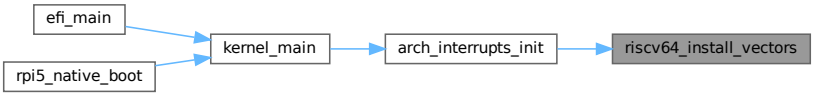


7.113.2.4 riscv64_install_vectors()

```
void riscv64_install_vectors (
            void  )  [extern]
```

Referenced by arch_interrupts_init().

Here is the caller graph for this function:



7.113.2.5 riscv64_interrupt_handler()

```
void riscv64_interrupt_handler (
            uint64_t scause )
```

Dispatch an asynchronous supervisor interrupt.

Called from `riscv64_trap_entry` in `isr.S` when `scause` has bit 63 set. Unlike `riscv64_exception_handler()` this **returns** — the trap entry restores the caller-saved register set and issues `SRET`.

Supervisor timer (cause 5) re-arms the one-shot SBI timer and then accounts the tick. **Re-arm comes first**: the SBI timer fires once per programmed deadline, so any return path that skips the re-arm stops the heartbeat permanently and silently. Ordering it ahead of `heartbeat_tick()` means a fault in the bookkeeping cannot also cost the next tick.

Supervisor external (cause 9): claim/dispatch/complete through the PLIC (substrate, item 4.3.5b). Source 10 (UART) stays unclaimed by anything here – 4.3.5b's own synthetic-interrupt self-test that used it was run once and reverted, not a standing consumer. virtio-input (item 4.3.5c) is the first real per-source dispatch, the same shape as i8042's keyboard branch in amd64's isr_common_handler().

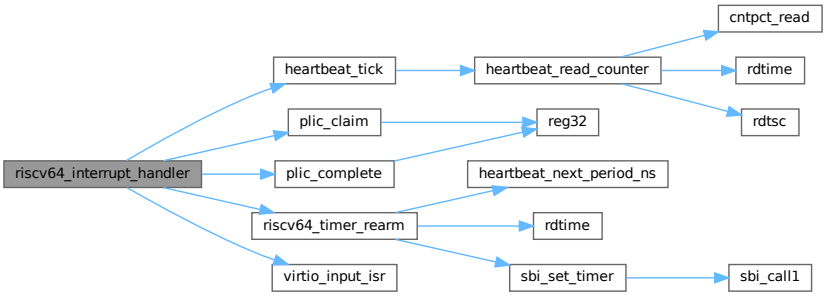
Any other cause is ignored rather than fatal: an unexpected-but-harmless asynchronous interrupt should not take the kernel down.

Parameters

scause	Raw `scause` value, interrupt bit still set.

References g_plic_claim_count, g_virtio_input_plic_source, heartbeat_tick(), plic_claim(), plic_complete(), riscv64_timer_rearm(), SCAUSE_S_EXTERNAL, SCAUSE_S_TIMER, and virtio_input_isr().

Here is the call graph for this function:



7.113.2.6 riscv64_timer_rearm()

```
void riscv64_timer_rearm (
            void  )  [extern]
```

Re-arm the one-shot SBI timer and account the tick.

The SBI timer is one-shot by nature. Servicing a timer interrupt without programming the next deadline leaves the heartbeat stopped permanently, with no error anywhere — the single most likely silent failure of this driver. Every path out of a timer interrupt must reach this function.

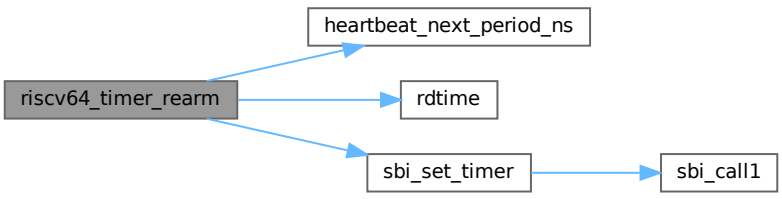
The step is `heartbeat_next_period_ns()` (item 0.8, ?26) converted to `time`-counter ticks via `s_time_hz`, not the fixed `s_timer_period_tsc` used to seed the very first deadline in `apic_timer_start()` – the latter remains `apic_timer_period_tsc()`'s return value for `heartbeat_init()`'s initial `expected_delta`, unchanged.

Called from `riscv64_interrupt_handler()` in `interrupts.c` on `scause` cause 5.

References heartbeat_next_period_ns(), rdtime(), s_next_deadline, s_sbi_time_ok, s_time_hz, s_timer_period_tsc, and sbi_set_timer().

Referenced by riscv64_interrupt_handler().

Here is the call graph for this function:



Here is the caller graph for this function:

7.113.2.7 virtio_input_isr()

```
void virtio_input_isr (
            void  )  [extern]
```

7.113.3 Variable Documentation

7.113.3.1 g_plic_claim_count

```
volatile uint32_t g_plic_claim_count = 0
```

Referenced by riscv64_interrupt_handler().

7.113.3.2 g_sk_fault_word

```
volatile const char* g_sk_fault_word = (void *)0
```

interrupts.c (riscv64) - Trap vector setup and common exception handler

interrupts.c (aarch64) - Exception vector setup and common handler

Referenced by execute_colon_word(), and vm_interpret_word().

7.113.3.3 g_virtio_input_plic_source

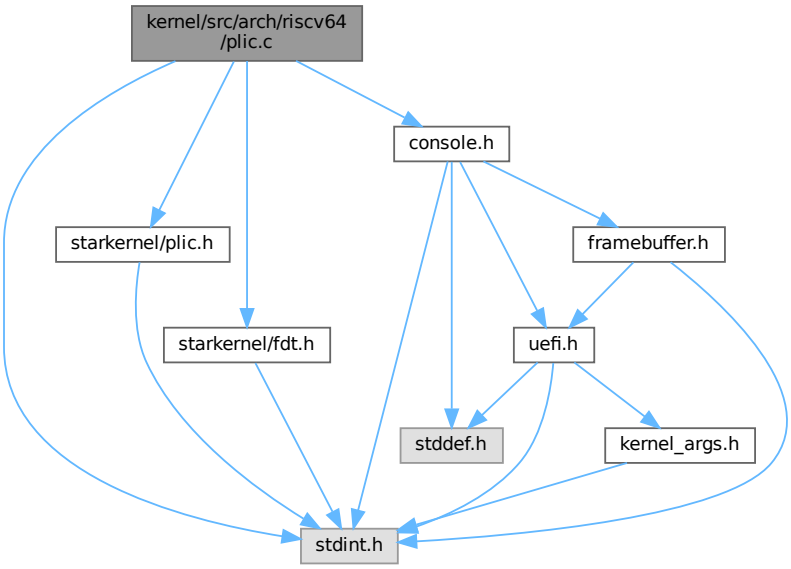
```
uint32_t g_virtio_input_plic_source  [extern]
```

Referenced by riscv64_interrupt_handler().

7.114 kernel/src/arch/riscv64/plic.c File Reference

```
#include <stdint.h>
#include "starkernel/plic.h"
#include "starkernel/fdt.h"
#include "console.h"
```

Include dependency graph for plic.c:



Macros

- #define PLIC_BASE_QEMU_DEFAULT 0x0C000000UL

  plic.c (riscv64) - Platform-Level Interrupt Controller driver
- #define PLIC_CONTEXT_S 1u /* S-mode, hart 0 -- verified, see file header */
- #define PLIC_PRIORITY(irq) (s_plic_base + 4u * (irq))
- #define PLIC_ENABLE(ctx) (s_plic_base + 0x2000u + 0x80u * (ctx))
- #define PLIC_THRESHOLD(ctx) (s_plic_base + 0x200000u + 0x1000u * (ctx))
- #define PLIC_CLAIM(ctx) (s_plic_base + 0x200004u + 0x1000u * (ctx))

Functions

- static volatile uint32_t * reg32 (uintptr_t addr)
- static uint32_t be32_cell (const unsigned char *b)

  Read one big-endian 32-bit cell from a devicetree property blob.
- static uint64_t be64_addr_cell_pair (const unsigned char *b)

  Read one two-cell (64-bit) big-endian address from a `reg` entry.
- static int plic_base_from_dtb (const void *dtb, uintptr_t *base_out)

  Try to discover the real PLIC base address from the DTB.
- int plic_init (const void *dtb)

  plic.h - Platform-Level Interrupt Controller interface (riscv64 only)

- void plic_set_priority (uint32_t irq, uint32_t priority)

  Set a source's interrupt priority (1-7; 0 means "never interrupt").
- void plic_enable (uint32_t irq)

  Enable a source for the S-mode hart-0 context.
- void plic_disable (uint32_t irq)

  Disable a source for the S-mode hart-0 context.
- uint32_t plic_claim (void)

  Claim the highest-priority pending interrupt for the S-mode hart-0 context.
- void plic_complete (uint32_t irq)

  Signal completion of the source previously returned by plic_claim().

Variables

- static uintptr_t s_plic_base = 0x0C000000UL

7.114.1 Macro Definition Documentation

7.114.1.1 PLIC_BASE_QEMU_DEFAULT

```
#define PLIC_BASE_QEMU_DEFAULT 0x0C000000UL
```

plic.c (riscv64) - Platform-Level Interrupt Controller driver

Item 4.3.5b. Register layout (priority/pending/enable/threshold/claim- complete) is the standard SiFive PLIC-1.0.0 layout QEMU implements – architectural, not board-specific, same status as the GICD/GICC register *offsets* item 0.6 cross-checked against the Linux arm-gic header.

PLIC_BASE_QEMU_DEFAULT and PLIC_CONTEXT_S are QEMU-virt-specific and were NOT hardcoded from memory: verified 2026-08-08 via `qemu-system-riscv64 -machine virt,dumpdtb=...`, decoded with a script using this tree's own fdt.c parsing logic (not dtc, which is not installed). The real devicetree shows: soc/plic@c000000: compatible = "sifive,plic-1.0.0","riscv,plic0", reg = <0x0 0xc000000 0x0 0x600000>, riscv,ndev = 95 soc/plic@c000000: interrupts-extended = <&cpu0-intc 11 &cpu0-intc 9> – two contexts for hart 0: context 0 feeds cause 11 (MEIE, M-mode external), context 1 feeds cause 9 (SEIE, S-mode external). Context index is position in this list, confirmed rather than assumed. soc/serial@10000000 (ns16550a): interrupts = <10>, interrupt-parent = &plic – PLIC source 10, matching console.c's existing NS16550_BASE. Confirmed for this exact QEMU 10.2.1 build; not assumed stable across versions, same caveat item 0.6 recorded for its own dumped values.

DTB-discovered base address, as of FABRIC-3.md ?V.3 item 3's fix, 2026-09-05: the node-scoped ("reg" is not globally unique the way "timebase-frequency" is) devicetree lookup this file's own comment previously flagged as "a bigger addition than this item's stated scope" was subsequently built anyway – `fdt_find_node_by_compatible()` + `fdt_find_prop_in_node()` (`fdt.c`/`fdt.h`), added for the aarch64 GIC base fix (FABRIC-3.md ?IV.3 item 7) and reused here unchanged. `plic_base_from_dtb()` below tries `"sifive,plic-1.0.0"` first; on failure (no DTB, e.g. this system's QEMU/UEFI firmware – same `fdt_valid()` failure mode apic.c's GIC discovery already documents) it falls back to the QEMU constant above, unchanged from this function's previous unconditional behaviour. Not yet confirmed against the Milk-V Mars's real DTB – the compatible string is architectural (SiFive PLIC-1.0.0 is the standard binding), not board-specific, but the actual JH7110 `reg` value is unverified until real hardware arrives (2026-09-17).

7.114.1.2 PLIC_CLAIM

```
#define PLIC_CLAIM(
              ctx ) (s_plic_base + 0x200004u + 0x1000u * (ctx))
```

7.114.1.3 PLIC_CONTEXT_S

```
#define PLIC_CONTEXT_S 1u /* S-mode, hart 0 -- verified, see file header */
```

7.114.1.4 PLIC_ENABLE

```
#define PLIC_ENABLE(
            ctx ) (s_plic_base + 0x2000u + 0x80u * (ctx))
```

7.114.1.5 PLIC_PRIORITY

```
#define PLIC_PRIORITY(
            irq ) (s_plic_base + 4u * (irq))
```

7.114.1.6 PLIC_THRESHOLD

```
#define PLIC_THRESHOLD(
            ctx ) (s_plic_base + 0x200000u + 0x1000u * (ctx))
```

7.114.2 Function Documentation

7.114.2.1 be32_cell()

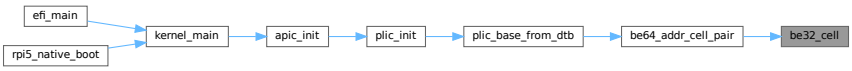
```
static uint32_t be32_cell (
            const unsigned char * b )  [static]
```

Read one big-endian 32-bit cell from a devicetree property blob.

Duplicated from aarch64 apic.c's identical helper rather than shared – same "a few lines is simpler than a new shared dependency" precedent that file's own comment already set for this exact situation.

Referenced by be64_addr_cell_pair().

Here is the caller graph for this function:



7.114.2.2 be64_addr_cell_pair()

```
static uint64_t be64_addr_cell_pair (
            const unsigned char * b )  [static]
```

Read one two-cell (64-bit) big-endian address from a `reg` entry.

QEMU-virt's `plic` node's own dumped `reg = <0x0 0xc000000 0x0 0x600000>` (see file header) is address(2 cells) + size(2 cells) = 4 cells total, same shape as aarch64's `gicv2` node – reads only the two address cells, ignoring the size pair that follows.

References be32_cell().

Referenced by plic_base_from_dtb().

Here is the call graph for this function:

Here is the caller graph for this function:

7.114.2.3 plic_base_from_dtb()

```
static int plic_base_from_dtb (
            const void * dtb,
            uintptr_t * base_out )  [static]
```

Try to discover the real PLIC base address from the DTB.

Looks up the "`sifive,plic-1.0.0`" node and reads its `reg` property's first (address) cell pair. Returns 0 (leaving `base_out` untouched) if `dtb` is not a valid FDT, no matching node exists, or `reg` is shorter than one address-cell pair – the caller keeps the QEMU constant default in that case, exactly as before this function existed.

Parameters

dtb	Candidate devicetree blob (`BootInfo->dtb`); NULL-safe.
base_out	Receives the PLIC base address on success.

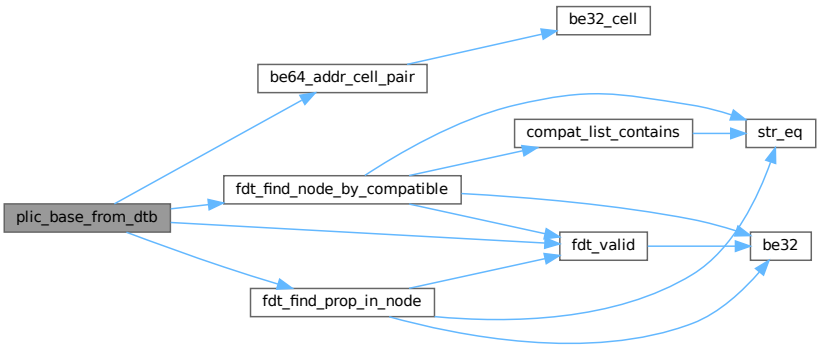
Returns

1 on success, 0 if discovery failed for any reason.

References be64_addr_cell_pair(), fdt_find_node_by_compatible(), fdt_find_prop_in_node(), and fdt_valid().

Referenced by plic_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.114.2.4 plic_claim()

```
uint32_t plic_claim (
            void  )
```

Claim the highest-priority pending interrupt for the S-mode hart-0 context.

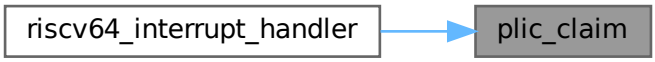
Returns the source ID, or 0 if none is pending.

References PLIC_CLAIM, PLIC_CONTEXT_S, and reg32().

Referenced by riscv64_interrupt_handler().

Here is the call graph for this function:

Here is the caller graph for this function:



7.114.2.5 plic_complete()

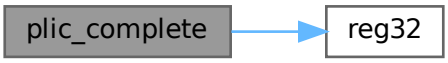
```
void plic_complete (
            uint32_t irq )
```

Signal completion of the source previously returned by plic_claim().

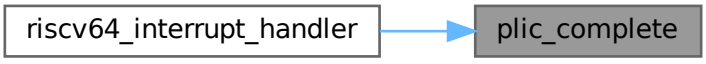
References PLIC_CLAIM, PLIC_CONTEXT_S, and reg32().

Referenced by riscv64_interrupt_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.114.2.6 plic_disable()

```
void plic_disable (
            uint32_t irq )
```

Disable a source for the S-mode hart-0 context.

References PLIC_CONTEXT_S, PLIC_ENABLE, and reg32().

Here is the call graph for this function:

7.114.2.7 plic_enable()

```
void plic_enable (
            uint32_t irq )
```

Enable a source for the S-mode hart-0 context.

References PLIC_CONTEXT_S, PLIC_ENABLE, and reg32().

Here is the call graph for this function:

7.114.2.8 plic_init()

```
int plic_init (
            const void * dtb )
```

plic.h - Platform-Level Interrupt Controller interface (riscv64 only)

Item 4.3.5a (FABRIC-0.md ?27.5): Phase 0 (0.2/0.3) only ever enabled the S-mode *timer* interrupt (sie.STIE). External interrupts (sie.SEIE) were never touched, and the PLIC – the only external-interrupt path on RISC-V, there is no legacy PIC or I/O APIC equivalent – had no driver at all before this item. Pure substrate: this item wires

the mechanism (claim/dispatch/complete) with no permanent source enabled by default; a real consumer (4.3.5b, virtio-keyboard) enables its own source later.

plic_init() takes the boot DTB as of FABRIC-3.md ?V.3 item 3's fix, 2026-09-05: the base address was previously a QEMU-virt-specific hardcoded constant, unconditionally wrong on the Milk-V Mars's real JH7110 PLIC. See plic.c's file header for the discovery mechanism. Map the PLIC and set the S-mode hart-0 context's priority threshold to 0 (maximally permissive – safe because nothing is enabled at any source by default; enabling a source is what actually lets it reach claim()). Also sets sie.SEIE. Does not touch sstatus.SIE – arch_enable_interrupts() still owns that, same as the timer.

Parameters

dtb	Candidate devicetree blob (`BootInfo->dtb`); NULL-safe. When a real `"sifive,plic-1.0.0"` node is found, the PLIC base address is read from its `reg` property; otherwise falls back to the QEMU-virt-machine constant, unchanged from this function's previous unconditional behaviour.

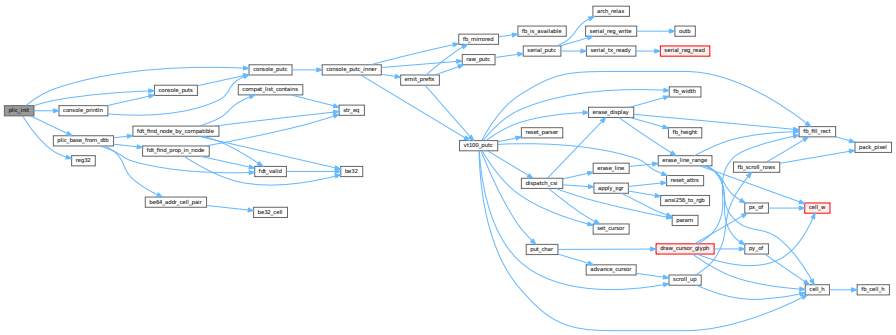
Returns

0 on success.

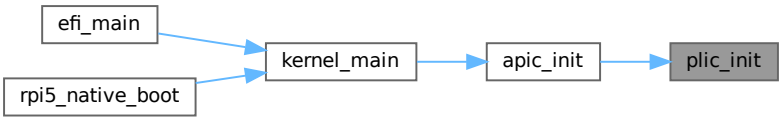
References console_println(), console_putc(), console_puts(), plic_base_from_dtb(), PLIC_CONTEXT_S, PLIC_THRESHOLD, reg32(), and s_plic_base.

Referenced by apic_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.114.2.9 plic_set_priority()

```
void plic_set_priority (
            uint32_t irq,
            uint32_t priority )
```

Set a source's interrupt priority (1-7; 0 means "never interrupt").

References PLIC_PRIORITY, and reg32().

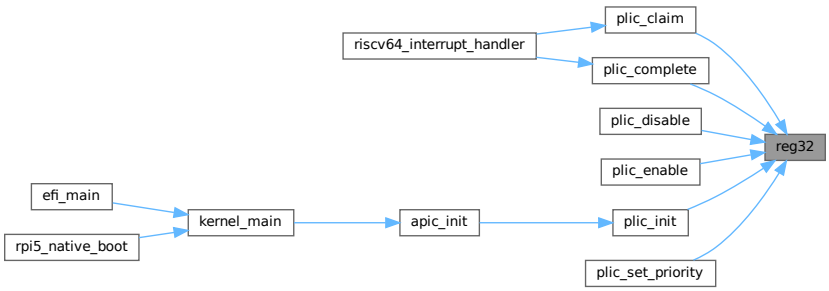
Here is the call graph for this function:

7.114.2.10 reg32()

```
static volatile uint32_t * reg32 (
            uintptr_t addr )  [inline], [static]
```

Referenced by plic_claim(), plic_complete(), plic_disable(), plic_enable(), plic_init(), and plic_set_priority().

Here is the caller graph for this function:



7.114.3 Variable Documentation

7.114.3.1 s_plic_base

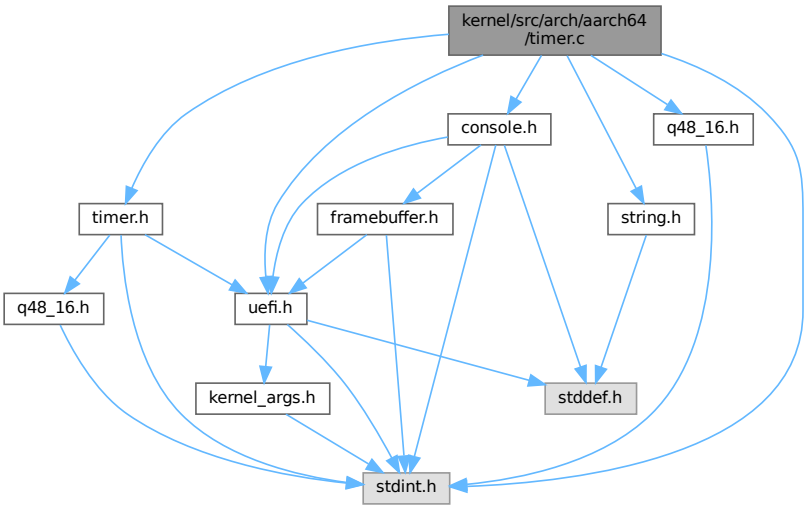
```
uintptr_t s_plic_base = 0x0C000000UL  [static]
```

Referenced by plic_init().

7.115 kernel/src/arch/aarch64/timer.c File Reference

```
#include "timer.h"
#include "console.h"
#include "q48_16.h"
#include "uefi.h"
#include <stdint.h>
#include <string.h>
```

Include dependency graph for timer.c:



Functions

- static int running_under_hypervisor (BootInfo *boot_info)

  timer.c (aarch64) - Timer subsystem using ARM Generic Timer (CNTPCT_EL0)
- static uint64_t cntpct_read (void)

  Read the AArch64 Physical System Counter with ISB serialisation.
- static uint64_t cntfrq_read (void)

  Read the AArch64 Counter Frequency Register (`CNTFRQ_EL0`).
- int timer_init (BootInfo *boot_info)
- uint64_t timer_tsc_hz (void)

  Return the system counter frequency in Hz.
- uint64_t timer_now_ns (void)

  Return a monotonic nanosecond timestamp.
- int timer_check_drift_now (void)

  Check for timer drift (AArch64 stub — always returns 0).
- const timer_calibration_record_t * timer_calibration_record (void)

  Return a pointer to the timer calibration record.
- uint64_t heartbeat_read_counter (void)

  Read the raw counter the aarch64 heartbeat is paced against.

Variables

- static uint64_t s_counter_hz = 0
- static uint64_t s_ns_per_tick = 0
- static uint64_t s_base_count = 0
- static uint64_t s_base_ns = 0
- static timer_calibration_record_t s_cal

7.115.1 Function Documentation

7.115.1.1 cntfrq_read()

```
static uint64_t cntfrq_read (
            void  )  [inline], [static]
```

Read the AArch64 Counter Frequency Register (CNTFRQ_EL0).

CNTFRQ_EL0 is a read-only EL0-accessible register whose value is written by EL3/EL2 firmware at boot to advertise the system counter frequency in Hz. It is the architectural way to discover CNTPCT_EL0's tick rate without out-of-band knowledge of the board.

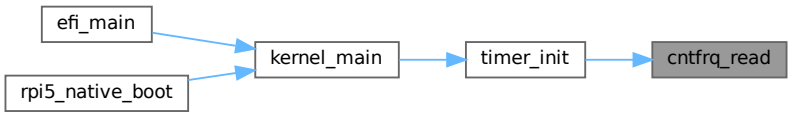
QEMU's SBSA/virt machine models typically set this to 62,500,000 Hz (62.5 MHz) for Cortex-A57 compatibility. Real hardware varies widely (10 MHz on Raspberry Pi 4, 24 MHz on Cortex-A53 dev boards, etc.). timer_init() falls back to 62,500,000 if the register reads zero.

Returns

Counter frequency in Hz as set by firmware; 0 if firmware did not initialise the register (uncommon but possible in bare-metal bring-up without a proper firmware stack).

Referenced by timer_init().

Here is the caller graph for this function:



7.115.1.2 cntpct_read()

```
static uint64_t cntpct_read (
            void  )  [inline], [static]
```

Read the AArch64 Physical System Counter with ISB serialisation.

Issues an `ISB` (Instruction Synchronisation Barrier) before reading `CNTPCT_EL0` to ensure all preceding instructions have completed before the counter value is sampled. This prevents the counter read from being speculated ahead of a preceding store or flag update, which would produce an erroneously early timestamp.

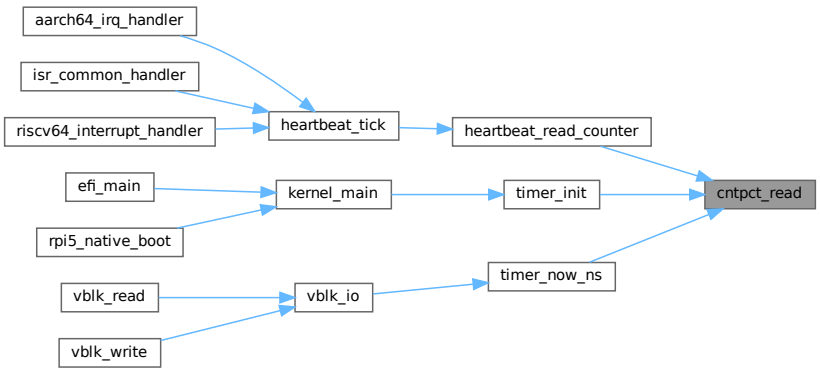
The ISB is heavier than strictly necessary for most timing uses but provides the same ordering guarantee as `RDTSCP` on x86-64. For interval measurements where start and end reads are in the same path, the overhead (1–3 cycles) is negligible.

Returns

Current 64-bit `CNTPCT_EL0` counter value in system-counter ticks.

Referenced by heartbeat_read_counter(), timer_init(), and timer_now_ns().

Here is the caller graph for this function:



7.115.1.3 heartbeat_read_counter()

```
uint64_t heartbeat_read_counter (
            void  )
```

Read the raw counter the aarch64 heartbeat is paced against.

Read the raw hardware counter this architecture's heartbeat is paced against – the same clock `timer_now_ns()` and calibration already use internally (rdtsc on amd64, the `time` CSR on riscv64, CNTPCT_EL0 on aarch64), not a separate/different source.

Item 0.8 (FABRIC-0.md ?25.1): the shared heartbeat.c now owns heartbeat_init()/heartbeat_tick()/heartbeat_service()/heartbeat_ticks()/ heartbeat_trust()/heartbeat_state() and the per-arch `g_heartbeat` state that used to live in this file. This is the one piece that stays per-architecture – the same `CNTPCT_EL0` the rest of this file's calibration already reads via `cntpct_read()`.

Returns

Current `CNTPCT_EL0` value.

7.115.1.4 running_under_hypervisor()

```
static int running_under_hypervisor (
            BootInfo * boot_info )  [static]
```

timer.c (aarch64) - Timer subsystem using ARM Generic Timer (CNTPCT_EL0)

Uses the 64-bit system counter available as CNTPCT_EL0. The counter frequency is read from CNTFRQ_EL0 (set by firmware).

References BootInfo::acpi_table, and memcmp().

Here is the call graph for this function:

7.115.1.5 timer_calibration_record()

```
const timer_calibration_record_t * timer_calibration_record (
            void  )
```

Return a pointer to the timer calibration record.

Returns `&s_cal` which is populated by `timer_init()` with:

- `tsc_hz_mean` = `CNTFRQ_EL0` (or 62.5 MHz fallback)
- `hpet_hz` = 0 (no HPET on AArch64)
- `pit_hz_mean` = 0 (no PIT on AArch64)
- `converged` = 1 (Generic Timer is already calibrated by firmware)
- `vm_mode` = real hypervisor-vs-hardware detection as of 2026-09-04 (ACPI RSDP OEM ID check, running_under_hypervisor() above) – the Generic Timer itself works identically either way, so this field's *timing* meaning here is purely informational, unlike amd64 where vm_mode gates a real calibration-path choice
- `trust` = `TIMER_TRUST_ABSOLUTE`

The record is exposed to `kernel_main()` and `timer.h` consumers for boot-log reporting and for the heartbeat subsystem's trust initialisation.

Returns

Pointer to the module-static calibration record; valid for the lifetime of the kernel.

7.115.1.6 timer_check_drift_now()

```
int timer_check_drift_now (
            void  )
```

Check for timer drift (AArch64 stub — always returns 0).

On x86-64 this function cross-checks the TSC against the HPET or PM Timer to detect frequency drift. The AArch64 Generic Timer is architecturally invariant and synchronised across all cores in the system; it cannot drift against itself. This stub always returns 0 (no drift detected) to satisfy the common `timer_check_drift_now()` call site.

Returns

0 always.

7.115.1.7 timer_init()

```
int timer_init (
            BootInfo * boot_info )
```

7.115.1.8 timer_now_ns()

```
uint64_t timer_now_ns (
            void  )
```

Return a monotonic nanosecond timestamp.

Reads `CNTPCT_EL0`, computes the delta from `s_base_count` (captured at `timer_init()`), and converts to nanoseconds using the Q16.16 fixed-point `s_ns_per_tick`:

```
ns = s_base_ns + ((delta * s_ns_per_tick) » 16)
```

The multiplication uses 64-bit arithmetic; at typical counter frequencies ($\leq$ 100 MHz) and reasonable uptime intervals (days to weeks) there is no overflow risk in the delta-to-ns conversion.

Returns

Monotonic nanosecond count since `timer_init()`, starting at 0.

7.115.1.9 timer_tsc_hz()

```
uint64_t timer_tsc_hz (
            void  )
```

Return the system counter frequency in Hz.

Returns `s_counter_hz` as set from `CNTFRQ_EL0` by `timer_init()`. If `timer_init()` has not been called yet (or firmware reported 0 and the fallback was not applied), returns 62,500,000 Hz as a safe default.

Used by `apic_timer_init()` to compute `s_timer_period_tsc` and by the shim time backend to convert counter ticks to nanoseconds before the full timer subsystem is available.

Returns

System counter frequency in Hz; at least 62,500,000.

7.115.2 Variable Documentation

7.115.2.1 s_base_count

```
uint64_t s_base_count = 0  [static]
```

Referenced by timer_init(), and timer_now_ns().

7.115.2.2 s_base_ns

```
uint64_t s_base_ns = 0  [static]
```

Referenced by timer_init(), and timer_now_ns().

7.115.2.3 s_cal

```
timer_calibration_record_t s_cal  [static]
```

Referenced by timer_calibration_record(), and timer_init().

7.115.2.4 s_counter_hz

```
uint64_t s_counter_hz = 0  [static]
```

Referenced by timer_init(), and timer_tsc_hz().

7.115.2.5 s_ns_per_tick

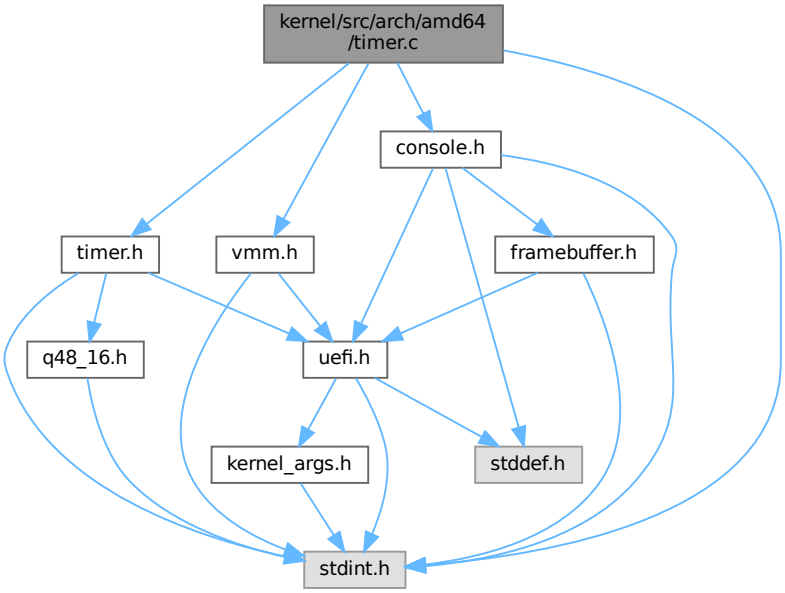
```
uint64_t s_ns_per_tick = 0  [static]
```

Referenced by timer_init(), and timer_now_ns().

7.116 kernel/src/arch/amd64/timer.c File Reference

```
#include <stdint.h>
#include "timer.h"
#include "console.h"
#include "vmm.h"
```

Include dependency graph for timer.c:



Data Structures

- struct __attribute__

  ioapic.c - I/O APIC driver (amd64)

Macros

- #define HPET_GEN_CAP_ID 0x000

  timer.c - Timer policy + TSC calibration (amd64)
- #define HPET_GEN_CONFIG 0x010
- #define HPET_MAIN_COUNTER 0x0F0
- #define HPET_PHYS_BASE 0xFED00000ull
- #define HPET_VIRT_BASE 0xFED00000ull /∗ identity-mapped in this setup ∗/
- #define PIT_CHANNEL0_DATA 0x40
- #define PIT_COMMAND 0x43
- #define PIT_FREQ_HZ 1193182u
- #define PMTIMER_IO_PORT_DEFAULT 0x408
- #define PMTIMER_FREQ_HZ 3579545ull
- #define PMTIMER_WIDTH_BITS 24
- #define PMTIMER_MASK ((1ull << PMTIMER_WIDTH_BITS) - 1ull)
- #define WARMUP_WINDOWS 2

- #define SAMPLE_WINDOWS 8
- #define WINDOW_TICKS_HPET 10000ull /∗ ∼1ms at 10MHz nominal ∗/
- #define WINDOW_TICKS_PIT 4000u /∗ ∼3.3ms nominal ∗/
- #define CV_EPSILON_PPM 1000ull /∗ 0.1% ∗/
- #define CV_STABLE_COUNT 6
- #define DRIFT_EPSILON_PPM 2000ull
- #define TIMER_REQUIRE_PIT 0
- #define TIMER_VM_STRICT_INVARIANT_TSC 0

Functions

- static uint64_t muldiv64 (uint64_t a, uint64_t b, uint64_t c)

  Compute (a × b) / c using 128-bit intermediate precision.
- static void print_dec (uint64_t val)

  Print a `uint64_t` value as decimal digits to the kernel console.
- static uint64_t rdtsc (void)

  Read the x86-64 Time Stamp Counter without serialisation (RDTSC).
- static uint64_t rdtscp_aux (uint32_t ∗aux)

  Read TSC and processor ID with weak serialisation (RDTSCP).
- static void cpuid (uint32_t leaf, uint32_t subleaf, uint32_t ∗a, uint32_t ∗b, uint32_t ∗c, uint32_t ∗d)

  Execute the x86-64 `CPUID` instruction and capture all output registers.
- static void outb (uint16_t port, uint8_t val)

  Write a byte to an x86 I/O port (OUT instruction).
- static uint8_t inb (uint16_t port)

  Read a byte from an x86 I/O port (IN instruction).
- static uint32_t inl (uint16_t port)

  Read a 32-bit dword from an x86 I/O port (INL instruction).
- static uint64_t hpet_read (uint32_t offset)

  Read a 64-bit HPET register at `offset` bytes from the HPET base.
- static void hpet_write (uint32_t offset, uint64_t val)

  Write a 64-bit value to an HPET register at `offset` bytes from base.
- static void timer_fatal (const char ∗msg)

  Emit a fatal timer error message and halt the processor permanently.
- static int cpu_has_invariant_tsc (void)

  Test whether the processor supports Invariant TSC (non-stop TSC).
- static int cpu_has_rdtscp (void)

  Test whether the processor supports the RDTSCP instruction.
- static int running_under_hypervisor (void)

  Detect whether the kernel is running under a hypervisor.
- static int timer_sig4_eq (const uint8_t ∗p, const char ∗s)
- static int timer_sig8_eq (const uint8_t ∗p, const char ∗s)
- static uint16_t fadt_find_pm_tmr_port (void ∗rsdp)

  Look up the ACPI PM_TMR_BLK I/O port from the FADT ("FACP" table).
- static uint32_t pmtimer_read (void)

  Read the raw ACPI PM Timer counter from I/O port `pmtimer_port`.
- static uint64_t pmtimer_delta (uint32_t start, uint32_t cur)

  Compute elapsed PM Timer ticks from `start` to `cur`, handling wrap-around.
- static uint64_t calibrate_tsc_with_pmtimer (void)

  Calibrate TSC frequency using the ACPI PM Timer as reference.
- static void vm_timebase_init_from_pmtimer (void)

  Initialise the VM relative-time base anchored to the PM Timer.

- static void enable_hpet (void)

  Enable the HPET main counter by setting the ENABLE_CNF bit.
- static void sanity_check_hpet_running_quick (void)

  Verify the HPET main counter is advancing (quick sanity check).
- static void pit_program_reload (uint16_t reload)

  Programme the 8253/8254 PIT Channel 0 with a 16-bit reload value.
- static uint16_t pit_read_counter16 (void)

  Read the current 16-bit counter value from PIT Channel 0.
- static int pit_probe_running (void)

  Probe whether the PIT is available and counting.
- static void update_stats (const uint64_t *samples, uint32_t count, uint64_t *mean, uint64_t *stddev)

  Compute mean and standard deviation of an array of `uint64_t` samples.
- static uint64_t calibrate_window_hpet (uint64_t target_ticks)

  Measure TSC frequency over a fixed HPET tick window.
- static uint64_t calibrate_window_pit (uint32_t target_ticks)

  Measure TSC frequency over a fixed PIT tick window.
- static int converge_tsc_bare_metal (uint64_t *locked_hz, uint64_t *pit_mean_out)

  Converge TSC frequency on bare metal using HPET + optional PIT cross-check.
- static uint64_t derive_tsc_hz_from_cpuid (void)

  Derive TSC frequency from CPUID leaves 0x15 and 0x16.
- static void init_timer_vm_mode (void)

  Initialise the timer subsystem in VM (hypervisor-present) mode.
- int timer_init (BootInfo *boot_info)

  Initialise the kernel timer subsystem (M5).
- uint64_t timer_tsc_hz (void)

  Return the locked TSC frequency in Hz.
- static uint64_t timer_now_ns_vm_relative (void)

  Return nanoseconds since boot using the ACPI PM Timer (VM relative mode).
- uint64_t timer_now_ns (void)

  Return the current monotonic time in nanoseconds.
- static int timer_drift_check_bare_metal (void)

  Check TSC drift against the HPET reference on bare metal.
- int timer_check_drift_now (void)

  Perform a runtime TSC drift check (no-op in VM mode).
- const timer_calibration_record_t * timer_calibration_record (void)

  Return a pointer to the current timer calibration record.
- uint64_t heartbeat_read_counter (void)

  Read the raw counter the amd64 heartbeat is paced against.

Variables

- static uint16_t pmtimer_port = 0x408
- static volatile uint64_t * hpet_regs = 0
- static uint64_t hpet_freq_hz = 0
- static uint64_t tsc_hz_locked = 0
- static uint64_t pit_tsc_hz_mean = 0
- static uint32_t windows_used = 0
- static timer_calibration_record_t calib_record = {0}
- static int pit_available = 0
- static uint32_t vm_pm_start = 0
- static uint64_t vm_ns_base = 0

7.116.1 Macro Definition Documentation

7.116.1.1 CV_EPSILON_PPM

```
#define CV_EPSILON_PPM 1000ull /* 0.1% */
```

7.116.1.2 CV_STABLE_COUNT

```
#define CV_STABLE_COUNT 6
```

7.116.1.3 DRIFT_EPSILON_PPM

```
#define DRIFT_EPSILON_PPM 2000ull
```

7.116.1.4 HPET_GEN_CAP_ID

```
#define HPET_GEN_CAP_ID 0x000
```

timer.c - Timer policy + TSC calibration (amd64)

The kernel will always run with the best time source available, but will only make determinism claims when the hardware contract is provably satisfied.

Policy split:

- VM path: avoid HPET MMIO. Prefer invariant TSC for ABSOLUTE time. Derive TSC Hz from CPUID leaves (0x15/0x16) when available; otherwise fall back to ACPI PM Timer (port I/O, stable under QEMU/KVM) to obtain TSC Hz. If invariant TSC is absent, kernel continues in RELATIVE time mode (PM Timer-based), unless strict mode is enabled.
- Bare metal path: calibrate TSC using HPET as primary reference, with optional PIT cross-check. Fail hard on non-convergence.

Failure mode: PANIC/HALT when invariants are not met.

Strict VM policy: -DTIMER_VM_STRICT_INVARIANT_TSC=1 enforces invariant TSC in VM mode (fail-hard).

7.116.1.5 HPET_GEN_CONFIG

```
#define HPET_GEN_CONFIG 0x010
```

7.116.1.6 HPET_MAIN_COUNTER

```
#define HPET_MAIN_COUNTER 0x0F0
```

7.116.1.7 HPET_PHYS_BASE

```
#define HPET_PHYS_BASE 0xFED00000ull
```

7.116.1.8 HPET_VIRT_BASE

```
#define HPET_VIRT_BASE 0xFED00000ull /* identity-mapped in this setup */
```

7.116.1.9 PIT_CHANNEL0_DATA

```
#define PIT_CHANNEL0_DATA 0x40
```

7.116.1.10 PIT_COMMAND

```
#define PIT_COMMAND 0x43
```

7.116.1.11 PIT_FREQ_HZ

```
#define PIT_FREQ_HZ 1193182u
```

7.116.1.12 PMTIMER_FREQ_HZ

```
#define PMTIMER_FREQ_HZ 3579545ull
```

7.116.1.13 PMTIMER_IO_PORT_DEFAULT

```
#define PMTIMER_IO_PORT_DEFAULT 0x408
```

7.116.1.14 PMTIMER_MASK

```
#define PMTIMER_MASK ((1ull << PMTIMER_WIDTH_BITS) - 1ull)
```

7.116.1.15 PMTIMER_WIDTH_BITS

```
#define PMTIMER_WIDTH_BITS 24
```

7.116.1.16 SAMPLE_WINDOWS

```
#define SAMPLE_WINDOWS 8
```

7.116.1.17 TIMER_REQUIRE_PIT

```
#define TIMER_REQUIRE_PIT 0
```

7.116.1.18 TIMER_VM_STRICT_INVARIANT_TSC

```
#define TIMER_VM_STRICT_INVARIANT_TSC 0
```

7.116.1.19 WARMUP_WINDOWS

```
#define WARMUP_WINDOWS 2
```

7.116.1.20 WINDOW_TICKS_HPET

```
#define WINDOW_TICKS_HPET 10000ull /* ~1ms at 10MHz nominal */
```

7.116.1.21 WINDOW_TICKS_PIT

```
#define WINDOW_TICKS_PIT 4000u /* ~3.3ms nominal */
```

7.116.2 Function Documentation

7.116.2.1 calibrate_tsc_with_pmtimer()

```
static uint64_t calibrate_tsc_with_pmtimer (
            void  )  [static]
```

Calibrate TSC frequency using the ACPI PM Timer as reference.

Waits for 1000 PM Timer ticks (~279 ?s at 3.58 MHz), measures the TSC delta over the same interval, then converts to Hz using `muldiv64()`. Includes a 5-million-iteration timeout to avoid hanging under TCG or broken PM Timer implementations.

Used by the VM path when CPUID leaves 0x15/0x16 do not provide a TSC frequency. The PM Timer is port I/O-based and does not cause VM-exits on most hypervisors (unlike HPET MMIO), making it suitable for VM calibration.

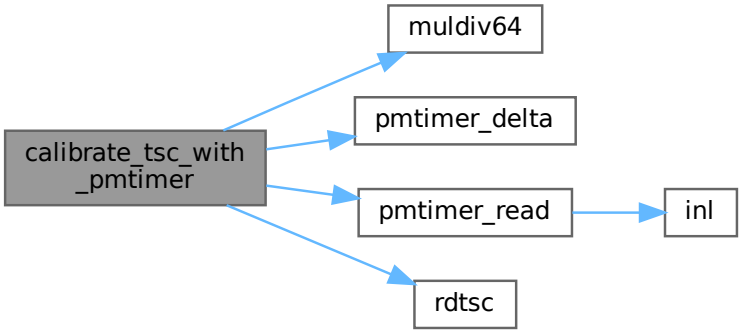
Returns

TSC frequency in Hz, or 0 if calibration timed out or produced an invalid result.

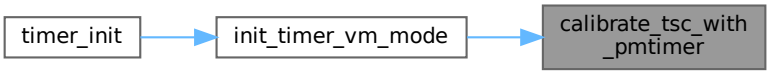
References muldiv64(), pmtimer_delta(), PMTIMER_FREQ_HZ, PMTIMER_MASK, pmtimer_read(), and rdtsc().

Referenced by init_timer_vm_mode().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.2 calibrate_window_hpet()

```
static uint64_t calibrate_window_hpet (
            uint64_t target_ticks )  [static]
```

Measure TSC frequency over a fixed HPET tick window.

Spins until the HPET main counter advances by `target_ticks` ticks, measuring the TSC delta over the same interval. Converts the result to Hz via `muldiv64()`. Returns 0 if the elapsed nanoseconds are zero (which indicates an invalid HPET frequency or overflow).

Used exclusively on the bare-metal path; the tight spin loop would cause excessive VM-exits if called under a hypervisor.

Parameters

target_ticks	HPET ticks to wait (~1 ms at 10 MHz HPET).

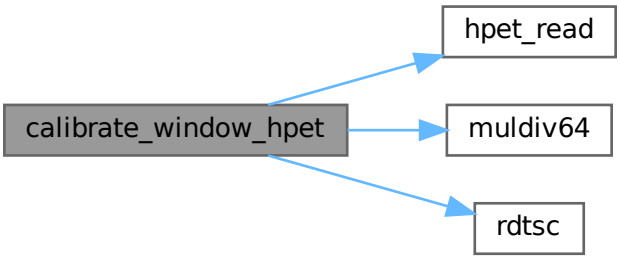
Returns

Estimated TSC frequency in Hz, or 0 on error.

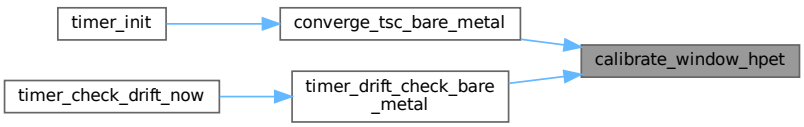
References hpet_freq_hz, HPET_MAIN_COUNTER, hpet_read(), muldiv64(), and rdtsc().

Referenced by converge_tsc_bare_metal(), and timer_drift_check_bare_metal().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.3 calibrate_window_pit()

```
static uint64_t calibrate_window_pit (
            uint32_t target_ticks )  [static]
```

Measure TSC frequency over a fixed PIT tick window.

Programs Channel 0 with reload 0xFFFF and polls until `target_ticks` PIT ticks have elapsed (up to 200,000 poll iterations). Returns the TSC Hz measured over the interval, or 0 if `pit_available` is false or the poll timed out.

Parameters

target_ticks	Number of PIT ticks to measure over (~3.3 ms at 1.19 MHz).

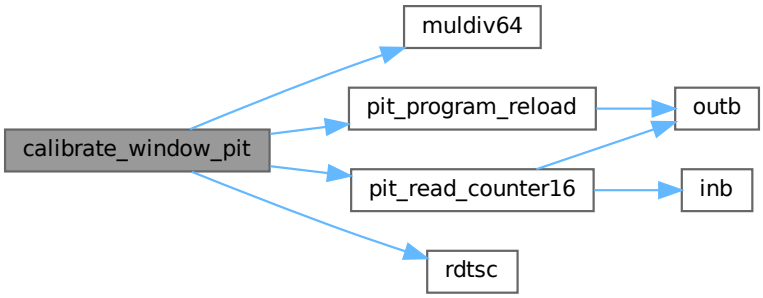
Returns

Estimated TSC frequency in Hz, or 0 if PIT is unavailable or timed out.

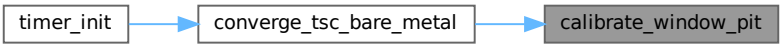
References muldiv64(), pit_available, PIT_FREQ_HZ, pit_program_reload(), pit_read_counter16(), and rdtsc().

Referenced by converge_tsc_bare_metal().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.4 converge_tsc_bare_metal()

```
static int converge_tsc_bare_metal (
            uint64_t * locked_hz,
            uint64_t * pit_mean_out )  [static]
```

Converge TSC frequency on bare metal using HPET + optional PIT cross-check.

Runs WARMUP_WINDOWS (2) warm-up windows then up to SAMPLE_WINDOWS (8) measurement windows, alternating HPET and PIT samples. Declares convergence when CV_STABLE_COUNT (6) consecutive windows all have CV below CV_EPSILON_PPM (1000 ppm). If both HPET and PIT are active the inter-source drift must also be < CV_EPSILON_PPM before the stable counter increments.

Fills calib_record on success. Calls timer_fatal() — never returns — on any hard error (HPET window failure, PIT required but unavailable).

Parameters

locked_hz	Output: converged TSC frequency in Hz.
pit_mean_out	Output: mean PIT-derived TSC Hz (0 if PIT disabled).

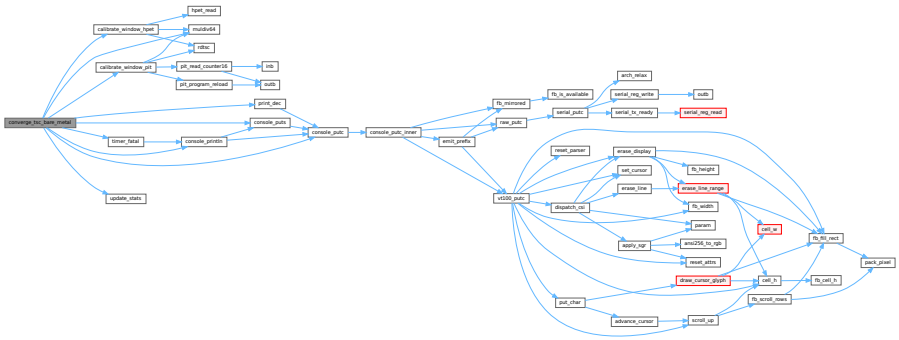
Returns

0 on successful convergence, -1 if the sample limit was exhausted.

References calib_record, calibrate_window_hpet(), calibrate_window_pit(), console_println(), console_putc(), console_puts(), timer_calibration_record::converged, CV_EPSILON_PPM, timer_calibration_record::cv_hpet_ppm, timer_calibration_record::cv_pit_ppm, CV_STABLE_COUNT, timer_calibration_record::diff_ppm, hpet_freq_hz, timer_calibration_record::hpet_hz, muldiv64(), pit_available, timer_calibration_record::pit_hz_mean, print_dec(), SAMPLE_WINDOWS, timer_fatal(), TIMER_REQUIRE_PIT, TIMER_TRUST_ABSOLUTE, timer_calibration_record::trust, timer_calibration_record::tsc_hz_mean, update_stats(), timer_calibration_record::vm_mode, WARMUP_WINDOWS, WINDOW_TICKS_HPET, WINDOW_TICKS_PIT, timer_calibration_record::windows_used, and windows_used.

Referenced by timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.116.2.5 cpu_has_invariant_tsc()

```
static int cpu_has_invariant_tsc (
            void  )  [static]
```

Test whether the processor supports Invariant TSC (non-stop TSC).

Reads `CPUID.80000007H`:EDX[8]. Invariant TSC means the TSC counts at a constant rate regardless of CPU frequency scaling or sleep states — a requirement for using RDTSC as a reliable nanosecond source.

Returns

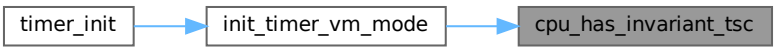
1 if invariant TSC is supported, 0 otherwise.

References cpuid().

Referenced by init_timer_vm_mode().

Here is the call graph for this function:

Here is the caller graph for this function:



7.116.2.6 cpu_has_rdtscp()

```
static int cpu_has_rdtscp (
            void  )  [static]
```

Test whether the processor supports the RDTSCP instruction.

Reads `CPUID.80000001H`:EDX[27]. RDTSCP provides weak serialisation and simultaneously reads both the TSC and the `IA32_TSC_AUX` MSR (which the OS programs to hold a per-CPU/socket identifier). Used in `timer_now_ns()` to prefer RDTSCP over the non-serialising RDTSC when available.

Returns

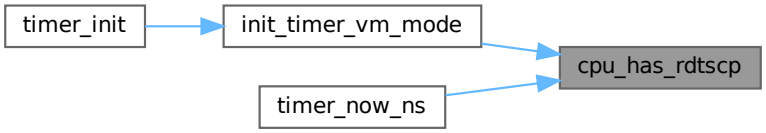
1 if RDTSCP is supported, 0 otherwise.

References cpuid().

Referenced by init_timer_vm_mode(), and timer_now_ns().

Here is the call graph for this function:

Here is the caller graph for this function:



7.116.2.7 cpuid()

```
static void cpuid (
            uint32_t leaf,
            uint32_t subleaf,
            uint32_t * a,
            uint32_t * b,
            uint32_t * c,
            uint32_t * d )  [inline], [static]
```

Execute the x86-64 `CPUID` instruction and capture all output registers.

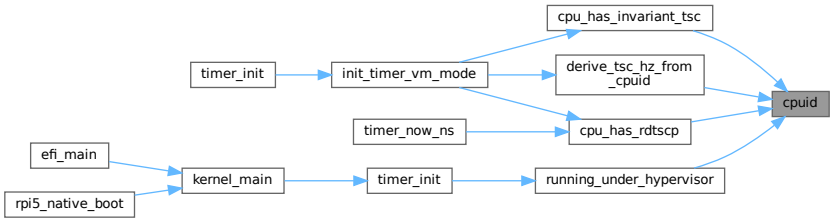
Passes `leaf` in `EAX` and `subleaf` in `ECX` before executing `CPUID`. Any of the output pointers may be `NULL`; the corresponding register value is then discarded.

Parameters

leaf	CPUID function leaf (EAX input).
subleaf	CPUID sub-leaf (ECX input, for structured extended leaves).
a	If non-NULL, receives `EAX` on return.
b	If non-NULL, receives `EBX` on return.
c	If non-NULL, receives `ECX` on return.
d	If non-NULL, receives `EDX` on return.

Referenced by cpu_has_invariant_tsc(), cpu_has_rdtscp(), derive_tsc_hz_from_cpuid(), and running_under_hypervisor().

Here is the caller graph for this function:



7.116.2.8 derive_tsc_hz_from_cpuid()

```
static uint64_t derive_tsc_hz_from_cpuid (
            void  )  [static]
```

Derive TSC frequency from CPUID leaves 0x15 and 0x16.

Checks leaf 0x15 (TSC/Crystal Ratio):

- If EAX, EBX, and ECX are all non-zero: $hz = ECX * EBX / EAX$.

Falls back to leaf 0x16 (base frequency in MHz) if leaf 0x15 gives 0.

Both leaves are available on Broadwell and later Intel CPUs; older CPUs and AMD processors may return 0. Under QEMU/KVM, leaf 0x15 is usually populated; under QEMU/TCG it often is not.

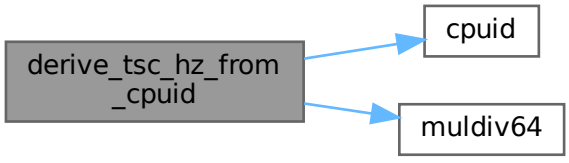
Returns

TSC frequency in Hz if derivable, 0 otherwise.

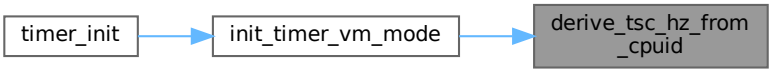
References cpuid(), and muldiv64().

Referenced by init_timer_vm_mode().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.9 enable_hpet()

```
static void enable_hpet (
            void  )  [static]
```

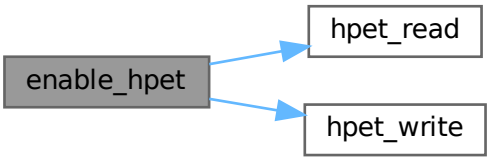
Enable the HPET main counter by setting the ENABLE_CNF bit.

Reads the `HPET_GEN_CONFIG` register, sets bit 0 (ENABLE_CNF), and writes it back. Must be called before the main counter is used for calibration. The HPET must be identity-mapped at `HPET_VIRT_BASE` before this function is called.

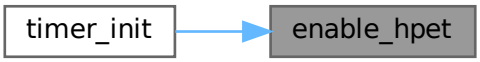
References HPET_GEN_CONFIG, hpet_read(), and hpet_write().

Referenced by timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.10 fadt_find_pm_tmr_port()

```
static uint16_t fadt_find_pm_tmr_port (
            void * rsdp )  [static]
```

Look up the ACPI PM_TMR_BLK I/O port from the FADT ("FACP" table).

Walks RSDP -> XSDT -> FADT exactly like pci.c's MCFG lookup. Prefers the ACPI 2.0+ X_PM_TMR_BLK Generic Address Structure (offset 208) when it names a valid SystemIO address; falls back to the legacy 32-bit PM_TMR_↩ BLK field (offset 76) otherwise. This port is machine-type dependent – QEMU's i440fx puts it at 0x408, Q35's ICH9 LPC at 0x608 – so it must be discovered rather than assumed.

Parameters

rsdp	BootInfo->acpi_table pointer from the UEFI configuration table (ACPI 2.0 GUID), or NULL.

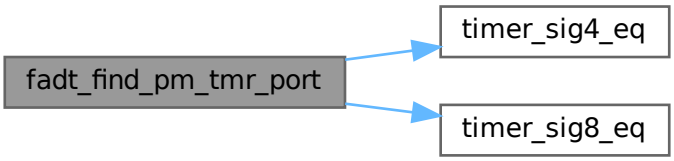
Returns

The discovered I/O port, or 0 if the FADT/XSDT/RSDP could not be parsed (caller should keep its hardcoded fallback in that case).

References timer_sig4_eq(), timer_sig8_eq(), and __attribute__::xsdt_address.

Referenced by timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.116.2.11 heartbeat_read_counter()

```
uint64_t heartbeat_read_counter (
            void  )
```

Read the raw counter the amd64 heartbeat is paced against.

Read the raw hardware counter this architecture's heartbeat is paced against – the same clock `timer_now_ns()` and calibration already use internally (rdtsc on amd64, the `time` CSR on riscv64, CNTPCT_EL0 on aarch64), not a separate/different source.

Item 0.8 (FABRIC-0.md ?25.1): the shared heartbeat.c owns heartbeat_init()/heartbeat_tick()/heartbeat_↩ service()/heartbeat_ticks()/ heartbeat_trust()/heartbeat_state() and the variance/trust math that used to live in this file. This is the one piece that stays per-architecture – the same `rdtsc()` the rest of this file's TSC calibration already uses, not a separate/different source.

Returns

Current TSC value.

References rdtsc().

Here is the call graph for this function:

7.116.2.12 hpet_read()

```
static uint64_t hpet_read (
            uint32_t offset )  [inline], [static]
```

Read a 64-bit HPET register at `offset` bytes from the HPET base.

Accesses `hpet_regs`[] (the identity-mapped MMIO region) using a naturally-aligned 64-bit read. `offset` must be a multiple of 8.

Parameters

offset	Byte offset of the target register (e.g., `HPET_MAIN_COUNTER` = 0x0F0).

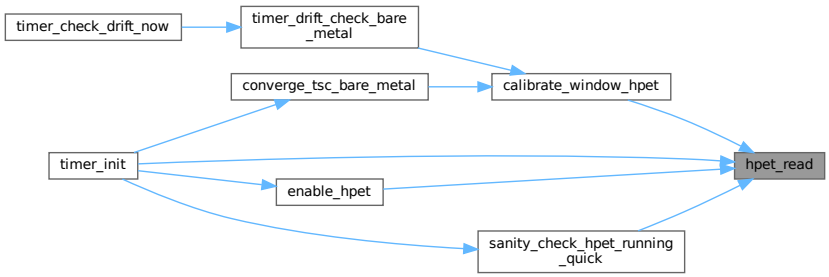
Returns

64-bit register value.

References hpet_regs.

Referenced by calibrate_window_hpet(), enable_hpet(), sanity_check_hpet_running_quick(), and timer_init().

Here is the caller graph for this function:



7.116.2.13 hpet_write()

```
static void hpet_write (
            uint32_t offset,
            uint64_t val )  [inline], [static]
```

Write a 64-bit value to an HPET register at `offset` bytes from base.

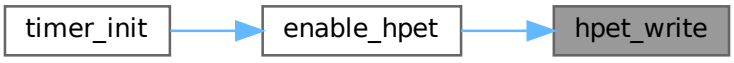
Parameters

offset	Byte offset of the target register (must be a multiple of 8).
val	64-bit value to write.

References hpet_regs.

Referenced by enable_hpet().

Here is the caller graph for this function:



7.116.2.14 inb()

```
static uint8_t inb (
            uint16_t port )  [inline], [static]
```

Read a byte from an x86 I/O port (IN instruction).

Parameters

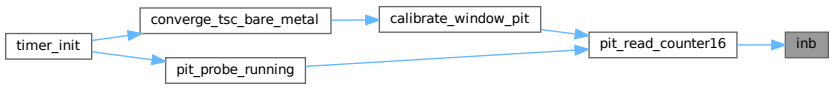
port	I/O port address.

Returns

Byte value read from the port.

Referenced by pit_read_counter16().

Here is the caller graph for this function:



7.116.2.15 init_timer_vm_mode()

```
static void init_timer_vm_mode (
            void  )  [static]
```

Initialise the timer subsystem in VM (hypervisor-present) mode.

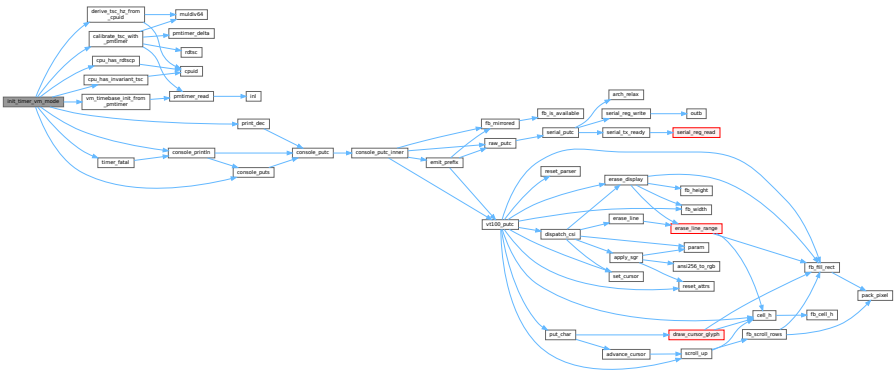
Skips HPET MMIO entirely (VM-exits would corrupt timing measurements). Instead:

1. Checks for invariant TSC. If absent and `TIMER_VM_STRICT_INVARIANT_TSC` is set, calls `timer_fatal()`. Otherwise logs a warning and uses `TIMER_TRUST_RELATIVE` mode.
2. Attempts to derive TSC Hz from CPUID leaves 0x15/0x16.
3. Falls back to `calibrate_tsc_with_pmtimer()` if CPUID yields 0.
4. If TSC Hz is still 0, logs a warning and forces `TIMER_TRUST_RELATIVE` (PM Timer only; no TSC→ns conversion).
5. Calls `vm_timebase_init_from_pmtimer()` to anchor relative time.
6. Fills `calib_record` with `vm_mode` = 1.

References calib_record, calibrate_tsc_with_pmtimer(), console_println(), console_puts(), timer_calibration_record::converged, cpu_has_invariant_tsc(), cpu_has_rdtscp(), timer_calibration_record::cv_hpet_ppm, timer_calibration_record::cv_pit_ppm, derive_tsc_hz_from_cpuid(), timer_calibration_record::diff_ppm, timer_calibration_record::hpet_hz, timer_calibration_record::pit_hz_r print_dec(), timer_fatal(), TIMER_TRUST_ABSOLUTE, TIMER_TRUST_RELATIVE, TIMER_VM_STRICT_INVARIANT_TSC, timer_calibration_record::trust, tsc_hz_locked, timer_calibration_record::tsc_hz_mean, timer_calibration_record::vm_mode, vm_timebase_init_from_pmtimer(), and timer_calibration_record::windows_used.

Referenced by timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.116.2.16 inl()

```
static uint32_t inl (
            uint16_t port )  [inline], [static]
```

Read a 32-bit dword from an x86 I/O port (INL instruction).

Used to read the ACPI PM Timer register which is a 24-bit counter accessed as a 32-bit I/O port.

Parameters

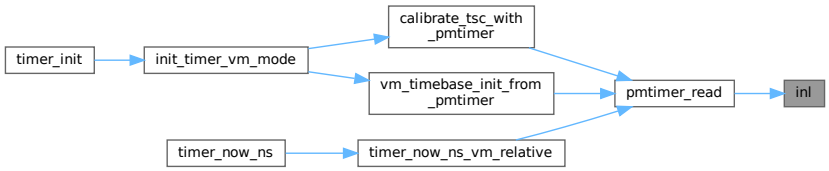
port	I/O port address.

Returns

32-bit value read from the port.

Referenced by pmtimer_read().

Here is the caller graph for this function:



7.116.2.17 muldiv64()

```
static uint64_t muldiv64 (
            uint64_t a,
            uint64_t b,
            uint64_t c )  [inline], [static]
```

Compute (a × b) / c using 128-bit intermediate precision.

Uses the x86-64 `MUL` instruction to produce a 128-bit product in `RDX:RAX`, then `DIV` to produce a 64-bit quotient. This avoids the overflow that would occur with a 64-bit-only intermediate when `a` and `b` are both large (e.g., nanosecond conversion of multi-GHz tick counts). No libgcc dependency — pure inline assembly.

FABRIC-0.md item 4.5d, 2026-08-11: this asm previously declared `RDX` as a plain output (`"=d"(hi)`), which tells GCC only "I want to read RDX's value after this block" — nothing told it that `mulq` writes RDX *before* `divq` needs to read a *different* value (the divisor `c`) out of it. Nothing stopped the register allocator from placing `c` itself in RDX, which `mulq` then overwrote with the product's high 64 bits before `divq` ever read it. At `-O0` this happened not to manifest; at `-O2`, GCC's more aggressive allocation actually did it, and for a product like `1000 * 1000000000` (fits entirely in the low 64 bits, so mulq's high-word output is 0), that corrupted the intended divisor to 0 – a genuine #DE (divide error) confirmed via QEMU `-d int` tracing, cascading to a double fault and then a triple fault (silently exited by `-no-reboot`, which is why this looked like an infinite hang rather than a crash at first). A `unsigned __int128` rewrite was tried first but needs libgcc's `__udivti3` for the general 128÷64 DIVISION case, undefined in this freestanding, `-nostdlib` build — not viable for that reason specifically. (Confirmed narrower during the Ed25519 crypto work, 2026-08-22: `__int128` multiply/add/shift-by-constant DO compile cleanly with zero undefined symbols on all three target toolchains — it's only division, needing a multi-instruction libgcc routine the compiler won't inline, that's unavailable here. Don't read this comment as "avoid __↩ int128 entirely" — src/starkernel/crypto/fe25519.c uses it safely for exactly that reason.) Fixed instead the way the Linux kernel's own `mul_u64_u64_div_u64` does it: declare `rdx` as a pure *clobber*, not an output. A clobber tells GCC the register is used internally by the whole asm block and must never be allocated to any input or output operand for its duration — exactly the guarantee the previous constraint list was missing.

Undefined behaviour if `a` × b / c overflows `uint64_t` (i.e., the true quotient exceeds `UINT64_MAX`). Callers must ensure this cannot happen with their inputs.

Parameters

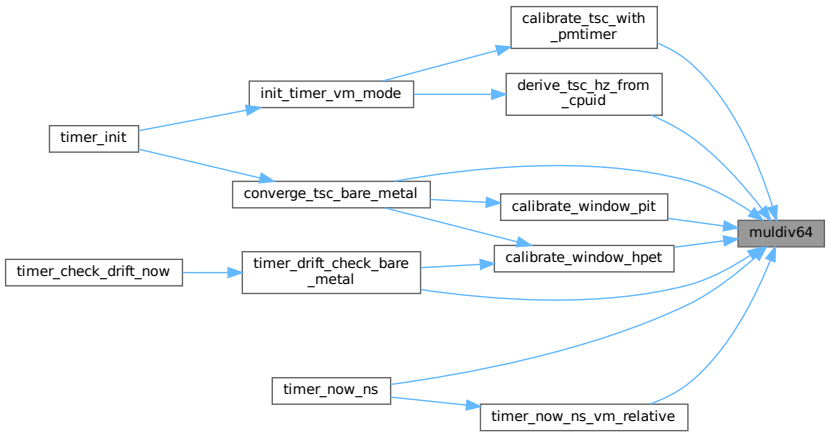
a	Multiplicand.
b	Multiplier.
c	Divisor (must be non-zero; division by zero causes a #DE fault).

Returns

Exact quotient (a ∗ b) / c, truncated toward zero.

Referenced by calibrate_tsc_with_pmtimer(), calibrate_window_hpet(), calibrate_window_pit(), converge_tsc_bare_metal(), derive_tsc_hz_from_cpuid(), timer_drift_check_bare_metal(), timer_now_ns(), and timer_now_ns_vm_relative().

Here is the caller graph for this function:



7.116.2.18 outb()

```
static void outb (
            uint16_t port,
            uint8_t val )  [inline], [static]
```

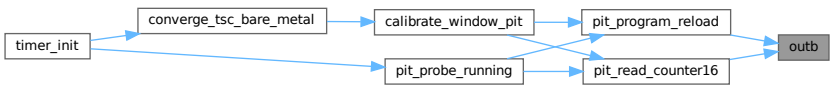
Write a byte to an x86 I/O port (OUT instruction).

Parameters

port	I/O port address.
val	Byte value to write.

Referenced by pit_program_reload(), and pit_read_counter16().

Here is the caller graph for this function:



7.116.2.19 pit_probe_running()

```
static int pit_probe_running (
            void  )  [static]
```

Probe whether the PIT is available and counting.

Programs the PIT with reload value 0xFFFF and polls up to 20,000 times for a counter change. Returns 1 if the counter advances (PIT running), 0 if it never changes (PIT absent, frozen, or emulation disabled). Used on the bare-metal path to decide whether to enable the PIT cross-check for TSC convergence.

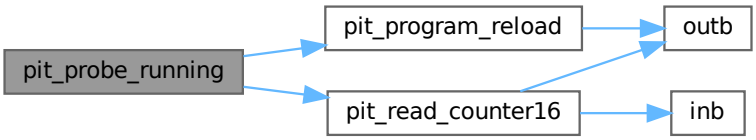
Returns

1 if the PIT is running, 0 if it is unavailable.

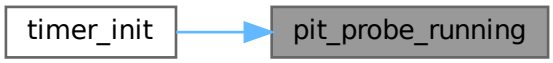
References pit_program_reload(), and pit_read_counter16().

Referenced by timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.20 pit_program_reload()

```
static void pit_program_reload (
            uint16_t reload )  [static]
```

Programme the 8253/8254 PIT Channel 0 with a 16-bit reload value.

Issues the mode command byte (0x34 = channel 0, lo/hi access, mode 2 rate generator) followed by the low and high bytes of `reload` to the PIT data port. After this call the PIT downcounts at `PIT_FREQ_HZ` (1193182 Hz) and reloads automatically.

Parameters

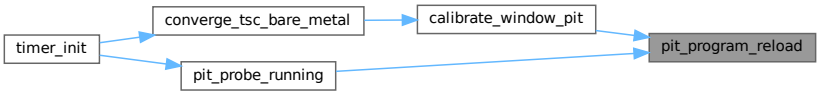
reload	16-bit initial counter value (higher = longer period).

References outb(), PIT_CHANNEL0_DATA, and PIT_COMMAND.

Referenced by calibrate_window_pit(), and pit_probe_running().

Here is the call graph for this function:

Here is the caller graph for this function:



7.116.2.21 pit_read_counter16()

```
static uint16_t pit_read_counter16 (
            void  )  [static]
```

Read the current 16-bit counter value from PIT Channel 0.

Sends a latch command (0x00) to the PIT command port to atomically snap the current count, then reads the low and high bytes from the Channel 0 data port in two `INB` instructions. Returns the counter as a 16-bit value; note the PIT counter counts down from the reload value.

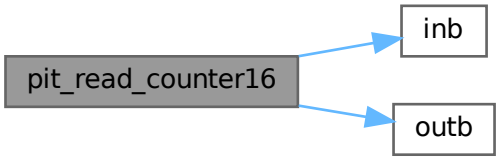
Returns

Current PIT Channel 0 counter value (decreasing, wraps at 0).

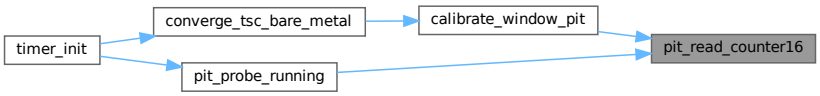
References inb(), outb(), PIT_CHANNEL0_DATA, and PIT_COMMAND.

Referenced by calibrate_window_pit(), and pit_probe_running().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.22 pmtimer_delta()

```
static uint64_t pmtimer_delta (
            uint32_t start,
            uint32_t cur )  [static]
```

Compute elapsed PM Timer ticks from `start` to `cur`, handling wrap-around.

Masks both values to `PMTIMER_WIDTH_BITS` (24 bits) before comparison. When the counter wraps — i.e., `cur` < `start` after masking — the function returns the distance through the wrap: (MASK - start + 1 + cur).

Parameters

start	PM Timer value at the beginning of the interval (unmasked).
cur	PM Timer value at the end of the interval (unmasked).

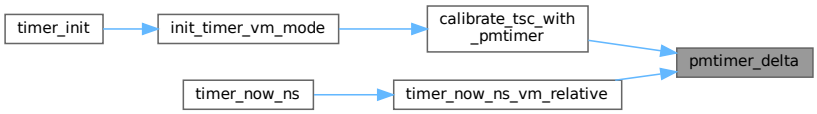
Returns

Number of PM Timer ticks elapsed, accounting for a single wrap.

References PMTIMER_MASK.

Referenced by calibrate_tsc_with_pmtimer(), and timer_now_ns_vm_relative().

Here is the caller graph for this function:



7.116.2.23 pmtimer_read()

```
static uint32_t pmtimer_read (
            void  )  [inline], [static]
```

Read the raw ACPI PM Timer counter from I/O port `pmtimer_port`.

The PM Timer runs at 3.579545 MHz and is a 24-bit counter (bits [23:0]) in most systems. The returned 32-bit value may contain non-zero bits above bit 23 on 32-bit PM Timer implementations; callers must mask with `PMTIMER_MASK` before use.

Returns

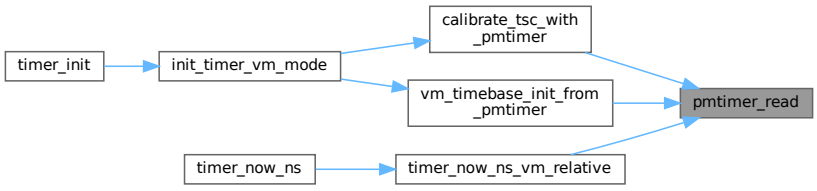
Raw 32-bit PM Timer register value.

References inl(), and pmtimer_port.

Referenced by calibrate_tsc_with_pmtimer(), timer_now_ns_vm_relative(), and vm_timebase_init_from_pmtimer().

Here is the call graph for this function:

Here is the caller graph for this function:



7.116.2.24 print_dec()

```
static void print_dec (
            uint64_t val )  [static]
```

Print a `uint64_t` value as decimal digits to the kernel console.

Uses a local reverse-digit buffer and `console_putc()`; no libc or heap allocation required. Prints `"0"` for a zero value. Used throughout the timer subsystem to emit calibration results without depending on `printf()`.

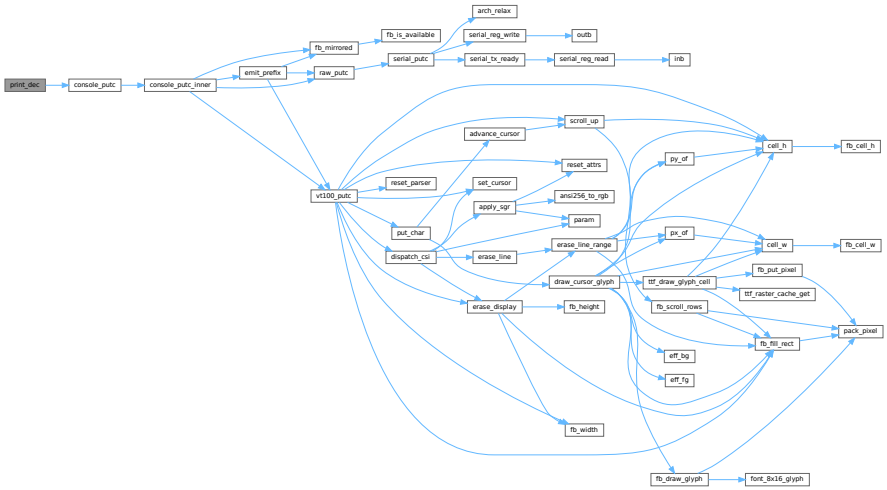
Parameters

val	Value to print in decimal.

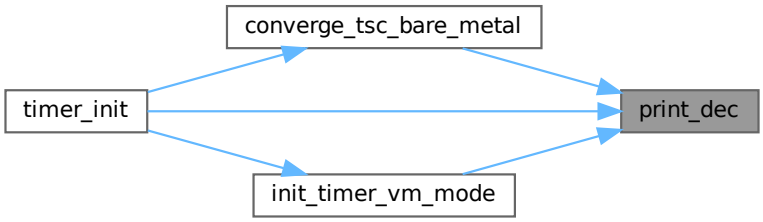
References console_putc().

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), and timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.25 rdtsc()

```
static uint64_t rdtsc (
            void  )  [inline], [static]
```

Read the x86-64 Time Stamp Counter without serialisation (RDTSC).

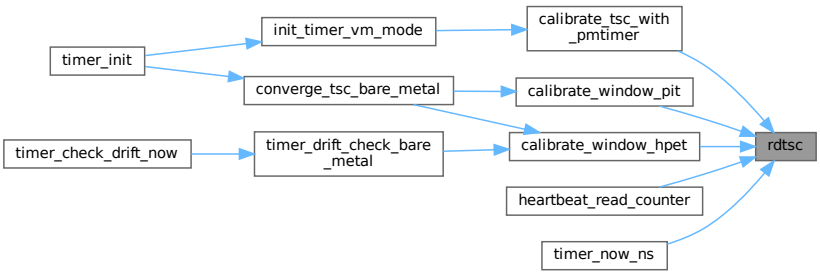
Executes the `RDTSC` instruction and combines `EDX:EAX` into a 64-bit result. May be reordered relative to surrounding instructions by the out-of-order execution unit; use `rdtscp_aux()` where ordering matters.

Returns

Current TSC value in CPU cycles.

Referenced by calibrate_tsc_with_pmtimer(), calibrate_window_hpet(), calibrate_window_pit(), heartbeat_read_counter(), and timer_now_ns().

Here is the caller graph for this function:



7.116.2.26 rdtscp_aux()

```
static uint64_t rdtscp_aux (
            uint32_t * aux )  [inline], [static]
```

Read TSC and processor ID with weak serialisation (RDTSCP).

`RDTSCP` is weakly serialising: all prior loads are complete before the read, but it does not wait for prior stores. Optionally captures the `IA32_TSC_AUX` MSR value (processor ID / NUMA node) in `aux`. Falls back to `rdtsc()` at call sites that check `cpu_has_rdtscp()` first.

Parameters

aux	If non-NULL, receives the `ECX` value from `RDTSCP` (processor/socket ID programmed by the OS or hypervisor).

Returns

Current TSC value in CPU cycles.

Referenced by timer_now_ns().

Here is the caller graph for this function:

7.116.2.27 running_under_hypervisor()

```
static int running_under_hypervisor (
            void  )  [static]
```

Detect whether the kernel is running under a hypervisor.

Reads `CPUID.1`:ECX[31] (Hypervisor Present bit). When set, the kernel switches to VM mode: HPET MMIO calibration is disabled (VM-exits would poison timing), and TSC frequency is derived from CPUID leaves 0x15/0x16 or the ACPI PM Timer instead.

Returns

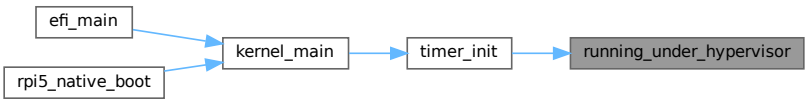
1 if a hypervisor is present (including QEMU/TCG), 0 on bare metal.

References cpuid().

Referenced by timer_init().

Here is the call graph for this function:

Here is the caller graph for this function:



7.116.2.28 sanity_check_hpet_running_quick()

```
static void sanity_check_hpet_running_quick (
            void ) [static]
```

Verify the HPET main counter is advancing (quick sanity check).

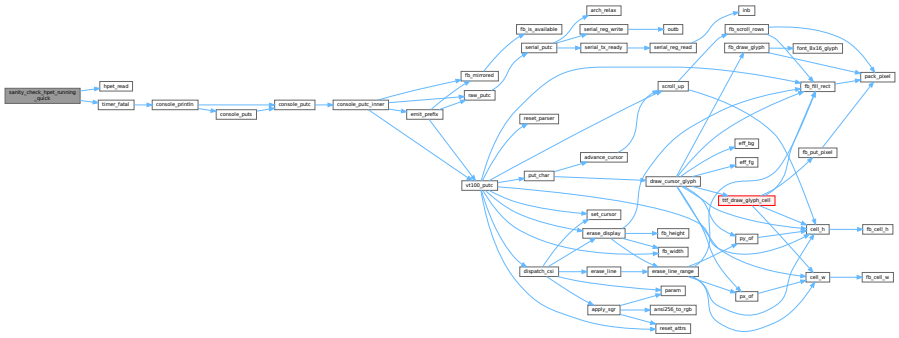
Reads the HPET_MAIN_COUNTER repeatedly up to 200,000 times. If the counter value changes at least once, the check passes silently. If the counter never changes — indicating a hardware fault, frozen HPET, or mapping error — calls timer_fatal() which halts the processor.

Only used on the bare-metal path; skipped in VM mode.

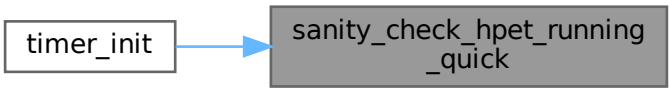
References HPET_MAIN_COUNTER, hpet_read(), and timer_fatal().

Referenced by timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.29 timer_calibration_record()

```
const timer_calibration_record_t * timer_calibration_record (
            void )
```

Return a pointer to the current timer calibration record.

Return a pointer to the timer calibration record.

Provides read-only access to the calib_record populated by timer_init(). Callers can inspect fields such as trust, tsc_hz_mean, cv_hpet_ppm, and converged for diagnostic output or to determine whether absolute timing guarantees are available.

Returns

Pointer to the static timer_calibration_record_t (never NULL).

References calib_record.

7.116.2.30 timer_check_drift_now()

```
int timer_check_drift_now (
            void  )
```

Perform a runtime TSC drift check (no-op in VM mode).

Check for timer drift (AArch64 stub — always returns 0).

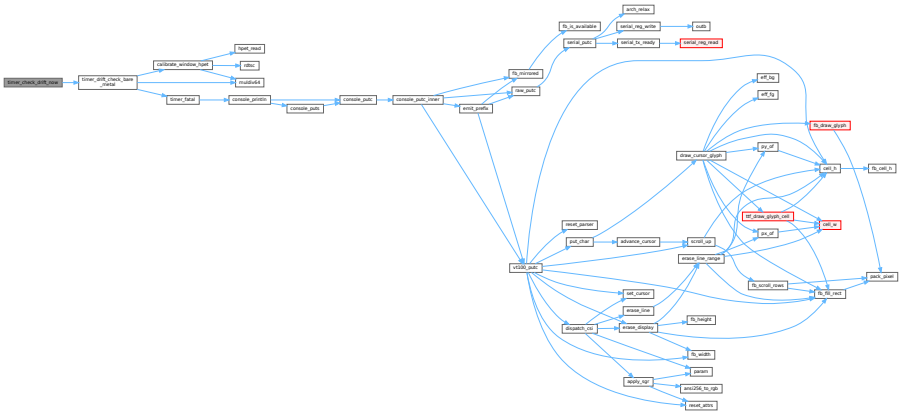
In bare-metal mode delegates to `timer_drift_check_bare_metal()`. In VM mode returns 0 immediately: VM drift checking requires a VM-friendly reference (such as pvclock) that is not yet implemented.

Returns

0 if drift is acceptable or in VM mode, -1 on bare-metal measurement failure.

References calib_record, timer_drift_check_bare_metal(), and timer_calibration_record::vm_mode.

Here is the call graph for this function:



7.116.2.31 timer_drift_check_bare_metal()

```
static int timer_drift_check_bare_metal (
            void  )  [static]
```

Check TSC drift against the HPET reference on bare metal.

Takes a single HPET calibration window and compares the measured TSC Hz to `tsc_hz_locked`. If the drift exceeds `DRIFT_EPSILON_PPM` (2000 ppm), calls `timer_fatal()` — never returns. Returns -1 if either reading is invalid.

Skipped in VM mode (`timer_check_drift_now()` returns 0 immediately).

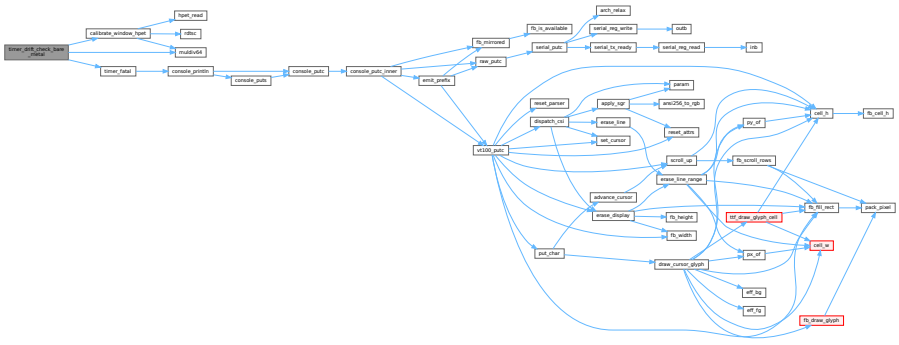
Returns

0 if drift is within bound, -1 on measurement failure.

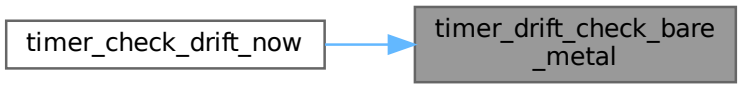
References calibrate_window_hpet(), DRIFT_EPSILON_PPM, muldiv64(), timer_fatal(), tsc_hz_locked, and WINDOW_TICKS_HPET.

Referenced by timer_check_drift_now().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.32 timer_fatal()

```
static void timer_fatal (
            const char * msg )  [static]
```

Emit a fatal timer error message and halt the processor permanently.

Prints `msg` followed by `"Timer subsystem fatal.  Halting."` to the kernel console, then enters an infinite `HLT` loop. Never returns. Called when a hard timer invariant cannot be satisfied (e.g., HPET counter not advancing, TSC calibration failed to converge).

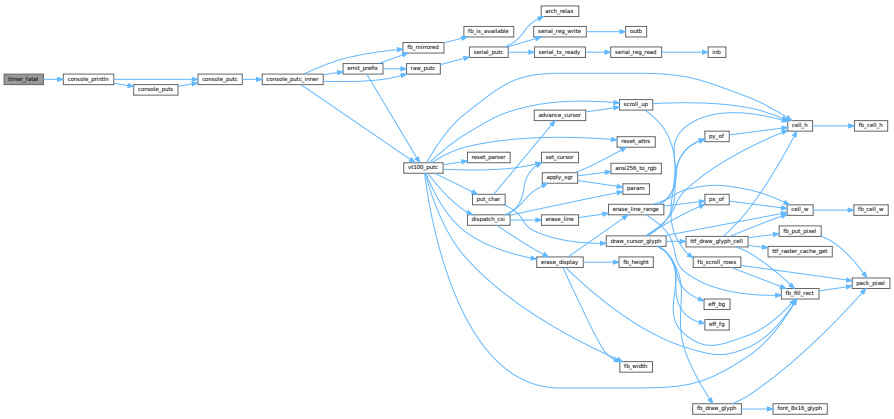
Parameters

msg	Null-terminated error description to print before halting.

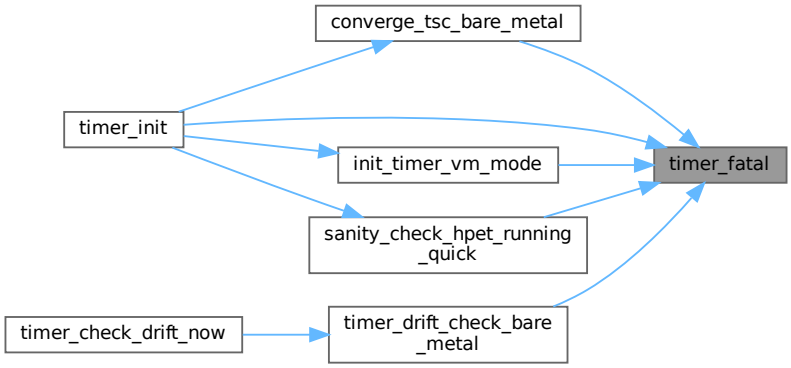
References console_println().

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), sanity_check_hpet_running_quick(), timer_drift_check_bare_meta and timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.2.33 timer_init()

```
int timer_init (
            BootInfo * boot_info )
```

Initialise the kernel timer subsystem (M5).

Detects the execution environment (bare metal vs. hypervisor) and follows the appropriate calibration path:

- **VM / hypervisor path** (CPUID.1:ECX[31] set): calls `init_timer_vm_mode()` — HPET MMIO disabled, TSC from CPUID/PM Timer.
- **Bare metal path**: maps HPET MMIO, enables the counter, optionally probes the PIT, then calls `converge_tsc_bare_metal()` until TSC Hz converges within `CV_EPSILON_PPM` ppm. Calls `timer_fatal()` on any hard failure.

On success, `tsc_hz_locked` holds the calibrated TSC frequency and `calib_record` is fully populated. `timer_now_ns()` is safe to call after this function returns.

Parameters

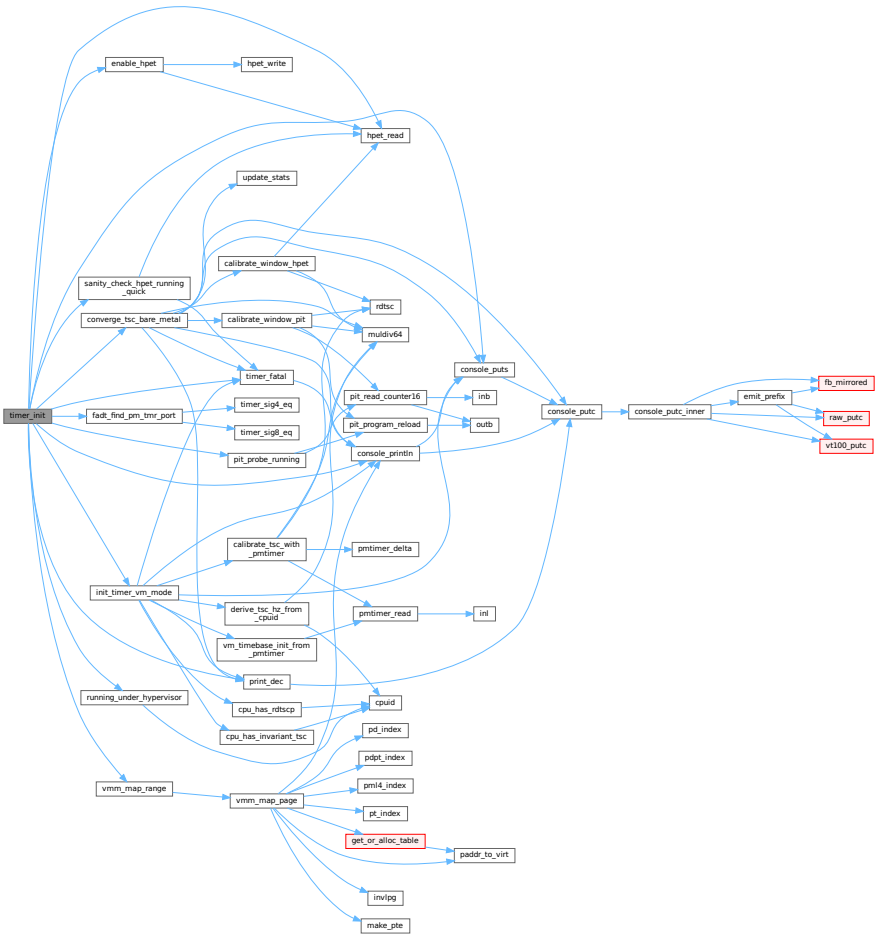
boot_info	BootInfo pointer from the UEFI loader; `acpi_table` is used to discover the ACPI PM_TMR_BLK port at runtime (see `fadt_find_pm_tmr_port()`) rather than assuming a fixed machine type's legacy address.

Returns

0 on success; calls `timer_fatal()` (never returns) on hard failure.

References BootInfo::acpi_table, calib_record, console_println(), console_puts(), converge_tsc_bare_metal(), timer_calibration_record::converged, enable_hpet(), fadt_find_pm_tmr_port(), hpet_freq_hz, HPET_GEN_CAP_ID, HPET_PHYS_BASE, hpet_read(), hpet_regs, HPET_VIRT_BASE, init_timer_vm_mode(), pit_available, pit_probe_running(), pit_tsc_hz_mean, pmtimer_port, print_dec(), running_under_hypervisor(), sanity_check_hpet_running_quick(), timer_fatal(), TIMER_REQUIRE_PIT, TIMER_TRUST_NONE, timer_calibration_record::trust, tsc_hz_locked, timer_calibration_record::vm_mode, VMM_FLAG_WRITABLE, and vmm_map_range().

Here is the call graph for this function:



7.116.2.34 timer_now_ns()

```
uint64_t timer_now_ns (
            void  )
```

Return the current monotonic time in nanoseconds.

Return a monotonic nanosecond timestamp.

Selects the best available time source based on the calibration record:

- `TIMER_TRUST_NONE`: returns 0 (no time base yet).
- VM mode + trust < `TIMER_TRUST_ABSOLUTE`: returns PM Timer relative ns.
- Absolute mode (`tsc_hz_locked` > 0): reads TSC (RDTSCP if supported, else RDTSC) and converts to ns via `muldiv64()`.

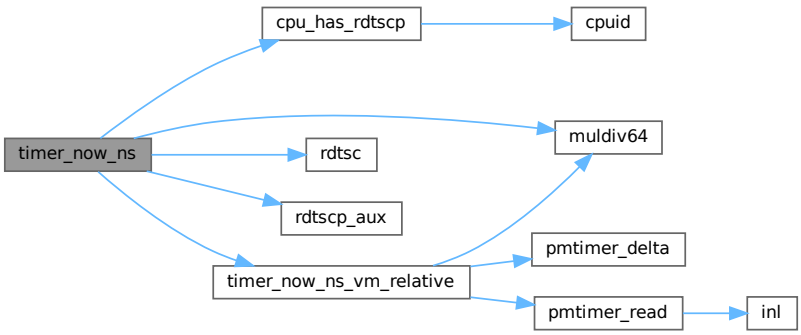
Called by the HAL (`sk_hal_time_ns()`) to feed the physics engine.

Returns

Nanoseconds since boot, or 0 if the timer is not yet calibrated.

References calib_record, cpu_has_rdtscp(), muldiv64(), rdtsc(), rdtscp_aux(), timer_now_ns_vm_relative(), TIMER_TRUST_ABSOLUTE, TIMER_TRUST_NONE, timer_calibration_record::trust, tsc_hz_locked, and timer_calibration_record::vm_mode.

Here is the call graph for this function:



7.116.2.35 timer_now_ns_vm_relative()

```
static uint64_t timer_now_ns_vm_relative (
            void  )  [static]
```

Return nanoseconds since boot using the ACPI PM Timer (VM relative mode).

Reads the current PM Timer counter, computes ticks elapsed since `vm_pm_start` (set by `vm_timebase_init_from_pmtime` converts to nanoseconds via `muldiv64()`, and adds `vm_ns_base`. Used when the calibration record shows `vm_mode` = 1 and `trust` < TIMER_TRUST_ABSOLUTE.

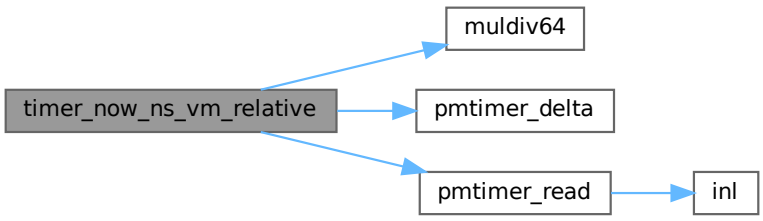
Returns

Nanoseconds elapsed since `vm_timebase_init_from_pmtimer()` was called.

References muldiv64(), pmtimer_delta(), PMTIMER_FREQ_HZ, PMTIMER_MASK, pmtimer_read(), vm_ns_base, and vm_pm_start.

Referenced by timer_now_ns().

Here is the call graph for this function:



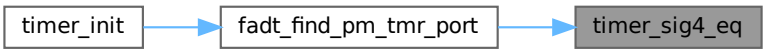
Here is the caller graph for this function:

7.116.2.36 timer_sig4_eq()

```
static int timer_sig4_eq (
            const uint8_t * p,
            const char * s )  [static]
```

Referenced by fadt_find_pm_tmr_port().

Here is the caller graph for this function:

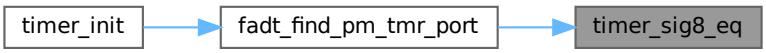


7.116.2.37 timer_sig8_eq()

```
static int timer_sig8_eq (
            const uint8_t * p,
            const char * s )  [static]
```

Referenced by fadt_find_pm_tmr_port().

Here is the caller graph for this function:



7.116.2.38 timer_tsc_hz()

```
uint64_t timer_tsc_hz (
            void  )
```

Return the locked TSC frequency in Hz.

Return the system counter frequency in Hz.

Returns the value established by `timer_init()`. Zero means calibration has not completed or failed to derive a TSC frequency (PM Timer relative mode only).

Returns

TSC frequency in Hz, or 0 if unknown.

References tsc_hz_locked.

7.116.2.39 update_stats()

```
static void update_stats (
             const uint64_t * samples,
             uint32_t count,
             uint64_t * mean,
             uint64_t * stddev )  [static]
```

Compute mean and standard deviation of an array of `uint64_t` samples.

Uses 64-bit integer arithmetic throughout. For TSC-frequency samples (∼3 GHz) with ≤16 windows, both the sum and variance accumulator remain within `uint64_t` range so long as deviations are below ∼1 MHz (which is well within the `CV_EPSILON_PPM` = 1000 ppm convergence bound).

Standard deviation is approximated via integer Newton-Raphson square root (10 iterations) over the variance.

Sets `*mean` = `*stddev` = 0 when `count` is 0.

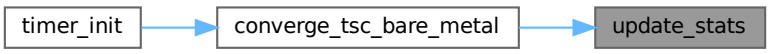
Parameters

samples	Array of `count` frequency samples in Hz.
count	Number of valid samples.
mean	Output: arithmetic mean of `samples`.
stddev	Output: population standard deviation of `samples`.

References count.

Referenced by converge_tsc_bare_metal().

Here is the caller graph for this function:



7.116.2.40 vm_timebase_init_from_pmtimer()

```
static void vm_timebase_init_from_pmtimer (
             void  )  [static]
```

Initialise the VM relative-time base anchored to the PM Timer.

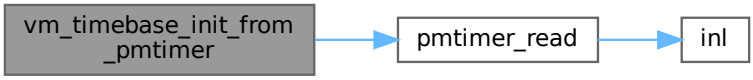
Records the current PM Timer counter value in `vm_pm_start` and resets `vm_ns_base` to zero. Subsequent calls to `timer_now_ns_vm_relative()` compute elapsed nanoseconds from this anchor point.

Called during `init_timer_vm_mode()` after the calibration phase, so the first call to `timer_now_ns()` returns a small positive value rather than a large absolute-time offset.

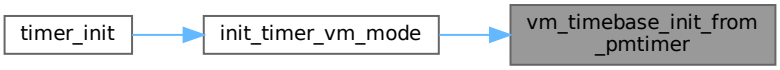
References PMTIMER_MASK, pmtimer_read(), vm_ns_base, and vm_pm_start.

Referenced by init_timer_vm_mode().

Here is the call graph for this function:



Here is the caller graph for this function:



7.116.3 Variable Documentation

7.116.3.1 calib_record

```
timer_calibration_record_t calib_record = {0}  [static]
```

Referenced by converge_tsc_bare_metal(), init_timer_vm_mode(), timer_calibration_record(), timer_check_drift_now(), timer_init(), and timer_now_ns().

7.116.3.2 hpet_freq_hz

```
uint64_t hpet_freq_hz = 0  [static]
```

Referenced by calibrate_window_hpet(), converge_tsc_bare_metal(), and timer_init().

7.116.3.3 hpet_regs

```
volatile uint64_t* hpet_regs = 0  [static]
```

Referenced by hpet_read(), hpet_write(), and timer_init().

7.116.3.4 pit_available

```
int pit_available = 0  [static]
```

Referenced by calibrate_window_pit(), converge_tsc_bare_metal(), and timer_init().

7.116.3.5 pit_tsc_hz_mean

```
uint64_t pit_tsc_hz_mean = 0  [static]
```

Referenced by timer_init().

7.116.3.6 pmtimer_port

```
uint16_t pmtimer_port = 0x408  [static]
```

Referenced by pmtimer_read(), and timer_init().

7.116.3.7 tsc_hz_locked

```
uint64_t tsc_hz_locked = 0  [static]
```

Referenced by init_timer_vm_mode(), timer_drift_check_bare_metal(), timer_init(), timer_now_ns(), and timer_tsc_hz().

7.116.3.8 vm_ns_base

```
uint64_t vm_ns_base = 0  [static]
```

Referenced by timer_now_ns_vm_relative(), and vm_timebase_init_from_pmtimer().

7.116.3.9 vm_pm_start

```
uint32_t vm_pm_start = 0  [static]
```

Referenced by timer_now_ns_vm_relative(), and vm_timebase_init_from_pmtimer().

7.116.3.10 windows_used

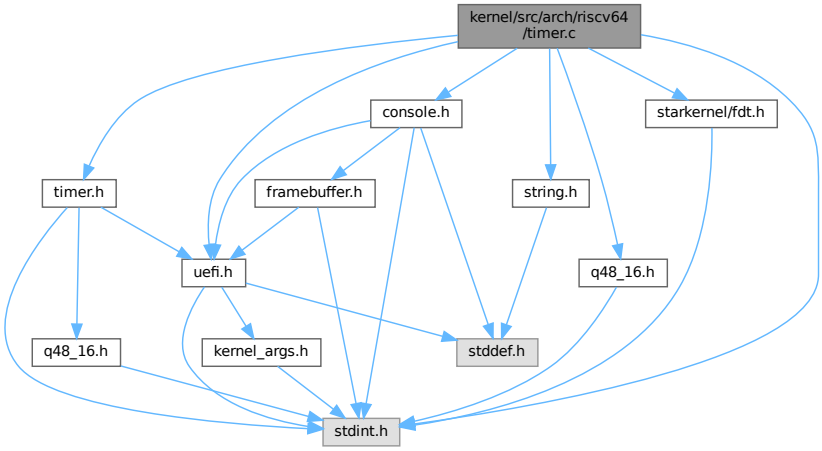
```
uint32_t windows_used = 0  [static]
```

Referenced by converge_tsc_bare_metal().

7.117 kernel/src/arch/riscv64/timer.c File Reference

```
#include "timer.h"
#include "console.h"
#include "q48_16.h"
#include "uefi.h"
#include "starkernel/fdt.h"
#include <stdint.h>
#include <string.h>
```

Include dependency graph for timer.c:



Macros

- #define RISCV_TIMEBASE_HZ_FALLBACK 10000000ULL

Functions

- static int bytes_contain (const uint8_t *hay, uint32_t haylen, const char *needle)

  timer.c (riscv64) - Timer using the `time` CSR.
- static int running_under_hypervisor (BootInfo *boot_info)
- static uint64_t rdtime (void)

  Read the RISC-V wall-clock counter (`rdtime`, CSR `time` 0xC01).
- static void print_dec (uint64_t val)

  Print `val` in decimal via `console_putc()`.
- int timer_init (BootInfo *boot_info)

  Initialise the kernel timer subsystem (M5).
- uint64_t timer_tsc_hz (void)

  Return the `time` counter frequency in Hz.
- uint64_t timer_now_ns (void)

  Return a monotonic nanosecond timestamp.
- int timer_check_drift_now (void)

  Check for `time` counter drift (RISC-V stub — always returns 0).
- const timer_calibration_record_t * timer_calibration_record (void)

  Return a pointer to the timer calibration record.
- uint64_t heartbeat_read_counter (void)

  Read the raw counter the riscv64 heartbeat is paced against.

Variables

- static uint64_t s_counter_hz = 10000000ULL
- static uint64_t s_ns_per_tick = 0
- static uint64_t s_base_count = 0
- static uint64_t s_base_ns = 0
- static timer_calibration_record_t s_cal

7.117.1 Macro Definition Documentation

7.117.1.1 RISCV_TIMEBASE_HZ_FALLBACK

```
#define RISCV_TIMEBASE_HZ_FALLBACK 10000000ULL
```

7.117.2 Function Documentation

7.117.2.1 bytes_contain()

```
static int bytes_contain (
            const uint8_t * hay,
            uint32_t haylen,
            const char * needle )  [static]
```

timer.c (riscv64) - Timer using the `time` CSR.

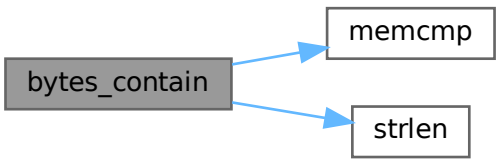
RISC-V provides `cycle` (per-hart clock cycles) and `time` (fixed-frequency wall clock). The primary timestamp source is `time`, read via rdtime, and its rate comes from the devicetree property timebase-frequency.

This was `cycle` at an assumed 1 GHz until punch-list item 0.3. Two things forced the change: the SBI TIME extension arms deadlines against `time`, so mixing the two counters would compare unrelated clocks; and `cycle` has no discoverable frequency, so every heartbeat variance and TIME-TRUST figure riscv64 produced before this was measured against a wrong expected interval (FABRIC-0.md ?16.2). Figures from before and after are not comparable.

References memcmp(), and strlen().

Referenced by running_under_hypervisor().

Here is the call graph for this function:



Here is the caller graph for this function:

7.117.2.2 heartbeat_read_counter()

```
uint64_t heartbeat_read_counter (
            void  )
```

Read the raw counter the riscv64 heartbeat is paced against.

Read the raw hardware counter this architecture's heartbeat is paced against – the same clock `timer_now_ns()` and calibration already use internally (rdtsc on amd64, the `time` CSR on riscv64, CNTPCT_EL0 on aarch64), not a separate/different source.

Item 0.8 (FABRIC-0.md ?25.1): the shared heartbeat.c now owns heartbeat_init()/heartbeat_tick()/heartbeat_↩ service()/heartbeat_ticks()/ heartbeat_trust()/heartbeat_state(). This is the one piece that stays per-architecture – the same `rdtime()` the timer deadline is armed against, not `rdcycle()` or any other source. Must read the same counter the deadline was programmed against: `expected_delta` is derived from timebase-frequency and is therefore in `time` units; measuring the interval with `cycle` instead would difference two unrelated clocks.

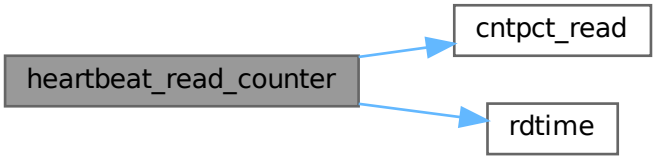
Returns

Current `time` CSR value.

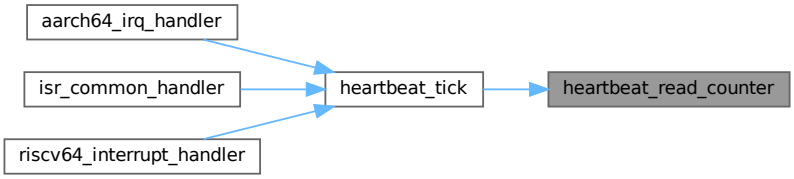
References cntpct_read(), and rdtime().

Referenced by heartbeat_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.117.2.3 print_dec()

```
static void print_dec (
            uint64_t val )  [static]
```

Print `val` in decimal via `console_putc()`.

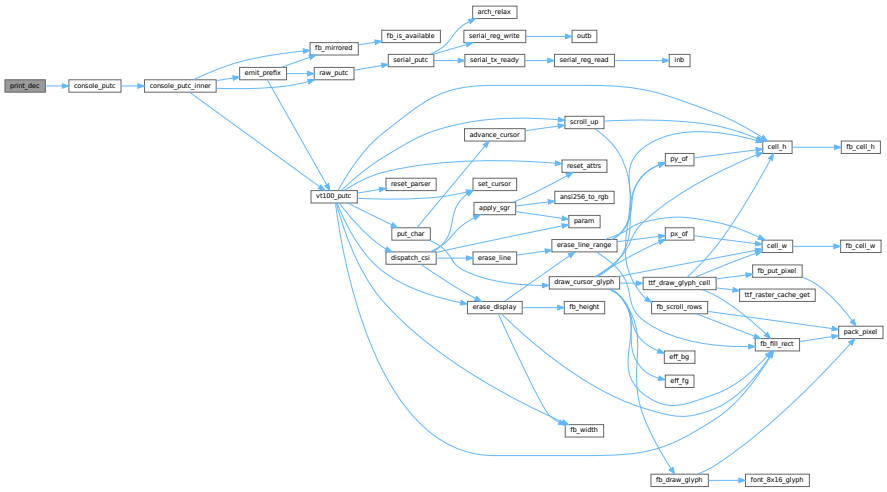
Same shape as the file-local helper in `arch/amd64/timer.c` — there is no shared decimal printer in `console.h`, and the freestanding build has no `printf()`. Prints "0" for zero.

Parameters

val	Value to print.

References console_putc().

Here is the call graph for this function:



7.117.2.4 rdtime()

```
static uint64_t rdtime (
            void  )  [inline], [static]
```

Read the RISC-V wall-clock counter (`rdtime`, CSR `time` 0xC01).

`time` is the memory-mapped real-time counter mandated by the privileged spec: fixed frequency, common to all harts, and — unlike `cycle` — its rate is discoverable, published by firmware as the devicetree property `timebase-frequency`.

This replaces the earlier `rdcycle()` source. Two reasons, and the first is not optional:

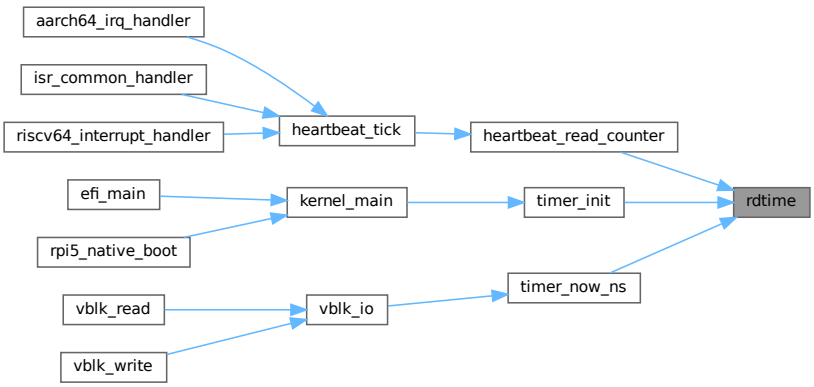
- **The SBI TIME extension is defined against `time`.** `sbi_set_timer()` takes an absolute value on this counter, so arming a timer from a `cycle` reading would compare two unrelated clocks.
- **`cycle`'s frequency is not discoverable**, which is why the previous implementation hardcoded an assumed 1 GHz. Every heartbeat variance and TIME-TRUST figure riscv64 has produced was therefore computed against a wrong expected interval.

Returns

Current 64-bit `time` value.

Referenced by heartbeat_read_counter(), timer_init(), and timer_now_ns().

Here is the caller graph for this function:

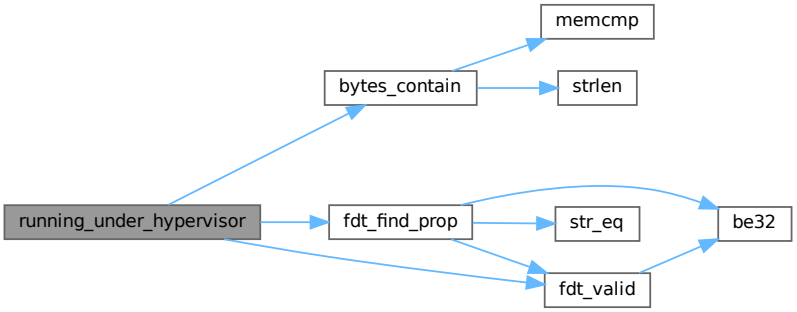


7.117.2.5 running_under_hypervisor()

```
static int running_under_hypervisor (
            BootInfo * boot_info )  [static]
```

References bytes_contain(), BootInfo::dtb, fdt_find_prop(), and fdt_valid().

Here is the call graph for this function:



7.117.2.6 timer_calibration_record()

```
const timer_calibration_record_t * timer_calibration_record (
            void  )
```

Return a pointer to the timer calibration record.

Returns `&s_cal`, populated by `timer_init()` with the discovered (or fallback) frequency and the corresponding trust level — `TIMER_TRUST_ABSOLUTE` when `timebase-frequency` came from the devicetree, `TIMER_TRUST_RELATIVE` on the fallback. Consumed by `kernel_main()` for boot-log reporting and by the heartbeat subsystem's trust init.

Returns

Pointer to the module-static calibration record; valid for the lifetime of the kernel.

References s_cal.

7.117.2.7 timer_check_drift_now()

```
int timer_check_drift_now (
            void  )
```

Check for `time` counter drift (RISC-V stub — always returns 0).

Check for timer drift (AArch64 stub — always returns 0).

On x86-64 this cross-checks the TSC against the HPET. The RISC-V `time` counter has no independent reference to check against at this milestone — and per the privileged spec it is architecturally required to be a fixed, synchronised rate across harts, unlike `cycle`. Always returns 0 (no drift) to satisfy the common call site.

Returns

0 always.

7.117.2.8 timer_init()

```
int timer_init (
            BootInfo * boot_info )
```

Initialise the kernel timer subsystem (M5).

Detects the execution environment (bare metal vs. hypervisor) and follows the appropriate calibration path:

- **VM / hypervisor path** (CPUID.1:ECX[31] set): calls `init_timer_vm_mode()` — HPET MMIO disabled, TSC from CPUID/PM Timer.
- **Bare metal path**: maps HPET MMIO, enables the counter, optionally probes the PIT, then calls `converge_tsc_bare_metal()` until TSC Hz converges within `CV_EPSILON_PPM` ppm. Calls `timer_fatal()` on any hard failure.

On success, `tsc_hz_locked` holds the calibrated TSC frequency and `calib_record` is fully populated. `timer_now_ns()` is safe to call after this function returns.

Parameters

boot_info	BootInfo pointer from the UEFI loader; `acpi_table` is used to discover the ACPI PM_TMR_BLK port at runtime (see `fadt_find_pm_tmr_port()`) rather than assuming a fixed machine type's legacy address.

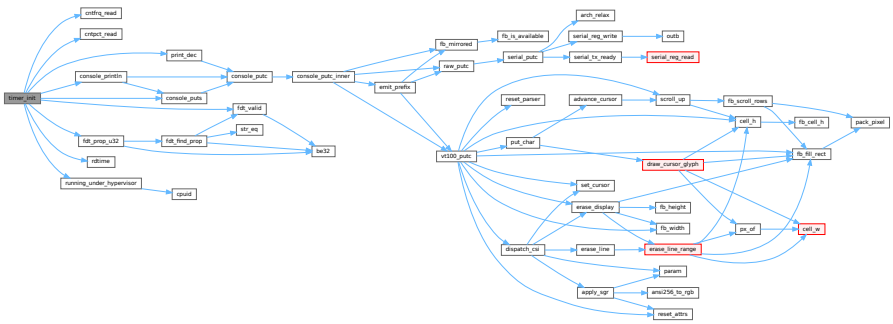
Returns

0 on success; calls `timer_fatal()` (never returns) on hard failure.

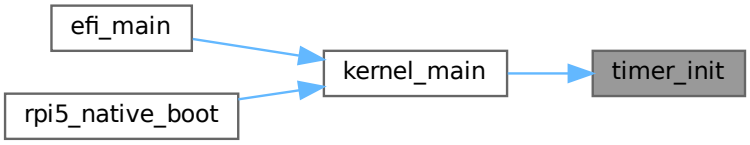
References cntfrq_read(), cntpct_read(), console_println(), console_puts(), timer_calibration_record::converged, BootInfo::dtb, fdt_prop_u32(), fdt_valid(), timer_calibration_record::hpet_hz, timer_calibration_record::pit_hz_mean, print_dec(), rdtime(), running_under_hypervisor(), s_base_count, s_base_ns, s_cal, s_counter_hz, s_ns_per_tick, TIMER_TRUST_ABSOLUTE, TIMER_TRUST_RELATIVE, timer_calibration_record::trust, timer_calibration_record::tsc_hz_mean, and timer_calibration_record::vm_mode.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.117.2.9 timer_now_ns()

```
uint64_t timer_now_ns (
            void  )
```

Return a monotonic nanosecond timestamp.

Reads `rdtime()`, computes the elapsed delta from `s_base_count` (captured at `timer_init()`), and converts to nanoseconds using the Q16.16 fixed-point `s_ns_per_tick`:

```
ns = s_base_ns + ((delta * s_ns_per_tick) » 16)
```

The 64-bit product of `delta` × `s_ns_per_tick` wraps once elapsed real time reaches 2^48 ns (~3.26 days) — this bound is **independent of `s_counter_hz`**, since `s_ns_per_tick` scales inversely with frequency and the two effects cancel. Verified by direct computation, not carried over from the earlier `cycle-based` comment here, which claimed "~18 seconds" and did not match the arithmetic either. Beyond ~3.26 days uptime `timer_now_ns()` wraps and produces incorrect values — a theoretical concern only for this single-hart POST-only kernel.

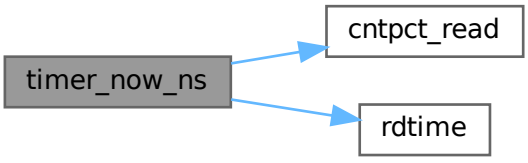
Returns

Monotonic nanosecond count since `timer_init()`, starting at 0.

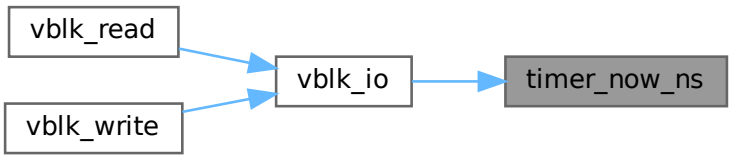
References cntpct_read(), rdtime(), s_base_count, s_base_ns, and s_ns_per_tick.

Referenced by vblk_io().

Here is the call graph for this function:



Here is the caller graph for this function:



7.117.2.10 timer_tsc_hz()

```
uint64_t timer_tsc_hz (
            void  )
```

Return the `time` counter frequency in Hz.

Return the system counter frequency in Hz.

Returns `s_counter_hz` — discovered from the devicetree's `timebase-frequency` when available, otherwise `RISCV_TIMEBASE_HZ_FALLBACK`. Used by `apic_timer_init()` to compute the expected inter-tick period and by the shim time backend to convert `time` counts to nanoseconds.

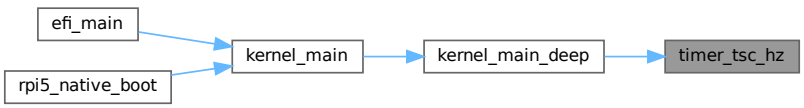
Returns

`time` counter frequency in Hz.

References s_counter_hz.

Referenced by kernel_main_deep().

Here is the caller graph for this function:



7.117.3 Variable Documentation

7.117.3.1 s_base_count

```
uint64_t s_base_count = 0  [static]
```

7.117.3.2 s_base_ns

```
uint64_t s_base_ns = 0  [static]
```

7.117.3.3 s_cal

```
timer_calibration_record_t s_cal  [static]
```

7.117.3.4 s_counter_hz

```
uint64_t s_counter_hz = 10000000ULL  [static]
```

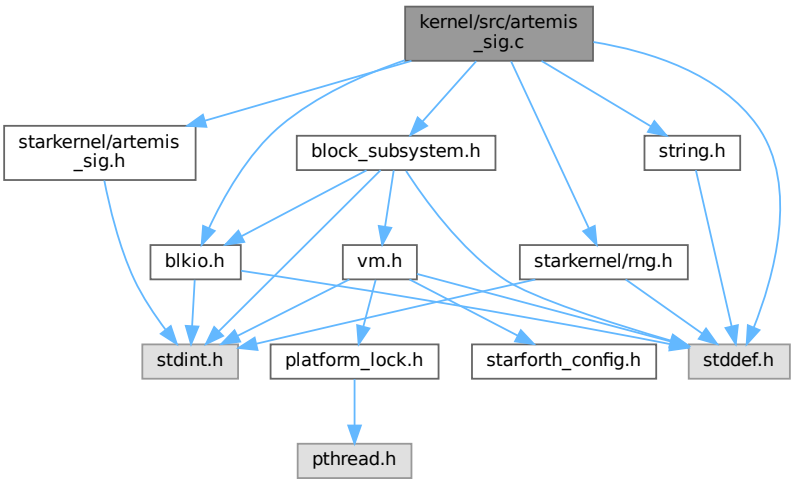
7.117.3.5 s_ns_per_tick

```
uint64_t s_ns_per_tick = 0  [static]
```

7.118 kernel/src/artemis_sig.c File Reference

```
#include "starkernel/artemis_sig.h"
#include <stddef.h>
#include <string.h>
#include "blkio.h"
#include "block_subsystem.h"
#include "starkernel/rng.h"
```

Include dependency graph for artemis_sig.c:



Functions

- uint64_t artemis_sig_compute_crc (const artemis_sig_t *sig)

  artemis_sig.c - Artemis disk signature check (FABRIC-3.md ?XXVI follow-on).
- static int artemis_sig_start_fblock (struct blkio_dev *dev, uint32_t *out_fblock)
- artemis_sig_result_t artemis_sig_check (struct blkio_dev *dev, artemis_sig_t *out_sig)
- int artemis_sig_genesis_stamp (struct blkio_dev *dev)

7.118.1 Function Documentation

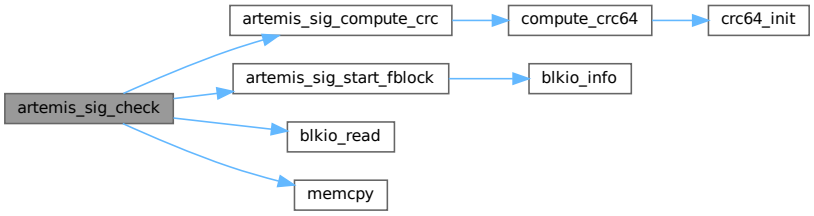
7.118.1.1 artemis_sig_check()

```
artemis_sig_result_t artemis_sig_check (
            struct blkio_dev * dev,
            artemis_sig_t * out_sig )
```

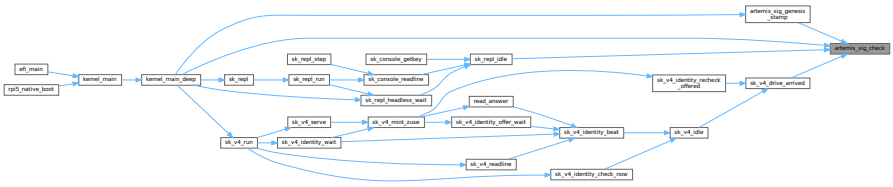
References ARTEMIS_SIG_BAD_CRC, ARTEMIS_SIG_BAD_VERSION, ARTEMIS_SIG_BLANK, artemis_sig_compute_crc(), ARTEMIS_SIG_GET_MAGIC, ARTEMIS_SIG_GET_VERSION, ARTEMIS_SIG_MAGIC, ARTEMIS_SIG_OK, ARTEMIS_SIG_READ_ERROR, artemis_sig_start_fblock(), ARTEMIS_SIG_VERSION_0, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_read(), artemis_sig_t::hdr_crc, artemis_sig_t::magic, and memcpy().

Referenced by artemis_sig_genesis_stamp(), kernel_main_deep(), sk_repl_idle(), and sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:



7.118.1.2 artemis_sig_compute_crc()

```
uint64_t artemis_sig_compute_crc (
            const artemis_sig_t * sig )
```

artemis_sig.c - Artemis disk signature check (FABRIC-3.md ?XXVI follow-on).

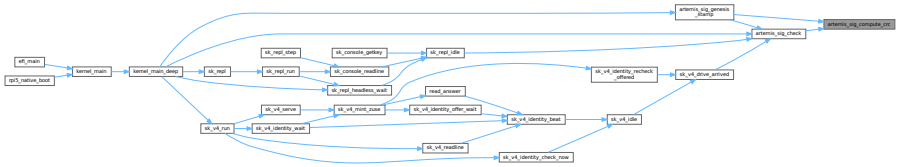
See starkernel/artemis_sig.h for the format and interface design.

References compute_crc64().

Referenced by artemis_sig_check(), and artemis_sig_genesis_stamp().

Here is the call graph for this function:

Here is the caller graph for this function:



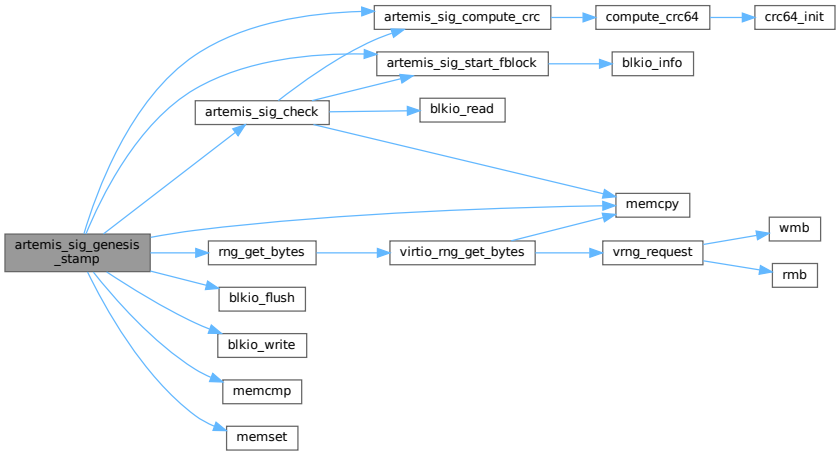
7.118.1.3 artemis_sig_genesis_stamp()

```
int artemis_sig_genesis_stamp (
            struct blkio_dev * dev )
```

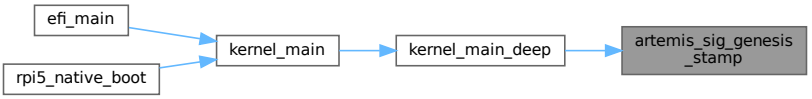
References artemis_sig_check(), artemis_sig_compute_crc(), ARTEMIS_SIG_OK, ARTEMIS_SIG_PACK, artemis_sig_start_fblock(), ARTEMIS_SIG_VERSION_0, blkio_flush(), BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_write(), artemis_sig_t::disk_uuid, artemis_sig_t::genesis_time_ns, artemis_sig_t::hdr_crc, artemis_sig_t::log_region_devblocks, artemis_sig_t::log_region_offset, artemis_sig_t::magic, memcmp(), memcpy(), memset(), artemis_sig_t::metadata_devblocks, and rng_get_bytes().

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.118.1.4 artemis_sig_start_fblock()

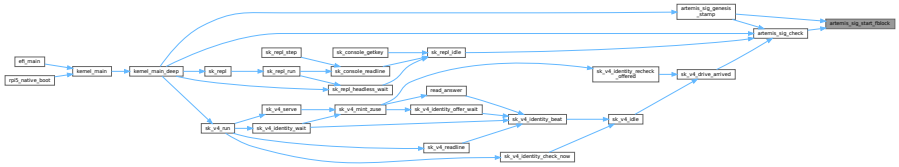
```
static int artemis_sig_start_fblock (
            struct blkio_dev * dev,
            uint32_t * out_fblock )  [static]
```

References ARTEMIS_SIG_DEVBLOCK_FROM_TOP, blkio_info(), BLKIO_OK, and blkio_info_t::total_blocks.

Referenced by artemis_sig_check(), and artemis_sig_genesis_stamp().

Here is the call graph for this function:

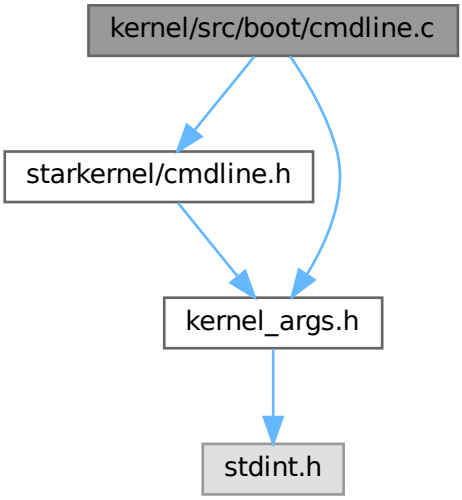
Here is the caller graph for this function:



7.119 kernel/src/boot/cmdline.c File Reference

```
#include "starkernel/cmdline.h"
#include "starkernel/kernel_args.h"
```

Include dependency graph for cmdline.c:



Functions

- static int sf_isspace (char c)

  Test whether a character is ASCII whitespace.
- static int sf_strncmp (const char *a, const char *b, unsigned int n)

  Compare at most `n` characters of two strings (freestanding).
- static void sf_strlcpy (char *dst, const char *src, unsigned int n)

  Copy a string into a fixed-size buffer with guaranteed NUL termination.
- uint64_t cmdline_parse_size (const char *s)

  Parse a decimal integer with optional K/M/G size suffix.
- static int parse_log_level (const char *s)

  Map a log-level name string to a `KARGS_LOG_*` constant.
- void cmdline_parse_ascii (const char *cmdline, KernelArgs *args)

  Parse a NUL-terminated ASCII command-line string into `KernelArgs`.

7.119.1 Function Documentation

7.119.1.1 cmdline_parse_ascii()

```
void cmdline_parse_ascii (
            const char * cmdline,
            KernelArgs * args )
```

Parse a NUL-terminated ASCII command-line string into KernelArgs.

Populates args from space-separated tokens in cmdline. Always initialises all fields to their defaults first (so partial or empty command lines produce safe values):

- stack_size = 0 (use BSS static stack)
- heap_size = 0 (use built-in kmalloc arena)
- run_doe = KARGS_DEFAULT_RUN_DOE
- log_level = KARGS_DEFAULT_LOG_LEVEL
- raw[0] = '\0'

Recognised flags:

- --doe — sets args->run_doe = 1.
- --log-level=<level> — sets args->log_level via parse_log_level().
- --stack=<size> — sets args->stack_size via cmdline_parse_size().
- --heap=<size> — sets args->heap_size via cmdline_parse_size().

Unknown flags are silently ignored for forward-compatibility with flags introduced in future kernel versions. The raw command-line string is copied verbatim into args->raw (truncated to KERNEL_ARGS_CMDLINE_MAX - 1) for diagnostic logging.

Called from efi_main() (in uefi_loader.c) after assembling the ASCII command line from the three priority sources: NVRAM variable, starforth.cfg on the ESP, and EFI LoadOptions.

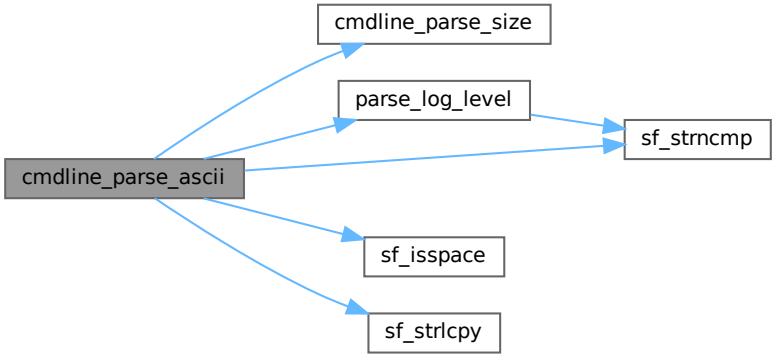
Parameters

cmdline	NUL-terminated ASCII command line; may be NULL or empty.
args	Output KernelArgs structure; must not be NULL.

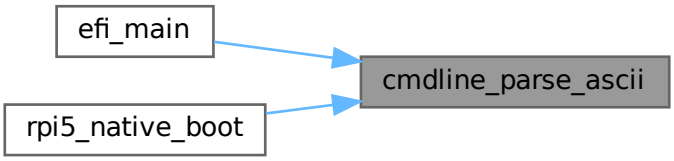
References cmdline_parse_size(), KernelArgs::heap_size, KARGS_DEFAULT_LOG_LEVEL, KARGS_DEFAULT_RUN_DOE, KARGS_SYSTEM_DEFAULT, KARGS_SYSTEM_V3, KARGS_SYSTEM_V4, KERNEL_ARGS_CMDLINE_MAX, KernelArgs::log_level, parse_log_level(), KernelArgs::raw, KernelArgs::run_doe, KernelArgs::screen_boot, sf_isspace(), sf_strlcpy(), sf_strncmp(), KernelArgs::stack_size, and KernelArgs::system.

Referenced by efi_main(), and rpi5_native_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.119.1.2 cmdline_parse_size()

```
uint64_t cmdline_parse_size (
            const char * s )
```

Parse a decimal integer with optional K/M/G size suffix.

Reads a non-negative decimal integer from `s` and multiplies it by:

- 1024 for 'K' or 'k' (kibibytes)
- 1024^2 for 'M' or 'm' (mebibytes)
- 1024^3 for 'G' or 'g' (gibibytes)
- 1 for any other or absent suffix

Returns 0 if `s` is NULL, empty, or does not begin with a digit. Overflow is not checked; caller must validate the resulting value is sensible for its intended use (stack size, heap size, etc.).

Used by `cmdline_parse_ascii()` to decode the `--stack=<size>` and `--heap=<size>` arguments from the boot command line.

Parameters

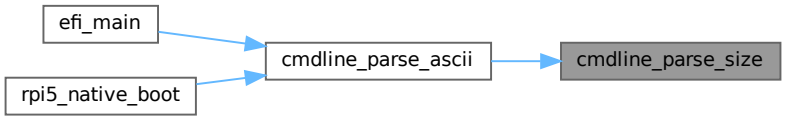
s	NUL-terminated string beginning with a decimal integer and optionally followed by a size suffix character.

Returns

Parsed byte count (0 on parse failure or zero input).

Referenced by cmdline_parse_ascii().

Here is the caller graph for this function:



7.119.1.3 parse_log_level()

```
static int parse_log_level (
            const char * s )  [static]
```

Map a log-level name string to a KARGS_LOG_* constant.

Performs case-insensitive prefix matching against the known level names:

- "debug" / "DEBUG" → KARGS_LOG_DEBUG
- "info" / "INFO" → KARGS_LOG_INFO
- "warn" / "WARN" → KARGS_LOG_WARN
- "error" / "ERROR" → KARGS_LOG_ERROR

Any unrecognised string (including empty) maps to KARGS_LOG_INFO as a safe default. Called from cmdline_parse_ascii() after recognising the --log-level= prefix.

Parameters

s	String beginning at the first character of the level name (i.e., after the "=" separator has been skipped by the caller).

Returns

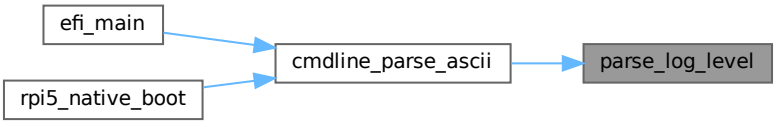
One of the KARGS_LOG_* constants defined in kernel_args.h; defaults to KARGS_LOG_INFO for unknown input.

References KARGS_LOG_DEBUG, KARGS_LOG_ERROR, KARGS_LOG_INFO, KARGS_LOG_WARN, and sf_strncmp().

Referenced by cmdline_parse_ascii().

Here is the call graph for this function:

Here is the caller graph for this function:



7.119.1.4 sf_isspace()

```
static int sf_isspace (
            char c )  [static]
```

Test whether a character is ASCII whitespace.

Returns non-zero for space (' '), horizontal tab ('\t'), carriage return ('\r'), and line feed ('\n'). Used by `cmdline_parse_ascii()` to skip inter-token whitespace and by `load_cfg_from_esp()` to strip trailing CR/LF from the config file. Replaces the libc `isspace()` which is not available in a freestanding build; this version does not consider locale.

Parameters

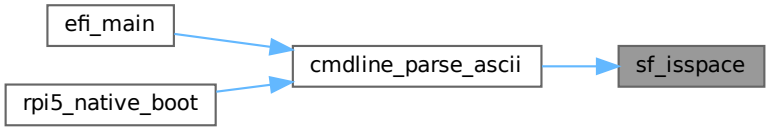
c	Character to test.

Returns

Non-zero if `c` is ASCII whitespace, 0 otherwise.

Referenced by cmdline_parse_ascii().

Here is the caller graph for this function:



7.119.1.5 sf_strlcpy()

```
static void sf_strlcpy (
            char * dst,
            const char * src,
            unsigned int n )  [static]
```

Copy a string into a fixed-size buffer with guaranteed NUL termination.

Copies at most `n-1` bytes from `src` into `dst` and always writes a NUL terminator at `dst[n-1]` (if `n` > 0). This is the BSD `strlcpy()` semantics without returning the source length. If `n` is 0 the function is a no-op (no write, no UB).

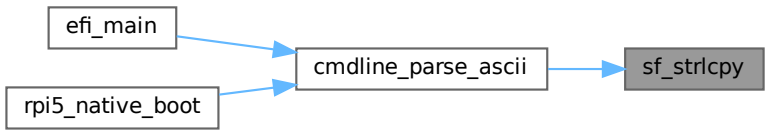
Used by `cmdline_parse_ascii()` to copy the raw command-line string into `KernelArgs.raw` with a guaranteed NUL, avoiding the truncation risk of `strncpy()`.

Parameters

dst	Destination buffer; receives at most `n-1` bytes of `src` followed by a NUL terminator.
src	NUL-terminated source string.
n	Size of `dst` in bytes (including the NUL slot).

Referenced by cmdline_parse_ascii().

Here is the caller graph for this function:



7.119.1.6 sf_strncmp()

```
static int sf_strncmp (
            const char * a,
            const char * b,
            unsigned int n )  [static]
```

Compare at most `n` characters of two strings (freestanding).

Iterates up to `n` bytes, stopping at the first differing character or when either string's null terminator is reached. Returns 0 if the first `n` characters are equal (or both strings end before `n`), positive if `a` is lexicographically greater, negative if `b` is. Comparison is on `unsigned char` values per C standard convention.

Used exclusively by `cmdline_parse_ascii()` and `parse_log_level()` to match fixed-length flag prefixes without a libc dependency.

Parameters

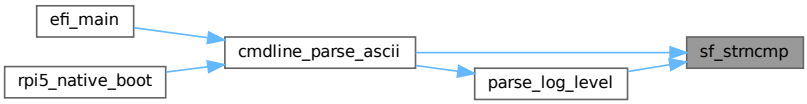
a	First string.
b	Second string.
n	Maximum number of characters to compare.

Returns

0 if equal, positive if `a` > `b`, negative if `a` < `b`.

Referenced by cmdline_parse_ascii(), and parse_log_level().

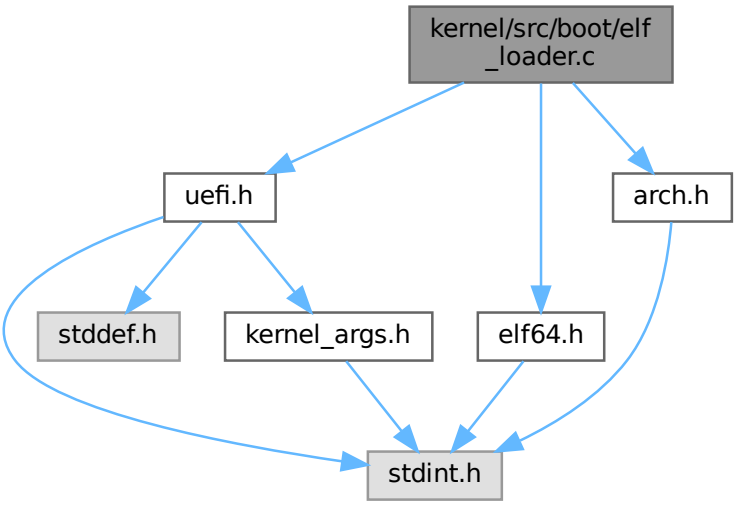
Here is the caller graph for this function:



7.120 kernel/src/boot/elf_loader.c File Reference

```
#include "uefi.h"
#include "elf64.h"
#include "arch.h"
```

Include dependency graph for elf_loader.c:



Functions

- static void * elf_memset (void *s, int c, uint64_t n)

  elf_loader.c - ELF64 kernel loader for StarKernel
- static void * elf_memcpy (void *dest, const void *src, uint64_t n)

  Copy a non-overlapping memory region (freestanding memcpy).
- static int elf_validate_header (const Elf64_Ehdr *ehdr)

  Validate an ELF64 header for the current target architecture.
- static int elf_load_segments (const uint8_t *elf_data, Elf64_Addr load_base)

  Map all PT_LOAD segments from an ELF64 image into memory.
- static int elf_apply_relocations (const uint8_t *elf_data, Elf64_Addr load_base)

  Apply all RELA relocation sections from an ELF64 image.
- int elf_load_kernel (const uint8_t *elf_data, uint64_t elf_size, Elf64_Addr *entry_out)

  Load, relocate, and return the entry point of a StarKernel ELF64 binary.

7.120.1 Function Documentation

7.120.1.1 elf_apply_relocations()

```
static int elf_apply_relocations (
            const uint8_t * elf_data,
            Elf64_Addr load_base )  [static]
```

Apply all RELA relocation sections from an ELF64 image.

Scans the section header table for sections of type `SHT_RELA`. For each such section, resolves the associated symbol table (via `sh_link`) and applies every relocation entry to the loaded image.

Supported relocation types are ISA-specific and compile-time selected:

- **amd64 (`ARCH_AMD64`)**:
  - `R_X86_64_NONE` — no-op.
  - `R_X86_64_RELATIVE` — absolute address = `load_base` + `r_addend`.
  - `R_X86_64_64` — `load_base` + symbol value + `r_addend` (64-bit).
  - `R_X86_64_32` / `R_X86_64_32S` — same, truncated to 32 bits.
  - `R_X86_64_PC32` / `R_X86_64_PLT32` — `symbol` + `r_addend` - P (PC-relative, 32 bits), where P is the relocation's own runtime address; PLT32 has no real PLT indirection in this static, non-PIE link, so it resolves the same as PC32.
- **aarch64 (`ARCH_AARCH64`)**:
  - `R_AARCH64_NONE` — no-op.
  - `R_AARCH64_RELATIVE` — `load_base` + `r_addend`.
  - `R_AARCH64_ABS64` — `load_base` + symbol + `r_addend`.
- **riscv64 (`ARCH_RISCV64`)**:
  - `R_RISCV_NONE` — no-op.
  - `R_RISCV_RELATIVE` — `load_base` + `r_addend`.
  - `R_RISCV_64` — `load_base` + symbol + `r_addend`.

Returns 0 (failure) if a symbol index in a relocation entry is out of bounds, if the symbol table entry size is zero, or if an unknown relocation type is encountered. Returns 1 on success.

Parameters

elf_data	Pointer to the ELF64 file image.
load_base	Load base address applied during `elf_load_segments()`.

Returns

1 on success, 0 on any relocation error.

References Elf64_Ehdr::e_shnum, Elf64_Ehdr::e_shoff, ELF64_R_SYM, ELF64_R_TYPE, R_AARCH64_ABS64, R_AARCH64_NONE, R_AARCH64_RELATIVE, Elf64_Rela::r_addend, Elf64_Rela::r_offset, R_RISCV_64, R_RISCV_NONE, R_RISCV_RELATIVE, R_X86_64_32, R_X86_64_32S, R_X86_64_64, R_X86_64_NONE, R_X86_64_PC32, R_X86_64_PLT32, R_X86_64_RELATIVE, Elf64_Shdr::sh_entsize, Elf64_Shdr::sh_link, Elf64_Shdr::sh_offset, Elf64_Shdr::sh_size, Elf64_Shdr::sh_type, SHT_RELA, and Elf64_Sym::st_value.

Referenced by elf_load_kernel().

Here is the caller graph for this function:

7.120.1.2 elf_load_kernel()

```
int elf_load_kernel (
            const uint8_t * elf_data,
            uint64_t elf_size,
            Elf64_Addr * entry_out )
```

Load, relocate, and return the entry point of a StarKernel ELF64 binary.

elf_loader.h - ELF64 kernel loader interface

Top-level ELF loader called by `efi_main()` (in the split-build path) after `ExitBootServices()`. Performs three operations in sequence:

1. **Header validation** via `elf_validate_header()` — checks magic, class, data encoding, version, type, and machine-specific architecture. Returns 0 immediately on failure.
2. **Segment loading** via `elf_load_segments()` — maps all `PT_LOAD` segments to their target addresses, initialises BSS to zero. The `load_base` is:
   - 0 for `ET_EXEC` (absolute-address kernel; typical for StarKernel).
   - 0x400000 (4 MB) for `ET_DYN` (position-independent kernel).
3. **Relocation processing** via `elf_apply_relocations()` — applies `SHT_RELA` sections to patch in the correct runtime addresses. Returns 0 on any relocation error.

On success, stores `load_base` + `e_entry` in `*entry_out` — the virtual address of the kernel's C entry function `kernel_main()` — and returns 1. The caller performs an indirect call to this address.

`elf_size` is currently unused (reserved for future bounds-checking of program-header and section-header offsets against the file boundary).

Parameters

elf_data	Pointer to the ELF64 file image in UEFI loader memory; must remain valid until after the call returns.
elf_size	Size of the ELF image in bytes (currently unchecked).
entry_out	Output: receives the kernel entry point address on success.

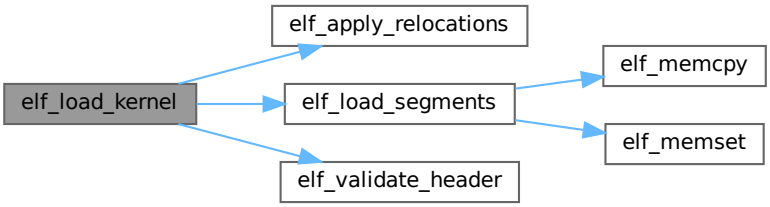
Returns

1 on success, 0 if validation, segment loading, or relocation fails.

References Elf64_Ehdr::e_entry, Elf64_Ehdr::e_type, elf_apply_relocations(), elf_load_segments(), elf_validate_header(), ET_DYN, and ET_EXEC.

Referenced by efi_main().

Here is the call graph for this function:



Here is the caller graph for this function:

7.120.1.3 elf_load_segments()

```
static int elf_load_segments (
             const uint8_t * elf_data,
             Elf64_Addr load_base )  [static]
```

Map all PT_LOAD segments from an ELF64 image into memory.

Iterates the program header table (`e_phnum` entries starting at offset `e_phoff`) and processes every entry with `p_type` == `PT_LOAD`:

1. Computes the destination address as `load_base` + `p_vaddr`.
2. Copies `p_filesz` bytes from `elf_data` + `p_offset` using `elf_memcpy()`.
3. If `p_memsz` > `p_filesz`, zero-fills the remaining (`p_memsz` - `p_filesz`) bytes with `elf_memset()` to initialise the BSS region within the segment.

The function does not check for address-range overlaps or that destination memory is physically available; the UEFI loader is responsible for allocating pages before calling `elf_load_kernel()`.

Parameters

elf_data	Pointer to the start of the ELF64 file in memory.
load_base	Base address at which to load the ELF image (0 for `ET_EXEC`, 0x400000 for `ET_DYN`).

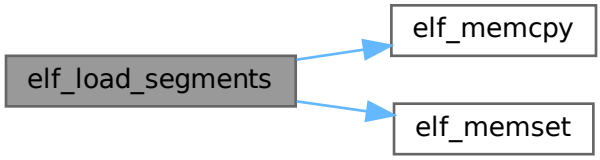
Returns

1 always (errors are silently ignored at this milestone; a future revision should return 0 on a failed memory access).

References Elf64_Ehdr::e_phnum, Elf64_Ehdr::e_phoff, elf_memcpy(), elf_memset(), Elf64_Phdr::p_offset, and PT_LOAD.

Referenced by elf_load_kernel().

Here is the call graph for this function:



Here is the caller graph for this function:

7.120.1.4 elf_memcpy()

```
static void * elf_memcpy (
            void * dest,
            const void * src,
            uint64_t n )  [static]
```

Copy a non-overlapping memory region (freestanding memcpy).

Copies `n` bytes from `src` to `dest` using a byte-at-a-time loop. Behaviour is undefined if the source and destination ranges overlap. Used by `elf_load_segments()` to copy each PT_LOAD segment's `p_filesz` bytes from the ELF image in memory to its load address.

Parameters

dest	Destination buffer; must not overlap with `src`.
src	Source buffer.
n	Number of bytes to copy.

Returns

`dest` (matches standard `memcpy` signature).

Referenced by elf_load_segments().

Here is the caller graph for this function:

7.120.1.5 elf_memset()

```
static void * elf_memset (
            void * s,
            int c,
            uint64_t n )  [static]
```

elf_loader.c - ELF64 kernel loader for StarKernel

Loads the StarKernel ELF binary from ESP, parses segments, applies relocations, and jumps to entry point.

Fill a memory region with a byte value (freestanding memset).

Writes `n` copies of the low byte of `c` to the memory starting at `s`. Used by `elf_load_segments()` to zero-fill BSS regions (`p_memsz` > `p_filesz`) without a libc dependency. The byte loop is correct for all sizes; a future optimisation could use SIMD or word-width stores for large BSS regions.

Parameters

s	Pointer to the start of the destination region.
c	Value to fill (only the low 8 bits are used).
n	Number of bytes to write.

Returns

`s` (matches standard `memset` signature).

Referenced by elf_load_segments().

Here is the caller graph for this function:

7.120.1.6 elf_validate_header()

```
static int elf_validate_header (
            const Elf64_Ehdr * ehdr )  [static]
```

Validate an ELF64 header for the current target architecture.

Checks the following fields in the ELF identification block and header:

- Magic bytes `e_ident`[0..3]: must be `0x7F` 'E' 'L' 'F'.
- Class `e_ident`[EI_CLASS]: must be `ELFCLASS64` (64-bit).
- Data encoding `e_ident`[EI_DATA]: must be `ELFDATA2LSB` (little-endian, the only byte order used by all three supported ISAs).
- Version `e_ident`[EI_VERSION]: must be `EV_CURRENT` (1).
- Type `e_type`: must be `ET_EXEC` (statically linked) or `ET_DYN` (position-independent executable).
- Machine `e_machine`: must match the build target — `EM_X86_64` for `ARCH_AMD64`, `EM_AARCH64` for `ARCH_AARCH64`, `EM_RISCV` for `ARCH_RISCV64`.

Parameters

ehdr	Pointer to the ELF64 header at the start of the image.

Returns

1 if the header is valid for this architecture, 0 otherwise.

References Elf64_Ehdr::e_ident, Elf64_Ehdr::e_machine, Elf64_Ehdr::e_type, EI_CLASS, EI_DATA, EI_MAG0, EI_MAG1, EI_MAG2, EI_MAG3, EI_VERSION, ELFCLASS64, ELFDATA2LSB, ELFMAG0, ELFMAG1, ELFMAG2, ELFMAG3, EM_AARCH64, EM_RISCV, EM_X86_64, ET_DYN, ET_EXEC, and EV_CURRENT.

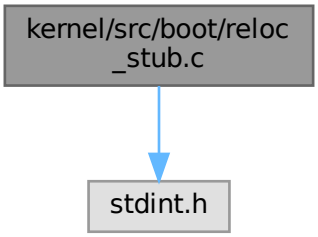
Referenced by elf_load_kernel().

Here is the caller graph for this function:

7.121 kernel/src/boot/reloc_stub.c File Reference

```
#include <stdint.h>
```

Include dependency graph for reloc_stub.c:



Data Structures

- struct reloc_block

  reloc_stub.c - Provide a minimal .reloc section for PE/COFF EFI loader

Functions

- struct reloc_block __attribute__ ((packed, aligned(4)))
- __attribute__ ((section(".reloc"), used))

  Minimal .reloc section stub to satisfy UEFI firmware loaders.

Variables

- uint32_t page_rva
- uint32_t block_size
- uint16_t entry

7.121.1 Function Documentation

7.121.1.1 __attribute__() [1/2]

```
struct reloc_block __attribute__ (
            (packed, aligned(4))  )
```

7.121.1.2 __attribute__() [2/2]

```
__attribute__ (
            (section(".reloc"), used)  )
```

Minimal .reloc section stub to satisfy UEFI firmware loaders.

Some UEFI implementations (notably certain hardware vendor firmware and strict OVMF builds) refuse to load a PE32+ image that lacks a Base Relocation Directory entry in the optional-header Data Directory. Even for monolithic position-independent images that require no run-time fix-ups, the presence of the .reloc section with a well-formed (but empty) relocation block is required to pass the firmware's image validation.

This single statically-allocated `reloc_block` satisfies the requirement:

- `page_rva` = 0 — applies to the first 4 KB page (convention for stubs).
- `block_size` = sizeof(reloc_block) — minimal well-formed block (8 bytes
  - 2 bytes for the single no-op entry = 10 bytes).
- `entry` = 0 — TYPE=0 (absolute, no-op) / OFFSET=0; ignored by the loader.

The `used` attribute prevents the compiler/linker from discarding the symbol as unreferenced; the `section`(".↩ reloc") attribute places it in the named PE section so the linker emits the correct Data Directory.

References entry, and reloc_block::page_rva.

7.121.2 Variable Documentation

7.121.2.1 block_size

```
uint32_t block_size
```

Referenced by blkio_usb_open_msc(), xhci_bot_read_block(), xhci_bot_send_read10(), xhci_bot_send_write10(), and xhci_bot_write_block().

7.121.2.2 entry

```
uint16_t entry
```

Referenced by __attribute__(), acl_recheck(), acl_recheck(), bus_to_phys(), capsule_birth_baby(), capsule_dict_hash_hook(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_vm_registry_get(), capsule_vm_registry_set_name(), capsule_vm_set_pending_reap(), capsule_vm_set_state(), capsule_wirebind_eject(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_reap_idle_check(), capsule_wirebind_unclean_detach(), cd_config_table_tick(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_variable(), defining_word_compile(), defining_word_constant(), defining_word_create(), defining_word_variable(), derive_state_flags(), dictionary_m_word_cfa(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), doe_log_heat_by_name(), execute_colon_word(), extract_heat_trajectory(), fdt_console_uart(), fdt_reg_address(), find_entry_from_name(), forth_FREEZE_CRITICAL(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_STORE(), forth_PIPELINING_ANALYZE_W forth_PIPELINING_RESET_ALL(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), forth_PIPELINING_STATS(), forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), get_body_address(), get_name_field(), get_or_alloc_table(), get_table(), get_word_stats(), hash_colon_body(), heartbeat_capture_tick_snapshot(), kernel_main_deep(), physics_execution_heat_increment(), physics_execution_heat_load(), physics_metadata_apply_linear_decay(),

physics_metadata_apply_seed(), physics_metadata_init(), physics_metadata_record_latency(), physics_metadata_refresh_state(), physics_metadata_set_mass(), physics_metadata_touch(), physics_on_lookup(), pop_xt(), print_word_name(), print_word_name_safe(), profiler_word_count(), profiler_word_enter(), profiler_word_exit(), register_word(), replace_entry(), rolling_window_seed_hotwords_cache(), rpi5_mmu_map_ram(), starforth_word_execution_heat_fetch(), starforth_word_execution_heat_store(), starforth_word_reset_execution_heat(), starforth_word_top_words(), starforth_word_word_execution_heat(), system_word_execute(), system_word_see(), vm_bootstrap_root_vocabulary(), vm_compile_call(), vm_compile_word(), vm_create_word(), vm_dictionary_get_data_field(), vm_dictionary_track_entry(), vm_dictionary_untrack_entry(), vm_docol_arm64(), vm_docol_asm(), vm_docol_rv64(), vm_interpret_word(), vm_physics_find_root_id(), vm_physics_status(), vmm_query_page(), vocabulary_create_vocabulary_direct(), and vocabulary_word_vocabulary().

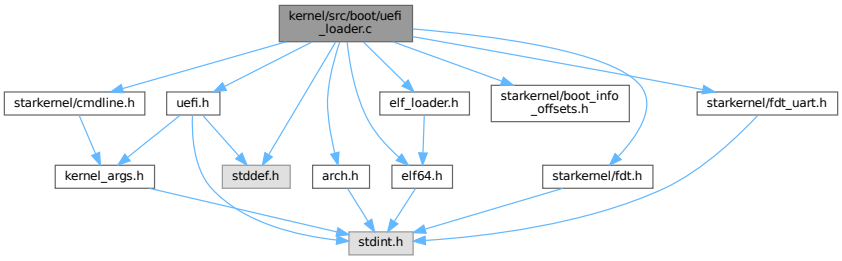
7.121.2.3 page_rva

```
uint32_t page_rva
```

7.122 kernel/src/boot/uefi_loader.c File Reference

```
#include "uefi.h"
#include "arch.h"
#include "elf_loader.h"
#include "elf64.h"
#include "starkernel/cmdline.h"
#include "starkernel/boot_info_offsets.h"
#include "starkernel/fdt.h"
#include "starkernel/fdt_uart.h"
#include <stddef.h>
```

Include dependency graph for uefi_loader.c:



Macros

- #define RAW_LOG(str) ((void)0)

  uefi_loader.c - UEFI boot loader for StarKernel
- #define UEFI_DEBUG_CHECKPOINTS 1
- #define DEBUG_CHECKPOINT_STALL_US (2u * 1000u * 1000u)
- #define KERNEL_MAX_SIZE (8 * 1024 * 1024)

Typedefs

- typedef void(* KernelEntry) (BootInfo *boot_info)

Functions

- static void debug_checkpoint (EFI_SYSTEM_TABLE *SystemTable, UINT32 id, const CHAR16 *label)
- static int guid_equals (const EFI_GUID *a, const EFI_GUID *b)

  Compare two EFI GUIDs for equality.
- static int load_cfg_from_esp (EFI_HANDLE ImageHandle, EFI_BOOT_SERVICES *BS, char *buf, UINTN buflen)

  Read `/starforth.cfg` from the EFI System Partition.
- static EFI_STATUS load_kernel_from_esp (EFI_HANDLE ImageHandle, EFI_BOOT_SERVICES *BS)

  Load `kernel.elf` from the ESP into UEFI-allocated memory (split build).
- EFI_STATUS EFIAPI efi_main (EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)

  UEFI application entry point — boot loader for StarKernel.

Variables

- static BootInfo g_boot_info = {0}
- static uint8_t * kernel_elf_buffer = NULL
- static uint64_t kernel_elf_size = 0

7.122.1 Macro Definition Documentation

7.122.1.1 DEBUG_CHECKPOINT_STALL_US

```
#define DEBUG_CHECKPOINT_STALL_US (2u * 1000u * 1000u)
```

7.122.1.2 KERNEL_MAX_SIZE

```
#define KERNEL_MAX_SIZE (8 * 1024 * 1024)
```

7.122.1.3 RAW_LOG

```
#define RAW_LOG(
              str ) ((void)0)
```

uefi_loader.c - UEFI boot loader for StarKernel

This loader loads the StarKernel ELF binary from the ESP, parses it, loads segments, applies relocations, and jumps to entry.

7.122.1.4 UEFI_DEBUG_CHECKPOINTS

```
#define UEFI_DEBUG_CHECKPOINTS 1
```

7.122.2 Typedef Documentation

7.122.2.1 KernelEntry

```
typedef void(* KernelEntry) (BootInfo *boot_info)
```

7.122.3 Function Documentation

7.122.3.1 debug_checkpoint()

```
static void debug_checkpoint (
            EFI_SYSTEM_TABLE * SystemTable,
            UINT32 id,
            const CHAR16 * label )  [static]
```

References _EFI_SYSTEM_TABLE::BootServices, _EFI_SYSTEM_TABLE::ConOut, DEBUG_CHECKPOINT_STALL_US, _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::OutputString, and _EFI_BOOT_SERVICES::Stall.

Referenced by efi_main().

Here is the caller graph for this function:

7.122.3.2 efi_main()

```
EFI_STATUS EFIAPI efi_main (
            EFI_HANDLE ImageHandle,
            EFI_SYSTEM_TABLE * SystemTable )
```

UEFI application entry point — boot loader for StarKernel.

`efi_main()` is the UEFI PE32+ entry point called by the firmware immediately after the image is loaded and validated. It drives the full boot sequence in two phases:

Phase A — UEFI Boot Services are active:

A-0. Resets the UEFI ConOut console and emits an identification banner. On amd64, initialises COM1 via `raw_serial_init()` for raw-serial boot logging (`RAW_LOG()` macro).

A-1. Assembles the ASCII kernel command line from three sources in priority order:

1. `StarForthBootArgs` NVRAM variable (one-shot, set by REBOOT word; immediately cleared after reading to prevent re-use on next boot).
2. `/starforth`.cfg on the ESP (installer / build-time config).
3. `LoadOptions` from the UEFI boot manager (typically the EFI shell command line or boot entry options). Parses the result via `cmdline_parse_ascii()` into `g_boot_info.args`.

A-2. If `--stack=<size>` was given, allocates `stack_size` bytes of `EfiLoaderData` pages for the kernel stack and stores the base address and size in `g_boot_info`.

A-3. Pre-fills non-EBS-dependent `BootInfo` fields: runtime services pointer, ACPI table (found in `ConfigurationTable` by GUID), and GOP framebuffer geometry.

A-4. (Split build only) Loads `kernel.elf` from the ESP via `load_kernel_from_esp()` into the pre-allocated 8 MB buffer.

Phase B — ExitBootServices:

Calls `GetMemoryMap()` to obtain the current map key, then immediately calls `ExitBootServices()`. Retries up to 16 times on `EFI_INVALID_PARAMETER` (map key stale) as required by the UEFI spec. No UEFI Boot Services calls are made between `GetMemoryMap` and `ExitBootServices` — this is the UEFI "golden path" requirement.

Phase C — Post-ExitBootServices:

- **Monolithic build**: calls `kernel_main(&g_boot_info)` directly (kernel is linked into the same PE image).
- **Split build**: parses and loads the kernel ELF via `elf_load_kernel()`, then performs an indirect call to the resolved `kernel_main()` entry point address.

If the kernel returns (which it must not), `efi_main()` halts the processor in an infinite `arch_halt()` loop.

Parameters

ImageHandle	UEFI handle for this loaded application image.
SystemTable	Pointer to the UEFI system table (Boot/Runtime Services, ConOut, ConfigurationTable, etc.).

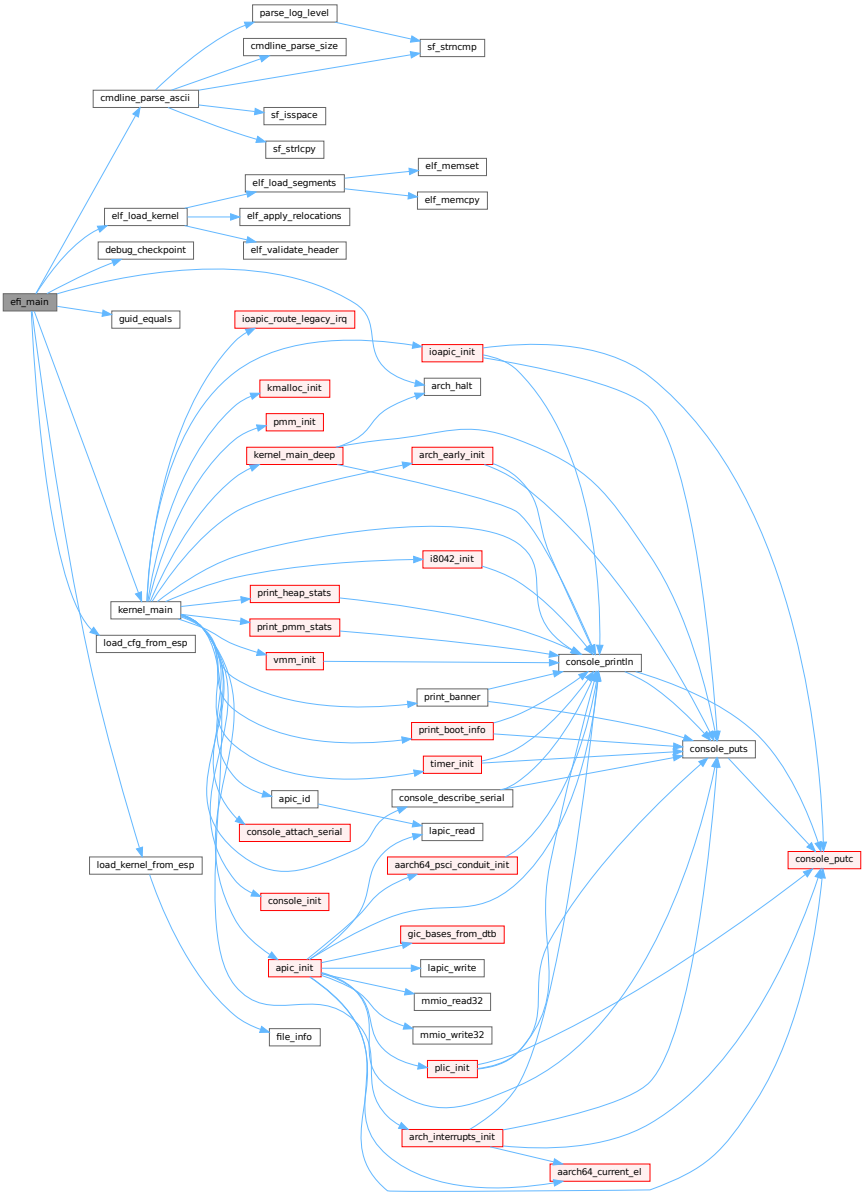
Returns

`EFI_SUCCESS` if the sequence completes without a fatal error before `ExitBootServices` (in practice, control never returns here because the kernel takes over permanently).

References BootInfo::acpi_table, AllocateAnyPages, _EFI_BOOT_SERVICES::AllocatePages, arch_halt(), BootInfo::args, FramebufferInfo::base, BootInfo::boot_disk, _EFI_SYSTEM_TABLE::BootServices, cmdline_parse_ascii(), _EFI_SYSTEM_TABLE::ConfigurationTable, _EFI_SYSTEM_TABLE::ConOut, debug_checkpoint(), BootInfo::dtb, EFI_ACPI_20_TABLE_GUID, EFI_ACPI_TABLE_GUID, EFI_BUFFER_TOO_SMALL, EFI_DTB_TABLE_GUID, EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID, EFI_INVALID_PARAMETER, EFI_LOADED_IMAGE_PROTOCOL_GUID, EFI_SUCCESS, EFI_VARIABLE_BOOTSERVICE_ACCESS, EFI_VARIABLE_NON_VOLATILE, EFI_VARIABLE_RUNTIME_ACCES EfiLoaderData, elf_load_kernel(), _EFI_BOOT_SERVICES::ExitBootServices, FALSE, BootInfo::framebuffer, EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::FrameBufferBase, EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::FrameBufferS g_boot_info, _EFI_BOOT_SERVICES::GetMemoryMap, _EFI_RUNTIME_SERVICES::GetVariable, guid_equals(), _EFI_BOOT_SERVICES::HandleProtocol, FramebufferInfo::height, EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::HorizontalR EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE::Info, KERNEL_ARGS_CMDLINE_MAX, kernel_elf_buffer, kernel_elf_size, kernel_main(), BootInfo::kernel_stack_base, BootInfo::kernel_stack_size, load_cfg_from_esp(), load_kernel_from_esp(), EFI_LOADED_IMAGE_PROTOCOL::LoadOptions, EFI_LOADED_IMAGE_PROTOCOL::LoadOptionsSize, _EFI_BOOT_SERVICES::LocateProtocol, BootInfo::memory_map, BootInfo::memory_map_descriptor_size, BootInfo::memory_map_size, _EFI_GRAPHICS_OUTPUT_PROTOCOL::Mode, _EFI_SYSTEM_TABLE::NumberOfTableEntries, _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::OutputString, FramebufferInfo::pixel_format, PixelBltOnly, EFI_GRAPHICS_OUTPUT_ FramebufferInfo::pixels_per_scanline, EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::PixelsPerScanLine, RAW_LOG, _EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL::Reset, BootInfo::runtime_services, _EFI_SYSTEM_TABLE::RuntimeServ

_EFI_RUNTIME_SERVICES::SetVariable, SF_VAR_BOOT_ARGS, FramebufferInfo::size, KernelArgs::stack_size, STARFORTH_VENDOR_GUID, BootInfo::uefi_boot_services_exited, EFI_CONFIGURATION_TABLE::VendorTable, EFI_GRAPHICS_OUTPUT_MODE_INFORMATION::VerticalResolution, and FramebufferInfo::width.

Here is the call graph for this function:



7.122.3.3 guid_equals()

```
static int guid_equals (
            const EFI_GUID * a,
            const EFI_GUID * b )  [static]
```

Compare two EFI GUIDs for equality.

Compares all 128 bits of the two GUIDs field by field: `Data1` (32-bit), `Data2` (16-bit), `Data3` (16-bit), and all 8 bytes of `Data4`. Returns non-zero only if every field matches.

Used by `efi_main()` to search the UEFI `ConfigurationTable` for the ACPI 2.0 and ACPI 1.0 table GUIDs (`EFI_ACPI_20_TABLE_GUID` and `EFI_ACPI_TABLE_GUID`) to populate `g_boot_info.acpi_table` before calling `ExitBootServices()`.

Parameters

a	Pointer to the first EFI GUID.
b	Pointer to the second EFI GUID.

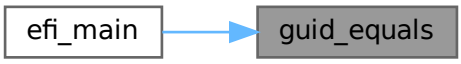
Returns

Non-zero if the GUIDs are equal, 0 otherwise.

References EFI_GUID::Data1, EFI_GUID::Data2, EFI_GUID::Data3, and EFI_GUID::Data4.

Referenced by efi_main().

Here is the caller graph for this function:



7.122.3.4 load_cfg_from_esp()

```
static int load_cfg_from_esp (
             EFI_HANDLE ImageHandle,
             EFI_BOOT_SERVICES * BS,
             char * buf,
             UINTN buflen )  [static]
```

Read `/starforth.cfg` from the EFI System Partition.

Opens the ESP root directory using `EFI_SIMPLE_FILE_SYSTEM_PROTOCOL` via the device handle of the loaded image, then reads the file `"starforth.cfg"` into `buf`. On success, `buf` contains a NUL-terminated ASCII command-line string (at most `buflen` - 1 bytes) with trailing CR and LF characters stripped.

This is the second-priority command-line source in `efi_main()` (after the NVRAM one-shot variable `StarForthBootArgs`). It is intended for installer-supplied or build-time-configured boot options that persist across reboots, unlike the ephemeral NVRAM variable.

Both `MONOLITHIC_BUILD` and split-build paths call this function; it is not gated by a preprocessor condition.

Parameters

ImageHandle	UEFI image handle for the current application; used to retrieve EFI_LOADED_IMAGE_PROTOCOL.
BS	UEFI Boot Services table pointer; must not be NULL.
buf	Caller-supplied buffer; receives the NUL-terminated ASCII config string on success.
buflen	Size of buf in bytes; at most buflen−1 bytes of the file are read.

Returns

1 if the file was read and contained non-empty content; 0 on any UEFI error or if the file is empty after stripping.

References _EFI_FILE_PROTOCOL::Close, EFI_LOADED_IMAGE_PROTOCOL::DeviceHandle, EFI_FILE_MODE_READ, EFI_LOADED_IMAGE_PROTOCOL_GUID, EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID, EFI_SUCCESS, _EFI_BOOT_SERVICES::HandleProtocol, _EFI_FILE_PROTOCOL::Open, _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL::OpenVolum and _EFI_FILE_PROTOCOL::Read.

Referenced by efi_main().

Here is the caller graph for this function:

7.122.3.5 load_kernel_from_esp()

```
static EFI_STATUS load_kernel_from_esp (
            EFI_HANDLE ImageHandle,
            EFI_BOOT_SERVICES * BS )  [static]
```

Load kernel.elf from the ESP into UEFI-allocated memory (split build).

Used only in the split-build path (!MONOLITHIC_BUILD) where the kernel is a separate ELF file on the EFI System Partition rather than being linked directly into the loader image. Execution order:

1. Allocates KERNEL_MAX_SIZE (8 MB) of EfiLoaderData pages via AllocatePages. Stores the result in the module-static kernel_elf_buffer pointer.
2. Retrieves EFI_LOADED_IMAGE_PROTOCOL from ImageHandle to obtain the boot device handle.
3. Opens EFI_SIMPLE_FILE_SYSTEM_PROTOCOL on the device handle and opens the ESP volume root directory.
4. Opens "kernel.elf" for read-only access.
5. Queries file size via EFI_FILE_INFO; returns EFI_BUFFER_TOO_SMALL if the file exceeds KERNEL↩ _MAX_SIZE.
6. Reads the entire file into kernel_elf_buffer and stores the byte count in kernel_elf_size.

Must be called before ExitBootServices() — file system access via Boot Services is unavailable after EBS. The allocated buffer and size are then passed to elf_load_kernel() in Phase C.

Only compiled when MONOLITHIC_BUILD is not defined.

Parameters

ImageHandle	UEFI image handle for the current application.
BS	UEFI Boot Services table pointer.

Returns

`EFI_SUCCESS` on success, or a UEFI status code on failure.

References AllocateAnyPages, _EFI_BOOT_SERVICES::AllocatePages, _EFI_FILE_PROTOCOL::Close, EFI_LOADED_IMAGE_PROTOCOL::DeviceHandle, EFI_ABORTED, EFI_BUFFER_TOO_SMALL, EFI_FILE_INFO_GUID, EFI_FILE_MODE_READ, EFI_LOADED_IMAGE_PROTOCOL_GUID, EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID, EFI_SUCCESS, EfiLoaderData, file_info(), EFI_FILE_INFO::FileSize, _EFI_FILE_PROTOCOL::GetInfo, _EFI_BOOT_SERVICES::HandleProtocol, kernel_elf_buffer, kernel_elf_size, KERNEL_MAX_SIZE, _EFI_FILE_PROTOCOL::Open, _EFI_SIMPLE_FILE_SYSTEM_PROTOCOL::OpenVolume, RAW_LOG, and _EFI_FILE_PROTOCOL::Read.

Referenced by efi_main().

Here is the call graph for this function:

Here is the caller graph for this function:

7.122.4 Variable Documentation

7.122.4.1 g_boot_info

```
BootInfo g_boot_info = {0}  [static]
```

Referenced by efi_main().

7.122.4.2 kernel_elf_buffer

```
uint8_t* kernel_elf_buffer = NULL  [static]
```

Referenced by efi_main(), and load_kernel_from_esp().

7.122.4.3 kernel_elf_size

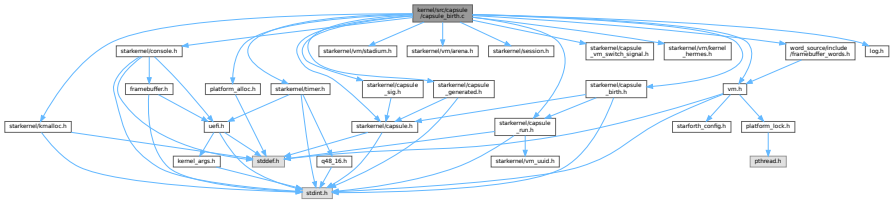
```
uint64_t kernel_elf_size = 0  [static]
```

Referenced by efi_main(), and load_kernel_from_esp().

7.123 kernel/src/capsule/capsule_birth.c File Reference

```
#include "starkernel/capsule_birth.h"
#include "starkernel/capsule.h"
#include "starkernel/capsule_run.h"
#include "starkernel/capsule_generated.h"
#include "starkernel/capsule_sig.h"
#include "starkernel/kmalloc.h"
#include "starkernel/console.h"
#include "starkernel/vm/stadium.h"
#include "starkernel/vm/arena.h"
#include "starkernel/session.h"
#include "starkernel/timer.h"
#include "starkernel/capsule_vm_switch_signal.h"
#include "starkernel/vm/kernel_hermes.h"
#include "word_source/include/framebuffer_words.h"
#include "vm.h"
#include "platform_alloc.h"
#include "log.h"
```

Include dependency graph for capsule_birth.c:



Data Structures

- struct vm_node

Typedefs

- typedef struct vm_node vm_node_t

Functions

- void capsule_birth_set_hooks (CapsuleExecFn exec_fn, CapsuleDictHashFn dict_hash_fn, CapsuleVMAllocFn vm_alloc_fn_arg)

  capsule_birth_set_hooks - Configure VM execution hooks
- static void vm_name_copy (char *dst, const char *src)
- static int vm_name_eq (const char *a, const char *b)
- static VMRegistryEntry * vm_find_entry_ptr (VMUuid vm_id)
- static void dictionary_snapshot_acl_from_parent (VM *child, VM *parent)
- void capsule_vm_registry_init (void *mama_vm_ptr)

  capsule_vm_registry_init - Initialize VM registry
- static VMRegistryEntry * vm_registry_alloc (void)
- int capsule_vm_registry_get (VMUuid vm_id, VMRegistryEntry *out)

  capsule_vm_registry_get - Get VM registry entry by ID
- uint32_t capsule_vm_registry_count (void)

  capsule_vm_registry_count - Get number of registered VMs
- int capsule_vm_registry_get_by_index (uint32_t index, VMRegistryEntry *out)

  capsule_vm_registry_get_by_index - Get registry entry by list position (birth order, stable within a boot session – the registry is append-only).
- static uint32_t vm_registry_live_count (void)
- int capsule_vm_find_by_name (const char *name, VMRegistryEntry *out)

  capsule_vm_find_by_name - Find VM registry entry by symbolic name
- static char vm_to_lower (char c)
- static int vm_name_eq_nocase (const char *a, const char *b)
- static int vm_name_prefix_eq_nocase (const char *str, const char *prefix)
- int capsule_vm_find_by_name_nocase (const char *name, VMRegistryEntry *out)

  capsule_vm_find_by_name_nocase - Find VM registry entry by name (case-insensitive)
- void capsule_vm_set_state (VMUuid vm_id, uint32_t state)

  capsule_vm_set_state - Update a VM's state in the registry
- void capsule_vm_set_pending_reap (VMUuid vm_id, int pending)

  capsule_vm_set_pending_reap - FABRIC-3.md ?XXVIII Stage 4 (2026-09-14): mark vm_id for deferred teardown once it is no longer worth resuming – see VMRegistryEntry.pending_reap's own doc comment for the full rationale.
- void capsule_vm_force_reap (VMUuid vm_id)

  capsule_vm_force_reap - FABRIC-3.md ?XXVIII Stage 4 (2026-09-14): unconditionally tear down vm_id regardless of VM_STATE_SWITCHED_OUT – the one caller allowed to bypass capsule_vm_kill()'s own refusal there, because this is called by the Stage 3 switcher itself (vm_core.c's checkpoint, on noticing pending_reap set), at the one point that genuinely knows the parked native-stack context will never be resumed.
- void capsule_vm_registry_set_name (VMUuid vm_id, const char *name)

  capsule_vm_registry_set_name - Assign a symbolic name to a registered VM
- static void dispatch_init_forth (void *vm_ctx)
- int capsule_vm_kill (const char *name)

  capsule_vm_kill - Destroy a named VM and release all its resources
- void capsule_vm_kill_all_nonmama (void)

  capsule_vm_kill_all_nonmama - Kill every non-Mama VM in the registry.
- static int contrib_capsule_refused (uint32_t flags, CapsuleSigResult sr)
- CapsuleRunResult capsule_birth_mama (void *mama_vm, const CapsuleDirHeader *dir, const CapsuleDesc *descs, const CapsuleNameEntry *names, const uint8_t *arena)

  capsule_birth_mama - Execute Mama's init capsule
- CapsuleRunResult capsule_birth_baby (const char *capsule_name, const CapsuleDirHeader *dir, const CapsuleDesc *descs, const CapsuleNameEntry *names, const uint8_t *arena, VMUuid parent, int skip↩ _pki_sig, VMUuid *out_vm_id, void **out_vm_ctx)

  capsule_birth_baby - Birth a new VM from a named (p) capsule
- CapsuleRunResult capsule_run_experiment (void *mama_vm, const char *capsule_name, const CapsuleDirHeader *dir, const CapsuleDesc *descs, const CapsuleNameEntry *names, const uint8_t *arena, uint64_t *out_run_id)

  capsule_run_experiment - Execute a named (e) capsule on Mama

Variables

- static CapsuleExecFn vm_exec_fn = 0

  capsule_birth.c - VM Birth Protocol Implementation (M7.1)
- static CapsuleDictHashFn vm_dict_hash_fn = 0
- static CapsuleVMAllocFn vm_alloc_fn = 0
- static vm_node_t * vm_registry_head = (void *)0
- static uint32_t vm_registry_count = 0

7.123.1 Typedef Documentation

7.123.1.1 vm_node_t

```
typedef struct vm_node vm_node_t
```

7.123.2 Function Documentation

7.123.2.1 capsule_birth_baby()

```
CapsuleRunResult capsule_birth_baby (
            const char * capsule_name,
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena,
            VMUuid parent,
            int skip_pki_sig,
            VMUuid * out_vm_id,
            void ** out_vm_ctx )
```

capsule_birth_baby - Birth a new VM from a named (p) capsule

Finds the capsule by colon-separated name, validates it, allocates a new VM, executes the capsule payload as IDENTITY, then (if present) executes unit.4th from the baby's block space as PERSONALITY.

Parameters

capsule_name	Colon-separated capsule name, e.g. "production:myvm.4th"
dir	Capsule directory header
descs	Capsule descriptor array
names	Capsule name entry array (parallel to descs)
arena	Capsule payload arena
parent	Who is birthing this VM (FABRIC-2.md ?H.12 step 7) – the caller's own VMUuid (e.g. vm->stadium_vm_id for a FORTH word handler), recorded on the new VM's Session.parent. Every current call site has one in scope, directly or one level up; traced live rather than assumed (checked all 6 call sites across mama_forth_words.c/capsule_console.c/ capsule_runcap.c/capsule_wirebind.c).

Parameters

skip_pki_sig	0 for every build-time capsule (the normal case – checked against the compile-time-baked signature array via capsule_get_signatures()). Non-zero only for RUNCAP (FABRIC-2.md ?F.6/F.18): a heap-built, single-entry directory sourced from a user's own thumbdrive has no entry in that array at all – index 0 would silently compare against whatever real capsule happens to occupy slot 0, which is not a security check, just a guaranteed-wrong one. Trust for that content comes from CERTVERIFY (a separate root, the user's own Zuse-signed cert) already having run before RUNCAP is ever called, not from this flag – this only skips a check that was never meaningful for that content in the first place. Deliberately a plain flag, not a new entry point, so the policy is one call-site decision, trivially reversible.
out_vm_id	Output: assigned VM ID
out_vm_ctx	Output: new VM context

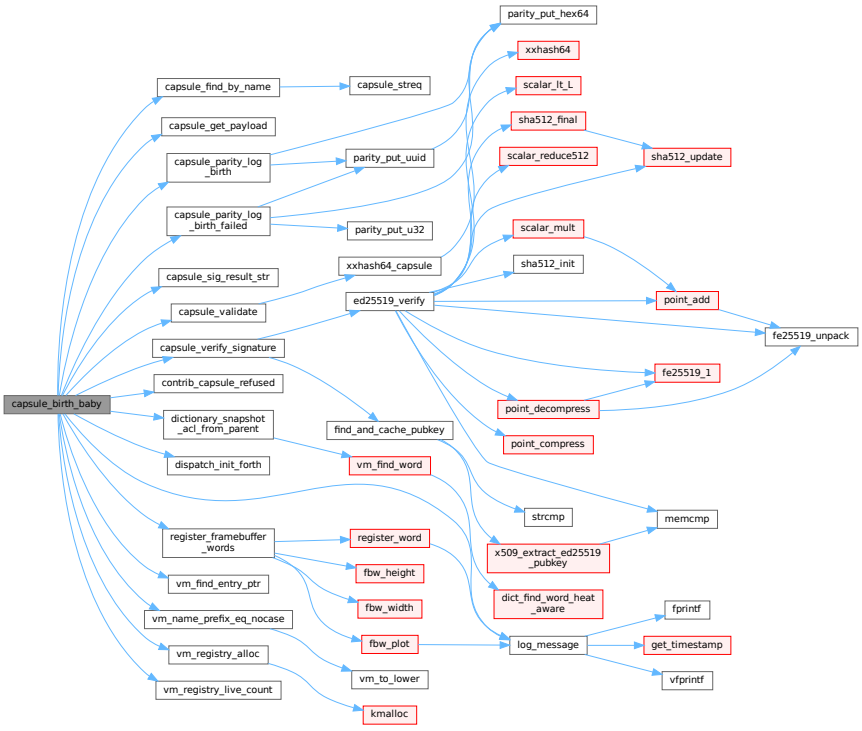
Returns

CAPSULE_RUN_OK on success, error code otherwise

References CapsuleDirHeader::arena_size, CAPSULE_BIRTH_ELIGIBLE, capsule_find_by_name(), capsule_get_payload(), capsule_parity_log_birth(), capsule_parity_log_birth_failed(), CAPSULE_RUN_ERR_EXEC_FAIL, CAPSULE_RUN_ERR_FLEET_FU CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_ERR_NOT_ELIGIBLE, CAPSULE_RUN_ERR_STILLBORN, CAPSULE_RUN_OK, CAPSULE_SIG_INVALID, CAPSULE_SIG_OK, capsule_sig_result_str(), CAPSULE_VALID, capsule_validate(), capsule_verify_signature(), CapsuleDesc, contrib_capsule_refused(), CapsuleDirHeader::desc_count, dictionary_snapshot_acl_from_parent(), dispatch_init_forth(), entry, log_message(), LOG_WARN, name, register_framebuffer_words(), vm_alloc_fn, vm_dict_hash_fn, vm_exec_fn, vm_find_entry_ptr(), vm_name_prefix_eq_nocase(), VMRegistryEntry::vm_ptr, vm_registry_alloc(), vm_registry_live_count(), VM_STATE_EMBRYO, VM_STATE_LIVE, and VM_STATE_STILLBORN.

Referenced by capsule_console_birth(), and capsule_runcap_birth().

Here is the call graph for this function:



Here is the caller graph for this function:

7.123.2.2 capsule_birth_mama()

```
CapsuleRunResult capsule_birth_mama (
            void * mama_vm,
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena )
```

capsule_birth_mama - Execute Mama's init capsule

Finds the MAMA_INIT capsule by flag, validates it, executes it on Mama's VM.

Parameters

mama_vm	Mama's VM context
dir	Capsule directory header
descs	Capsule descriptor array
names	Capsule name entry array (parallel to descs)
arena	Capsule payload arena

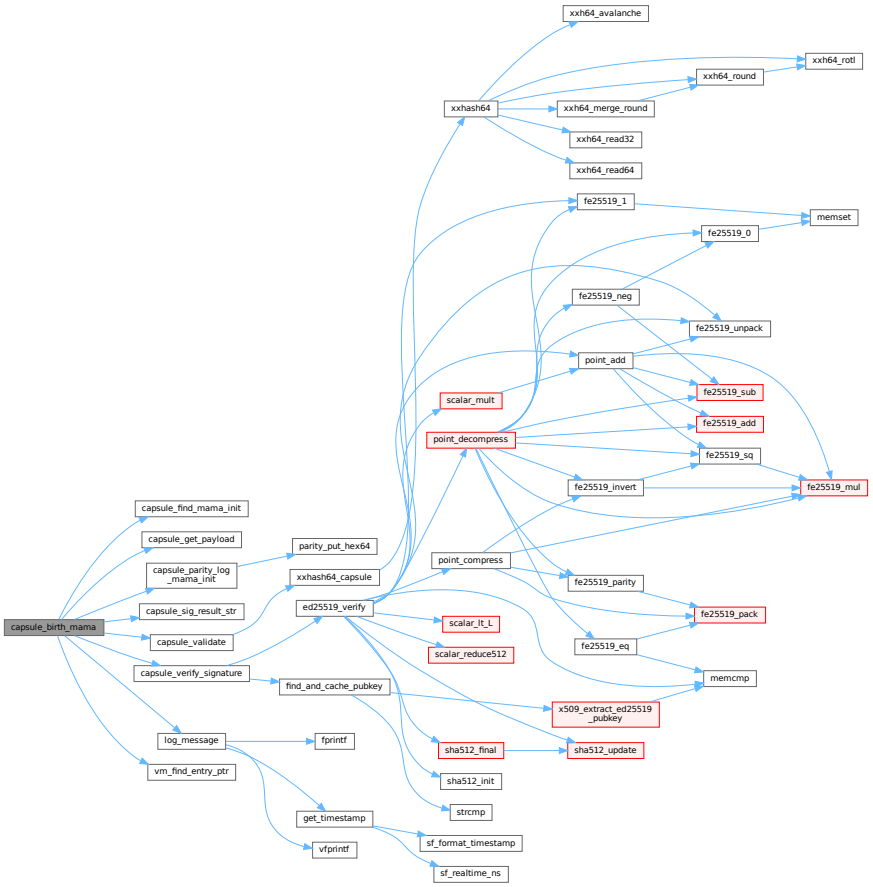
Returns

CAPSULE_RUN_OK on success, error code otherwise

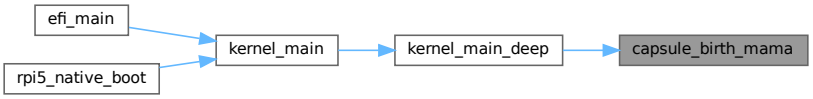
References CapsuleDirHeader::arena_size, VMRegistryEntry::birth_capsule_id, VMRegistryEntry::birth_dict_hash, capsule_find_mama_init(), capsule_get_payload(), capsule_parity_log_mama_init(), CAPSULE_RUN_ERR_EXEC_FAIL, CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_OK, CAPSULE_SIG_INVALID, CAPSULE_SIG_OK, capsule_sig_result_str(), CAPSULE_VALID, capsule_validate(), capsule_verify_signature(), CapsuleDesc, CapsuleDirHeader::desc_count, log_message(), LOG_WARN, name, vm_dict_hash_fn, vm_exec_fn, and vm_find_entry_ptr().

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.123.2.3 capsule_birth_set_hooks()

```
void capsule_birth_set_hooks (
            CapsuleExecFn exec_fn,
            CapsuleDictHashFn dict_hash_fn,
            CapsuleVMAllocFn vm_alloc_fn )
```

capsule_birth_set_hooks - Configure VM execution hooks

Must be called before any birth operations.

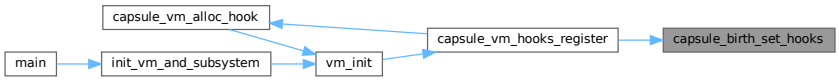
Parameters

exec_fn	Function to execute FORTH code on VM
dict_hash↩ _fn	Function to compute dictionary hash
vm_alloc_fn	Function to allocate new VM (for babies)

References vm_alloc_fn, vm_dict_hash_fn, and vm_exec_fn.

Referenced by capsule_vm_hooks_register().

Here is the caller graph for this function:



7.123.2.4 capsule_run_experiment()

```
CapsuleRunResult capsule_run_experiment (
            void * mama_vm,
            const char * capsule_name,
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena,
            uint64_t * out_run_id )
```

capsule_run_experiment - Execute a named (e) capsule on Mama

Finds the experiment capsule by colon-separated name, validates it, executes it on Mama's VM without creating a new VM.

Parameters

mama_vm	Mama's VM context
capsule_name	Colon-separated capsule name, e.g. "experiments:doe-l8:init-l8-stable.4th"
dir	Capsule directory header
descs	Capsule descriptor array
names	Capsule name entry array (parallel to descs)
arena	Capsule payload arena
out_run_id	Output: assigned run ID

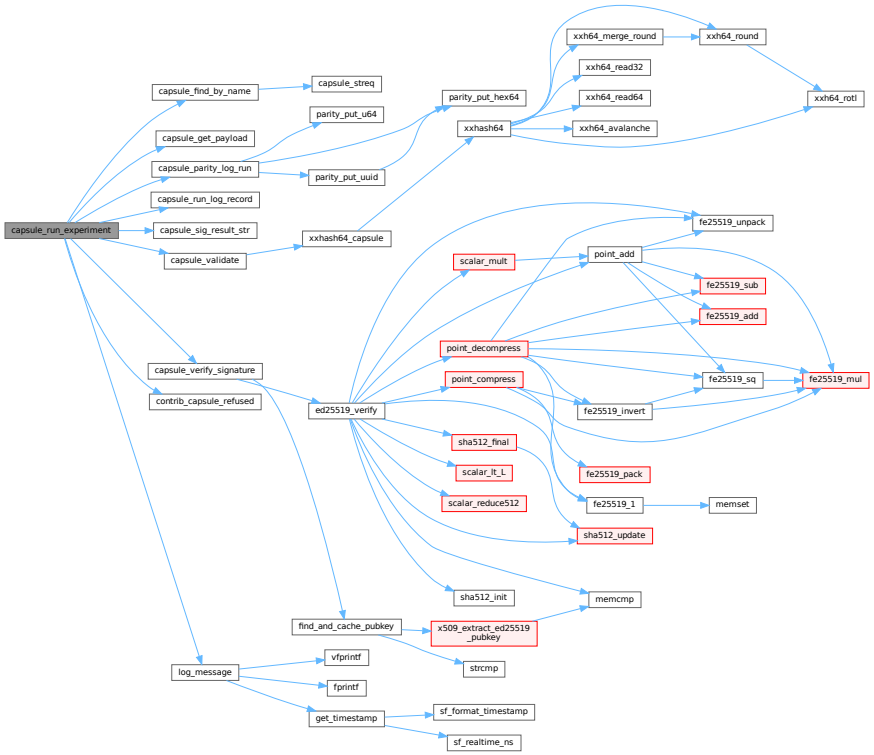
Returns

CAPSULE_RUN_OK on success, error code otherwise

References CapsuleDirHeader::arena_size, CAPSULE_DOE_ELIGIBLE, capsule_find_by_name(), capsule_get_payload(), CapsuleRunRecord::capsule_hash, CapsuleRunRecord::capsule_id, capsule_parity_log_run(), CAPSULE_RUN_ERR_EXEC_FAIL,

CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_ERR_NOT_ELIGIBLE, capsule_run_log_record(), CAPSULE_RUN_OK, CAPSULE_SIG_INVALID, CAPSULE_SIG_OK, capsule_sig_result_str(), CAPSULE_VALID, capsule_validate(), capsule_verify_signature(), CapsuleDesc, contrib_capsule_refused(), CapsuleDirHeader::desc_count, CapsuleRunRecord::ended_ns, CapsuleRunRecord::flags, log_message(), LOG_WARN, name, CapsuleRunRecord::post_dict_hash, CapsuleRunRecord::pre_dict_h, CapsuleRunRecord::reserved, CapsuleRunRecord::result_code, CapsuleRunRecord::run_id, CapsuleRunRecord::started_ns, vm_dict_hash_fn, vm_exec_fn, and CapsuleRunRecord::vm_id.

Here is the call graph for this function:



7.123.2.5 capsule_vm_find_by_name()

```
int capsule_vm_find_by_name (
            const char * name,
            VMRegistryEntry * out )
```

capsule_vm_find_by_name - Find VM registry entry by symbolic name

Case-sensitive. Returns the first match.

Parameters

name	Symbolic VM name, e.g. "Hermes"
out	Output: registry entry copy

Returns

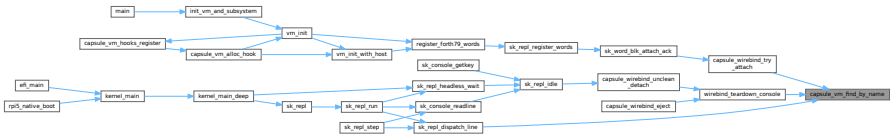
0 if found, -1 if not found

References vm_node::entry, VMRegistryEntry::name, name, vm_node::next, vm_name_eq(), and vm_registry_head.

Referenced by capsule_wirebind_try_attach(), sk_repl_dispatch_line(), and wirebind_teardown_console().

Here is the call graph for this function:

Here is the caller graph for this function:



7.123.2.6 capsule_vm_find_by_name_nocase()

```
int capsule_vm_find_by_name_nocase (
            const char * name,
            VMRegistryEntry * out )
```

capsule_vm_find_by_name_nocase - Find VM registry entry by name (case-insensitive)

Used by BIRTH for idempotency: prevents birthing a second VM with the same name regardless of case differences.

Parameters

name	Symbolic VM name (compared case-insensitively)
out	Output: registry entry copy

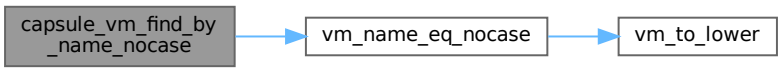
Returns

0 if found, -1 if not found

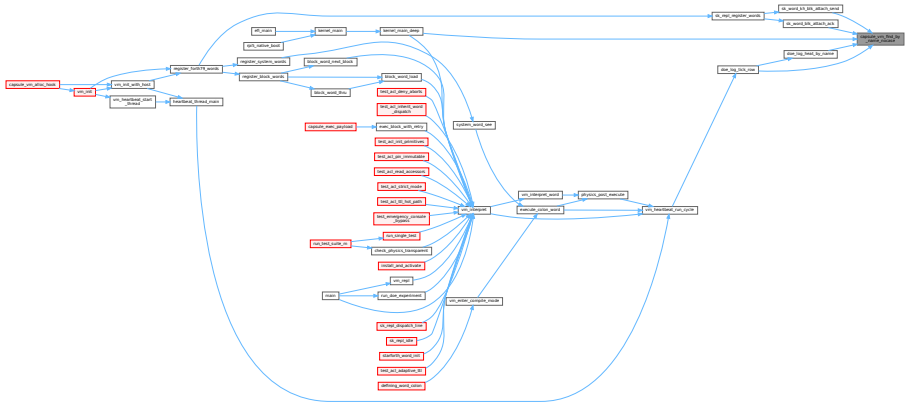
References vm_node::entry, VMRegistryEntry::name, name, vm_node::next, vm_name_eq_nocase(), and vm_registry_head.

Referenced by doe_log_heat_by_name(), doe_log_tick_row(), kernel_main_deep(), sk_word_blk_attach_ack(), and sk_word_kh_blk_attach_send().

Here is the call graph for this function:



Here is the caller graph for this function:



7.123.2.7 capsule_vm_force_reap()

```
void capsule_vm_force_reap (
            VMUuid vm_id )
```

capsule_vm_force_reap - FABRIC-3.md ?XXVIII Stage 4 (2026-09-14): unconditionally tear down vm_id regardless of VM_STATE_SWITCHED_OUT – the one caller allowed to bypass capsule_vm_kill()'s own refusal there, because this is called *by* the Stage 3 switcher itself (vm_core.c's checkpoint, on noticing pending_reap set), at the one point that genuinely knows the parked native-stack context will never be resumed.

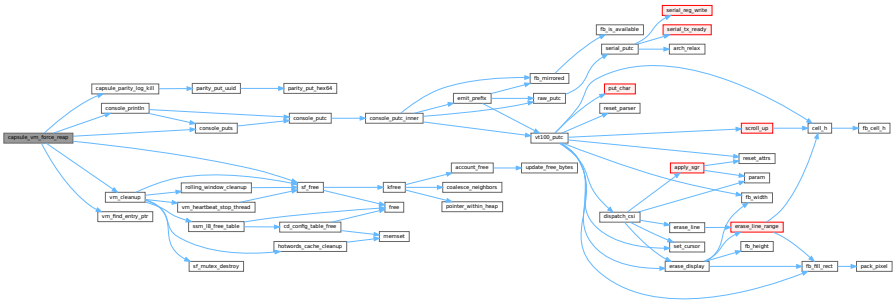
Also releases the VM's own switch-signal slot (sk_vm_switch_signal_unregister()) – generic cleanup, independent of whatever subsystem (WIREBIND today) set pending_reap in the first place. No-op if vm_id isn't registered or is already DEAD.

Parameters

vm_id	VM ID to reap.

References capsule_parity_log_kill(), console_println(), console_puts(), entry, sf_free(), vm_cleanup(), vm_find_entry_ptr(), VM_NAME_MAX, and VM_STATE_DEAD.

Here is the call graph for this function:



7.123.2.8 capsule_vm_kill()

```
int capsule_vm_kill (
            const char * name )
```

capsule_vm_kill - Destroy a named VM and release all its resources

Looks up the VM by name (case-insensitive). Hera (VM 0) cannot be killed. If the VM is already DEAD the call is a no-op. On success: vm_cleanup + sf_free, state → VM_STATE_DEAD, name cleared.

Parameters

name	Symbolic name of the VM to kill (case-insensitive)

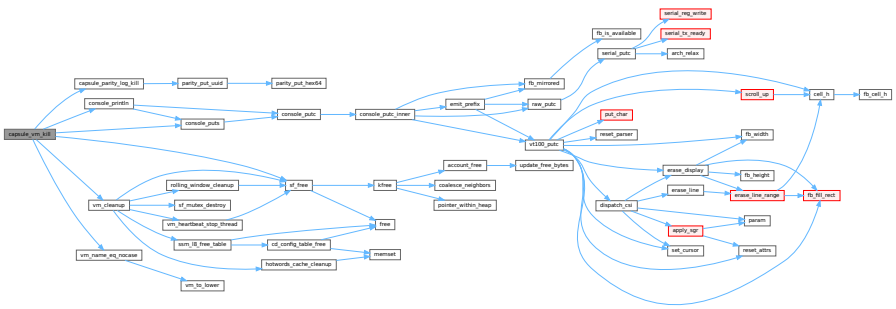
Returns

0 on success (or already dead), -1 if not found or refused

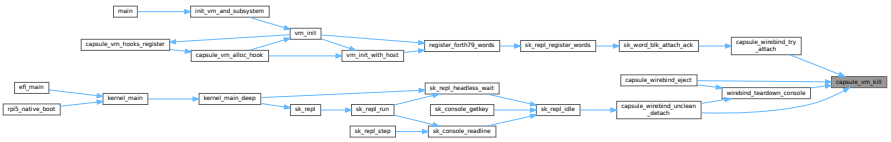
References capsule_parity_log_kill(), console_println(), console_puts(), entry, vm_node::entry, VMRegistryEntry::name, name, vm_node::next, sf_free(), vm_cleanup(), vm_name_eq_nocase(), VM_NAME_MAX, vm_registry_head, VM_STATE_DEAD, and VM_STATE_SWITCHED_OUT.

Referenced by capsule_wirebind_eject(), capsule_wirebind_try_attach(), capsule_wirebind_unclean_detach(), and wirebind_teardown_console().

Here is the call graph for this function:



Here is the caller graph for this function:



7.123.2.9 capsule_vm_kill_all_nonmama()

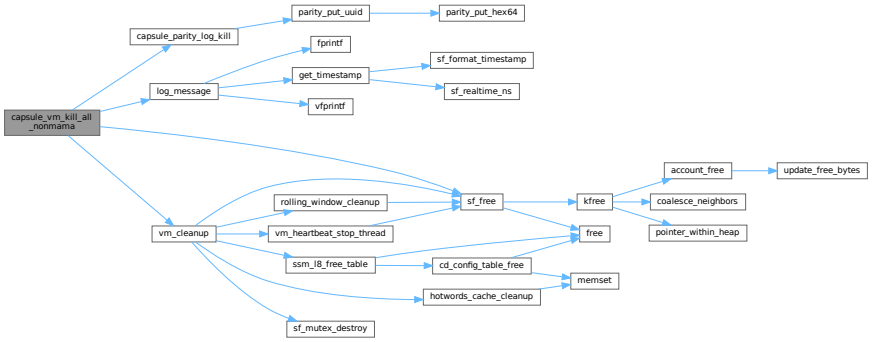
```
void capsule_vm_kill_all_nonmama (
            void  )
```

capsule_vm_kill_all_nonmama - Kill every non-Mama VM in the registry.

Sets halted, calls vm_cleanup + sf_free, marks state DEAD. Used by Hera's BYE immediately before arch_cold_reset() to reap all children.

References capsule_parity_log_kill(), vm_node::entry, VM::halted, log_message(), LOG_WARN, VMRegistryEntry::name, vm_node::next, sf_free(), VMRegistryEntry::stadium_patron_cell, VMRegistryEntry::state, vm_cleanup(), VMRegistryEntry::vm_id, VMRegistryEntry::vm_ptr, vm_registry_head, VM_STATE_DEAD, and VM_STATE_SWITCHED_OUT.

Here is the call graph for this function:



7.123.2.10 capsule_vm_registry_count()

```
uint32_t capsule_vm_registry_count (
            void  )
```

capsule_vm_registry_count - Get number of registered VMs

References vm_registry_count.

7.123.2.11 capsule_vm_registry_get()

```
int capsule_vm_registry_get (
            VMUuid vm_id,
            VMRegistryEntry * out )
```

capsule_vm_registry_get - Get VM registry entry by ID

Parameters

vm↩ _id	VM ID to look up
out	Output: registry entry copy

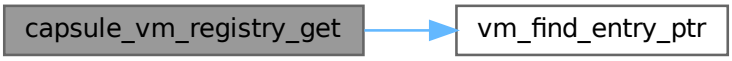
Returns

0 on success, -1 if not found

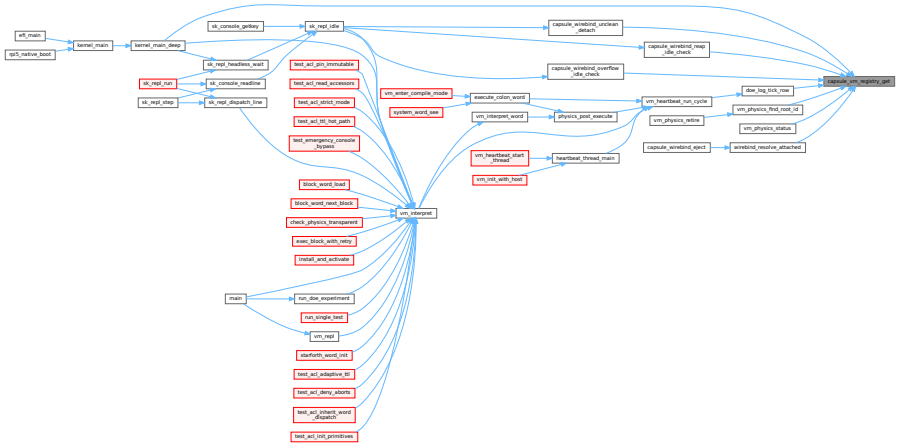
References entry, and vm_find_entry_ptr().

Referenced by capsule_wirebind_overflow_idle_check(), capsule_wirebind_reap_idle_check(), capsule_wirebind_unclean_detach(), doe_log_tick_row(), kernel_main_deep(), vm_physics_find_root_id(), vm_physics_status(), and wirebind_resolve_attached().

Here is the call graph for this function:



Here is the caller graph for this function:



7.123.2.12 capsule_vm_registry_get_by_index()

```
int capsule_vm_registry_get_by_index (
            uint32_t index,
            VMRegistryEntry * out )
```

capsule_vm_registry_get_by_index - Get registry entry by list position (birth order, stable within a boot session – the registry is append-only).

For enumeration (e.g. the idle-loop messaging pump, FABRIC-2.md Phase C, 2026-08-28), where no vm_id is known up front. Index range is [0, capsule_vm_registry_count()).

Parameters

index	Zero-based position in birth order
out	Output: registry entry copy

Returns

0 on success, -1 if index is out of range

References vm_node::entry, vm_node::next, and vm_registry_head.

7.123.2.13 capsule_vm_registry_init()

```
void capsule_vm_registry_init (
            void * mama_vm_ptr )
```

capsule_vm_registry_init - Initialize VM registry

Allocates Mama's registry node (VM 0, name "Hera") via kmalloc and stores mama_vm_ptr so KILL can guard against destroying Mama. Must be called after kmalloc_init().

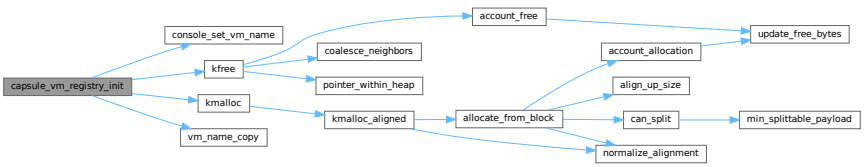
Parameters

mama_vm_ptr	Pointer to Mama's VM object (e.g. &sk_mama_vm)

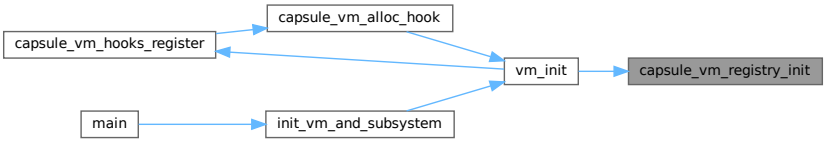
References VMRegistryEntry::birth_capsule_id, VMRegistryEntry::birth_dict_hash, VMRegistryEntry::birth_timestamp_ns, console_set_vm_name(), vm_node::entry, VMRegistryEntry::flags, kfree(), kmalloc(), VMRegistryEntry::name, vm_node::next, VMRegistryEntry::parent_vm_id, VMRegistryEntry::stadium_patron_cell, VMRegistryEntry::state, VMRegistryEntry::vm_id, vm_name_copy(), VM_NAME_MAX, VMRegistryEntry::vm_ptr, vm_registry_count, vm_registry_head, and VM_STATE_LIVE.

Referenced by vm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.123.2.14 capsule_vm_registry_set_name()

```
void capsule_vm_registry_set_name (
            VMUuid vm_id,
            const char * name )
```

capsule_vm_registry_set_name - Assign a symbolic name to a registered VM

Truncates to VM_NAME_MAX-1 characters. No-op if vm_id not found.

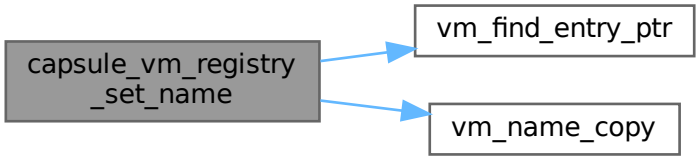
Parameters

vm↩ _id	VM ID to name
name	Symbolic name string

References entry, name, vm_find_entry_ptr(), and vm_name_copy().

Referenced by capsule_console_birth(), and capsule_runcap_birth().

Here is the call graph for this function:



Here is the caller graph for this function:

7.123.2.15 capsule_vm_set_pending_reap()

```
void capsule_vm_set_pending_reap (
            VMUuid vm_id,
            int pending )
```

capsule_vm_set_pending_reap - FABRIC-3.md ?XXVIII Stage 4 (2026-09-14): mark vm_id for deferred teardown once it is no longer worth resuming – see VMRegistryEntry.pending_reap's own doc comment for the full rationale.

No-op if vm_id isn't registered.

Parameters

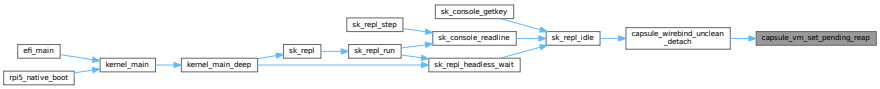
vm_id	VM ID to mark.
pending	1 to mark, 0 to clear (e.g. a re-attach of the same identity before the switcher ever reaped it).

References entry, and vm_find_entry_ptr().

Referenced by capsule_wirebind_unclean_detach().

Here is the call graph for this function:

Here is the caller graph for this function:



7.123.2.16 capsule_vm_set_state()

```
void capsule_vm_set_state (
            VMUuid vm_id,
            uint32_t state )
```

capsule_vm_set_state - Update a VM's state in the registry

Parameters

vm_id	VM ID to update
state	New VMState value

References entry, and vm_find_entry_ptr().

Here is the call graph for this function:

7.123.2.17 contrib_capsule_refused()

```
static int contrib_capsule_refused (
            uint32_t flags,
            CapsuleSigResult sr )  [static]
```

References CAPSULE_FLAG_CONTRIB, CAPSULE_SIG_OK, and flags.

Referenced by capsule_birth_baby(), and capsule_run_experiment().

Here is the caller graph for this function:

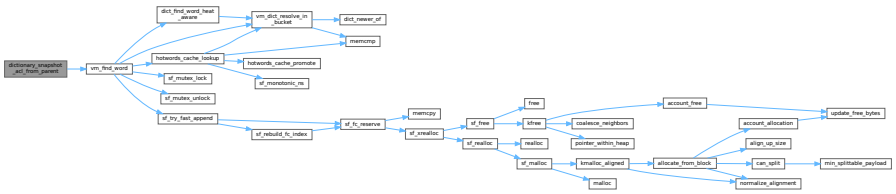
7.123.2.18 dictionary_snapshot_acl_from_parent()

```
static void dictionary_snapshot_acl_from_parent (
            VM * child,
            VM * parent )  [static]
```

References DictEntry::acl_allow, DictEntry::acl_mode, DictEntry::acl_pinned, DictEntry::acl_ttl, VM::latest, DictEntry::link, DictEntry::name, DictEntry::name_len, and vm_find_word().

Referenced by capsule_birth_baby().

Here is the call graph for this function:



Here is the caller graph for this function:

7.123.2.19 dispatch_init_forth()

```
static void dispatch_init_forth (
            void * vm_ctx )  [static]
```

Referenced by capsule_birth_baby().

Here is the caller graph for this function:

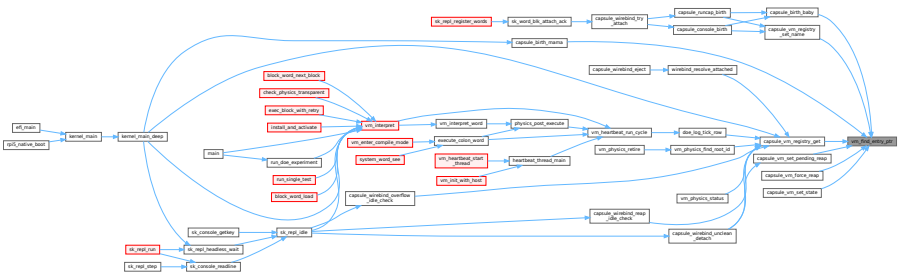
7.123.2.20 vm_find_entry_ptr()

```
static VMRegistryEntry * vm_find_entry_ptr (
            VMUuid vm_id )  [static]
```

References vm_node::entry, vm_node::next, VMRegistryEntry::vm_id, and vm_registry_head.

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_vm_force_reap(), capsule_vm_registry_get(), capsule_vm_registry_set_name(), capsule_vm_set_pending_reap(), and capsule_vm_set_state().

Here is the caller graph for this function:



7.123.2.21 vm_name_copy()

```
static void vm_name_copy (
            char * dst,
            const char * src )  [static]
```

References VM_NAME_MAX.

Referenced by capsule_vm_registry_init(), and capsule_vm_registry_set_name().

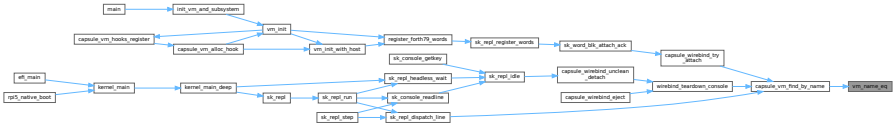
Here is the caller graph for this function:

7.123.2.22 vm_name_eq()

```
static int vm_name_eq (
            const char * a,
            const char * b )  [static]
```

Referenced by capsule_vm_find_by_name().

Here is the caller graph for this function:



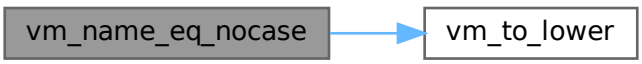
7.123.2.23 vm_name_eq_nocase()

```
static int vm_name_eq_nocase (
            const char * a,
            const char * b )  [static]
```

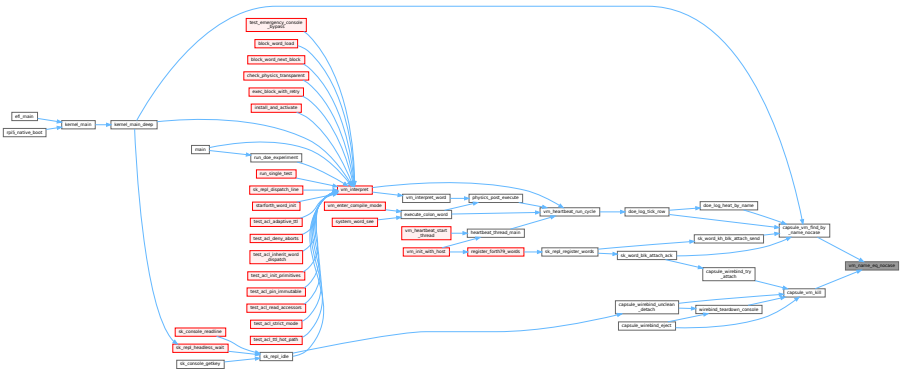
References vm_to_lower().

Referenced by capsule_vm_find_by_name_nocase(), and capsule_vm_kill().

Here is the call graph for this function:



Here is the caller graph for this function:



7.123.2.24 vm_name_prefix_eq_nocase()

```
static int vm_name_prefix_eq_nocase (
            const char * str,
            const char * prefix )  [static]
```

References vm_to_lower().

Referenced by capsule_birth_baby().

Here is the call graph for this function:

Here is the caller graph for this function:

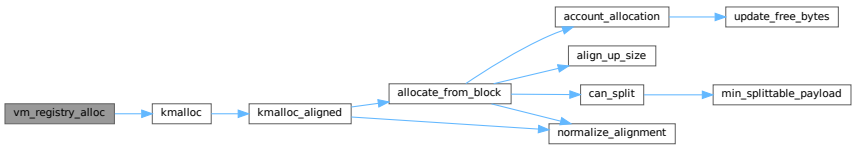
7.123.2.25 vm_registry_alloc()

```
static VMRegistryEntry * vm_registry_alloc (
            void  )  [static]
```

References VMRegistryEntry::birth_capsule_id, VMRegistryEntry::birth_dict_hash, VMRegistryEntry::birth_timestamp_ns, vm_node::entry, VMRegistryEntry::flags, kmalloc(), VMRegistryEntry::name, vm_node::next, VMRegistryEntry::parent_vm_id, VMRegistryEntry::stadium_patron_cell, VMRegistryEntry::state, VMRegistryEntry::vm_id, VM_NAME_MAX, VMRegistryEntry::vm_ptr, vm_registry_count, vm_registry_head, and VM_STATE_EMBRYO.

Referenced by capsule_birth_baby().

Here is the call graph for this function:



Here is the caller graph for this function:

7.123.2.26 vm_registry_live_count()

```
static uint32_t vm_registry_live_count (
            void  )  [static]
```

References vm_node::entry, vm_node::next, VMRegistryEntry::state, vm_registry_head, and VM_STATE_LIVE.

Referenced by capsule_birth_baby().

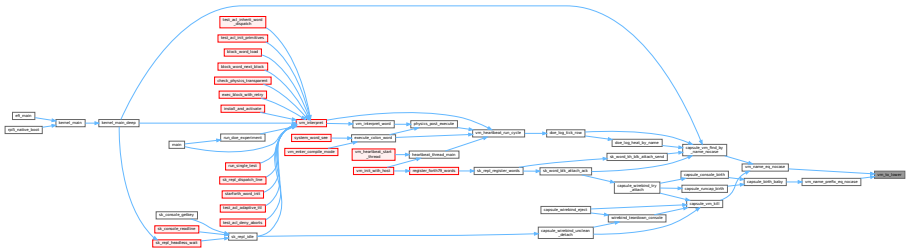
Here is the caller graph for this function:

7.123.2.27 vm_to_lower()

```
static char vm_to_lower (
            char c )  [static]
```

Referenced by vm_name_eq_nocase(), and vm_name_prefix_eq_nocase().

Here is the caller graph for this function:



7.123.3 Variable Documentation

7.123.3.1 vm_alloc_fn

```
CapsuleVMAllocFn vm_alloc_fn = 0  [static]
```

Referenced by capsule_birth_baby(), and capsule_birth_set_hooks().

7.123.3.2 vm_dict_hash_fn

```
CapsuleDictHashFn vm_dict_hash_fn = 0  [static]
```

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_birth_set_hooks(), and capsule_run_experiment().

7.123.3.3 vm_exec_fn

```
CapsuleExecFn vm_exec_fn = 0  [static]
```

capsule_birth.c - VM Birth Protocol Implementation (M7.1)

Mama init, baby birth, and experiment execution. Freestanding - no libc dependency.

Birth sequence for a baby VM:

1. Find capsule by name (capsule_find_by_name)
2. Assert CAPSULE_BIRTH_ELIGIBLE
3. Validate content hash
4. vm_alloc_hook() — fresh VM
5. vm_exec_hook(payload) — IDENTITY (init capsule from Hera's store)
6. vm_exec_hook("1 LOAD") — PERSONALITY (baby's personal init.4th from block 1, if present)
7. Log parity record

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_birth_set_hooks(), and capsule_run_experiment().

7.123.3.4 vm_registry_count

```
uint32_t vm_registry_count = 0  [static]
```

Referenced by capsule_vm_registry_count(), capsule_vm_registry_init(), and vm_registry_alloc().

7.123.3.5 vm_registry_head

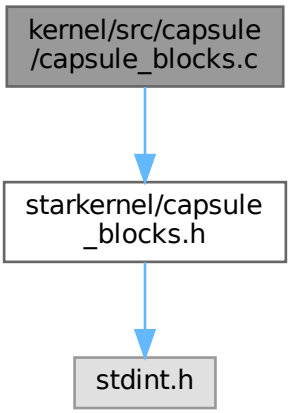
```
vm_node_t* vm_registry_head = (void *)0  [static]
```

Referenced by capsule_vm_find_by_name(), capsule_vm_find_by_name_nocase(), capsule_vm_kill(), capsule_vm_kill_all_nonmama capsule_vm_registry_get_by_index(), capsule_vm_registry_init(), vm_find_entry_ptr(), vm_registry_alloc(), and vm_registry_live_count().

7.124 kernel/src/capsule/capsule_blocks.c File Reference

```
#include "starkernel/capsule_blocks.h"
```

Include dependency graph for capsule_blocks.c:



Functions

- int capsule_block_header (const uint8_t *p, const uint8_t *end, uint32_t *out_num, const uint8_t **out_after)

7.124.1 Function Documentation

7.124.1.1 capsule_block_header()

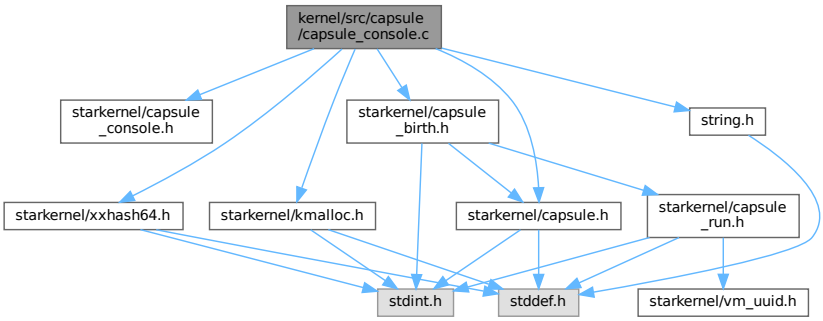
```
int capsule_block_header (
            const uint8_t * p,
            const uint8_t * end,
            uint32_t * out_num,
            const uint8_t ** out_after )
```

7.125 kernel/src/capsule/capsule_console.c File Reference

```
#include "starkernel/capsule_console.h"
#include "starkernel/capsule.h"
#include "starkernel/capsule_birth.h"
#include "starkernel/xxhash64.h"
#include "starkernel/kmalloc.h"
```

```
#include <string.h>
```

Include dependency graph for capsule_console.c:



Functions

- CapsuleRunResult capsule_console_birth (const char ∗console_name, VMUuid parent, VMUuid ∗out_vm_id, void ∗∗out_vm_ctx)

Variables

- static const char CONSOLE_IDENTITY_SRC [ ]

7.125.1 Function Documentation

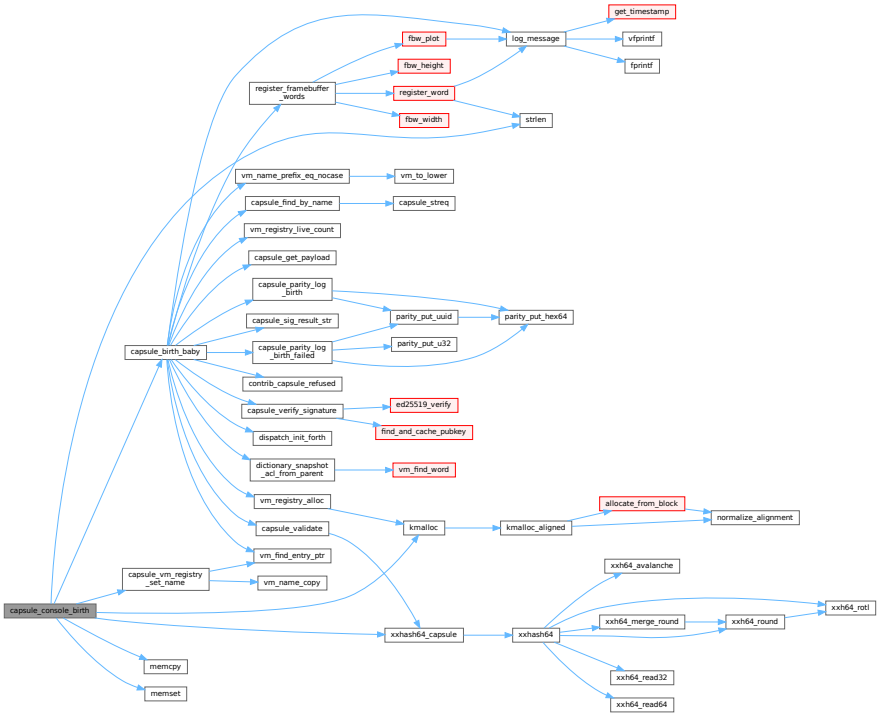
7.125.1.1 capsule_console_birth()

```
CapsuleRunResult capsule_console_birth (
            const char * console_name,
            VMUuid parent,
            VMUuid * out_vm_id,
            void ** out_vm_ctx )
```

References CapsuleDirHeader::arena_base, CapsuleDirHeader::arena_size, capsule_birth_baby(), CAPSULE_DIR_MAGIC, CAPSULE_FLAG_ACTIVE, CAPSULE_FLAG_PRODUCTION, CAPSULE_HASH_XXHASH64, CAPSULE_MAGIC_PACK, CAPSULE_NAME_MAX, CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_ERR_STILLBORN, CAPSULE_RUN_OK, CAPSULE_VERSION_0, capsule_vm_registry_set_name(), CapsuleDesc, CONSOLE_IDENTITY_SRC, CapsuleDirHeader::desc_capacity, CapsuleDirHeader::desc_count, CapsuleDirHeader::dir_hash, kmalloc(), CapsuleDirHeader::magic, memcpy(), memset(), CapsuleNameEntry::name, CapsuleDirHeader::name_count, strlen(), and xxhash64_capsule().

Referenced by capsule_wirebind_try_attach().

Here is the call graph for this function:



Here is the caller graph for this function:

7.125.2 Variable Documentation

7.125.2.1 CONSOLE_IDENTITY_SRC

```
const char CONSOLE_IDENTITY_SRC[]  [static]
```

Initial value:

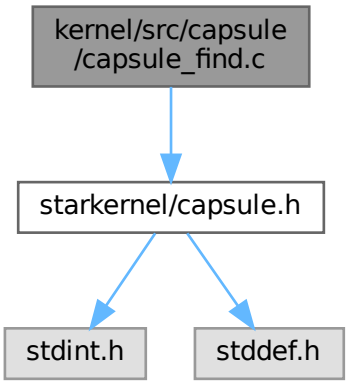
```
=
    "Block 4997\n"
```

Referenced by capsule_console_birth().

7.126 kernel/src/capsule/capsule_find.c File Reference

```
#include "starkernel/capsule.h"
```

Include dependency graph for capsule_find.c:



Functions

- static int capsule_streq (const char *a, const char *b)

  capsule_find.c - Capsule Lookup (M7.1)
- const CapsuleDesc * capsule_find_by_id (const CapsuleDirHeader *dir, const CapsuleDesc *descs, uint64_t id)

  capsule_find_by_id - Find capsule by content hash ID
- const CapsuleDesc * capsule_find_by_name (const CapsuleDirHeader *dir, const CapsuleDesc *descs, const CapsuleNameEntry *names, const char *name)

  capsule_find_by_name - Find capsule by colon-separated name
- const CapsuleDesc * capsule_find_mama_init (const CapsuleDirHeader *dir, const CapsuleDesc *descs)

  capsule_find_mama_init - Find the Mama init capsule
- const uint8_t * capsule_get_payload (const CapsuleDesc *desc, const uint8_t *arena_base)

  capsule_get_payload - Get pointer to capsule payload bytes

7.126.1 Function Documentation

7.126.1.1 capsule_find_by_id()

```
const CapsuleDesc * capsule_find_by_id (
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            uint64_t id )
```

capsule_find_by_id - Find capsule by content hash ID

Parameters

dir	Directory header
descs	Descriptor array
id	Capsule ID (content hash) to find

Returns

Pointer to descriptor, or NULL if not found

References CapsuleDirHeader::desc_count.

7.126.1.2 capsule_find_by_name()

```
const CapsuleDesc * capsule_find_by_name (
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const char * name )
```

capsule_find_by_name - Find capsule by colon-separated name

Parameters

dir	Directory header
descs	Descriptor array
names	Name entry array (parallel to descs)
name	Colon-separated capsule name, e.g. "core:init.4th"

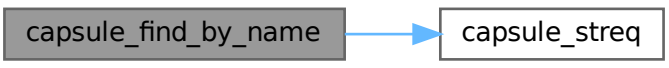
Returns

Pointer to descriptor, or NULL if not found

References capsule_streq(), CapsuleDirHeader::desc_count, and name.

Referenced by capsule_birth_baby(), capsule_exec_init(), and capsule_run_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

7.126.1.3 capsule_find_mama_init()

```
const CapsuleDesc * capsule_find_mama_init (
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs )
```

capsule_find_mama_init - Find the Mama init capsule

Searches the descriptor array for the capsule with CAPSULE_FLAG_MAMA_INIT. There must be exactly one such capsule.

Parameters

dir	Directory header
descs	Descriptor array

Returns

Pointer to Mama's init descriptor, or NULL if not found

References CAPSULE_IS_MAMA_INIT, CapsuleDirHeader::desc_count, and flags.

Referenced by capsule_birth_mama(), and kernel_main_deep().

Here is the caller graph for this function:

7.126.1.4 capsule_get_payload()

```
const uint8_t * capsule_get_payload (
            const CapsuleDesc * desc,
            const uint8_t * arena_base )
```

capsule_get_payload - Get pointer to capsule payload bytes

Parameters

desc	Capsule descriptor
arena_base	Base address of payload arena

Returns

Pointer to payload bytes, or NULL on error

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_exec_init(), and capsule_run_experiment().

Here is the caller graph for this function:

7.126.1.5 capsule_streq()

```
static int capsule_streq (
            const char * a,
            const char * b )  [static]
```

capsule_find.c - Capsule Lookup (M7.1)

Lookup functions for capsule descriptors by ID, name, and role. Freestanding - no libc dependency.

Referenced by capsule_find_by_name().

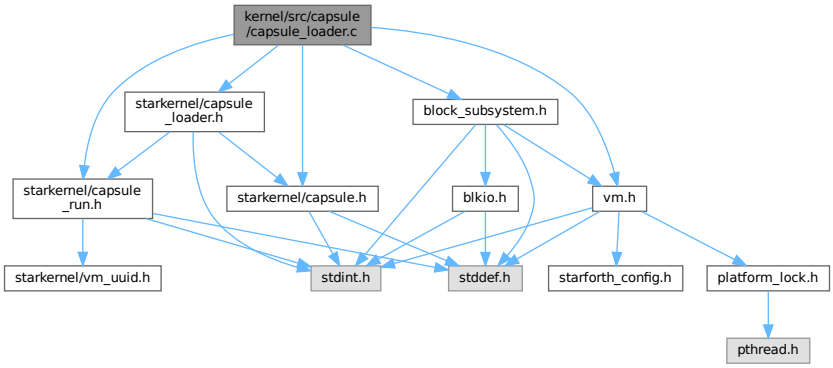
Here is the caller graph for this function:

7.127 kernel/src/capsule/capsule_loader.c File Reference

```
#include "starkernel/capsule_loader.h"
#include "starkernel/capsule.h"
#include "starkernel/capsule_run.h"
#include "block_subsystem.h"
```

```
#include "vm.h"
```

Include dependency graph for capsule_loader.c:



Macros

- #define CAPSULE_RAM_OFFSET 2048u

  capsule_loader.c - Block Capsule Loader and Executor (M7.1)
- #define LOADER_BLOCK_SIZE 1024u
- #define CAPSULE_BLOCK_LINES 16
- #define CAPSULE_LINE_COLS 64
- #define CAPSULE_EXEC_LINE_MAX 256
- #define CAPSULE_MAX_DEFERRED CAPSULE_BLOCK_LINES

Functions

- static int is_block_header (const uint8_t *p, const uint8_t *end, uint32_t *out_num, const uint8_t **out_after)
- static void write_ramdrive_block (uint32_t block_num, const uint8_t *content, uint32_t content_len)
- static size_t u32_to_dec (uint32_t val, char *buf, size_t size)
- int capsule_load_blocks (const uint8_t *payload, uint64_t length, uint32_t *out_entry_block)

  capsule_loader.h - Block Capsule Loader and Executor (M7.1)
- static int popcount16 (uint16_t v)
- static int exec_block_with_retry (VM *vm, const uint8_t *block_start, size_t block_len, uint32_t block_num)
- int capsule_exec_payload (void *vm_opaque, const uint8_t *payload, uint64_t length)

  capsule_exec_payload - Parse, populate block device, and execute a payload
- void capsule_clear_blocks (const uint8_t *payload, uint64_t length)

  capsule_clear_blocks - Zero all ramdrive slots referenced in a capsule payload
- CapsuleRunResult capsule_exec_init (void *vm_opaque, const char *capsule_name, const CapsuleDirHeader *dir, const CapsuleDesc *descs, const CapsuleNameEntry *names, const uint8_t *arena)

  capsule_exec_init - Load and execute an init capsule

7.127.1 Macro Definition Documentation

7.127.1.1 CAPSULE_BLOCK_LINES

```
#define CAPSULE_BLOCK_LINES 16
```

7.127.1.2 CAPSULE_EXEC_LINE_MAX

```
#define CAPSULE_EXEC_LINE_MAX 256
```

7.127.1.3 CAPSULE_LINE_COLS

```
#define CAPSULE_LINE_COLS 64
```

7.127.1.4 CAPSULE_MAX_DEFERRED

```
#define CAPSULE_MAX_DEFERRED CAPSULE_BLOCK_LINES
```

7.127.1.5 CAPSULE_RAM_OFFSET

```
#define CAPSULE_RAM_OFFSET 2048u
```

capsule_loader.c - Block Capsule Loader and Executor (M7.1)

Parses a capsule payload for "Block <num>" headers and implements the full init sequence:

1. Write blocks to ramdrive (2048+) — establishes device content so FORTH can call LOAD N later if needed.
2. Execute each block directly from payload bytes via exec_block_with_retry. exec_block_with_retry receives a C pointer into the payload; it does NOT use the block device, so no dedicated-RAM copy is needed here.
3. Zero dedicated RAM slots so userspace can reuse them.

The ramdrive (2048+) is the fallback device when no physical block device is mounted.

7.127.1.6 LOADER_BLOCK_SIZE

```
#define LOADER_BLOCK_SIZE 1024u
```

7.127.2 Function Documentation

7.127.2.1 capsule_clear_blocks()

```
void capsule_clear_blocks (
            const uint8_t * payload,
            uint64_t length )
```

capsule_clear_blocks - Zero all ramdrive slots referenced in a capsule payload

Re-parses the payload for "Block <num>" headers and zeros each corresponding 1024-byte ramdrive slot. NOT called by capsule_exec_init() (fixed 2026-09-10 – see its own doc comment): callers that specifically want a capsule's block range freed for reuse (e.g. kernel_main.c, right after Mama's own init.4th birth, to free that range for interactive block-editor use) call this themselves, explicitly, after exec returns.

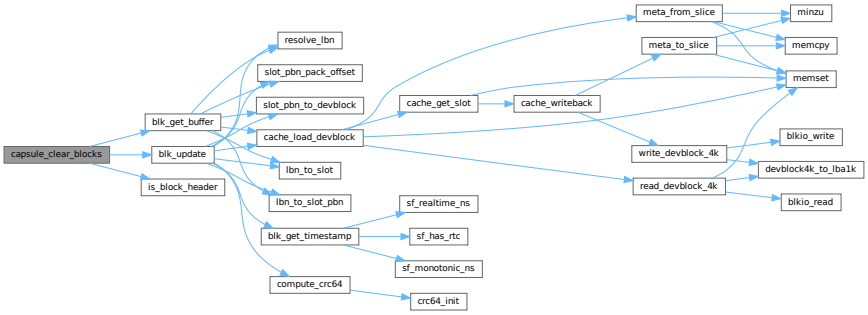
Parameters

payload	Raw capsule payload bytes
length	Payload length in bytes

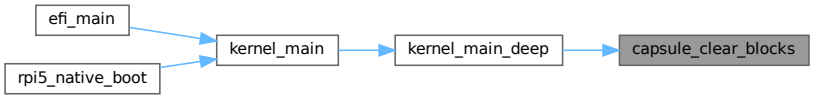
References blk_get_buffer(), blk_update(), is_block_header(), and LOADER_BLOCK_SIZE.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.127.2.2 capsule_exec_init()

```
CapsuleRunResult capsule_exec_init (
            void * vm,
            const char * capsule_name,
            const CapsuleDirHeader * dir,
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena )
```

capsule_exec_init - Load and execute an init capsule

Full init sequence:

1. Locate capsule by colon-separated name in the capsule directory
2. Validate content hash

3. capsule_exec_payload: populate block device + execute content

Block content is NOT cleared afterward (fixed 2026-09-10): it stays resident in ramdrive storage so a subsequent Standard BLOCK/LOAD on the same block number reads back what EXEC just wrote, matching FORTH-79 block-persistence semantics. Callers that specifically want the old "free this capsule's block range" behavior (e.g. kernel_main.c freeing init.4th's range for later interactive block-editor use) call capsule_clear_blocks() themselves, explicitly, after this returns.

Parameters

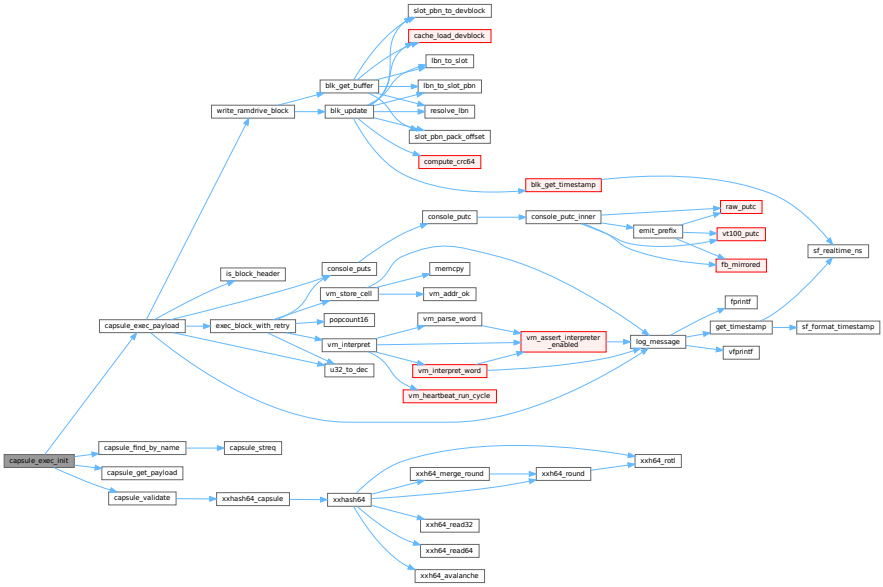
vm	VM to execute on
capsule_name	Colon-separated capsule name, e.g. "init.4th"
dir	Capsule directory header
descs	Capsule descriptor array
names	Capsule name entry array (parallel to descs)
arena	Capsule payload arena

Returns

CAPSULE_RUN_OK on success, error code otherwise

References CapsuleDirHeader::arena_size, capsule_exec_payload(), capsule_find_by_name(), capsule_get_payload(), CAPSULE_RUN_ERR_EXEC_FAIL, CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_OK, CAPSULE_VALID, capsule_validate(), and CapsuleDesc.

Here is the call graph for this function:



7.127.2.3 capsule_exec_payload()

```
int capsule_exec_payload (
            void * vm_opaque,
            const uint8_t * payload,
            uint64_t length )
```

capsule_exec_payload - Parse, populate block device, and execute a payload

For each "Block <num>" section: write content to block device (warn+truncate if > 1KB), then execute line-by-line via vm_interpret. No LOAD is injected.

Parameters

vm_opaque	VM to execute on
payload	Raw payload bytes
length	Payload length in bytes

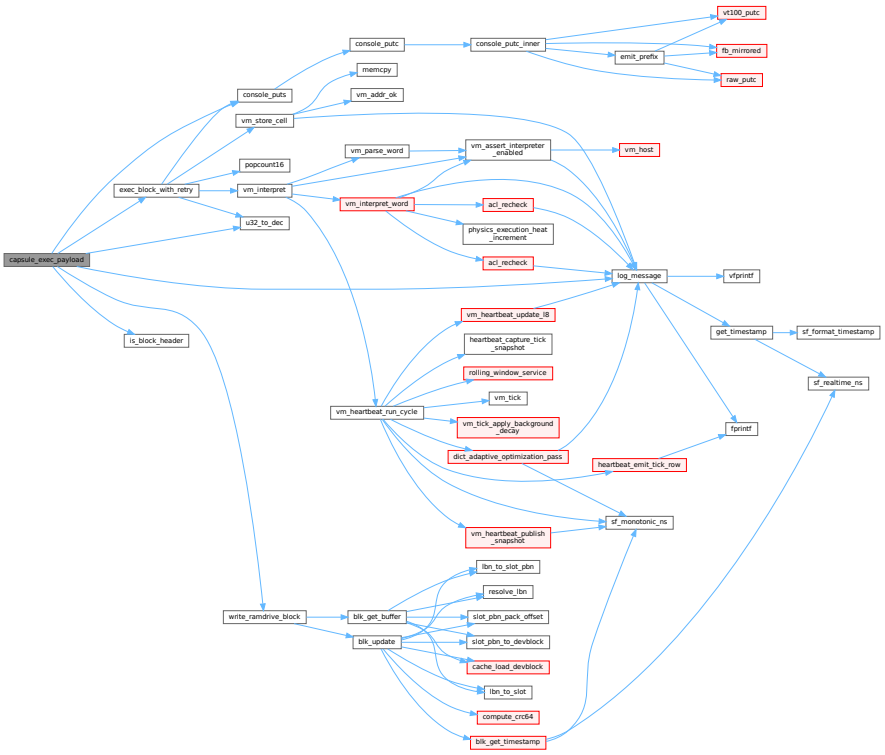
Returns

0 on success, -1 on execution error

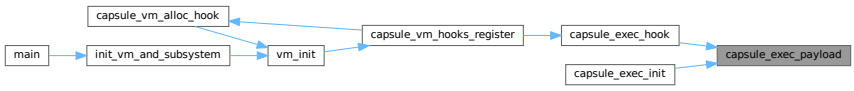
References console_puts(), exec_block_with_retry(), is_block_header(), LOADER_BLOCK_SIZE, LOG_INFO, log_message(), u32_to_dec(), and write_ramdrive_block().

Referenced by capsule_exec_hook(), and capsule_exec_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.127.2.4 capsule_load_blocks()

```
int capsule_load_blocks (
            const uint8_t * payload,
            uint64_t length,
            uint32_t * out_entry_block )
```

capsule_loader.h - Block Capsule Loader and Executor (M7.1)

Parses a capsule payload for "Block <num>" headers and writes each block's content into the correct ramdrive slot, then executes the entry block and zeros the ramdrive slots afterward.

Format: Block <decimal>
<content: up to 1024 bytes of FORTH source> Block <decimal>
...

Blocks may appear in any order; block numbers are arbitrary. Content exceeding 1024 bytes is truncated with a warning. capsule_load_blocks - Parse a capsule payload and write blocks to the ramdrive

For each "Block <num>" header found in the payload:

1. Zero the 1024-byte ramdrive slot for block <num>
2. Copy content bytes into the slot (excess beyond 1024 silently dropped)
3. Mark the slot dirty via blk_update()

Content between headers may be text or binary. Anything before the first "Block <num>" header is ignored.

Parameters

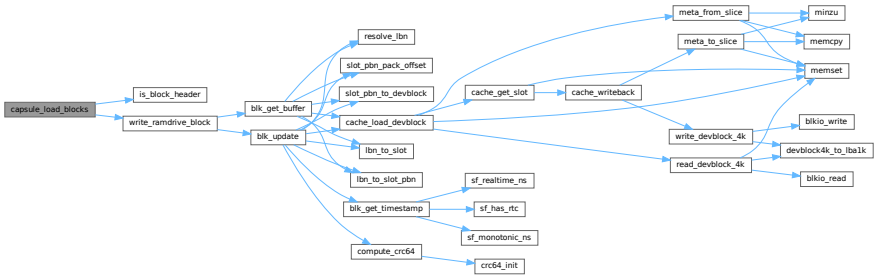
payload	Raw capsule payload bytes (not null-terminated)
length	Payload length in bytes
out_entry_block	If non-NULL, receives the first block number found (the execution entry point); set to 0 if no blocks found

Returns

Number of blocks written, or -1 on invalid input

References is_block_header(), and write_ramdrive_block().

Here is the call graph for this function:



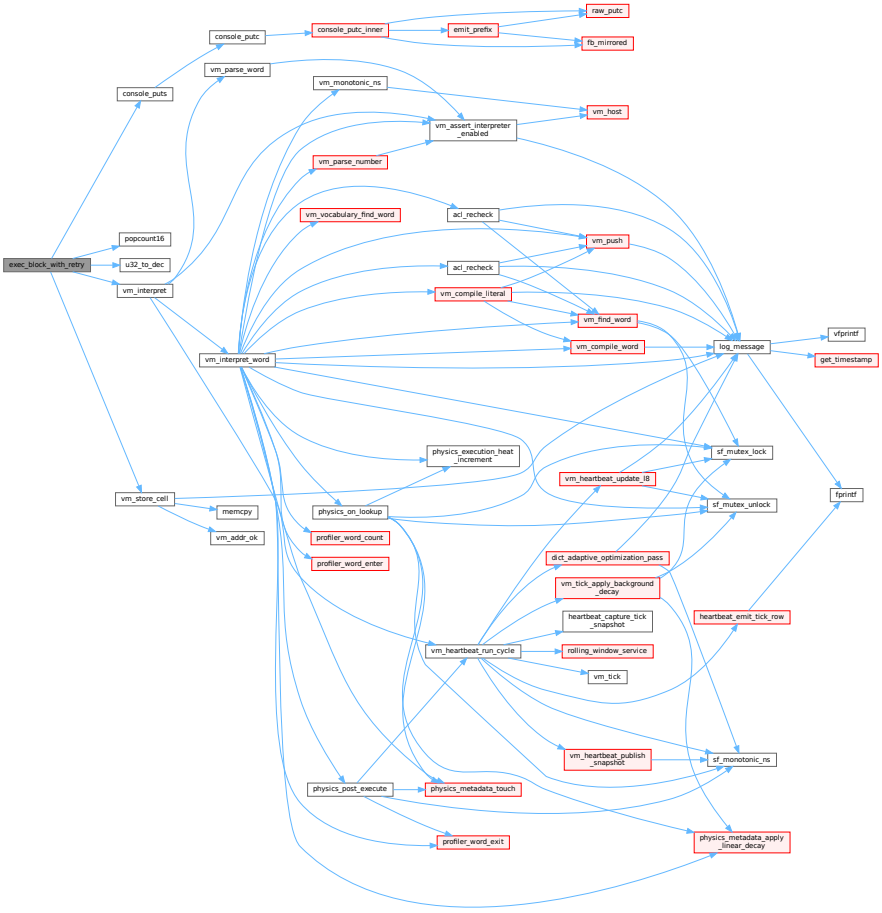
7.127.2.5 exec_block_with_retry()

```
static int exec_block_with_retry (
            VM * vm,
            const uint8_t * block_start,
            size_t block_len,
            uint32_t block_num )  [static]
```

References VM::abort_requested, CAPSULE_BLOCK_LINES, CAPSULE_EXEC_LINE_MAX, CAPSULE_LINE_COLS, console_puts(), VM::error, VM::mode, MODE_INTERPRET, popcount16(), VM::state_addr, VM::state_var, u32_to_dec(), vm_interpret(), and vm_store_cell().

Referenced by capsule_exec_payload().

Here is the call graph for this function:



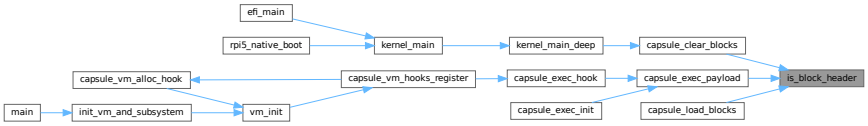
Here is the caller graph for this function:

7.127.2.6 is_block_header()

```
static int is_block_header (
            const uint8_t * p,
            const uint8_t * end,
            uint32_t * out_num,
            const uint8_t ** out_after )  [static]
```

Referenced by capsule_clear_blocks(), capsule_exec_payload(), and capsule_load_blocks().

Here is the caller graph for this function:

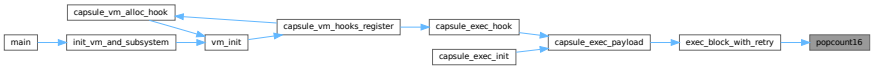


7.127.2.7 popcount16()

```
static int popcount16 (
            uint16_t v )  [static]
```

Referenced by exec_block_with_retry().

Here is the caller graph for this function:

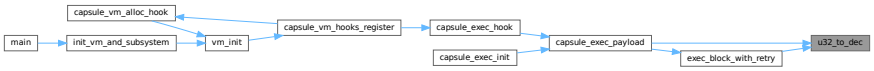


7.127.2.8 u32_to_dec()

```
static size_t u32_to_dec (
            uint32_t val,
            char * buf,
            size_t size )  [static]
```

Referenced by capsule_exec_payload(), and exec_block_with_retry().

Here is the caller graph for this function:



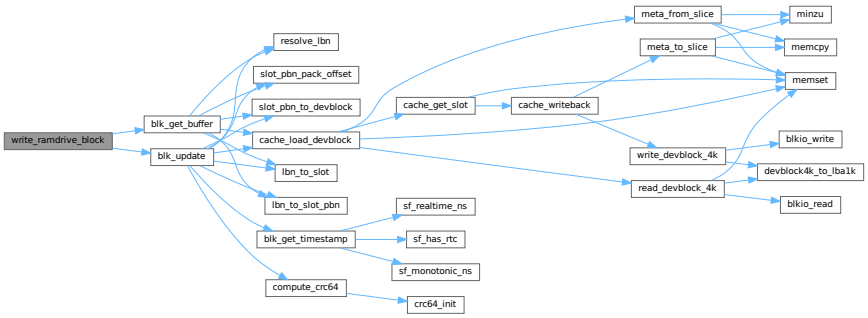
7.127.2.9 write_ramdrive_block()

```
static void write_ramdrive_block (
            uint32_t block_num,
            const uint8_t * content,
            uint32_t content_len )  [static]
```

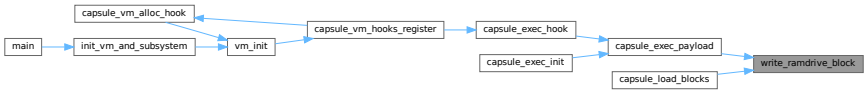
References blk_get_buffer(), blk_update(), and LOADER_BLOCK_SIZE.

Referenced by capsule_exec_payload(), and capsule_load_blocks().

Here is the call graph for this function:



Here is the caller graph for this function:

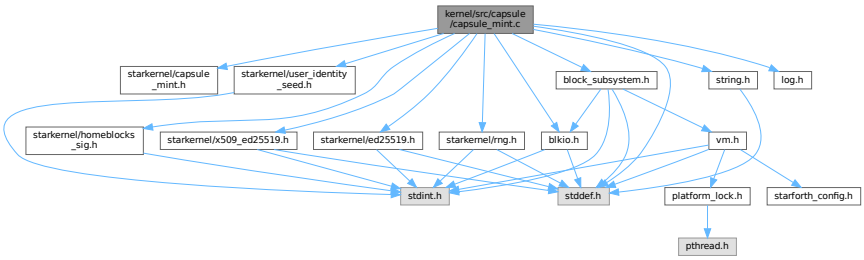


7.128 kernel/src/capsule/capsule_mint.c File Reference

```
#include "starkernel/capsule_mint.h"
#include "starkernel/homeblocks_sig.h"
#include "starkernel/user_identity_seed.h"
#include "starkernel/x509_ed25519.h"
#include "starkernel/ed25519.h"
#include "starkernel/rng.h"
#include "block_subsystem.h"
#include "blkio.h"
#include "log.h"
#include <string.h>
```

```
#include <stddef.h>
```

Include dependency graph for capsule_mint.c:



Macros

- #define MINT_CERT_DEVBLOCK 2u
- #define MINT_CERT_DEVBLOCKS 1u
- #define MINT_IDENTITY_SRC_OFFSET 3u
- #define MINT_IDENTITY_SRC_DEVBLOCKS 2u /∗ 1 seed record + 1 personality source ∗/
- #define MINT_METADATA_DEVBLOCKS 5u /∗ devblocks 0..4 reserved ∗/

Functions

- size_t blkio_ram_state_size (void)

  Return the byte size required for a `blkio_ram_state_t`.
- const blkio_vtable_t ∗ blkio_ram_vtable (void)

  Return the vtable for the RAM block I/O backend.
- int blkio_ram_init_state (void ∗state_mem, size_t state_len, uint8_t ∗base, uint32_t total_blocks, uint32_t fbs, uint8_t read_only, void ∗∗out_opaque)

  Initialise a `blkio_ram_state_t` in caller-provided memory.
- static int write_devblock (struct blkio_dev ∗dev, uint32_t devblock, const uint8_t ∗buf4096)
- static int read_devblock (struct blkio_dev ∗dev, uint32_t devblock, uint8_t ∗buf4096)
- static int verify_mint (struct blkio_dev ∗dev, uint32_t identity_src_offset, const user_identity_seed_t ∗written↩ _idrec)
- MintResult capsule_mint_identity (struct blkio_dev ∗dev, VM ∗issuer_vm, const char ∗full_name, const char ∗username, const char ∗email, const char ∗phone, uint8_t out_pubkey[32], uint8_t out_seed[32], Mint↩ Personality personality, int drive_known_blank)
- MintResult capsule_mint_identity_scratch (VM ∗issuer_vm, const char ∗full_name, const char ∗username, const char ∗email, const char ∗phone, MintPersonality personality, uint8_t out_uuid[16], uint8_t out_↩ cert[4096])

Variables

- static const char MINT_DEFAULT_PERSONALITY [ ]
- static const char MINT_RESTRICTED_PERSONALITY [ ]

7.128.1 Macro Definition Documentation

7.128.1.1 MINT_CERT_DEVBLOCK

```
#define MINT_CERT_DEVBLOCK 2u
```

7.128.1.2 MINT_CERT_DEVBLOCKS

```
#define MINT_CERT_DEVBLOCKS 1u
```

7.128.1.3 MINT_IDENTITY_SRC_DEVBLOCKS

```
#define MINT_IDENTITY_SRC_DEVBLOCKS 2u /* 1 seed record + 1 personality source */
```

7.128.1.4 MINT_IDENTITY_SRC_OFFSET

```
#define MINT_IDENTITY_SRC_OFFSET 3u
```

7.128.1.5 MINT_METADATA_DEVBLOCKS

```
#define MINT_METADATA_DEVBLOCKS 5u /* devblocks 0..4 reserved */
```

7.128.2 Function Documentation

7.128.2.1 blkio_ram_init_state()

```
int blkio_ram_init_state (
            void * state_mem,
            size_t state_len,
            uint8_t * base,
            uint32_t total_blocks,
            uint32_t fbs,
            uint8_t read_only,
            void ** out_opaque )  [extern]
```

Initialise a `blkio_ram_state_t` in caller-provided memory.

Stores `base`, `total_blocks`, `fbs` (defaulting to `BLKIO_FORTH_BLOCK_SIZE` if zero), and `read_only` into the state struct at `state_mem`. Sets `*out_opaque` to point to the state. The caller owns the `base` memory and must ensure it remains valid for the device lifetime.

Parameters

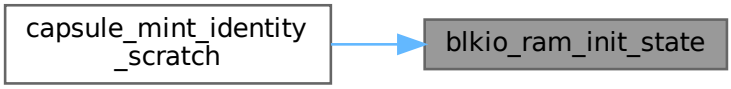
state_mem	Caller-allocated buffer (must be >= blkio_ram_state_size())
state_len	Size of `state_mem` in bytes
base	Pointer to the RAM block array
total_blocks	Number of FORTH blocks in `base`
fbs	FORTH block size in bytes (0 = use default)
read_only	1 = deny writes, 0 = allow reads and writes
out_opaque	Output: set to `state_mem` on success

Returns

`BLKIO_OK` on success, `BLKIO_EINVAL` on bad arguments

Referenced by capsule_mint_identity_scratch().

Here is the caller graph for this function:



7.128.2.2 blkio_ram_state_size()

```
size_t blkio_ram_state_size (
            void  )  [extern]
```

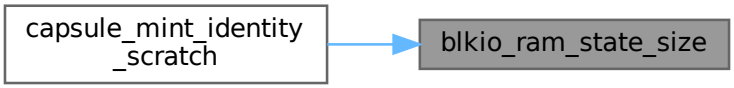
Return the byte size required for a `blkio_ram_state_t`.

Returns

sizeof(blkio_ram_state_t)

Referenced by capsule_mint_identity_scratch().

Here is the caller graph for this function:



7.128.2.3 blkio_ram_vtable()

```
const blkio_vtable_t * blkio_ram_vtable (
            void  )  [extern]
```

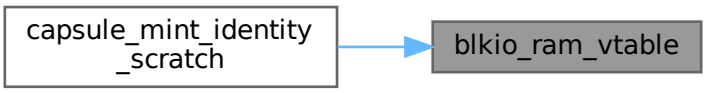
Return the vtable for the RAM block I/O backend.

Returns

Pointer to the static `blkio_vtable_t` for RAM devices

Referenced by capsule_mint_identity_scratch().

Here is the caller graph for this function:



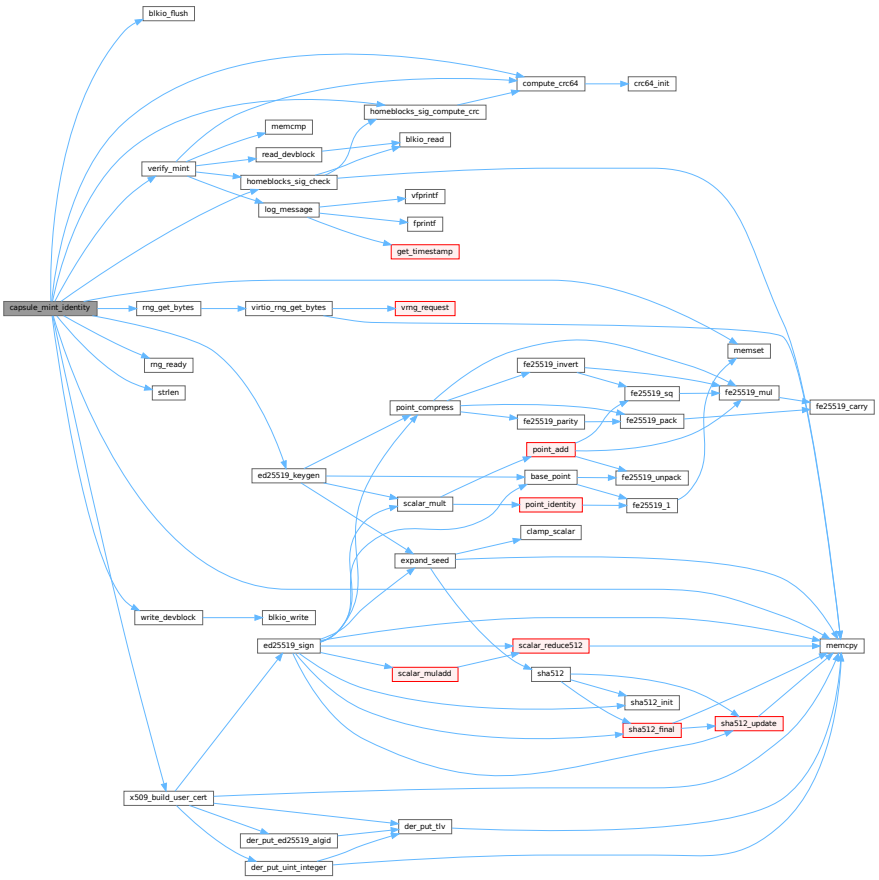
7.128.2.4 capsule_mint_identity()

```
MintResult capsule_mint_identity (
              struct blkio_dev * dev,
              VM * issuer_vm,
              const char * full_name,
              const char * username,
              const char * email,
              const char * phone,
              uint8_t out_pubkey[32],
              uint8_t out_seed[32],
              MintPersonality personality,
              int drive_known_blank )
```

References blkio_flush(), homeblocks_sig_t::cert_devblocks, homeblocks_sig_t::cert_offset, compute_crc64(), user_identity_seed_t::crc, homeblocks_sig_t::drive_uuid, ed25519_keygen(), user_identity_seed_t::email, user_identity_seed_t::full_name, homeblocks_sig_t::hdr_crc, homeblocks_sig_check(), homeblocks_sig_compute_crc(), HOMEBLOCKS_SIG_OK, HOMEBLOCKS_SIG_PACK, HOMEBLOCKS_SIG_START_FBLOCK, HOMEBLOCKS_SIG_VERSION_0, homeblocks_sig_t::identity_src_devblocks, homeblocks_sig_t::identity_src_offset, homeblocks_sig_t::magic, user_identity_seed_t::magic, memcpy(), memset(), homeblocks_sig_t::metadata_devblocks, MINT_CERT_DEVBLOCK, MINT_CERT_DEVBLOCKS, MINT_DEFAULT_PERSONALITY, MINT_IDENTITY_SRC_DEVBLOCKS, MINT_IDENTITY_SRC_OFF MINT_METADATA_DEVBLOCKS, MINT_RESTRICTED_PERSONALITY, homeblocks_sig_t::minted_time_ns, user_identity_seed_t::phone, user_identity_seed_t::pubkey, rng_get_bytes(), rng_ready(), user_identity_seed_t::seed, strlen(), USER_IDENTITY_EMAIL_MAX, USER_IDENTITY_FULL_NAME_MAX, USER_IDENTITY_PHONE_MAX, USER_IDENTITY_SEED_MAGIC, USER_IDENTITY_SEED_VERSION, USER_IDENTITY_USERNAME_MAX, user_identity_seed_t::username, verify_mint(), user_identity_seed_t::version, write_devblock(), x509_build_user_cert(), VM::zuse_cert_installed, and VM::zuse_cert_seed.

Referenced by capsule_mint_identity_scratch(), and capsule_zuse_boot_genesis_mint().

Here is the call graph for this function:



Here is the caller graph for this function:

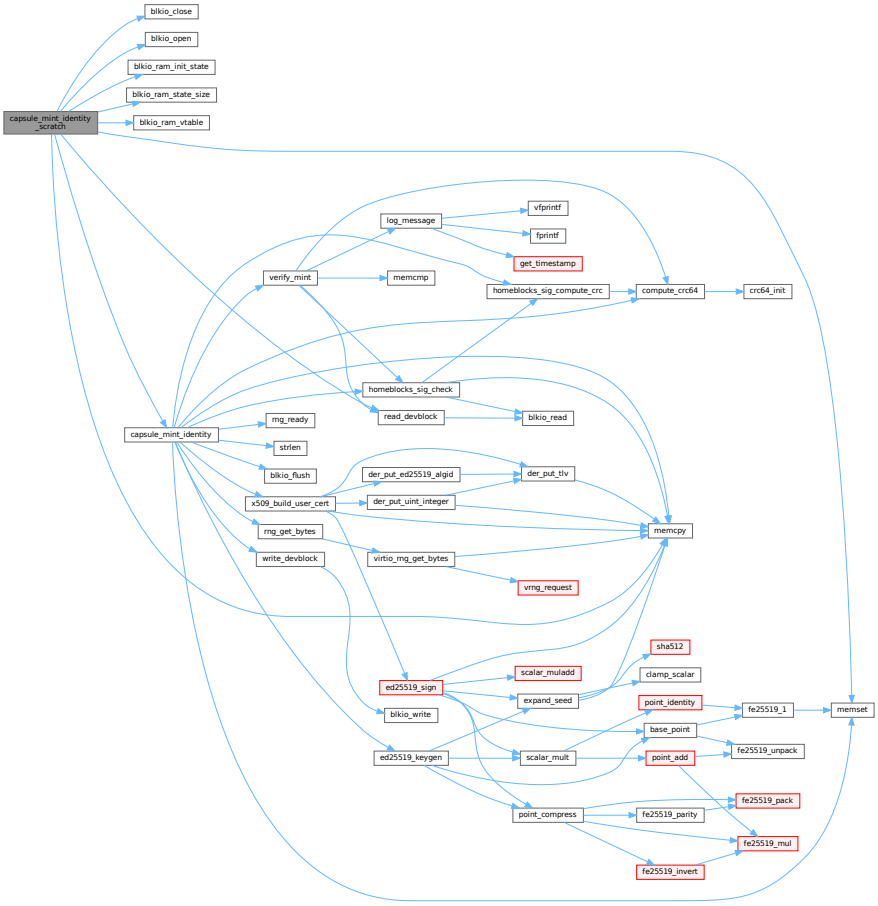
7.128.2.5 capsule_mint_identity_scratch()

```
MintResult capsule_mint_identity_scratch (
            VM * issuer_vm,
            const char * full_name,
            const char * username,
            const char * email,
            const char * phone,
            MintPersonality personality,
```

```
            uint8_t out_uuid[16],
            uint8_t out_cert[4096] )
```

References blkio_close(), BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_open(), blkio_ram_init_state(), blkio_ram_state_size(), blkio_ram_vtable(), capsule_mint_identity(), blkio_params_t::forth_block_size, HOMEBLOCKS_SIG_START_, memcpy(), memset(), MINT_CERT_DEVBLOCK, MINT_METADATA_DEVBLOCKS, and read_devblock().

Here is the call graph for this function:



7.128.2.6 read_devblock()

```
static int read_devblock (
            struct blkio_dev * dev,
            uint32_t devblock,
            uint8_t * buf4096 )  [static]
```

References base, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, and blkio_read().

Referenced by capsule_mint_identity_scratch(), and verify_mint().

Here is the call graph for this function:

Here is the caller graph for this function:

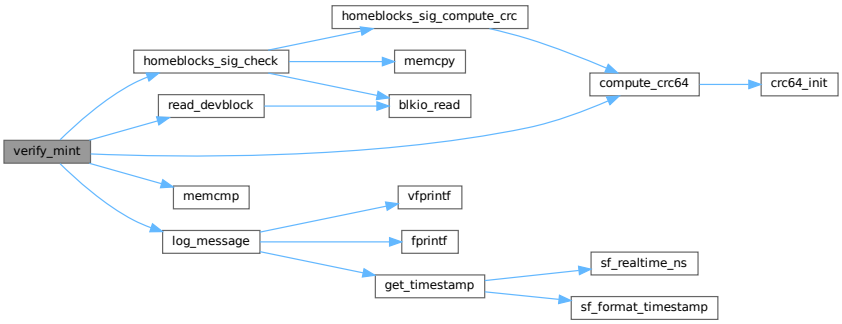
7.128.2.7 verify_mint()

```
static int verify_mint (
            struct blkio_dev * dev,
            uint32_t identity_src_offset,
            const user_identity_seed_t * written_idrec )  [static]
```

References compute_crc64(), user_identity_seed_t::crc, homeblocks_sig_check(), HOMEBLOCKS_SIG_OK, HOMEBLOCKS_SIG_START_FBLOCK, LOG_ERROR, LOG_INFO, log_message(), user_identity_seed_t::magic, memcmp(), read_devblock(), and user_identity_seed_t::version.

Referenced by capsule_mint_identity().

Here is the call graph for this function:



Here is the caller graph for this function:

7.128.2.8 write_devblock()

```
static int write_devblock (
            struct blkio_dev * dev,
            uint32_t devblock,
            const uint8_t * buf4096 )  [static]
```

References base, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, and blkio_write().

Referenced by capsule_mint_identity().

Here is the call graph for this function:

Here is the caller graph for this function:

7.128.3 Variable Documentation

7.128.3.1 MINT_DEFAULT_PERSONALITY

```
const char MINT_DEFAULT_PERSONALITY[]  [static]
```

Initial value:

```
=
    "Block 4999\n"
    ": WELCOME ( -- ) .\" Minted identity -- default personality\" CR ;\n"
    "WELCOME\n"
```

Referenced by capsule_mint_identity().

7.128.3.2 MINT_RESTRICTED_PERSONALITY

```
const char MINT_RESTRICTED_PERSONALITY[]  [static]
```

Initial value:

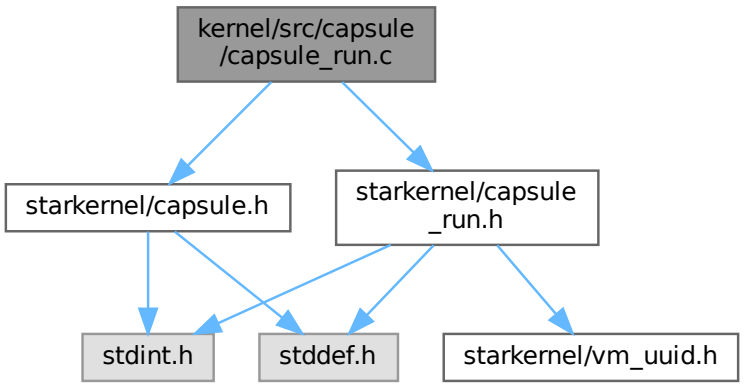
```
=
    "Block 4998\n"
    "S\" acl-std79.4th\" EXEC\n"
    "ACL-LOCKDOWN-STD79\n"
    ": WELCOME ( -- ) .\" Minted identity -- FORTH-79/83 standard words only\" CR ;\n"
    "WELCOME\n"
```

Referenced by capsule_mint_identity().

7.129 kernel/src/capsule/capsule_run.c File Reference

```
#include "starkernel/capsule_run.h"
#include "starkernel/capsule.h"
```

Include dependency graph for capsule_run.c:



Macros

- #define PARITY_HAVE_SINK (parity_puts != 0)

  capsule_run.c - DoE Run Logging and Parity (M7.1)
- #define PARITY_EMIT(s) do { if (parity_puts) parity_puts(s); } while (0)

Functions

- void capsule_run_log_init (void)

  capsule_run_log_init - Initialize run log
- uint64_t capsule_run_log_record (const CapsuleRunRecord *record)

  capsule_run_log_record - Log a run record
- int capsule_run_log_get (uint64_t run_id, CapsuleRunRecord *out)

  capsule_run_log_get - Get a run record by ID

- uint32_t capsule_run_log_count (void)

  capsule_run_log_count - Get number of logged runs
- void capsule_parity_set_output (void(*putc_fn)(char), void(*puts_fn)(const char *))

  capsule_parity_set_output - Set output hooks for parity logging
- static void parity_put_hex64 (uint64_t val)
- static void parity_put_uuid (VMUuid id)
- static void parity_put_u32 (uint32_t val)
- static void parity_put_u64 (uint64_t val)
- void capsule_parity_log_birth (VMUuid vm_id, uint64_t capsule_id, uint64_t capsule_hash, uint64_t dict_↩ hash)

  capsule_parity_log_birth - Log VM birth parity record
- void capsule_parity_log_birth_failed (VMUuid vm_id, uint64_t capsule_id, CapsuleRunResult error, uint64_t partial_dict_hash)

  capsule_parity_log_birth_failed - Log failed birth
- void capsule_parity_log_run (VMUuid vm_id, uint64_t run_id, uint64_t capsule_id, uint64_t pre_dict_hash, uint64_t post_dict_hash)

  capsule_parity_log_run - Log DoE run parity record
- void capsule_parity_log_mama_init (uint64_t capsule_id, uint64_t capsule_hash, uint64_t dict_hash)

  capsule_parity_log_mama_init - Log Mama init parity record
- void capsule_parity_log_kill (VMUuid vm_id, const char *name)

  capsule_parity_log_kill - Log VM kill parity record

Variables

- static CapsuleRunRecord run_log [1024]
- static uint32_t run_log_head = 0
- static uint32_t run_log_count = 0
- static uint64_t run_id_counter = 0
- static void(* parity_putc)(char c)=0
- static void(* parity_puts)(const char *s)=0

7.129.1 Macro Definition Documentation

7.129.1.1 PARITY_EMIT

```
#define PARITY_EMIT(
              s ) do { if (parity_puts) parity_puts(s); } while (0)
```

7.129.1.2 PARITY_HAVE_SINK

```
#define PARITY_HAVE_SINK (parity_puts != 0)
```

capsule_run.c - DoE Run Logging and Parity (M7.1)

Ring buffer for capsule execution logging and parity record emission. Freestanding - no libc dependency except for console output hooks.

7.129.2 Function Documentation

7.129.2.1 capsule_parity_log_birth()

```
void capsule_parity_log_birth (
            VMUuid vm_id,
            uint64_t capsule_id,
            uint64_t capsule_hash,
            uint64_t dict_hash )
```

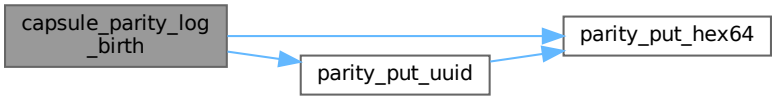
capsule_parity_log_birth - Log VM birth parity record

Output format: PARITY:BIRTH vm_id=N capsule_id=X mode=p capsule_hash=H dict_hash=D

References PARITY_EMIT, PARITY_HAVE_SINK, parity_put_hex64(), and parity_put_uuid().

Referenced by capsule_birth_baby().

Here is the call graph for this function:



Here is the caller graph for this function:

7.129.2.2 capsule_parity_log_birth_failed()

```
void capsule_parity_log_birth_failed (
            VMUuid vm_id,
            uint64_t capsule_id,
            CapsuleRunResult error,
            uint64_t partial_dict_hash )
```

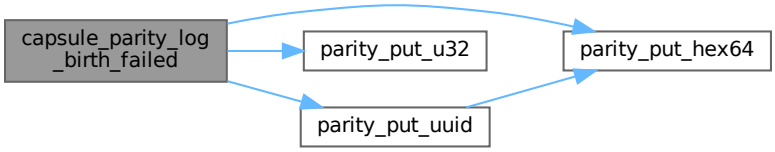
capsule_parity_log_birth_failed - Log failed birth

Output format: PARITY:BIRTH_FAILED vm_id=N capsule_id=X error=E partial_dict_hash=H

References PARITY_EMIT, PARITY_HAVE_SINK, parity_put_hex64(), parity_put_u32(), and parity_put_uuid().

Referenced by capsule_birth_baby().

Here is the call graph for this function:



Here is the caller graph for this function:

7.129.2.3 capsule_parity_log_kill()

```
void capsule_parity_log_kill (
            VMUuid vm_id,
            const char * name )
```

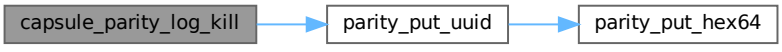
capsule_parity_log_kill - Log VM kill parity record

Output format: PARITY:KILL vm_id=N name=X

References name, PARITY_EMIT, PARITY_HAVE_SINK, and parity_put_uuid().

Referenced by capsule_vm_force_reap(), capsule_vm_kill(), and capsule_vm_kill_all_nonmama().

Here is the call graph for this function:



Here is the caller graph for this function:

7.129.2.4 capsule_parity_log_mama_init()

```
void capsule_parity_log_mama_init (
            uint64_t capsule_id,
            uint64_t capsule_hash,
            uint64_t dict_hash )
```

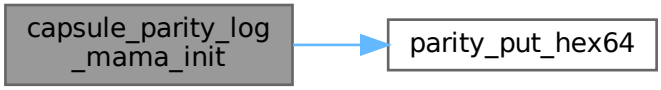
capsule_parity_log_mama_init - Log Mama init parity record

Output format: PARITY:MAMA_INIT capsule_id=X mode=m capsule_hash=H dict_hash=D

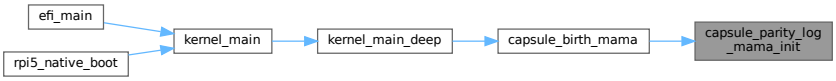
References PARITY_EMIT, PARITY_HAVE_SINK, and parity_put_hex64().

Referenced by capsule_birth_mama().

Here is the call graph for this function:



Here is the caller graph for this function:



7.129.2.5 capsule_parity_log_run()

```
void capsule_parity_log_run (
            VMUuid vm_id,
            uint64_t run_id,
            uint64_t capsule_id,
            uint64_t pre_dict_hash,
            uint64_t post_dict_hash )
```

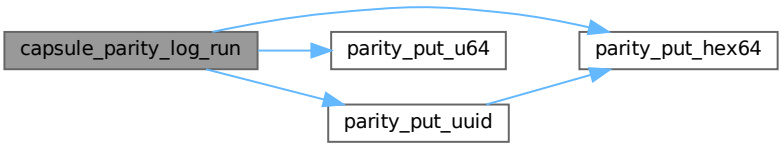
capsule_parity_log_run - Log DoE run parity record

Output format: PARITY:RUN vm_id=N run_id=R capsule_id=X mode=e pre_dict=P post_dict=Q

References PARITY_EMIT, PARITY_HAVE_SINK, parity_put_hex64(), parity_put_u64(), and parity_put_uuid().

Referenced by capsule_run_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

7.129.2.6 capsule_parity_set_output()

```
void capsule_parity_set_output (
            void(*)(char) putc_fn,
            void(*)(const char *) puts_fn )
```

capsule_parity_set_output - Set output hooks for parity logging

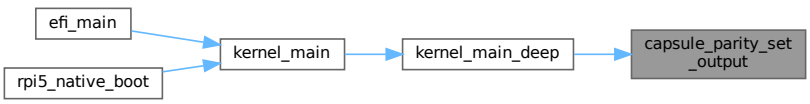
Parameters

putc↩ _fn	Function to output a single character
puts↩ _fn	Function to output a string

References parity_putc, and parity_puts.

Referenced by kernel_main_deep().

Here is the caller graph for this function:



7.129.2.7 capsule_run_log_count()

```
uint32_t capsule_run_log_count (
            void  )
```

capsule_run_log_count - Get number of logged runs

References run_log_count.

7.129.2.8 capsule_run_log_get()

```
int capsule_run_log_get (
            uint64_t run_id,
            CapsuleRunRecord * out )
```

capsule_run_log_get - Get a run record by ID

Parameters

run↩ _id	Run ID to retrieve
out	Output record

Returns

0 on success, -1 if not found

References CAPSULE_MAX_RUN_RECORDS, run_id_counter, and run_log.

7.129.2.9 capsule_run_log_init()

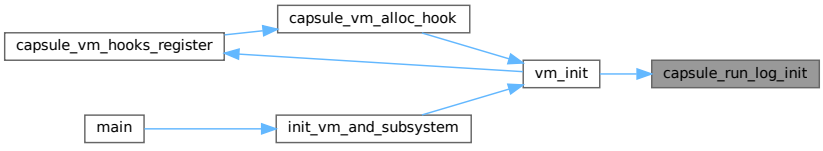
```
void capsule_run_log_init (
            void  )
```

capsule_run_log_init - Initialize run log

References CapsuleRunRecord::capsule_hash, CapsuleRunRecord::capsule_id, CAPSULE_MAX_RUN_RECORDS, CapsuleRunRecord::ended_ns, CapsuleRunRecord::flags, CapsuleRunRecord::post_dict_hash, CapsuleRunRecord::pre_dict_hash, CapsuleRunRecord::reserved, CapsuleRunRecord::result_code, CapsuleRunRecord::run_id, run_id_counter, run_log, run_log_count, run_log_head, CapsuleRunRecord::started_ns, and CapsuleRunRecord::vm_id.

Referenced by vm_init().

Here is the caller graph for this function:



7.129.2.10 capsule_run_log_record()

```
uint64_t capsule_run_log_record (
            const CapsuleRunRecord * record )
```

capsule_run_log_record - Log a run record

Parameters

record	Record to log

Returns

Run ID assigned, or 0 on failure

References CAPSULE_MAX_RUN_RECORDS, CapsuleRunRecord::run_id, run_id_counter, run_log, run_log_count, and run_log_head.

Referenced by capsule_run_experiment().

Here is the caller graph for this function:

7.129.2.11 parity_put_hex64()

```
static void parity_put_hex64 (
            uint64_t val )  [static]
```

References PARITY_EMIT.

Referenced by capsule_parity_log_birth(), capsule_parity_log_birth_failed(), capsule_parity_log_mama_init(), capsule_parity_log_run(), and parity_put_uuid().

Here is the caller graph for this function:

7.129.2.12 parity_put_u32()

```
static void parity_put_u32 (
            uint32_t val )  [static]
```

References PARITY_EMIT.

Referenced by capsule_parity_log_birth_failed().

Here is the caller graph for this function:

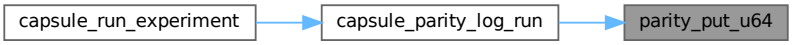
7.129.2.13 parity_put_u64()

```
static void parity_put_u64 (
            uint64_t val )  [static]
```

References PARITY_EMIT.

Referenced by capsule_parity_log_run().

Here is the caller graph for this function:



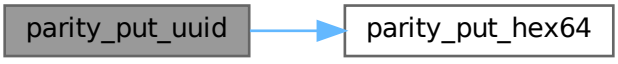
7.129.2.14 parity_put_uuid()

```
static void parity_put_uuid (
            VMUuid id )  [static]
```

References PARITY_EMIT, and parity_put_hex64().

Referenced by capsule_parity_log_birth(), capsule_parity_log_birth_failed(), capsule_parity_log_kill(), and capsule_parity_log_run().

Here is the call graph for this function:



Here is the caller graph for this function:

7.129.3 Variable Documentation

7.129.3.1 parity_putc

```
void(* parity_putc) (char c)=0 (
            char c )  [static]
```

Referenced by capsule_parity_set_output().

7.129.3.2 parity_puts

```
void(* parity_puts) (const char *s)=0 (
            const char * s )  [static]
```

Referenced by capsule_parity_set_output().

7.129.3.3 run_id_counter

```
uint64_t run_id_counter = 0  [static]
```

Referenced by capsule_run_log_get(), capsule_run_log_init(), and capsule_run_log_record().

7.129.3.4 run_log

```
CapsuleRunRecord run_log[1024]  [static]
```

Referenced by capsule_run_log_get(), capsule_run_log_init(), and capsule_run_log_record().

7.129.3.5 run_log_count

```
uint32_t run_log_count = 0  [static]
```

Referenced by capsule_run_log_count(), capsule_run_log_init(), and capsule_run_log_record().

7.129.3.6 run_log_head

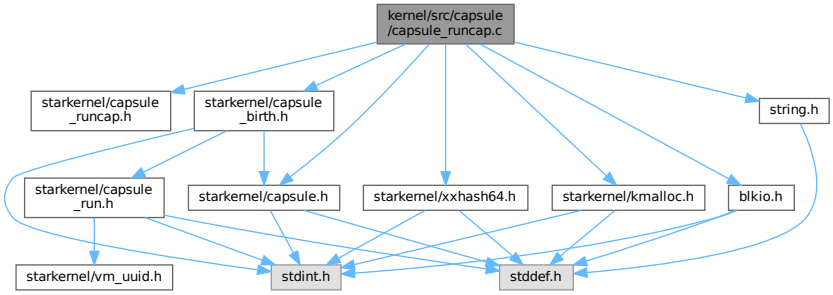
```
uint32_t run_log_head = 0  [static]
```

Referenced by capsule_run_log_init(), and capsule_run_log_record().

7.130 kernel/src/capsule/capsule_runcap.c File Reference

```
#include "starkernel/capsule_runcap.h"
#include "starkernel/capsule.h"
#include "starkernel/capsule_birth.h"
#include "starkernel/xxhash64.h"
#include "starkernel/kmalloc.h"
#include "blkio.h"
#include <string.h>
```

Include dependency graph for capsule_runcap.c:



Functions

- CapsuleRunResult capsule_runcap_birth (struct blkio_dev *dev, const homeblocks_sig_t *sig, const char *vm_name, VMUuid parent, VMUuid *out_vm_id, void **out_vm_ctx)

7.130.1 Function Documentation

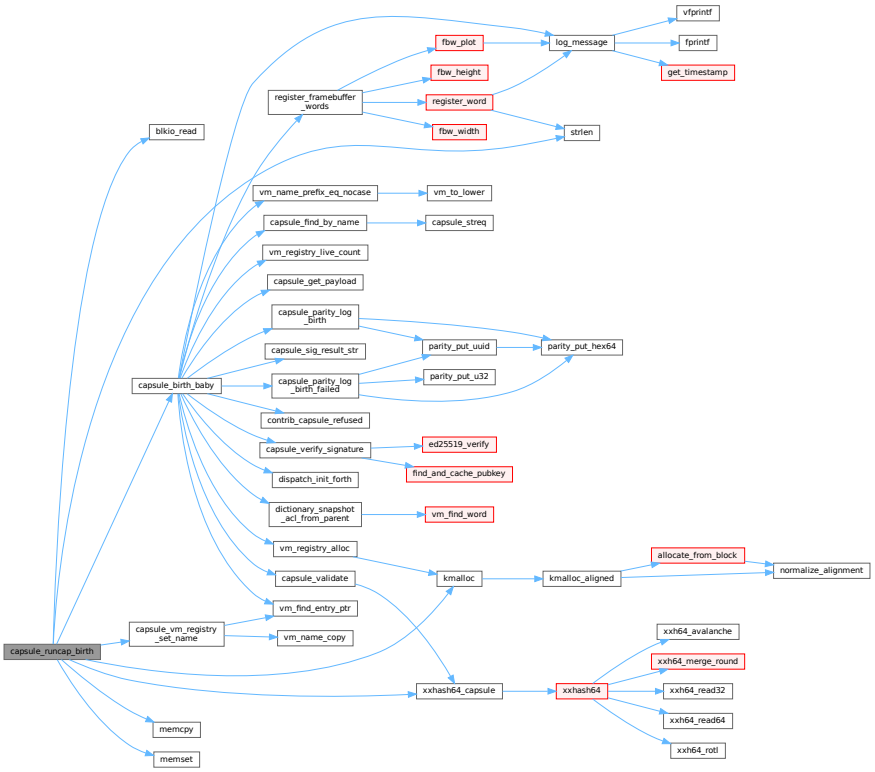
7.130.1.1 capsule_runcap_birth()

```
CapsuleRunResult capsule_runcap_birth (
            struct blkio_dev * dev,
            const homeblocks_sig_t * sig,
            const char * vm_name,
            VMUuid parent,
            VMUuid * out_vm_id,
            void ** out_vm_ctx )
```

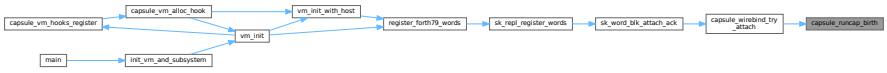
References CapsuleDirHeader::arena_base, CapsuleDirHeader::arena_size, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_read(), capsule_birth_baby(), CAPSULE_DIR_MAGIC, CAPSULE_FLAG_ACTIVE, CAPSULE_FLAG_PRODUCTI CAPSULE_HASH_XXHASH64, CAPSULE_MAGIC_PACK, CAPSULE_NAME_MAX, CAPSULE_RUN_ERR_INVALID, CAPSULE_RUN_ERR_STILLBORN, CAPSULE_RUN_OK, CAPSULE_VERSION_0, capsule_vm_registry_set_name(), CapsuleDesc, CapsuleDirHeader::desc_capacity, CapsuleDirHeader::desc_count, CapsuleDirHeader::dir_hash, homeblocks_sig_t::identity_src_devblocks, homeblocks_sig_t::identity_src_offset, kmalloc(), CapsuleDirHeader::magic, memcpy(), memset(), CapsuleNameEntry::name, CapsuleDirHeader::name_count, strlen(), and xxhash64_capsule().

Referenced by capsule_wirebind_try_attach().

Here is the call graph for this function:



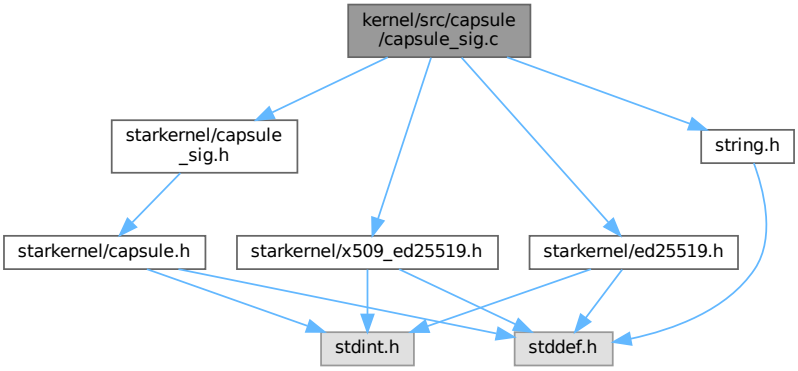
Here is the caller graph for this function:



7.131 kernel/src/capsule/capsule_sig.c File Reference

```
#include "starkernel/capsule_sig.h"
#include "starkernel/x509_ed25519.h"
#include "starkernel/ed25519.h"
#include <string.h>
```

Include dependency graph for capsule_sig.c:



Functions

- static int find_and_cache_pubkey (const CapsuleDesc *descs, const CapsuleNameEntry *names, const uint8_t *arena_base, uint32_t desc_count)
- CapsuleSigResult capsule_verify_signature (const CapsuleDesc *descs, const CapsuleNameEntry *names, const CapsuleSigEntry *sigs, const uint8_t *arena_base, uint32_t desc_count, int index)
- const char * capsule_sig_result_str (CapsuleSigResult result)

Variables

- static int pubkey_state = -1
- static uint8_t cached_pubkey [32]

7.131.1 Function Documentation

7.131.1.1 capsule_sig_result_str()

```
const char * capsule_sig_result_str (
            CapsuleSigResult result )
```

References CAPSULE_SIG_INVALID, CAPSULE_SIG_MISSING, CAPSULE_SIG_NO_ROOT_KEY, and CAPSULE_SIG_OK.

Referenced by capsule_birth_baby(), capsule_birth_mama(), and capsule_run_experiment().

Here is the caller graph for this function:

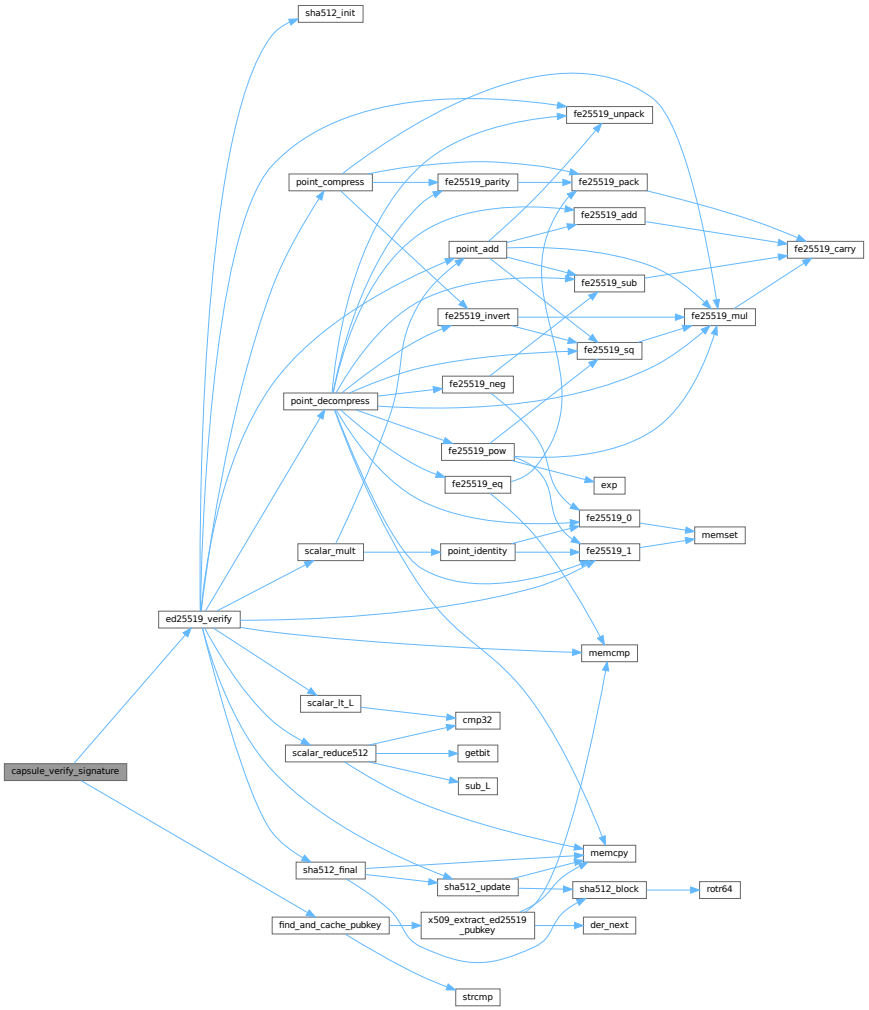
7.131.1.2 capsule_verify_signature()

```
CapsuleSigResult capsule_verify_signature (
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const CapsuleSigEntry * sigs,
            const uint8_t * arena_base,
            uint32_t desc_count,
            int index )
```

References cached_pubkey, CAPSULE_SIG_INVALID, CAPSULE_SIG_MISSING, CAPSULE_SIG_NO_ROOT_KEY, CAPSULE_SIG_OK, CapsuleDesc, ed25519_verify(), find_and_cache_pubkey(), and CapsuleSigEntry::sig.

Referenced by capsule_birth_baby(), capsule_birth_mama(), and capsule_run_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

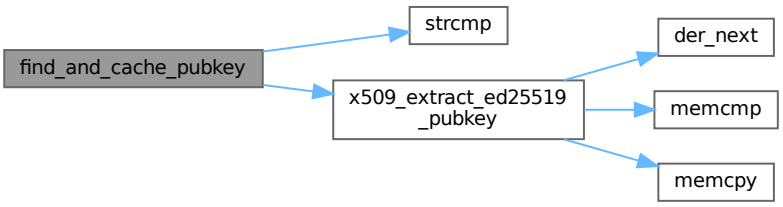
7.131.1.3 find_and_cache_pubkey()

```
static int find_and_cache_pubkey (
            const CapsuleDesc * descs,
            const CapsuleNameEntry * names,
            const uint8_t * arena_base,
            uint32_t desc_count )  [static]
```

References cached_pubkey, CAPSULE_SIGNING_CERT_NAME, CapsuleDesc, name, pubkey_state, strcmp(), and x509_extract_ed25519_pubkey().

Referenced by capsule_verify_signature().

Here is the call graph for this function:



Here is the caller graph for this function:

7.131.2 Variable Documentation

7.131.2.1 cached_pubkey

```
uint8_t cached_pubkey[32]  [static]
```

Referenced by capsule_verify_signature(), and find_and_cache_pubkey().

7.131.2.2 pubkey_state

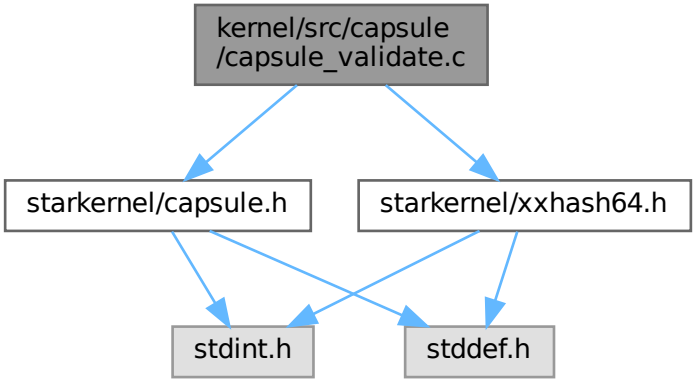
```
int pubkey_state = -1  [static]
```

Referenced by find_and_cache_pubkey().

7.132 kernel/src/capsule/capsule_validate.c File Reference

```
#include "starkernel/capsule.h"
#include "starkernel/xxhash64.h"
```

Include dependency graph for capsule_validate.c:



Functions

- const char * capsule_validate_result_str (CapsuleValidateResult result)

  capsule_validate_result_str - Get string for validation result
- CapsuleValidateResult capsule_validate (const CapsuleDesc *desc, const uint8_t *arena_base, uint64_t arena_size, int verify_hash)

  capsule_validate - Validate a capsule descriptor

Variables

- static const char * validate_result_strings [ ]

  capsule_validate.c - Capsule Validation (M7.1)

7.132.1 Function Documentation

7.132.1.1 capsule_validate()

```
CapsuleValidateResult capsule_validate (
            const CapsuleDesc * desc,
            const uint8_t * arena_base,
            uint64_t arena_size,
            int verify_hash )
```

capsule_validate - Validate a capsule descriptor

Parameters

desc	Capsule descriptor to validate
arena_base	Base address of payload arena
arena_size	Size of payload arena in bytes
verify_hash	If true, recompute and compare content hash

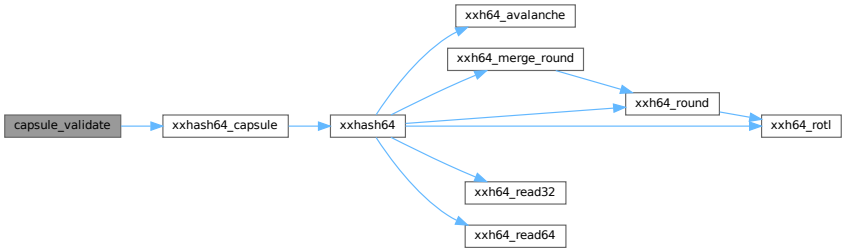
Returns

CAPSULE_VALID on success, error code otherwise

References CAPSULE_DESC_MAGIC, CAPSULE_ERR_BAD_HASH_ALG, CAPSULE_ERR_BAD_MAGIC, CAPSULE_ERR_BAD_VERSION, CAPSULE_ERR_BOUNDS, CAPSULE_ERR_HASH_MISMATCH, CAPSULE_ERR_MODE_INVA CAPSULE_ERR_NULL_PTR, CAPSULE_ERR_REVOKED_ACTIVE, CAPSULE_FLAG_ACTIVE, CAPSULE_FLAG_REVOKED, CAPSULE_HASH_BLAKE3, CAPSULE_HASH_XXHASH64, CAPSULE_MAGIC_GET_HASHALG, CAPSULE_MAGIC_GET_SIG, CAPSULE_MAGIC_GET_VERSION, CAPSULE_MODE_VALID, CAPSULE_VALID, CAPSULE_VERSION_0, and xxhash64_capsule().

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_exec_init(), and capsule_run_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

7.132.1.2 capsule_validate_result_str()

```
const char * capsule_validate_result_str (
            CapsuleValidateResult result )
```

capsule_validate_result_str - Get string for validation result

References validate_result_strings.

7.132.2 Variable Documentation

7.132.2.1 validate_result_strings

```
const char* validate_result_strings[]  [static]
```

Initial value:

```
= {
    "valid",
    "bad magic",
    "bad version",
    "bad hash algorithm",
    "bounds error",
    "mode invalid",
    "revoked but active",
    "hash mismatch",
    "null pointer",
}
```

capsule_validate.c - Capsule Validation (M7.1)

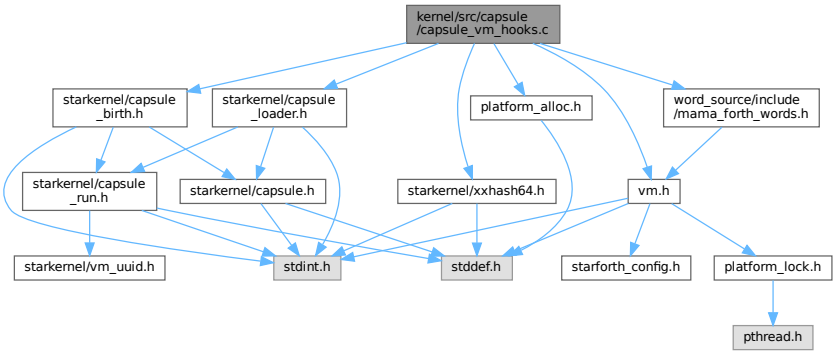
Validates capsule descriptors for structural integrity and content hash. Freestanding - no libc dependency.

Referenced by capsule_validate_result_str().

7.133 kernel/src/capsule/capsule_vm_hooks.c File Reference

```
#include "starkernel/capsule_birth.h"
#include "starkernel/capsule_loader.h"
#include "starkernel/xxhash64.h"
#include "vm.h"
#include "platform_alloc.h"
#include "word_source/include/mama_forth_words.h"
```

Include dependency graph for capsule_vm_hooks.c:



Functions

- static int capsule_exec_hook (void *vm_ctx, const char *code, uint64_t code_len)

  capsule_vm_hooks.c - VM Hook Implementations for Capsule Birth (M7.1)
- static uint64_t capsule_dict_hash_hook (void *vm_ctx)
- static void * capsule_vm_alloc_hook (void)
- void capsule_vm_hooks_register (void)

  capsule_vm_hooks_register - Wire the three hooks into the capsule subsystem.

7.133.1 Function Documentation

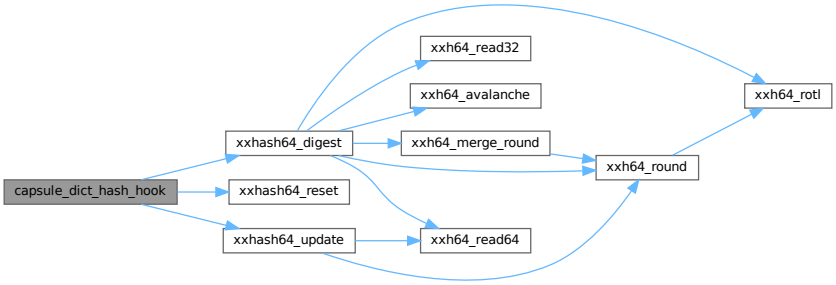
7.133.1.1 capsule_dict_hash_hook()

```
static uint64_t capsule_dict_hash_hook (
            void * vm_ctx )  [static]
```

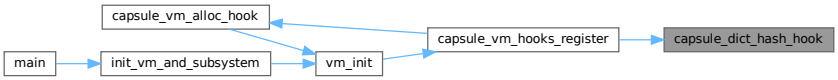
References entry, VM::latest, xxhash64_digest(), xxhash64_reset(), and xxhash64_update().

Referenced by capsule_vm_hooks_register().

Here is the call graph for this function:



Here is the caller graph for this function:



7.133.1.2 capsule_exec_hook()

```
static int capsule_exec_hook (
            void * vm_ctx,
            const char * code,
            uint64_t code_len )  [static]
```

capsule_vm_hooks.c - VM Hook Implementations for Capsule Birth (M7.1)

Concrete implementations of the three function pointers required by capsule_birth_set_hooks():

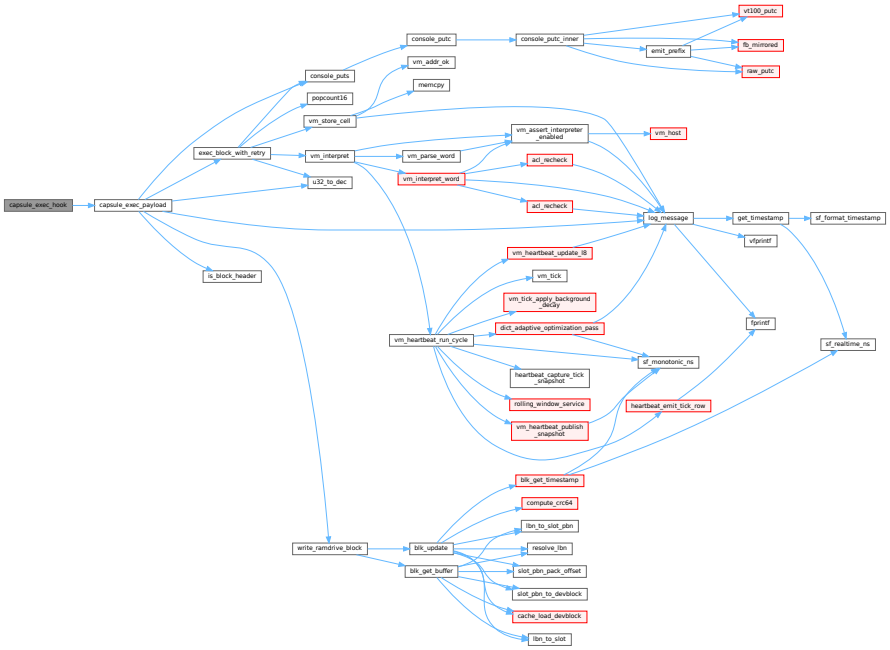
capsule_exec_hook – wraps vm_interpret() capsule_dict_hash_hook – xxhash64 walk of the live dictionary capsule_vm_alloc_hook – allocates and initialises a fresh VM

Call capsule_vm_hooks_register() once during VM bootstrap (after register_forth79_words) to wire everything up.

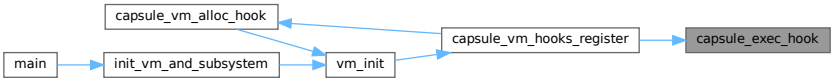
References capsule_exec_payload().

Referenced by capsule_vm_hooks_register().

Here is the call graph for this function:



Here is the caller graph for this function:



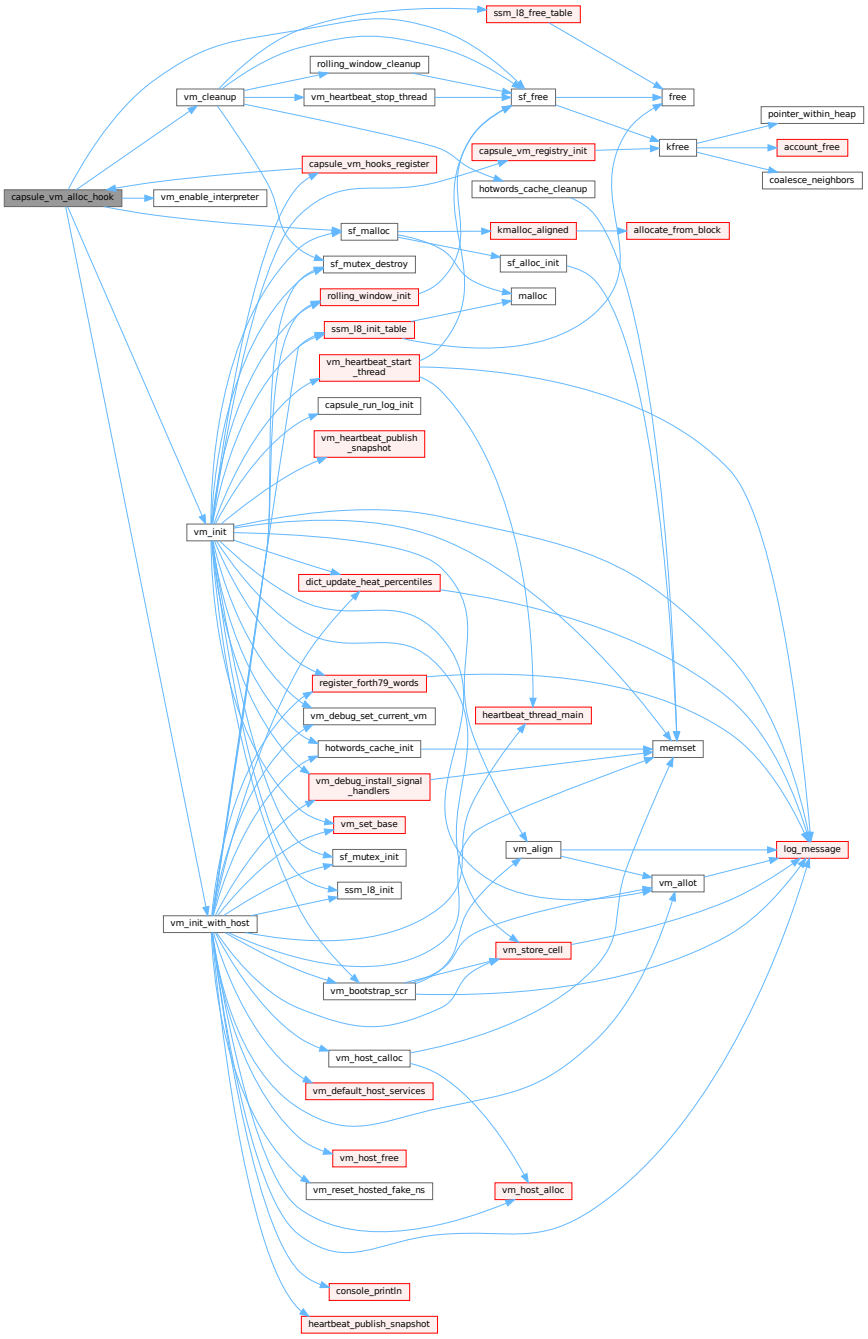
7.133.1.3 capsule_vm_alloc_hook()

```
static void * capsule_vm_alloc_hook (
            void  )  [static]
```

References VM::error, sf_free(), sf_malloc(), vm_cleanup(), vm_enable_interpreter(), vm_init(), and vm_init_with_host().

Referenced by capsule_vm_hooks_register().

Here is the call graph for this function:



Here is the caller graph for this function:

7.133.1.4 capsule_vm_hooks_register()

```
void capsule_vm_hooks_register (
            void  )
```

capsule_vm_hooks_register - Wire the three hooks into the capsule subsystem.

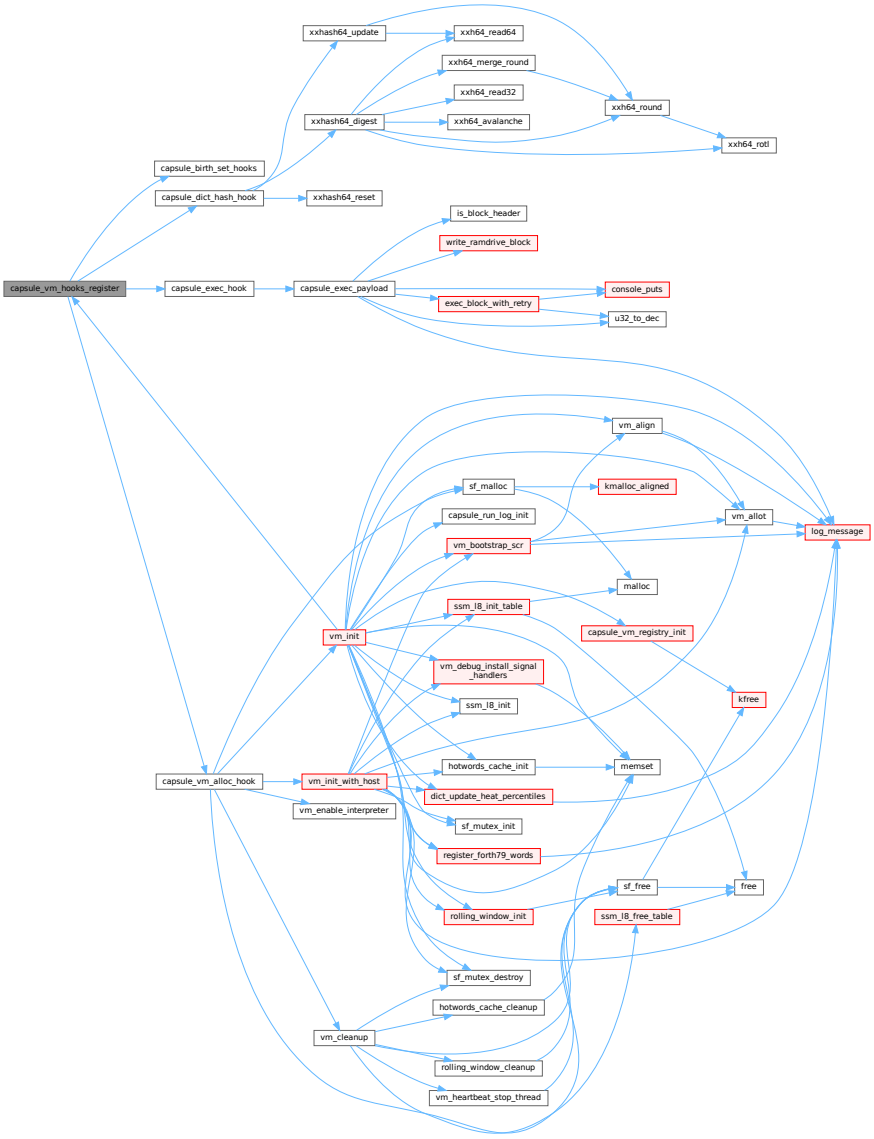
capsule_vm_hooks_register - Wire concrete VM hooks into the capsule subsystem

Must be called after vm_init() completes on Mama's VM, before any capsule birth or experiment operations.

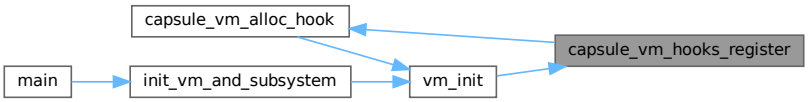
References capsule_birth_set_hooks(), capsule_dict_hash_hook(), capsule_exec_hook(), and capsule_vm_alloc_hook().

Referenced by vm_init().

Here is the call graph for this function:



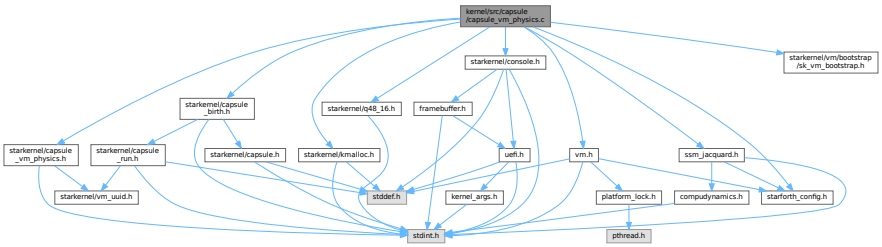
Here is the caller graph for this function:



7.134 kernel/src/capsule/capsule_vm_physics.c File Reference

```
#include "starkernel/capsule_vm_physics.h"
#include "starkernel/capsule_birth.h"
#include "starkernel/kmalloc.h"
#include "starkernel/q48_16.h"
#include "starkernel/console.h"
#include "starforth_config.h"
#include "starkernel/vm/bootstrap/sk_vm_bootstrap.h"
#include "vm.h"
#include "ssm_jacquard.h"
```

Include dependency graph for capsule_vm_physics.c:



Data Structures

- struct VMPhysics
- struct vm_physics_node
- struct VMFleetTouchSample
- struct VMFleetWindow

Macros

- #define VM_FLEET_WINDOW_DEPTH 64

  capsule_vm_physics.c - Dynamic VM Fleet Physics Implementation
- #define VM_PHYSICS_EPSILON_Q48 3277ULL
- #define VM_PHYSICS_MIN_INFORMATIVE_SAMPLES 8

Typedefs

- typedef struct vm_physics_node vm_physics_node_t

Functions

- static vm_physics_node_t * vm_physics_find (VMUuid vm_id)
- static void vm_physics_transfer (VMPhysics *from, VMPhysics *to, uint64_t amount_q48)

  vm_physics_transfer - The one conservative primitive everything else is a special case of.
- void vm_physics_init (VMUuid vm_id)

  capsule_vm_physics.h - Dynamic VM Fleet Physics
- static VMUuid vm_physics_find_root_id (VMUuid vm_id)

  vm_physics_find_root_id - Walk parent_vm_id edges up to Hera
- void vm_physics_retire (VMUuid vm_id)

  vm_physics_retire - Remove a killed VM, returning its heat to Hera
- void vm_physics_touch (VMUuid vm_id)

  vm_physics_touch - Record real dispatch activity against a VM
- static uint32_t l8_regime_modulation_q16 (void)

  l8_regime_modulation_q16 - Read-only Q16 multiplier from Hera's own L8
- void vm_physics_tick (uint64_t now_ns)

  vm_physics_tick - Heartbeat-gated inference pass
- void vm_physics_heartbeat_tick (uint64_t now_ns)

  vm_physics_heartbeat_tick - Fleet-wide heartbeat, gates vm_physics_tick()
- uint64_t vm_physics_fleet_heat_sum (void)

  vm_physics_fleet_heat_sum - Sum of execution_heat_q48 over all LIVE VMs
- int vm_physics_conserved (void)

  vm_physics_conserved - Conservation check
- uint64_t vm_physics_heat_of (VMUuid vm_id)

  vm_physics_heat_of - Current execution_heat_q48 for one VM
- static void console_put_u64 (uint64_t val)
- void vm_physics_status (void)

  vm_physics_status - Print a diagnostic status report

Variables

- static vm_physics_node_t * vm_physics_head = (void *)0
- static VMFleetWindow fleet_window = { { { 0, 0, 0 } }, 0, 0, 0 }
- static uint64_t fleet_transfer_slope_q48 = 65536ULL / 3
- static uint64_t slope_fit_quality_q48 = 0
- static uint64_t fleet_heartbeat_tick_count = 0
- static uint64_t fleet_last_capacity_tick = 0

7.134.1 Macro Definition Documentation

7.134.1.1 VM_FLEET_WINDOW_DEPTH

```
#define VM_FLEET_WINDOW_DEPTH 64
```

capsule_vm_physics.c - Dynamic VM Fleet Physics Implementation

See capsule_vm_physics.h and docs/working/architecture/VM-PHYSICS-DYNAMIC-FLEET-DESIGN-20260705.↩
md for the full design rationale.

Concurrency: no locking. The only interrupt-driven kernel path (heartbeat_tick, from the timer ISR) touches only its own isolated TimeTrustState and never calls into mama_word_*, vm_interpret, or any capsule/dictionary state. HEARTBEAT_THREAD_ENABLED=0 means no separate OS thread runs physics logic either. BIRTH/KILL/↩
VM-EXEC/VM-CALL/ VM-STEP all execute synchronously in Hera's single interpreter context with nothing able to preempt them mid-transfer.

L8 coupling (rev q): the only place this file reaches outside its own VMPhysics/VMFleetWindow state is l8_regime_modulation_q16(), a read-only look at Hera's own ssm_l8_state via sk_get_mama_vm() – modulating the fleet's empirically-recovered transfer rate uniformly, never deciding which VM receives heat. Still a passive observer: this changes how fast the fleet's own rate estimate responds, not who runs or who gets touched.

7.134.1.2 VM_PHYSICS_EPSILON_Q48

```
#define VM_PHYSICS_EPSILON_Q48 3277ULL
```

7.134.1.3 VM_PHYSICS_MIN_INFORMATIVE_SAMPLES

```
#define VM_PHYSICS_MIN_INFORMATIVE_SAMPLES 8
```

7.134.2 Typedef Documentation

7.134.2.1 vm_physics_node_t

```
typedef struct vm_physics_node vm_physics_node_t
```

7.134.3 Function Documentation

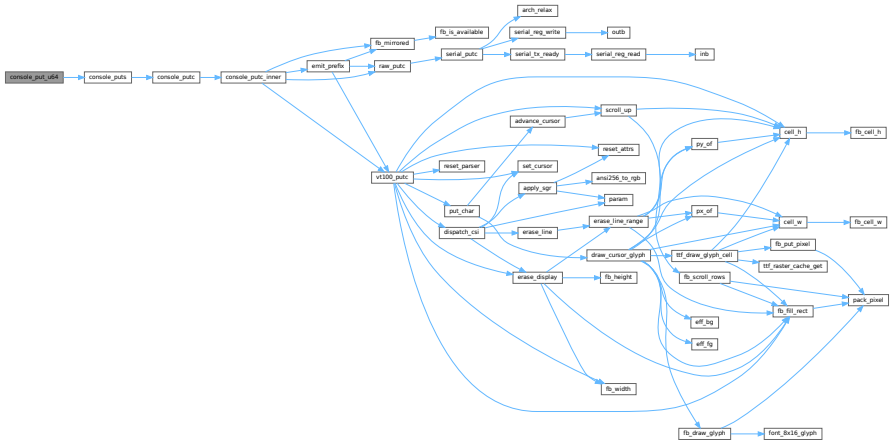
7.134.3.1 console_put_u64()

```
static void console_put_u64 (
            uint64_t val )  [static]
```

References console_puts().

Referenced by vm_physics_status().

Here is the call graph for this function:



Here is the caller graph for this function:

7.134.3.2 l8_regime_modulation_q16()

```
static uint32_t l8_regime_modulation_q16 (
            void  )  [static]
```

l8_regime_modulation_q16 - Read-only Q16 multiplier from Hera's own L8

Gives L8 a real, passive-observer-respecting causal channel into the fleet's transfer rate (rev q, VM-FLEET-↩ ATTRACTOR-DESIGN-20260705.md): previously nothing L8 did could affect fleet dynamics at all (rev j's architectural diagnosis for its null DoE result). Reuses Hera's adaptive table's current_regime – the same 3-bit entropy/cv/temporal classification the bandit already stratifies its own scores by, rather than inventing a new signal – as a proxy for fleet-wide dispatch activity, since all VM-EXEC/VM-CALL flows through her.

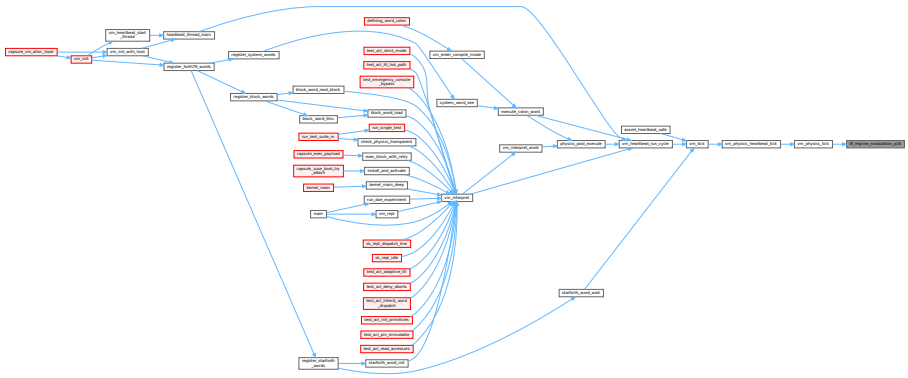
modulation = (popcount(regime) + 1) ∗ 0.5, i.e. 0.5x-2.0x: more of the three high-signal bits active scales the fleet's already-recovered rate up (track change more aggressively), fewer scales it down (damp toward the raw empirical rate). No per-VM favoritism – this scales whatever vm_physics_tick() already computed uniformly, it doesn't decide who receives heat.

Returns Q16 65536 (neutral 1.0x) if Hera or her adaptive table isn't available, so a missing signal never blocks or corrupts the estimate this multiplies.

References CDConfigTable::current_regime, VM::ssm_l8_state, and ssm_l8_state_t::table.

Referenced by vm_physics_tick().

Here is the caller graph for this function:



7.134.3.3 vm_physics_conserved()

```
int vm_physics_conserved (
            void  )
```

vm_physics_conserved - Conservation check

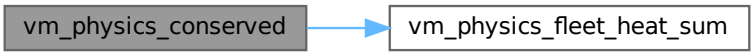
Returns

1 if $|$vm_physics_fleet_heat_sum() - Q.1$|$ $<$ epsilon, else 0

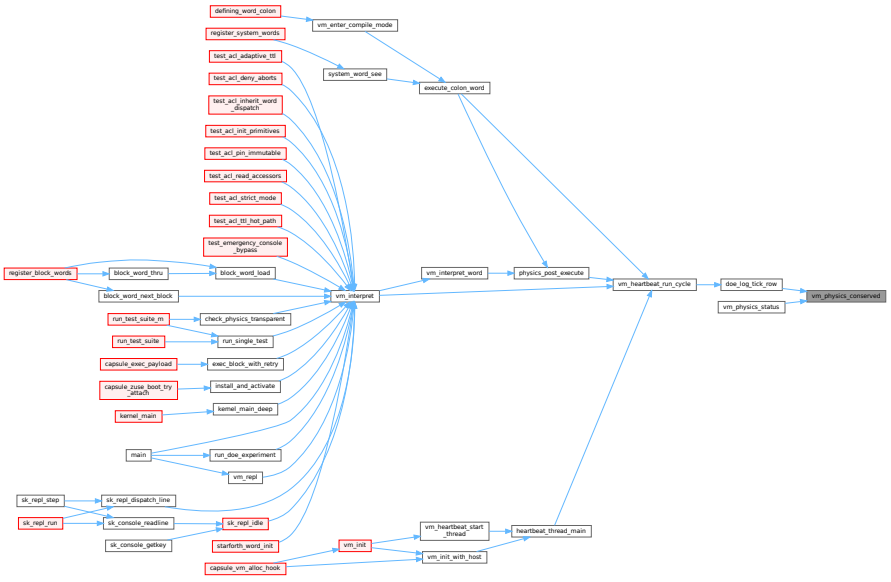
References Q48_ONE, VM_PHYSICS_EPSILON_Q48, and vm_physics_fleet_heat_sum().

Referenced by doe_log_tick_row(), and vm_physics_status().

Here is the call graph for this function:



Here is the caller graph for this function:



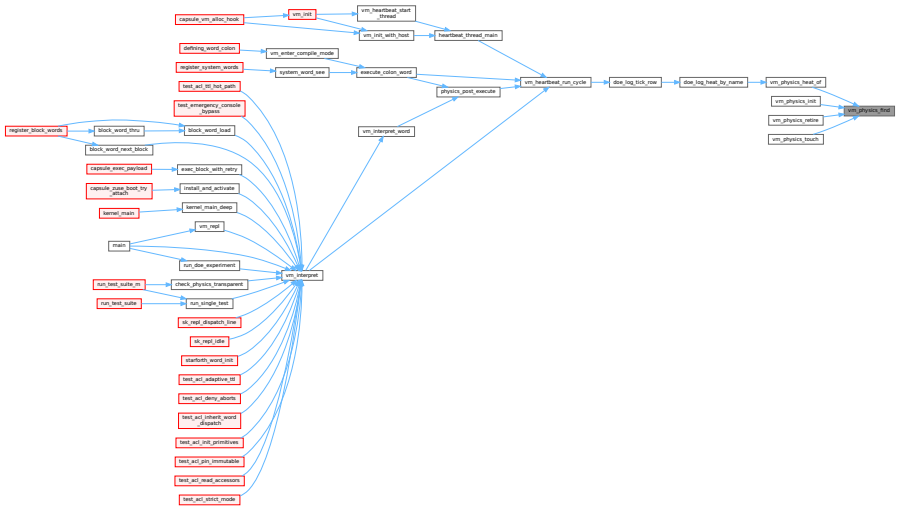
7.134.3.4 vm_physics_find()

```
static vm_physics_node_t * vm_physics_find (
            VMUuid vm_id )  [static]
```

References vm_physics_node::next, vm_physics_node::vm_id, and vm_physics_head.

Referenced by vm_physics_heat_of(), vm_physics_init(), vm_physics_retire(), and vm_physics_touch().

Here is the caller graph for this function:



7.134.3.5 vm_physics_find_root_id()

```
static VMUuid vm_physics_find_root_id (
            VMUuid vm_id )  [static]
```

vm_physics_find_root_id - Walk parent_vm_id edges up to Hera

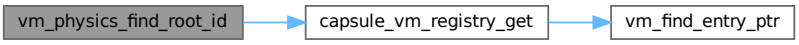
Every VM's parent_vm_id (VMRegistryEntry, capsule_run.h) is set once at its own birth and never rewritten – including a dead VM's, so the walk continues correctly through an already-killed intermediate parent without any special dead-parent handling. Hera's own entry is self-referential (parent_vm_id == vm_id == vm_uuid_hera()), which is what stops the walk: she is the only node with no outbound edge.

The 64-hop guard is defensive only – a well-formed chain terminates in O(fleet depth) hops, nowhere near 64. It exists so a corrupted chain (which should never happen) returns the starting vm_id instead of spinning.

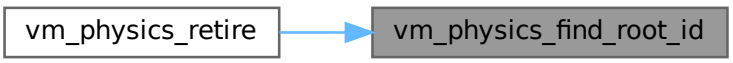
References capsule_vm_registry_get(), entry, and VMRegistryEntry::parent_vm_id.

Referenced by vm_physics_retire().

Here is the call graph for this function:



Here is the caller graph for this function:



7.134.3.6 vm_physics_fleet_heat_sum()

```
uint64_t vm_physics_fleet_heat_sum (
            void  )
```

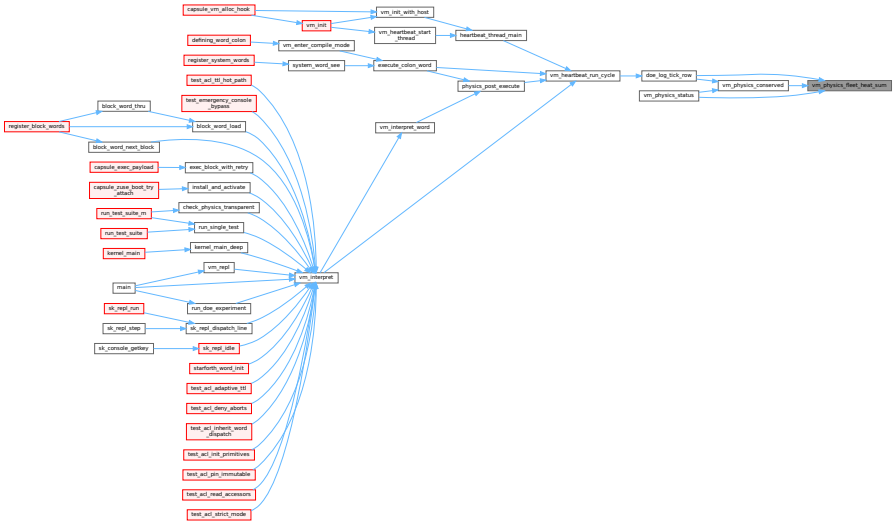
vm_physics_fleet_heat_sum - Sum of execution_heat_q48 over all LIVE VMs

Diagnostic / verification primitive. Should always read Q.1 (65536) if the conservation invariant holds.

References VMPhysics::execution_heat_q48, VMPhysics::is_live, vm_physics_node::next, vm_physics_node::physics, and vm_physics_head.

Referenced by doe_log_tick_row(), vm_physics_conserved(), and vm_physics_status().

Here is the caller graph for this function:



7.134.3.7 vm_physics_heartbeat_tick()

```
void vm_physics_heartbeat_tick (
            uint64_t now_ns )
```

vm_physics_heartbeat_tick - Fleet-wide heartbeat, gates vm_physics_tick()

Call from every VM's own vm_tick(), not just Hera's. Increments a single fleet-wide tick counter (distinct from any VM's own per-VM HeartbeatState.tick_count) and calls vm_physics_tick() once that shared counter has advanced by HEARTBEAT_INFERENCE_FREQUENCY since the last fit.

Fixes a real defect (VM-FLEET-ATTRACTOR-DESIGN-20260705.md rev k): gating this on Hera's own tick_count meant the inference pass almost never fired, because Hera-as-orchestrator mostly blocks on VM-EXEC/ VM-CALL dispatch – work that accrues to the *target* VM's own tick count, not hers. VM-PHYSICS-DYNAMIC-FLEET-DESIGN-20260705.md already specified the fix's direction: fleet_transfer_slope_q48 must be inferred "from aggregate fleet statistics... not one per VM" – the same principle applies to the readiness signal that gates computing it, not just the regression's own input trajectory.

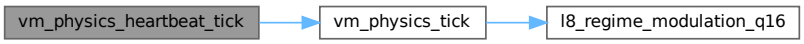
Parameters

now_ns	Current monotonic time in nanoseconds (of whichever VM is calling; unused by vm_physics_tick() today)

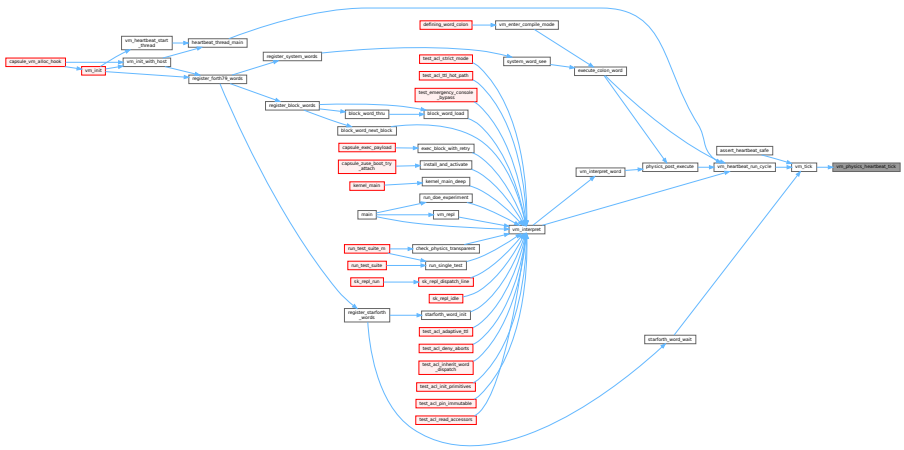
References fleet_heartbeat_tick_count, fleet_last_capacity_tick, STADIUM_CAPACITY_TICK, and vm_physics_tick().

Referenced by vm_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.134.3.8 vm_physics_heat_of()

```
uint64_t vm_physics_heat_of (
            VMUuid vm_id )
```

vm_physics_heat_of - Current execution_heat_q48 for one VM

Transparency primitive (VM-FLEET-ATTRACTOR-DESIGN-20260705.md): the fleet-wide conservation invariant always reads Q.1 by construction, so it carries no information about how heat is actually distributed among live VMs. This is the per-VM fact that sum alone can't provide.

Parameters

vm↩ _id	Registry VM ID

Returns

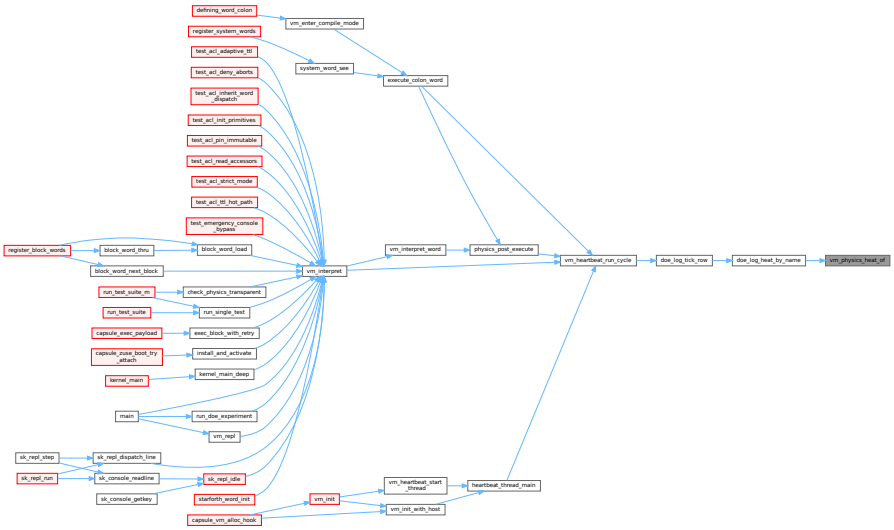
execution_heat_q48, or 0 if vm_id is unknown or not live

References VMPhysics::execution_heat_q48, VMPhysics::is_live, vm_physics_node::physics, and vm_physics_find().

Referenced by doe_log_heat_by_name().

Here is the call graph for this function:

Here is the caller graph for this function:



7.134.3.9 vm_physics_init()

```
void vm_physics_init (
            VMUuid vm_id )
```

capsule_vm_physics.h - Dynamic VM Fleet Physics

Structural parallel to the word-level physics engine (execution heat, decay, rolling-window history, regression-inferred slope) applied to VMs instead of dictionary words. Kernel-only: no hosted-build equivalent, since the hosted build has no multi-VM fleet.

sum(execution_heat_q48 for all LIVE VMs) == Q.1 is a conservation invariant, held by construction: every state change is a balanced transfer (see design doc VM-PHYSICS-DYNAMIC-FLEET-DESIGN-20260705.md).

This is a passive observer of VM activity, never a driver of it — there is no scheduler here. Recording happens only at existing dispatch points (BIRTH, KILL, VM-EXEC, VM-CALL, VM-STEP). vm_physics_init - Register a newly-born VM at zero heat

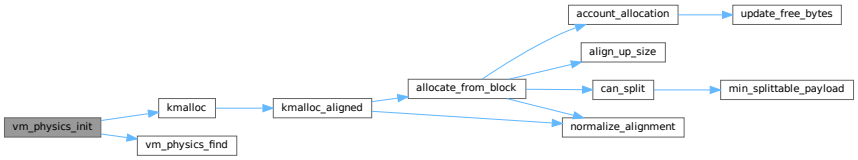
Called from mama_word_birth once a VM reaches VM_STATE_LIVE. Starts at execution_heat_q48 = 0, so conservation holds trivially — no fan-out from existing VMs is needed.

Parameters

vm↩_id	Registry VM ID assigned at birth

References VMPhysics::execution_heat_q48, VMPhysics::is_live, kmalloc(), VMPhysics::last_active_tick, vm_physics_node::next, vm_physics_node::physics, Q48_ONE, VMPhysics::touched, vm_physics_node::vm_id, vm_physics_find(), and vm_physics_head.

Here is the call graph for this function:



7.134.3.10 vm_physics_retire()

```
void vm_physics_retire (
            VMUuid vm_id )
```

vm_physics_retire - Remove a killed VM, returning its heat to Hera

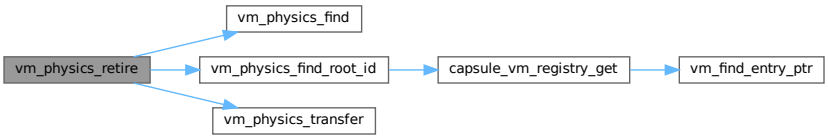
Called from mama_word_kill before the registry entry is torn down. The dying VM's entire remaining heat is transferred up its parent_vm_id chain (VMRegistryEntry, capsule_run.h) to the fleet's single structural root, Hera – no division, no per-survivor weighting. The chain walk tolerates already-dead intermediate parents (a parent's own parent_vm_id was set once at its birth and never rewritten, so the walk continues through it).

Parameters

vm↩_id	Registry VM ID of the VM being killed

References VMPhysics::execution_heat_q48, VMPhysics::is_live, vm_physics_node::physics, vm_physics_find(), vm_physics_find_root_id(), and vm_physics_transfer().

Here is the call graph for this function:



7.134.3.11 vm_physics_status()

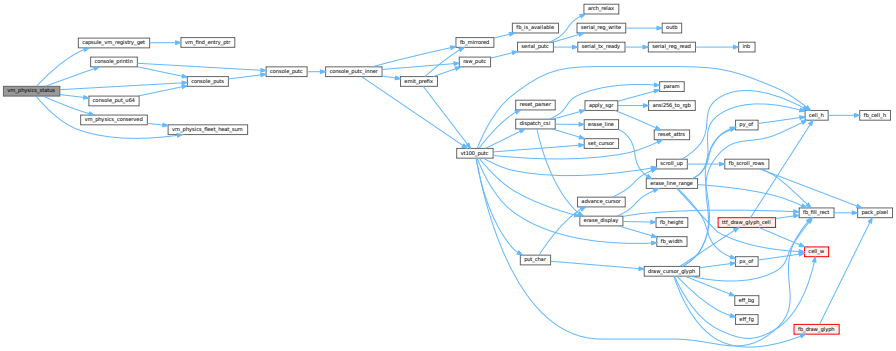
```
void vm_physics_status (
            void  )
```

vm_physics_status - Print a diagnostic status report

Replaces K-STATUS/VM-STATUS. Reports fleet heat sum, conservation verdict, and the current fleet-wide inferred slope / fit quality / warm-up state – the fleet-level analog of K-STATUS's per-VM report, adapted to a mechanism with no fixed VM count to enumerate.

References capsule_vm_registry_get(), console_println(), console_put_u64(), console_puts(), entry, VMPhysics::execution_heat_q48, fleet_transfer_slope_q48, fleet_window, VMPhysics::is_live, VMFleetWindow::is_warm, vm_physics_node::next, vm_physics_node::physics, slope_fit_quality_q48, vm_physics_node::vm_id, vm_physics_conserved(), vm_physics_fleet_heat_sum(), and vm_physics_head.

Here is the call graph for this function:



7.134.3.12 vm_physics_tick()

```
void vm_physics_tick (
            uint64_t now_ns )
```

vm_physics_tick - Heartbeat-gated inference pass

Mirrors vm_tick_inference_engine: when the fleet touch-history window is warm, replays it to re-fit fleet_transfer_↩ slope_q48 via the same closed-form log-linear OLS regression used for word-level decay. Skips the update (does not substitute a default) when unwarmed or when fit quality is not usable.

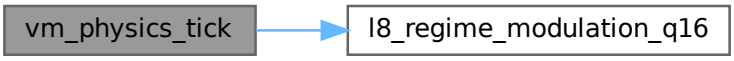
Parameters

now_ns	Current monotonic time in nanoseconds

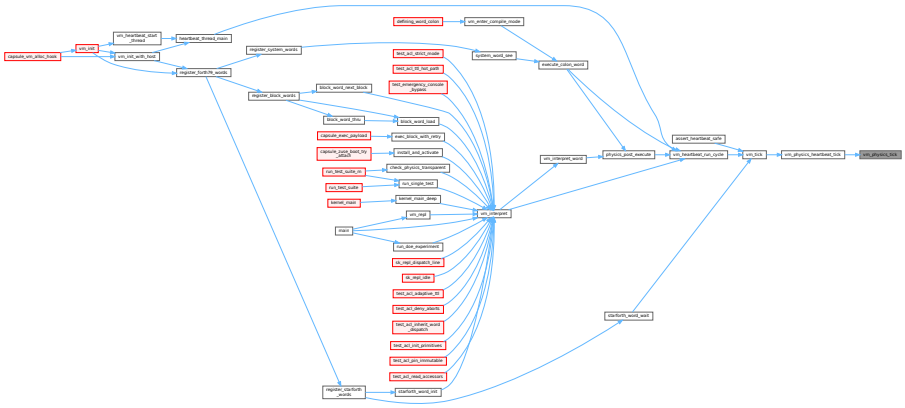
References VMFleetTouchSample::amount, VMFleetTouchSample::clamped, VMFleetTouchSample::elapsed_ticks, fleet_transfer_slope_q48, fleet_window, VMFleetWindow::is_warm, l8_regime_modulation_q16(), slope_fit_quality_q48, VMFleetWindow::touch_samples, VM_FLEET_WINDOW_DEPTH, and VM_PHYSICS_MIN_INFORMATIVE_SAMPLES.

Referenced by vm_physics_heartbeat_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.134.3.13 vm_physics_touch()

```
void vm_physics_touch (
            VMUuid vm_id )
```

vm_physics_touch - Record real dispatch activity against a VM

Called from the three existing dispatch primitives (VM-EXEC, VM-CALL, VM-STEP) at the point they've already confirmed the target VM is live. Pulls heat from the rest of the fleet toward the touched VM (amount = elapsed_ticks ∗ fleet_transfer_slope_q48 >> 16) and appends vm_id to the fleet's rolling touch-history window.

Restated on the virtual tick (FABRIC-0.md item 2.1, 2026-08-04): no longer takes a wall-clock timestamp. Reads fleet_heartbeat_tick_count internally, which is execution-paced (advanced once per vm_tick() call, see vm_↩ physics_heartbeat_tick), so the transfer this produces is a deterministic function of the execution stream, not of wall time.

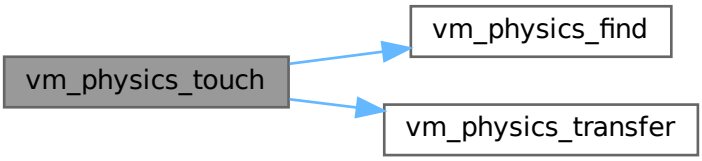
Parameters

vm↩ _id	Registry VM ID being dispatched to

References VMFleetTouchSample::amount, VMFleetTouchSample::clamped, VMFleetWindow::count, VMFleetTouchSample::elapsed, VMPhysics::execution_heat_q48, fleet_heartbeat_tick_count, fleet_transfer_slope_q48, fleet_window, VMFleetWindow::head, VMPhysics::is_live, VMFleetWindow::is_warm, VMPhysics::last_active_tick, vm_physics_node::next, vm_physics_node::physics,

VMFleetWindow::touch_samples, VMPhysics::touched, VM_FLEET_WINDOW_DEPTH, vm_physics_find(), vm_physics_head, and vm_physics_transfer().

Here is the call graph for this function:



7.134.3.14 vm_physics_transfer()

```
static void vm_physics_transfer (
            VMPhysics * from,
            VMPhysics * to,
            uint64_t amount_q48 )  [static]
```

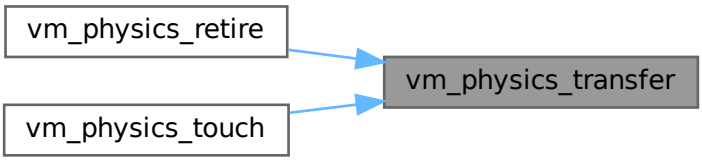
vm_physics_transfer - The one conservative primitive everything else is a special case of.

Subtracts amount from from (clamped at 0) and adds the same amount to to. Nothing is created or destroyed: sum(execution_heat for all LIVE VMs) is invariant across any call.

References VMPhysics::execution_heat_q48.

Referenced by vm_physics_retire(), and vm_physics_touch().

Here is the caller graph for this function:



7.134.4 Variable Documentation

7.134.4.1 fleet_heartbeat_tick_count

```
uint64_t fleet_heartbeat_tick_count = 0  [static]
```

Referenced by vm_physics_heartbeat_tick(), and vm_physics_touch().

7.134.4.2 fleet_last_capacity_tick

```
uint64_t fleet_last_capacity_tick = 0  [static]
```

Referenced by vm_physics_heartbeat_tick().

7.134.4.3 fleet_transfer_slope_q48

```
uint64_t fleet_transfer_slope_q48 = 65536ULL / 3  [static]
```

Referenced by vm_physics_status(), vm_physics_tick(), and vm_physics_touch().

7.134.4.4 fleet_window

```
VMFleetWindow fleet_window = { { { 0, 0, 0 } }, 0, 0, 0 }  [static]
```

Referenced by vm_physics_status(), vm_physics_tick(), and vm_physics_touch().

7.134.4.5 slope_fit_quality_q48

```
uint64_t slope_fit_quality_q48 = 0  [static]
```

Referenced by vm_physics_status(), and vm_physics_tick().

7.134.4.6 vm_physics_head

```
vm_physics_node_t* vm_physics_head = (void *)0  [static]
```

Referenced by vm_physics_find(), vm_physics_fleet_heat_sum(), vm_physics_init(), vm_physics_status(), and vm_physics_touch().

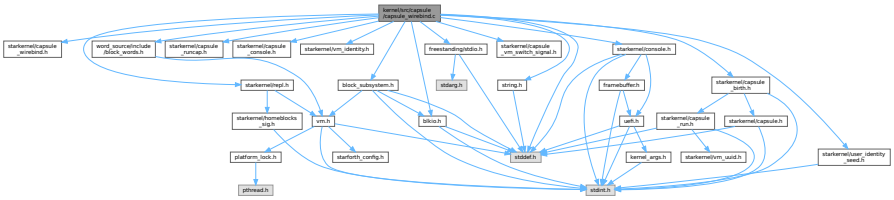
7.135 kernel/src/capsule/capsule_vm_switch_signal.c File Reference

7.136 kernel/src/capsule/capsule_wirebind.c File Reference

```
#include "starkernel/capsule_wirebind.h"
#include "starkernel/capsule_birth.h"
#include "starkernel/capsule_runcap.h"
#include "starkernel/capsule_console.h"
#include "starkernel/vm_identity.h"
#include "starkernel/user_identity_seed.h"
#include "starkernel/console.h"
#include "starkernel/repl.h"
#include "starkernel/capsule_vm_switch_signal.h"
#include "blkio.h"
#include "block_subsystem.h"
#include "word_source/include/block_words.h"
```

```
#include "freestanding/stdio.h"
#include <string.h>
#include <stddef.h>
```

Include dependency graph for capsule_wirebind.c:



Data Structures

- struct wirebind_live_entry_t

Macros

- #define WIREBIND_MAX_LIVE 13u
- #define WIREBIND_CERT_MAX_DEVBLOCKS 4u
- #define WIREBIND_OVERFLOW_FREE_THRESHOLD_PCT 10u
- #define WIREBIND_OVERFLOW_CLAIM_DEVBLOCKS 8u

Functions

- static wirebind_live_entry_t * wirebind_live_find_by_dev (struct blkio_dev *dev)
- static wirebind_live_entry_t * wirebind_live_alloc (void)
- static int read_devblock (struct blkio_dev *dev, uint32_t devblock, uint8_t *buf4096)
- int capsule_wirebind_verify_cert (struct blkio_dev *dev, const homeblocks_sig_t *sig, VM *mama_vm, VMIdentity *out)
- void capsule_wirebind_try_attach (struct blkio_dev *dev, const homeblocks_sig_t *sig, VM *mama_vm)
- static int wirebind_resolve_attached (VMRegistryEntry *out)
- static void wirebind_teardown_console (const char *username)
- static void wirebind_live_remove (wirebind_live_entry_t *e)
- int capsule_wirebind_eject (void)
- void capsule_wirebind_unclean_detach (struct blkio_dev *dev)
- void capsule_wirebind_reap_idle_check (void)
- const char * capsule_wirebind_attached_username (void)
- void capsule_wirebind_overflow_idle_check (void)

Variables

- static VMUuid g_wirebind_attached_vm_id
- static int g_wirebind_attached_valid = 0
- static char g_wirebind_attached_username [32u] = {0}
- static struct blkio_dev * g_wirebind_attached_dev = (struct blkio_dev *) 0
- static wirebind_live_entry_t g_wirebind_live [13u]

7.136.1 Macro Definition Documentation

7.136.1.1 WIREBIND_CERT_MAX_DEVBLOCKS

```
#define WIREBIND_CERT_MAX_DEVBLOCKS 4u
```

7.136.1.2 WIREBIND_MAX_LIVE

```
#define WIREBIND_MAX_LIVE 13u
```

7.136.1.3 WIREBIND_OVERFLOW_CLAIM_DEVBLOCKS

```
#define WIREBIND_OVERFLOW_CLAIM_DEVBLOCKS 8u
```

7.136.1.4 WIREBIND_OVERFLOW_FREE_THRESHOLD_PCT

```
#define WIREBIND_OVERFLOW_FREE_THRESHOLD_PCT 10u
```

7.136.2 Function Documentation

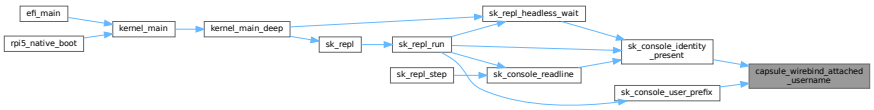
7.136.2.1 capsule_wirebind_attached_username()

```
const char * capsule_wirebind_attached_username (
            void  )
```

References g_wirebind_attached_username, and g_wirebind_attached_valid.

Referenced by sk_console_identity_present(), and sk_console_user_prefix().

Here is the caller graph for this function:

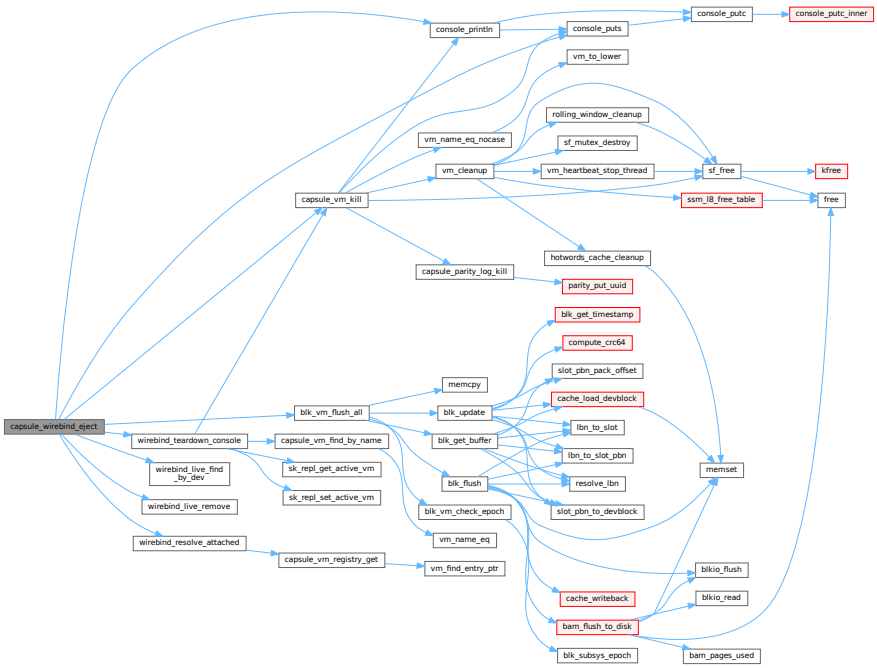


7.136.2.2 capsule_wirebind_eject()

```
int capsule_wirebind_eject (
            void  )
```

References blk_vm_flush_all(), capsule_vm_kill(), console_println(), console_puts(), entry, g_wirebind_attached_dev, g_wirebind_attached_username, g_wirebind_attached_valid, wirebind_live_find_by_dev(), wirebind_live_remove(), wirebind_resolve_attached(), and wirebind_teardown_console().

Here is the call graph for this function:



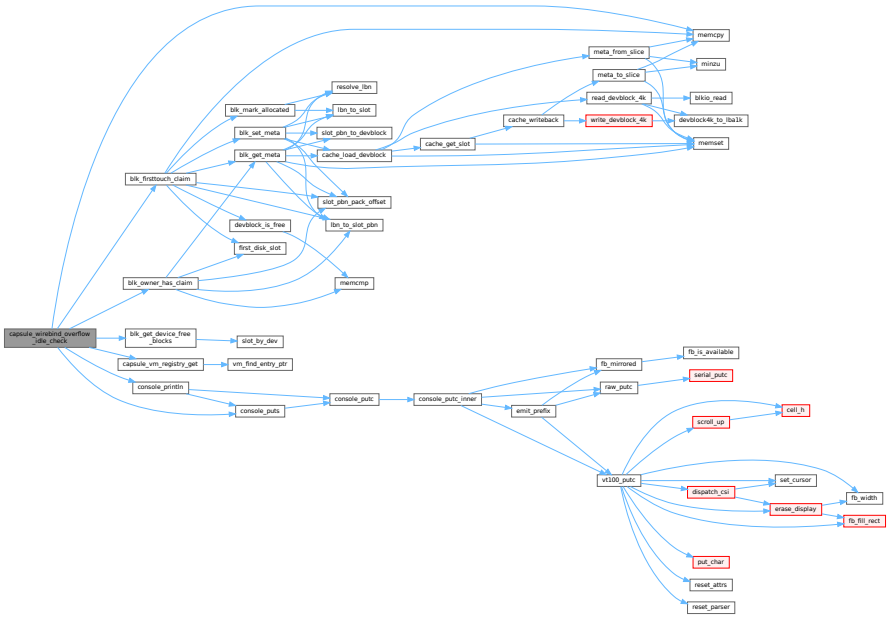
7.136.2.3 capsule_wirebind_overflow_idle_check()

```
void capsule_wirebind_overflow_idle_check (
            void  )
```

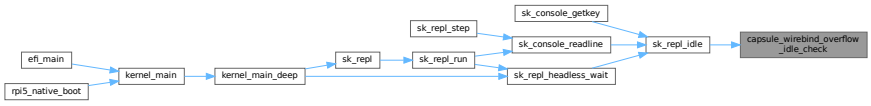
References blk_firsttouch_claim(), blk_get_device_free_blocks(), BLK_OK, blk_owner_has_claim(), capsule_vm_registry_get(), console_println(), console_puts(), entry, g_wirebind_attached_dev, g_wirebind_attached_username, g_wirebind_attached_valid, g_wirebind_attached_vm_id, memcpy(), blkio_dev::total_blocks, VM_STATE_LIVE, WIREBIND_OVERFLOW_CLAIM_DEVBLOCKS, and WIREBIND_OVERFLOW_FREE_THRESHOLD_PCT.

Referenced by sk_repl_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



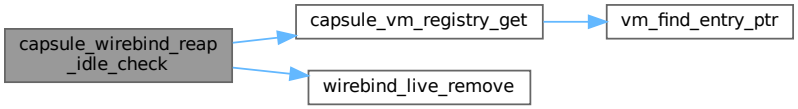
7.136.2.4 capsule_wirebind_reap_idle_check()

```
void capsule_wirebind_reap_idle_check (
            void  )
```

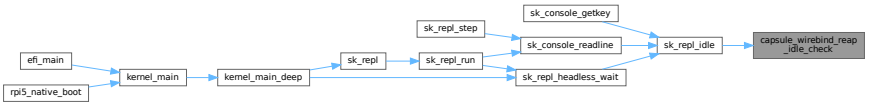
References capsule_vm_registry_get(), entry, g_wirebind_live, VM_STATE_DEAD, wirebind_live_remove(), and WIREBIND_MAX_LIVE.

Referenced by sk_repl_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



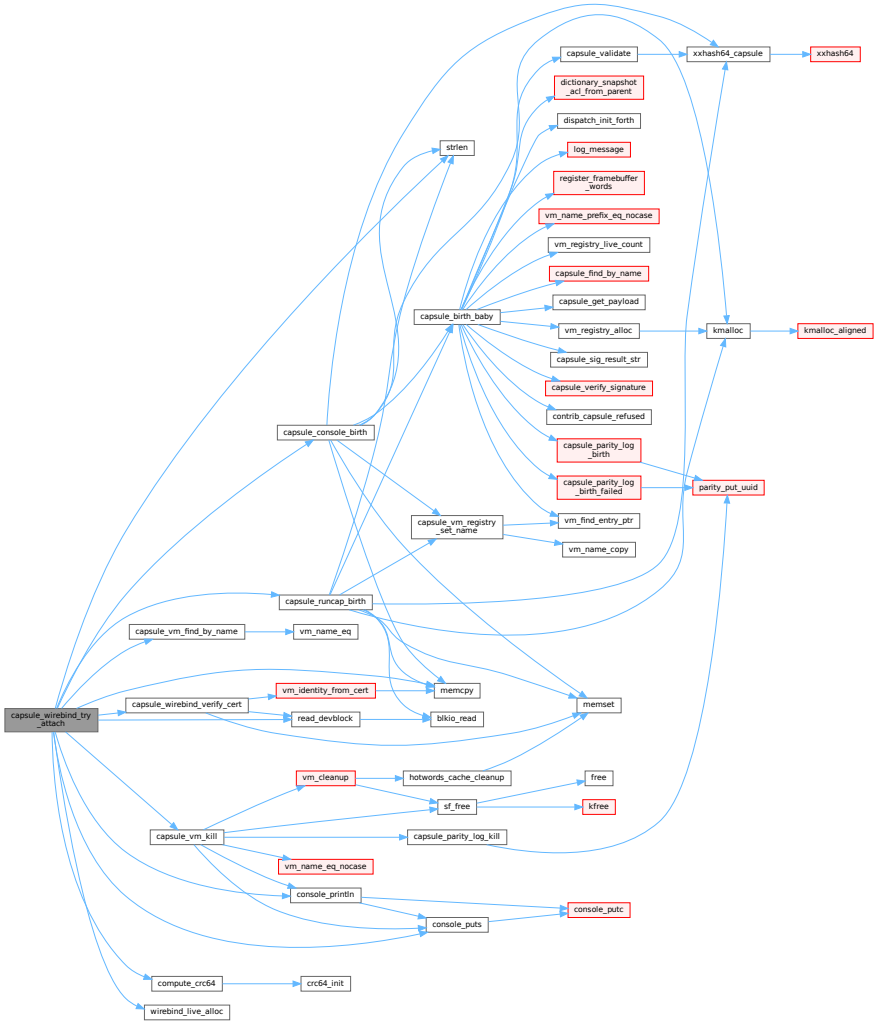
7.136.2.5 capsule_wirebind_try_attach()

```
void capsule_wirebind_try_attach (
            struct blkio_dev * dev,
            const homeblocks_sig_t * sig,
            VM * mama_vm )
```

References capsule_console_birth(), CAPSULE_RUN_OK, capsule_runcap_birth(), capsule_vm_find_by_name(), capsule_vm_kill(), capsule_wirebind_verify_cert(), homeblocks_sig_t::cert_devblocks, homeblocks_sig_t::cert_offset, compute_crc64(), console_println(), console_puts(), user_identity_seed_t::crc, wirebind_live_entry_t::dev, g_wirebind_attached_dev, g_wirebind_attached_username, g_wirebind_attached_valid, g_wirebind_attached_vm_id, homeblocks_sig_t::identity_src_devblocks, homeblocks_sig_t::identity_src_offset, wirebind_live_entry_t::in_use, user_identity_seed_t::magic, memcpy(), read_devblock(), VMRegistryEntry::state, strlen(), wirebind_live_entry_t::switch_slot, USER_IDENTITY_SEED_MAGIC, USER_IDENTITY_SEED_VERSION, USER_IDENTITY_USERNAME_MAX, wirebind_live_entry_t::user_vm_id, user_identity_seed_t::username, wirebind_live_entry_t::username, user_identity_seed_t::version, VM_STATE_LIVE, wirebind_live_alloc(), and VM::zuse_root_pubkey_known.

Referenced by sk_word_blk_attach_ack().

Here is the call graph for this function:



Here is the caller graph for this function:

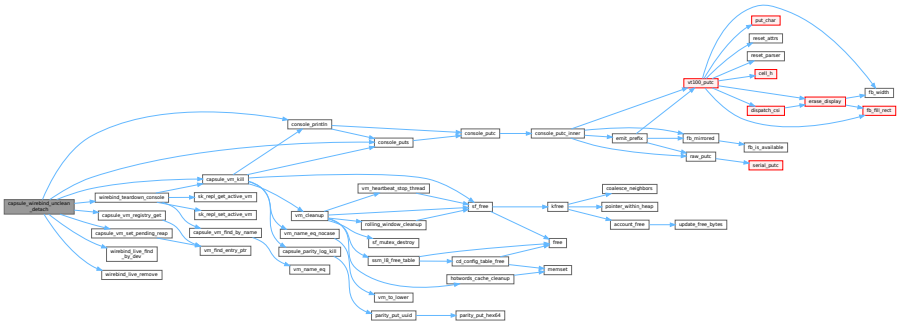
7.136.2.6 capsule_wirebind_unclean_detach()

```
void capsule_wirebind_unclean_detach (
            struct blkio_dev * dev )
```

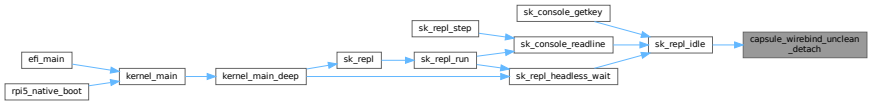
References capsule_vm_kill(), capsule_vm_registry_get(), capsule_vm_set_pending_reap(), console_println(), console_puts(), entry, g_wirebind_attached_dev, g_wirebind_attached_username, g_wirebind_attached_valid, wirebind_live_entry_t::user_vm_id, wirebind_live_entry_t::username, VM_STATE_DEAD, VM_STATE_STILLBORN, wirebind_live_find_by_dev(), wirebind_live_remove(), and wirebind_teardown_console().

Referenced by sk_repl_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



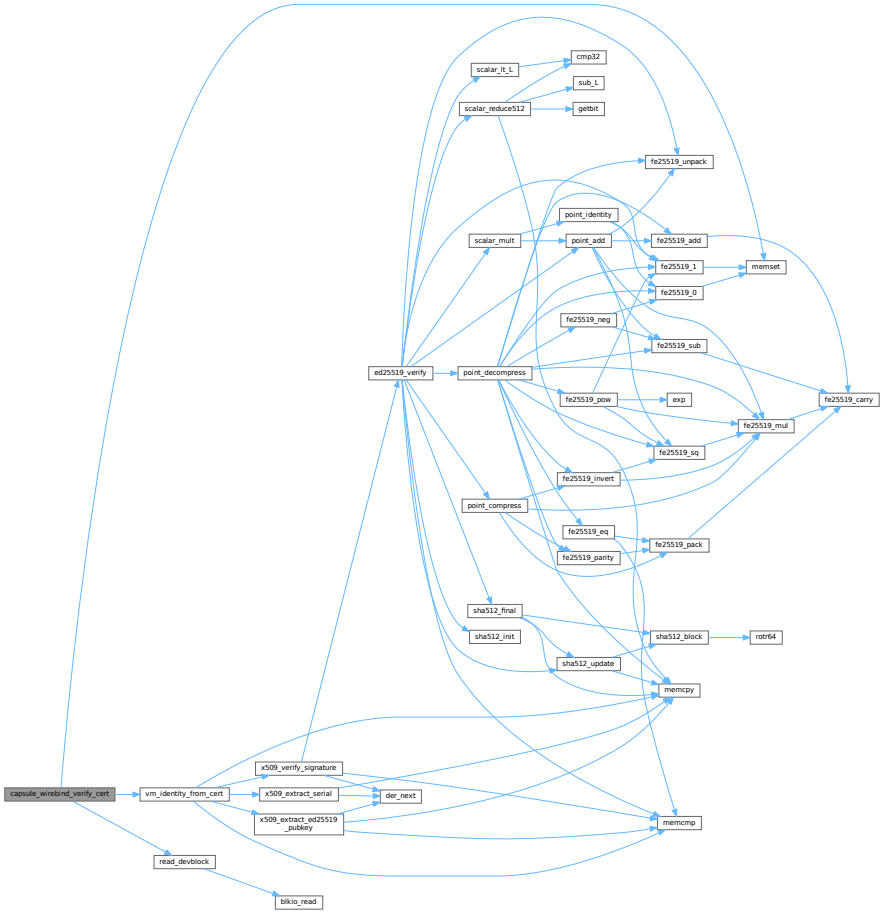
7.136.2.7 capsule_wirebind_verify_cert()

```
int capsule_wirebind_verify_cert (
            struct blkio_dev * dev,
            const homeblocks_sig_t * sig,
            VM * mama_vm,
            VMIdentity * out )
```

References homeblocks_sig_t::cert_devblocks, homeblocks_sig_t::cert_offset, homeblocks_sig_t::drive_uuid, memset(), read_devblock(), vm_identity_from_cert(), WIREBIND_CERT_MAX_DEVBLOCKS, VM::zuse_root_pubkey, and VM::zuse_root_pubkey_known.

Referenced by capsule_wirebind_try_attach().

Here is the call graph for this function:



Here is the caller graph for this function:

7.136.2.8 read_devblock()

```
static int read_devblock (
            struct blkio_dev * dev,
            uint32_t devblock,
            uint8_t * buf4096 )  [static]
```

References base, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, and blkio_read().

Referenced by capsule_wirebind_try_attach(), and capsule_wirebind_verify_cert().

Here is the call graph for this function:

Here is the caller graph for this function:

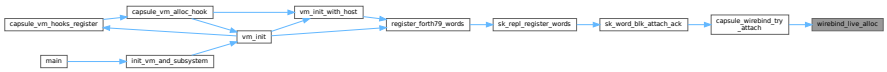
7.136.2.9 wirebind_live_alloc()

```
static wirebind_live_entry_t * wirebind_live_alloc (
            void  )  [static]
```

References g_wirebind_live, and WIREBIND_MAX_LIVE.

Referenced by capsule_wirebind_try_attach().

Here is the caller graph for this function:



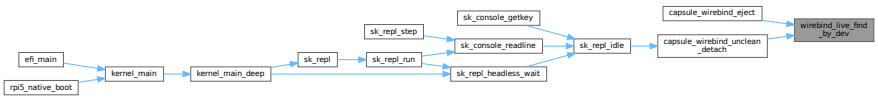
7.136.2.10 wirebind_live_find_by_dev()

```
static wirebind_live_entry_t * wirebind_live_find_by_dev (
            struct blkio_dev * dev )  [static]
```

References g_wirebind_live, and WIREBIND_MAX_LIVE.

Referenced by capsule_wirebind_eject(), and capsule_wirebind_unclean_detach().

Here is the caller graph for this function:



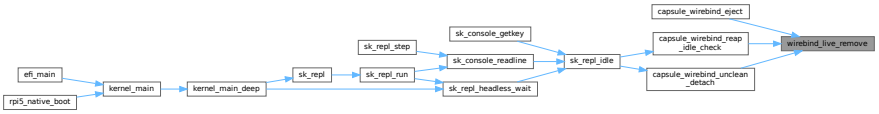
7.136.2.11 wirebind_live_remove()

```
static void wirebind_live_remove (
            wirebind_live_entry_t * e )  [static]
```

References wirebind_live_entry_t::dev, wirebind_live_entry_t::in_use, wirebind_live_entry_t::switch_slot, wirebind_live_entry_t::user_ and wirebind_live_entry_t::username.

Referenced by capsule_wirebind_eject(), capsule_wirebind_reap_idle_check(), and capsule_wirebind_unclean_detach().

Here is the caller graph for this function:



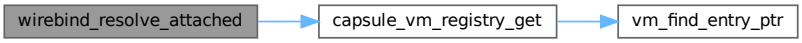
7.136.2.12 wirebind_resolve_attached()

```
static int wirebind_resolve_attached (
            VMRegistryEntry * out )  [static]
```

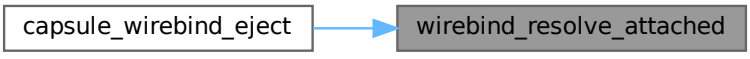
References capsule_vm_registry_get(), g_wirebind_attached_username, g_wirebind_attached_valid, g_wirebind_attached_vm_id, VMRegistryEntry::state, and VM_STATE_LIVE.

Referenced by capsule_wirebind_eject().

Here is the call graph for this function:



Here is the caller graph for this function:



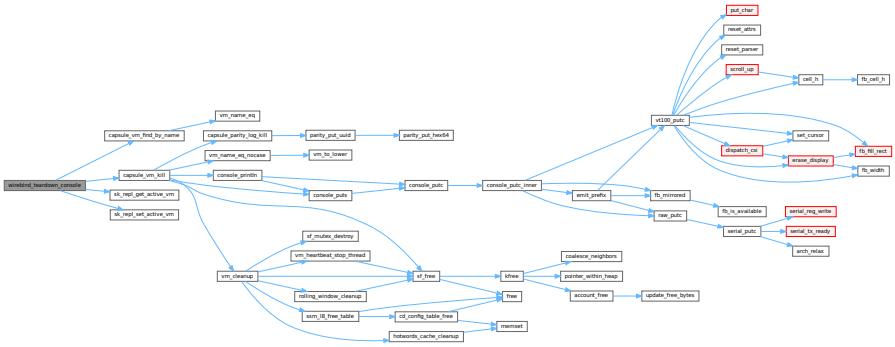
7.136.2.13 wirebind_teardown_console()

```
static void wirebind_teardown_console (
            const char * username )  [static]
```

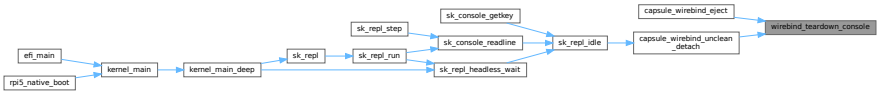
References capsule_vm_find_by_name(), capsule_vm_kill(), VMRegistryEntry::name, sk_repl_get_active_vm(), sk_repl_set_active_vm(), VMRegistryEntry::state, VMRegistryEntry::vm_ptr, and VM_STATE_LIVE.

Referenced by capsule_wirebind_eject(), and capsule_wirebind_unclean_detach().

Here is the call graph for this function:



Here is the caller graph for this function:



7.136.3 Variable Documentation

7.136.3.1 g_wirebind_attached_dev

```
struct blkio_dev* g_wirebind_attached_dev = (struct blkio_dev *) 0  [static]
```

Referenced by capsule_wirebind_eject(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), and capsule_wirebind_unclean_detach().

7.136.3.2 g_wirebind_attached_username

```
char g_wirebind_attached_username[32u] = {0}  [static]
```

Referenced by capsule_wirebind_attached_username(), capsule_wirebind_eject(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), capsule_wirebind_unclean_detach(), and wirebind_resolve_attached().

7.136.3.3 g_wirebind_attached_valid

```
int g_wirebind_attached_valid = 0  [static]
```

Referenced by capsule_wirebind_attached_username(), capsule_wirebind_eject(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), capsule_wirebind_unclean_detach(), and wirebind_resolve_attached().

7.136.3.4 g_wirebind_attached_vm_id

```
VMUuid g_wirebind_attached_vm_id  [static]
```

Referenced by capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), and wirebind_resolve_attached().

7.136.3.5 g_wirebind_live

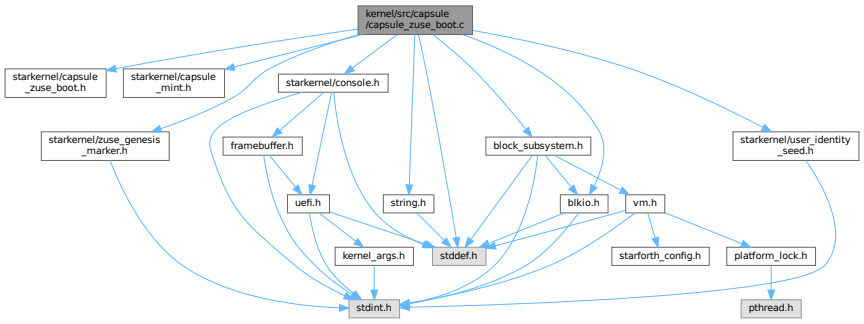
```
wirebind_live_entry_t g_wirebind_live[13u]  [static]
```

Referenced by capsule_wirebind_reap_idle_check(), wirebind_live_alloc(), and wirebind_live_find_by_dev().

7.137 kernel/src/capsule/capsule_zuse_boot.c File Reference

```
#include "starkernel/capsule_zuse_boot.h"
#include "starkernel/capsule_mint.h"
#include "starkernel/zuse_genesis_marker.h"
#include "starkernel/user_identity_seed.h"
#include "starkernel/console.h"
#include "block_subsystem.h"
#include "blkio.h"
#include <string.h>
#include <stddef.h>
```

Include dependency graph for capsule_zuse_boot.c:



Functions

- static int read_devblock (struct blkio_dev ∗dev, uint32_t devblock, uint8_t ∗buf4096)
- static int genesis_marker_read (zuse_genesis_marker_t ∗out)
- static void install_and_activate (VM ∗mama_vm, struct blkio_dev ∗dev, const uint8_t seed[32], const uint8_t pubkey[32])
- zuse_drive_t capsule_zuse_boot_check_key (struct blkio_dev ∗dev, homeblocks_sig_result_t sig_rc, const homeblocks_sig_t ∗sig, const uint8_t want[32], uint8_t seed[32], uint8_t pubkey[32])
- zuse_drive_t capsule_zuse_boot_check (struct blkio_dev ∗dev, homeblocks_sig_result_t sig_rc, const homeblocks_sig_t ∗sig, uint8_t seed[32], uint8_t pubkey[32])
- int capsule_zuse_boot_genesis_mint (struct blkio_dev ∗dev, uint8_t seed[32], uint8_t pubkey[32])
- int capsule_zuse_boot_root_pubkey (uint8_t pubkey[32])
- void capsule_zuse_boot_try_attach (struct blkio_dev ∗dev, homeblocks_sig_result_t sig_rc, const homeblocks_sig_t ∗sig, VM ∗mama_vm)
- void capsule_zuse_boot_logout (VM ∗mama_vm, struct blkio_dev ∗dev)
- struct blkio_dev ∗ capsule_zuse_boot_attached_dev (void)
- void capsule_zuse_boot_load_root_pubkey (VM ∗mama_vm)

Variables

- static int g_zuse_attached_this_device = 0
- static struct blkio_dev ∗ g_zuse_attached_dev = (struct blkio_dev ∗) 0

7.137.1 Function Documentation

7.137.1.1 capsule_zuse_boot_attached_dev()

```
struct blkio_dev * capsule_zuse_boot_attached_dev (
            void  )
```

References g_zuse_attached_dev, and g_zuse_attached_this_device.

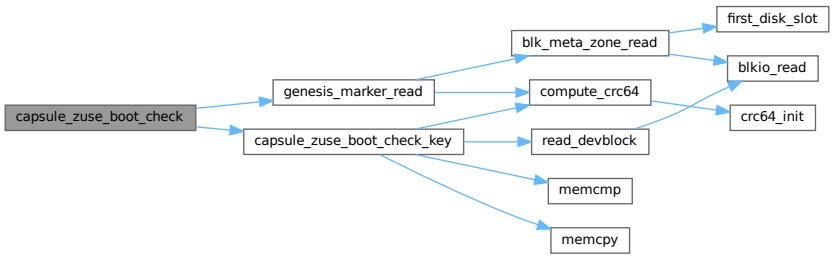
7.137.1.2 capsule_zuse_boot_check()

```
zuse_drive_t capsule_zuse_boot_check (
            struct blkio_dev * dev,
            homeblocks_sig_result_t sig_rc,
            const homeblocks_sig_t * sig,
            uint8_t seed[32],
            uint8_t pubkey[32] )
```

References capsule_zuse_boot_check_key(), genesis_marker_read(), HOMEBLOCKS_SIG_BLANK, and zuse_genesis_marker_t::zuse_pubkey.

Referenced by capsule_zuse_boot_try_attach().

Here is the call graph for this function:



Here is the caller graph for this function:

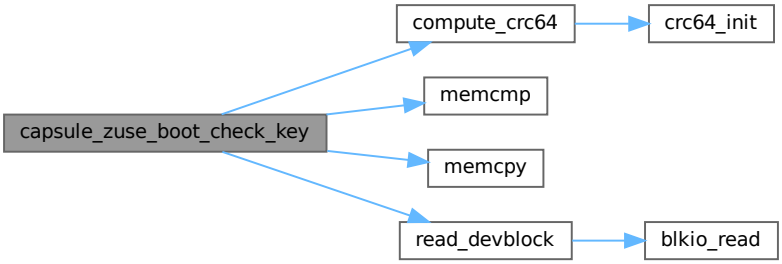
7.137.1.3 capsule_zuse_boot_check_key()

```
zuse_drive_t capsule_zuse_boot_check_key (
            struct blkio_dev * dev,
            homeblocks_sig_result_t sig_rc,
            const homeblocks_sig_t * sig,
            const uint8_t want[32],
            uint8_t seed[32],
            uint8_t pubkey[32] )
```

References compute_crc64(), user_identity_seed_t::crc, HOMEBLOCKS_SIG_OK, homeblocks_sig_t::identity_src_devblocks, homeblocks_sig_t::identity_src_offset, user_identity_seed_t::magic, memcmp(), memcpy(), user_identity_seed_t::pubkey, read_devblock(), user_identity_seed_t::seed, USER_IDENTITY_SEED_MAGIC, USER_IDENTITY_SEED_VERSION, and user_identity_seed_t::version.

Referenced by capsule_zuse_boot_check(), sk_v4_drive_arrived(), sk_v4_genesis(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

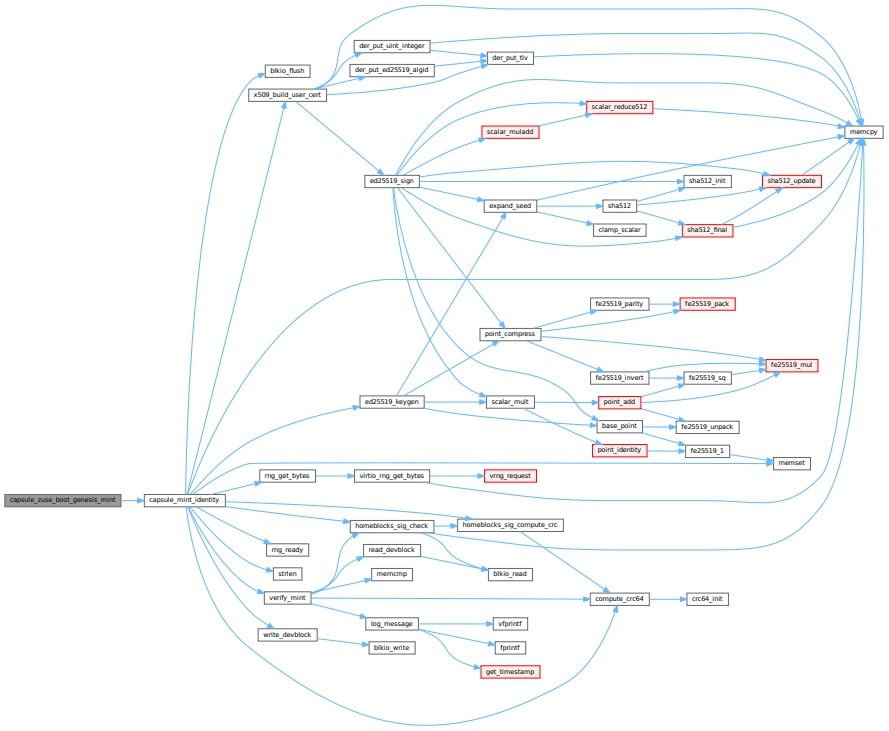
7.137.1.4 capsule_zuse_boot_genesis_mint()

```
int capsule_zuse_boot_genesis_mint (
            struct blkio_dev * dev,
            uint8_t seed[32],
            uint8_t pubkey[32] )
```

References capsule_mint_identity().

Referenced by capsule_zuse_boot_try_attach(), sk_v4_genesis(), and sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

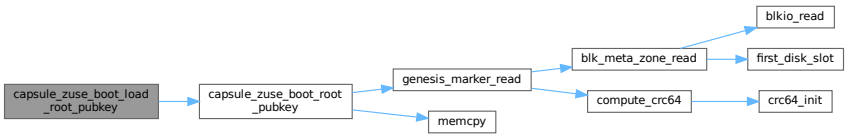
7.137.1.5 capsule_zuse_boot_load_root_pubkey()

```
void capsule_zuse_boot_load_root_pubkey (
            VM * mama_vm )
```

References capsule_zuse_boot_root_pubkey(), VM::zuse_root_pubkey, and VM::zuse_root_pubkey_known.

Referenced by kernel_main_deep(), and sk_word_blk_attach_ack().

Here is the call graph for this function:



Here is the caller graph for this function:

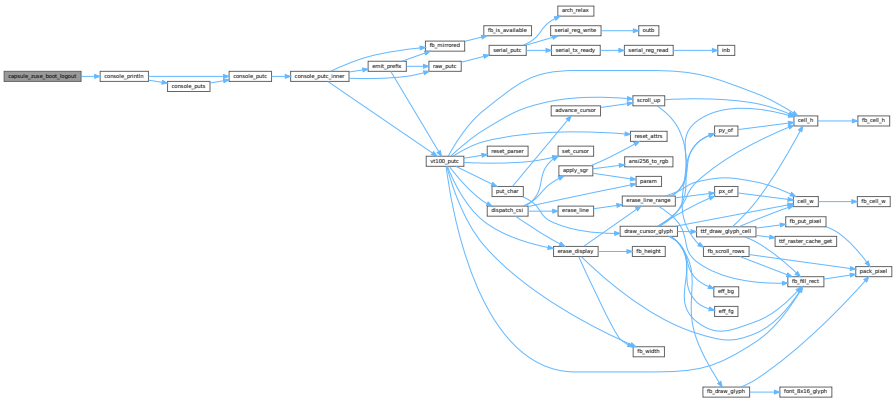
7.137.1.6 capsule_zuse_boot_logout()

```
void capsule_zuse_boot_logout (
            VM * mama_vm,
            struct blkio_dev * dev )
```

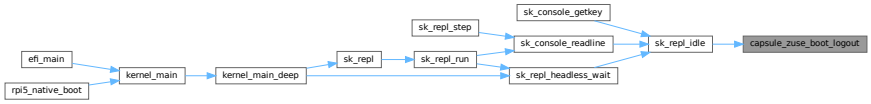
References console_println(), g_zuse_attached_dev, g_zuse_attached_this_device, and VM::zuse_session.

Referenced by sk_repl_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



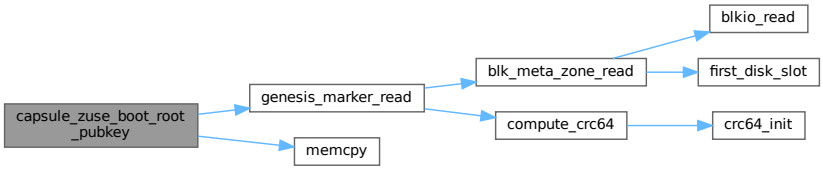
7.137.1.7 capsule_zuse_boot_root_pubkey()

```
int capsule_zuse_boot_root_pubkey (
            uint8_t pubkey[32] )
```

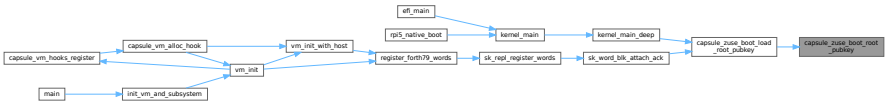
References genesis_marker_read(), memcpy(), and zuse_genesis_marker_t::zuse_pubkey.

Referenced by capsule_zuse_boot_load_root_pubkey().

Here is the call graph for this function:



Here is the caller graph for this function:



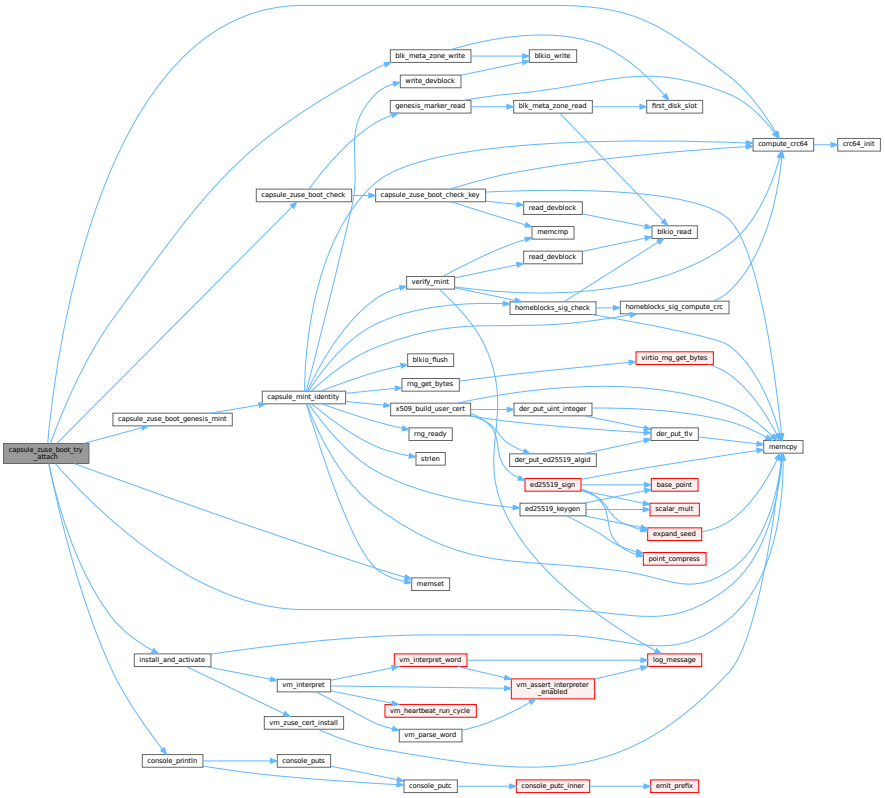
7.137.1.8 capsule_zuse_boot_try_attach()

```
void capsule_zuse_boot_try_attach (
            struct blkio_dev * dev,
            homeblocks_sig_result_t sig_rc,
            const homeblocks_sig_t * sig,
            VM * mama_vm )
```

References blk_meta_zone_write(), capsule_zuse_boot_check(), capsule_zuse_boot_genesis_mint(), compute_crc64(), console_println(), zuse_genesis_marker_t::crc, install_and_activate(), zuse_genesis_marker_t::magic, memcpy(), memset(), zuse_genesis_marker_t::version, ZUSE_GENESIS_MARKER_MAGIC, ZUSE_GENESIS_MARKER_VERSION, zuse_genesis_marker_t::zuse_pubkey, and VM::zuse_session.

Referenced by sk_word_blk_attach_ack().

Here is the call graph for this function:



Here is the caller graph for this function:

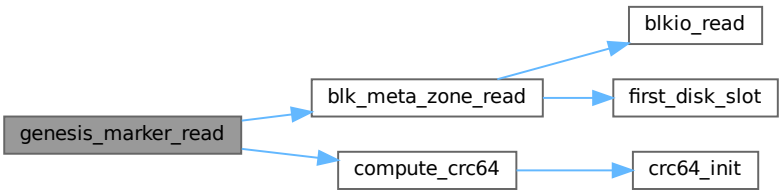
7.137.1.9 genesis_marker_read()

```
static int genesis_marker_read (
            zuse_genesis_marker_t * out )  [static]
```

References blk_meta_zone_read(), compute_crc64(), zuse_genesis_marker_t::crc, zuse_genesis_marker_t::magic, zuse_genesis_marker_t::version, ZUSE_GENESIS_MARKER_MAGIC, and ZUSE_GENESIS_MARKER_VERSION.

Referenced by capsule_zuse_boot_check(), and capsule_zuse_boot_root_pubkey().

Here is the call graph for this function:



Here is the caller graph for this function:

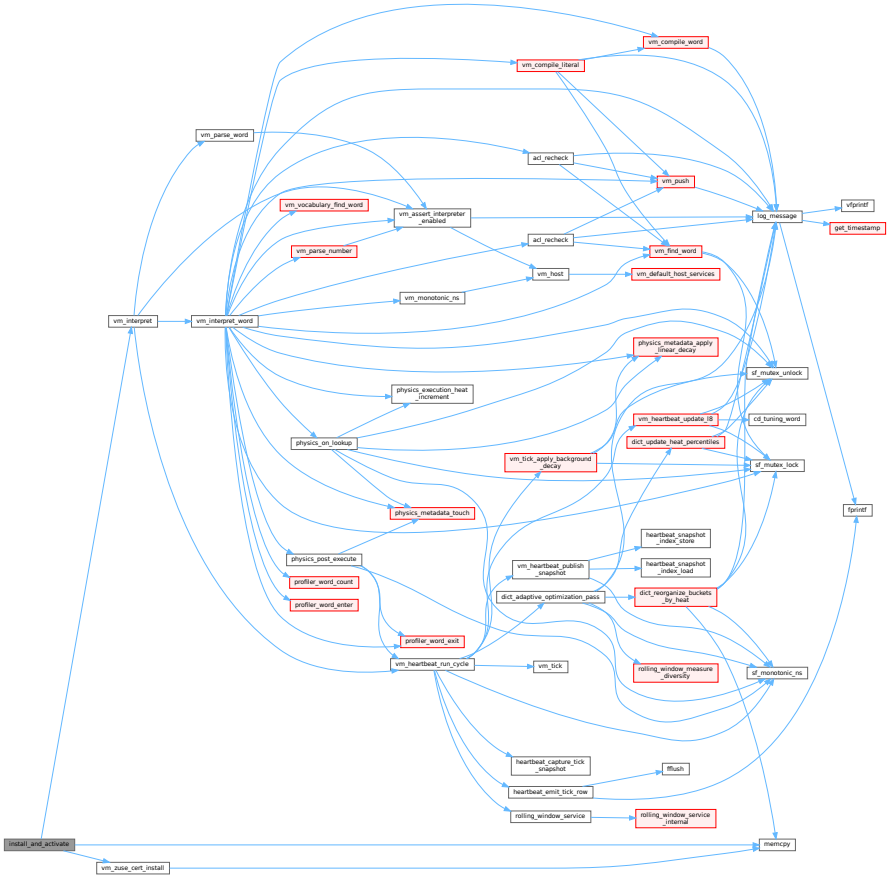
7.137.1.10 install_and_activate()

```
static void install_and_activate (
            VM * mama_vm,
            struct blkio_dev * dev,
            const uint8_t seed[32],
            const uint8_t pubkey[32] )  [static]
```

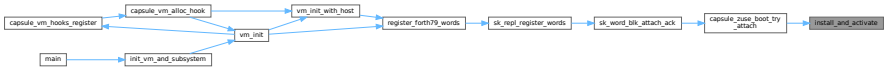
References g_zuse_attached_dev, g_zuse_attached_this_device, memcpy(), vm_interpret(), vm_zuse_cert_install(), VM::zuse_root_pubkey, and VM::zuse_root_pubkey_known.

Referenced by capsule_zuse_boot_try_attach().

Here is the call graph for this function:



Here is the caller graph for this function:



7.137.1.11 read_devblock()

```
static int read_devblock (
            struct blkio_dev * dev,
            uint32_t devblock,
            uint8_t * buf4096 )  [static]
```

References base, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, and blkio_read().

Referenced by capsule_zuse_boot_check_key().

Here is the call graph for this function:

Here is the caller graph for this function:

7.137.2 Variable Documentation

7.137.2.1 g_zuse_attached_dev

```
struct blkio_dev* g_zuse_attached_dev = (struct blkio_dev *) 0  [static]
```

Referenced by capsule_zuse_boot_attached_dev(), capsule_zuse_boot_logout(), and install_and_activate().

7.137.2.2 g_zuse_attached_this_device

```
int g_zuse_attached_this_device = 0  [static]
```

Referenced by capsule_zuse_boot_attached_dev(), capsule_zuse_boot_logout(), and install_and_activate().

7.138 kernel/src/capsule/mama_forth_words.c File Reference

Mama FORTH vocabulary implementation for kernel capsule system (M7.1)

7.138.1 Detailed Description

Mama FORTH vocabulary implementation for kernel capsule system (M7.1)

Implements the MAMA vocabulary - kernel-only words for the LithosAnanke capsule birth protocol. This file is ONLY compiled for **STARKERNEL** builds.

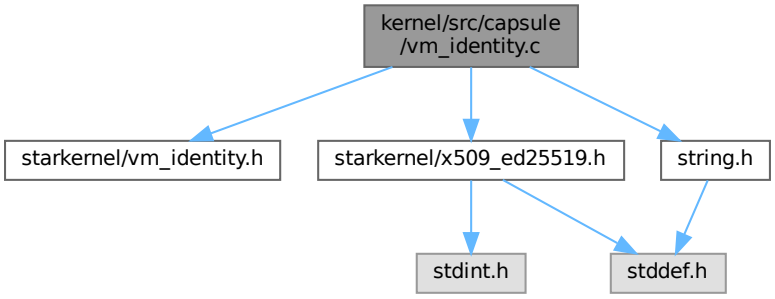
The MAMA vocabulary provides:

- Capsule directory enumeration
- Baby VM birth from production capsules
- Experiment execution on Mama
- VM registry queries

7.139 kernel/src/capsule/vm_identity.c File Reference

```
#include "starkernel/vm_identity.h"
#include "starkernel/x509_ed25519.h"
#include <string.h>
```

Include dependency graph for vm_identity.c:



Functions

- int vm_identity_has_cap (const VMIdentity *id, uint32_t cap)
- int vm_identity_from_cert (VMIdentity *out, const uint8_t *der, size_t der_len, const uint8_t issuer_↩ pubkey[32], const uint8_t drive_uuid[16], uint32_t acl_caps)

7.139.1 Function Documentation

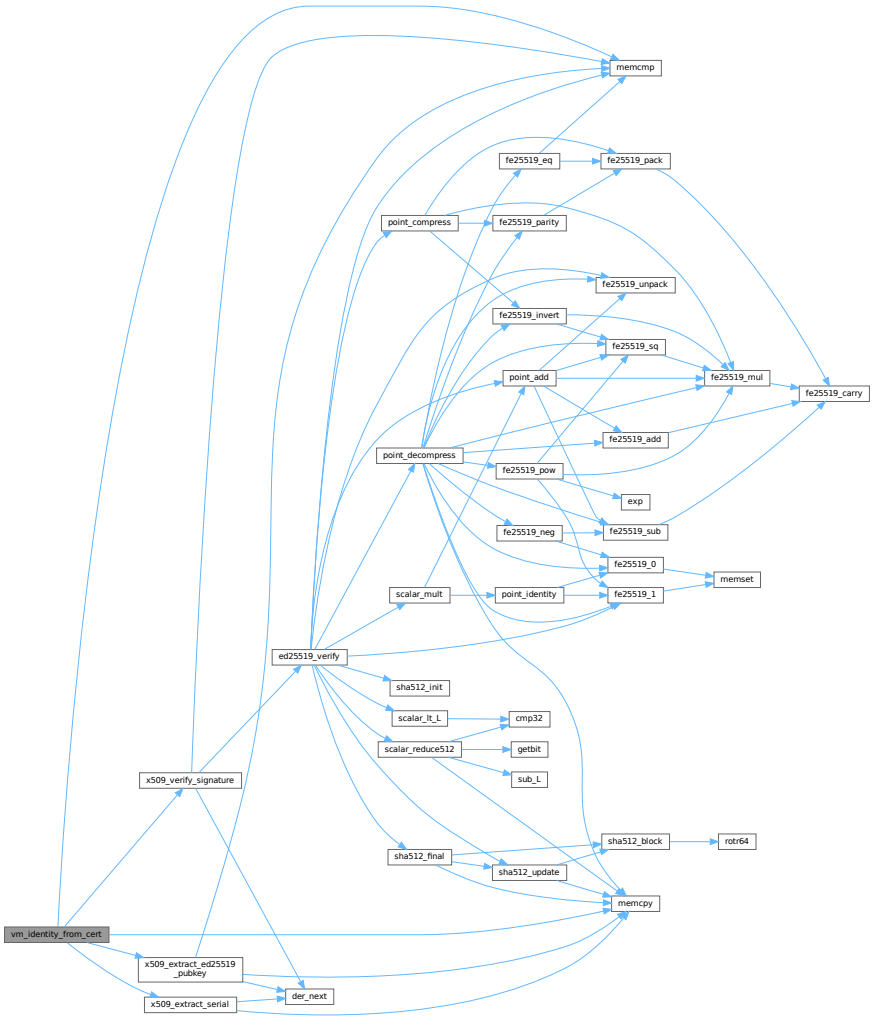
7.139.1.1 vm_identity_from_cert()

```
int vm_identity_from_cert (
            VMIdentity * out,
            const uint8_t * der,
            size_t der_len,
            const uint8_t issuer_pubkey[32],
            const uint8_t drive_uuid[16],
            uint32_t acl_caps )
```

References memcmp(), memcpy(), x509_extract_ed25519_pubkey(), x509_extract_serial(), and x509_verify_signature().

Referenced by capsule_wirebind_verify_cert().

Here is the call graph for this function:



Here is the caller graph for this function:

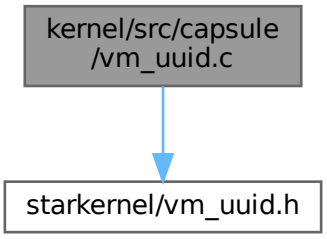
7.139.1.2 vm_identity_has_cap()

```
int vm_identity_has_cap (
            const VMIdentity * id,
            uint32_t cap )
```

7.140 kernel/src/capsule/vm_uuid.c File Reference

```
#include "starkernel/vm_uuid.h"
```

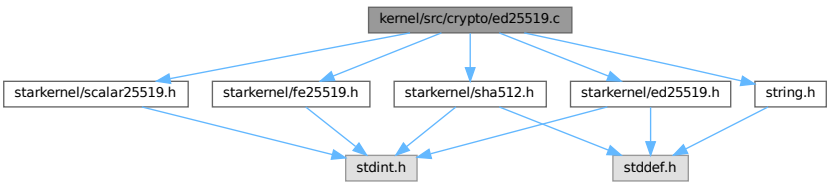
Include dependency graph for vm_uuid.c:



7.141 kernel/src/crypto/ed25519.c File Reference

```
#include "starkernel/ed25519.h"
#include "starkernel/fe25519.h"
#include "starkernel/scalar25519.h"
#include "starkernel/sha512.h"
#include <string.h>
```

Include dependency graph for ed25519.c:



Data Structures

- struct ed_point_t

Functions

- static void fe25519_pow (fe25519 *r, const fe25519 *a, const uint8_t exp[32])
- static void point_identity (ed_point_t *p)
- static void point_add (ed_point_t *r, const ed_point_t *p1, const ed_point_t *p2)
- static void scalar_mult (ed_point_t *r, const uint8_t scalar[32], const ed_point_t *p)
- static int point_decompress (ed_point_t *p, const uint8_t in[32])

- static void point_compress (uint8_t out[32], const ed_point_t *p)
- static void clamp_scalar (uint8_t a[32])
- static void expand_seed (const uint8_t seed[32], uint8_t a[32], uint8_t prefix[32])
- static void base_point (ed_point_t *b)
- void ed25519_keygen (const uint8_t seed[32], uint8_t pubkey_out[32])
- void ed25519_sign (const uint8_t seed[32], const uint8_t *msg, size_t msg_len, uint8_t sig_out[64])
- int ed25519_verify (const uint8_t pubkey[32], const uint8_t *msg, size_t msg_len, const uint8_t sig[64])

Variables

- static const uint8_t D_BYTES [32]
- static const uint8_t BX_BYTES [32]
- static const uint8_t BY_BYTES [32]
- static const uint8_t SQRTM1_BYTES [32]
- static const uint8_t POW_2252M2_EXP [32]

7.141.1 Function Documentation

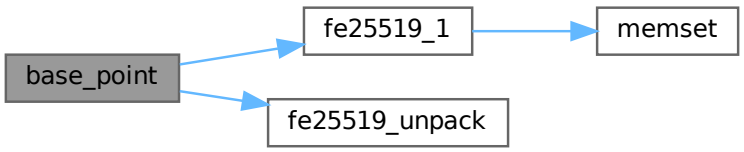
7.141.1.1 base_point()

```
static void base_point (
            ed_point_t * b )  [static]
```

References BX_BYTES, BY_BYTES, fe25519_1(), fe25519_unpack(), ed_point_t::x, ed_point_t::y, and ed_point_t::z.

Referenced by ed25519_keygen(), and ed25519_sign().

Here is the call graph for this function:



Here is the caller graph for this function:

7.141.1.2 clamp_scalar()

```
static void clamp_scalar (
            uint8_t a[32] )  [static]
```

Referenced by expand_seed().

Here is the caller graph for this function:

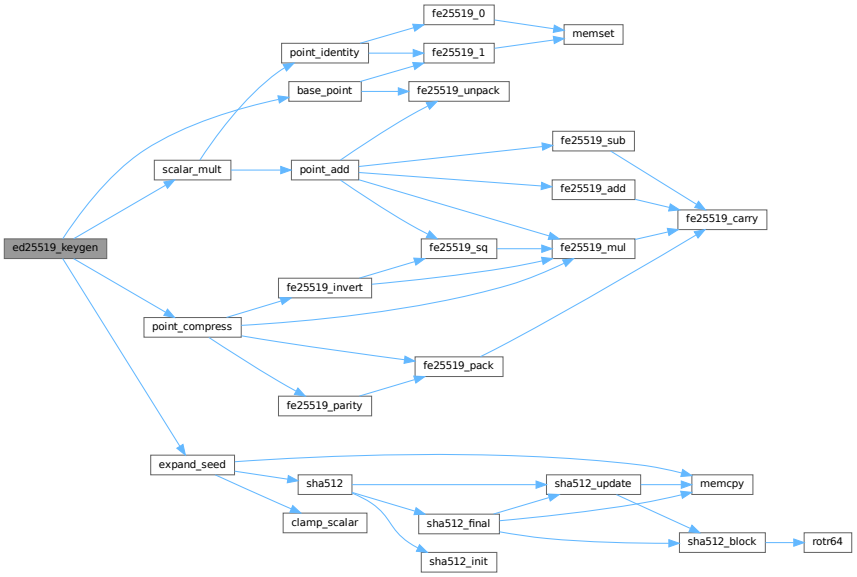
7.141.1.3 ed25519_keygen()

```
void ed25519_keygen (
            const uint8_t seed[32],
            uint8_t pubkey_out[32] )
```

References base_point(), expand_seed(), point_compress(), and scalar_mult().

Referenced by capsule_mint_identity().

Here is the call graph for this function:



Here is the caller graph for this function:

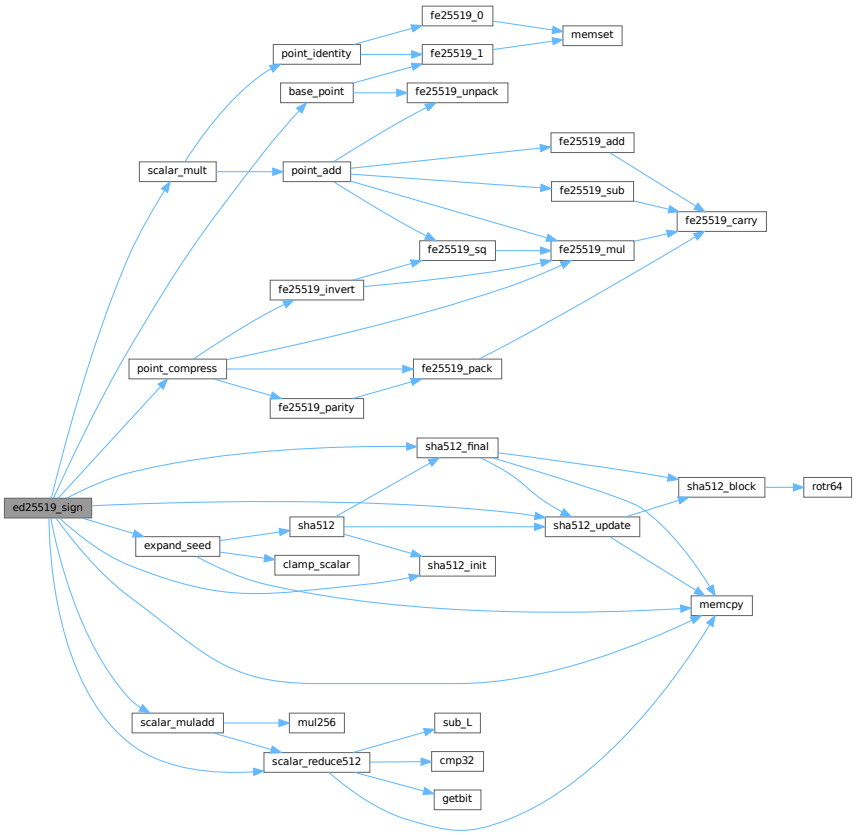
7.141.1.4 ed25519_sign()

```
void ed25519_sign (
            const uint8_t seed[32],
            const uint8_t * msg,
            size_t msg_len,
            uint8_t sig_out[64] )
```

References base_point(), expand_seed(), memcpy(), point_compress(), scalar_muladd(), scalar_mult(), scalar_reduce512(), sha512_final(), sha512_init(), and sha512_update().

Referenced by x509_build_user_cert().

Here is the call graph for this function:



Here is the caller graph for this function:

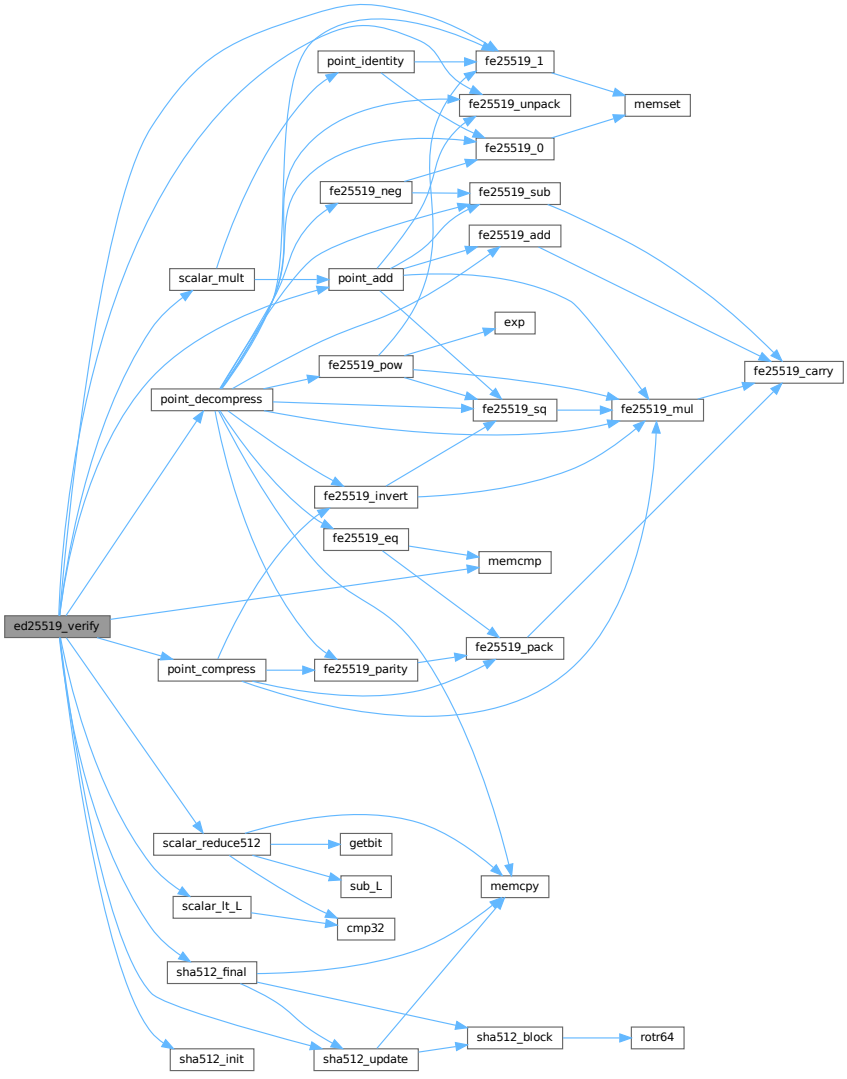
7.141.1.5 ed25519_verify()

```
int ed25519_verify (
            const uint8_t pubkey[32],
            const uint8_t * msg,
            size_t msg_len,
            const uint8_t sig[64] )
```

References BX_BYTES, BY_BYTES, fe25519_1(), fe25519_unpack(), memcmp(), point_add(), point_compress(), point_decompress(), scalar_lt_L(), scalar_mult(), scalar_reduce512(), sha512_final(), sha512_init(), sha512_update(), ed_point_t::x, ed_point_t::y, and ed_point_t::z.

Referenced by capsule_verify_signature(), and x509_verify_signature().

Here is the call graph for this function:



Here is the caller graph for this function:

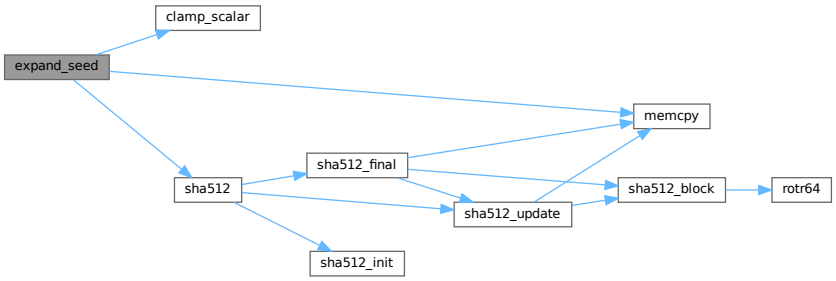
7.141.1.6 expand_seed()

```
static void expand_seed (
            const uint8_t seed[32],
            uint8_t a[32],
            uint8_t prefix[32] )  [static]
```

References clamp_scalar(), memcpy(), and sha512().

Referenced by ed25519_keygen(), and ed25519_sign().

Here is the call graph for this function:



Here is the caller graph for this function:

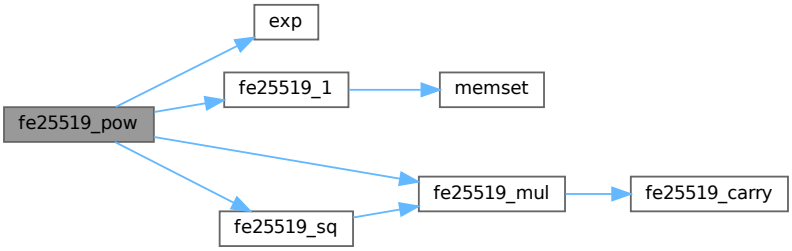
7.141.1.7 fe25519_pow()

```
static void fe25519_pow (
            fe25519 * r,
            const fe25519 * a,
            const uint8_t exp[32] )  [static]
```

References exp(), fe25519_1(), fe25519_mul(), and fe25519_sq().

Referenced by point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

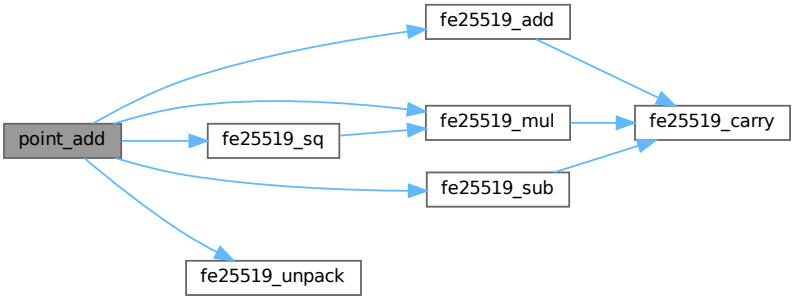
7.141.1.8 point_add()

```
static void point_add (
            ed_point_t * r,
            const ed_point_t * p1,
            const ed_point_t * p2 )  [static]
```

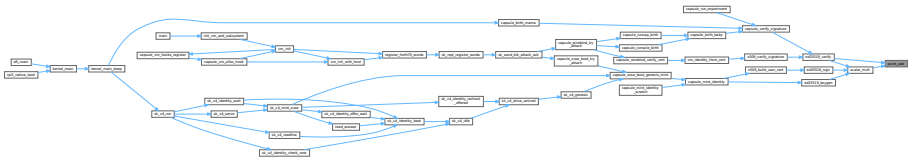
References D_BYTES, fe25519_add(), fe25519_mul(), fe25519_sq(), fe25519_sub(), fe25519_unpack(), ed_point_t::x, ed_point_t::y, and ed_point_t::z.

Referenced by ed25519_verify(), and scalar_mult().

Here is the call graph for this function:



Here is the caller graph for this function:



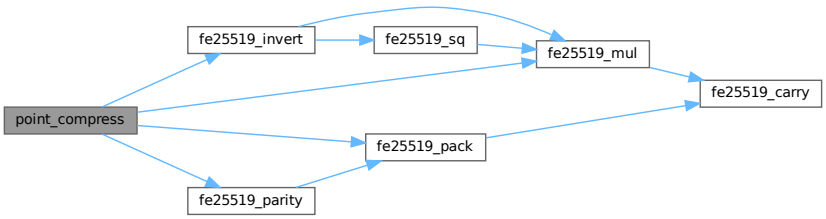
7.141.1.9 point_compress()

```
static void point_compress (
            uint8_t out[32],
            const ed_point_t * p )  [static]
```

References fe25519_invert(), fe25519_mul(), fe25519_pack(), fe25519_parity(), ed_point_t::x, ed_point_t::y, and ed_point_t::z.

Referenced by ed25519_keygen(), ed25519_sign(), and ed25519_verify().

Here is the call graph for this function:



Here is the caller graph for this function:

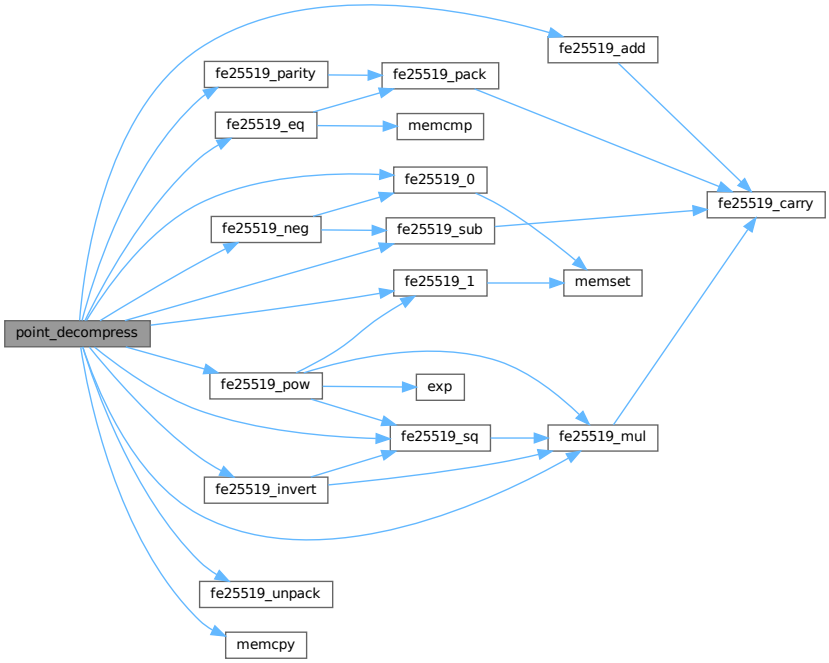
7.141.1.10 point_decompress()

```
static int point_decompress (
            ed_point_t * p,
            const uint8_t in[32] )  [static]
```

References D_BYTES, fe25519_0(), fe25519_1(), fe25519_add(), fe25519_eq(), fe25519_invert(), fe25519_mul(), fe25519_neg(), fe25519_parity(), fe25519_pow(), fe25519_sq(), fe25519_sub(), fe25519_unpack(), memcpy(), POW_2252M2_EXP, SQRTM1_BYTES, ed_point_t::x, ed_point_t::y, ed_point_t::z, and zero.

Referenced by ed25519_verify().

Here is the call graph for this function:



Here is the caller graph for this function:

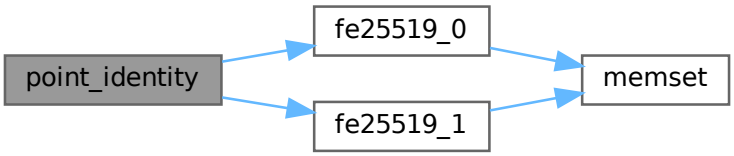
7.141.1.11 point_identity()

```
static void point_identity (
            ed_point_t * p )  [static]
```

References fe25519_0(), fe25519_1(), ed_point_t::x, ed_point_t::y, and ed_point_t::z.

Referenced by scalar_mult().

Here is the call graph for this function:



Here is the caller graph for this function:

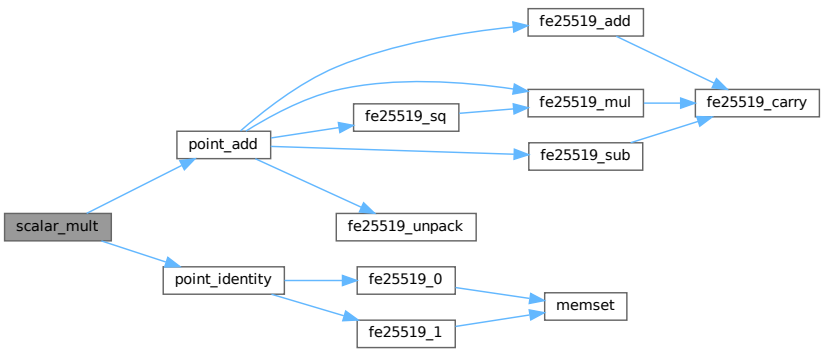
7.141.1.12 scalar_mult()

```
static void scalar_mult (
            ed_point_t * r,
            const uint8_t scalar[32],
            const ed_point_t * p )  [static]
```

References point_add(), and point_identity().

Referenced by ed25519_keygen(), ed25519_sign(), and ed25519_verify().

Here is the call graph for this function:



Here is the caller graph for this function:

7.141.2 Variable Documentation

7.141.2.1 BX_BYTES

```
const uint8_t BX_BYTES[32]  [static]
```

Initial value:

```
= {
    0x1a, 0xd5, 0x25, 0x8f, 0x60, 0x2d, 0x56, 0xc9,
    0xb2, 0xa7, 0x25, 0x95, 0x60, 0xc7, 0x2c, 0x69,
    0x5c, 0xdc, 0xd6, 0xfd, 0x31, 0xe2, 0xa4, 0xc0,
    0xfe, 0x53, 0x6e, 0xcd, 0xd3, 0x36, 0x69, 0x21,
}
```

Referenced by base_point(), and ed25519_verify().

7.141.2.2 BY_BYTES

```
const uint8_t BY_BYTES[32]  [static]
```

Initial value:

```
= {
    0x58, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
    0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
}
```

Referenced by base_point(), and ed25519_verify().

7.141.2.3 D_BYTES

```
const uint8_t D_BYTES[32]  [static]
```

Initial value:

```
= {
    0xa3, 0x78, 0x59, 0x13, 0xca, 0x4d, 0xeb, 0x75,
    0xab, 0xd8, 0x41, 0x41, 0x4d, 0x0a, 0x70, 0x00,
    0x98, 0xe8, 0x79, 0x77, 0x79, 0x40, 0xc7, 0x8c,
    0x73, 0xfe, 0x6f, 0x2b, 0xee, 0x6c, 0x03, 0x52,
}
```

Referenced by point_add(), and point_decompress().

7.141.2.4 POW_2252M2_EXP

```
const uint8_t POW_2252M2_EXP[32]  [static]
```

Initial value:

```
= {
    0xfe, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x0f,
}
```

Referenced by point_decompress().

7.141.2.5 SQRTM1_BYTES

```
const uint8_t SQRTM1_BYTES[32]  [static]
```

Initial value:

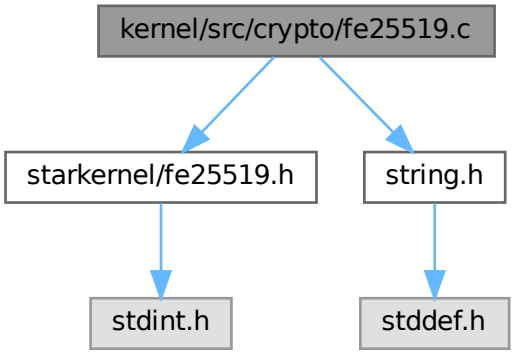
```
= {
    0xb0, 0xa0, 0x0e, 0x4a, 0x27, 0x1b, 0xee, 0xc4,
    0x78, 0xe4, 0x2f, 0xad, 0x06, 0x18, 0x43, 0x2f,
    0xa7, 0xd7, 0xfb, 0x3d, 0x99, 0x00, 0x4d, 0x2b,
    0x0b, 0xdf, 0xc1, 0x4f, 0x80, 0x24, 0x83, 0x2b,
}
```

Referenced by point_decompress().

7.142 kernel/src/crypto/fe25519.c File Reference

```
#include "starkernel/fe25519.h"
#include <string.h>
```

Include dependency graph for fe25519.c:



Macros

- #define WBITS 51
- #define MASK51 (((int64_t)1 << WBITS) - 1)

Typedefs

- typedef unsigned __int128 u128

Functions

- void fe25519_0 (fe25519 *r)
- void fe25519_1 (fe25519 *r)
- void fe25519_copy (fe25519 *r, const fe25519 *a)
- static void fe25519_carry (fe25519 *r)
- void fe25519_add (fe25519 *r, const fe25519 *a, const fe25519 *b)
- void fe25519_sub (fe25519 *r, const fe25519 *a, const fe25519 *b)
- void fe25519_neg (fe25519 *r, const fe25519 *a)
- void fe25519_mul (fe25519 *r, const fe25519 *a, const fe25519 *b)
- void fe25519_sq (fe25519 *r, const fe25519 *a)
- void fe25519_mul_small (fe25519 *r, const fe25519 *a, uint32_t c)
- void fe25519_invert (fe25519 *r, const fe25519 *a)
- void fe25519_pack (uint8_t out[32], const fe25519 *a)
- void fe25519_unpack (fe25519 *r, const uint8_t in[32])
- int fe25519_eq (const fe25519 *a, const fe25519 *b)
- int fe25519_parity (const fe25519 *a)

7.142.1 Macro Definition Documentation

7.142.1.1 MASK51

```
#define MASK51 (((int64_t)1 << WBITS) - 1)
```

7.142.1.2 WBITS

```
#define WBITS 51
```

7.142.2 Typedef Documentation

7.142.2.1 u128

```
typedef unsigned __int128 u128
```

7.142.3 Function Documentation

7.142.3.1 fe25519_0()

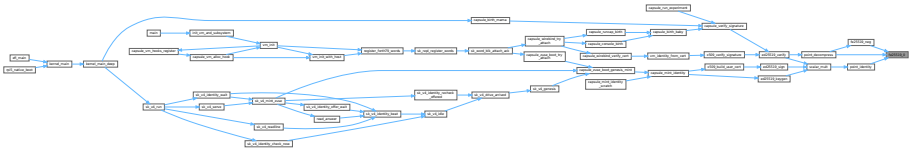
```
void fe25519_0 (
            fe25519 * r )
```

References memset(), and fe25519::v.

Referenced by fe25519_neg(), point_decompress(), and point_identity().

Here is the call graph for this function:

Here is the caller graph for this function:



7.142.3.2 fe25519_1()

```
void fe25519_1 (
            fe25519 * r )
```

References memset(), and fe25519::v.

Referenced by base_point(), ed25519_verify(), fe25519_pow(), point_decompress(), and point_identity().

Here is the call graph for this function:

Here is the caller graph for this function:

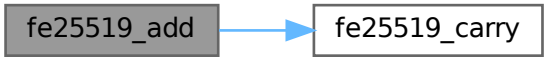
7.142.3.3 fe25519_add()

```
void fe25519_add (
            fe25519 * r,
            const fe25519 * a,
            const fe25519 * b )
```

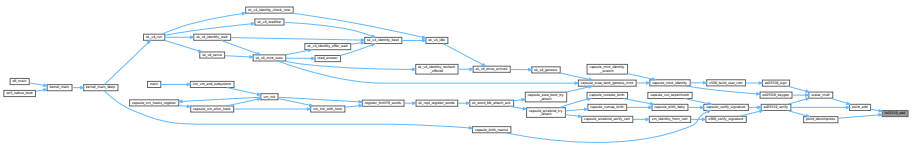
References fe25519_carry(), and fe25519::v.

Referenced by point_add(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:



7.142.3.4 fe25519_carry()

```
static void fe25519_carry (
            fe25519 * r )  [static]
```

References fe25519::v, and WBITS.

Referenced by fe25519_add(), fe25519_mul(), fe25519_mul_small(), fe25519_pack(), and fe25519_sub().

Here is the caller graph for this function:

7.142.3.5 fe25519_copy()

```
void fe25519_copy (
            fe25519 * r,
            const fe25519 * a )
```

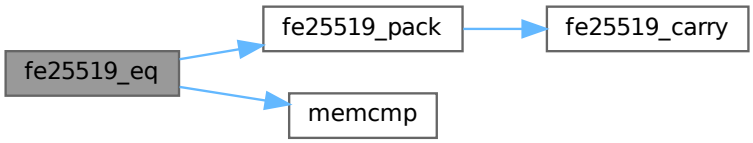
7.142.3.6 fe25519_eq()

```
int fe25519_eq (
            const fe25519 * a,
            const fe25519 * b )
```

References fe25519_pack(), and memcmp().

Referenced by point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

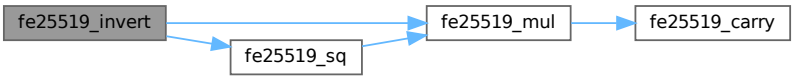
7.142.3.7 fe25519_invert()

```
void fe25519_invert (
            fe25519 * r,
            const fe25519 * a )
```

References fe25519_mul(), and fe25519_sq().

Referenced by point_compress(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

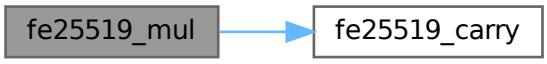
7.142.3.8 fe25519_mul()

```
void fe25519_mul (
            fe25519 * r,
            const fe25519 * a,
            const fe25519 * b )
```

References fe25519_carry(), MASK51, fe25519::v, and WBITS.

Referenced by fe25519_invert(), fe25519_pow(), fe25519_sq(), point_add(), point_compress(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

7.142.3.9 fe25519_mul_small()

```
void fe25519_mul_small (
            fe25519 * r,
            const fe25519 * a,
            uint32_t c )
```

References fe25519_carry(), MASK51, fe25519::v, and WBITS.

Here is the call graph for this function:

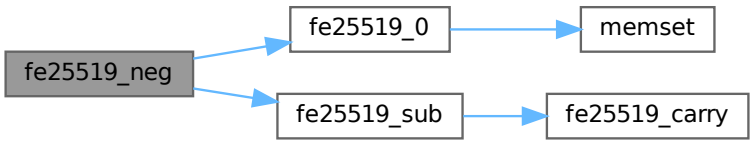
7.142.3.10 fe25519_neg()

```
void fe25519_neg (
            fe25519 * r,
            const fe25519 * a )
```

References fe25519_0(), fe25519_sub(), and zero.

Referenced by point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

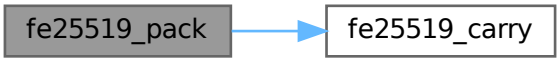
7.142.3.11 fe25519_pack()

```
void fe25519_pack (
            uint8_t out[32],
            const fe25519 * a )
```

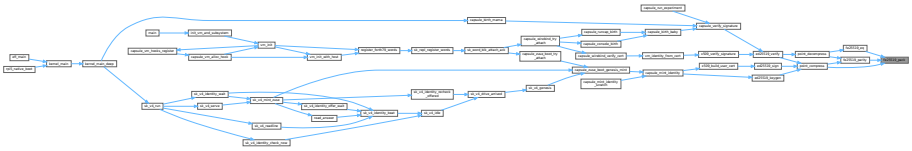
References fe25519_carry(), MASK51, fe25519::v, and WBITS.

Referenced by fe25519_eq(), fe25519_parity(), and point_compress().

Here is the call graph for this function:



Here is the caller graph for this function:



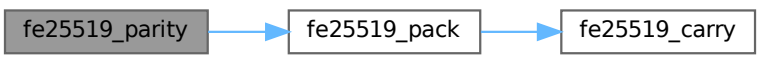
7.142.3.12 fe25519_parity()

```
int fe25519_parity (
            const fe25519 * a )
```

References fe25519_pack().

Referenced by point_compress(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

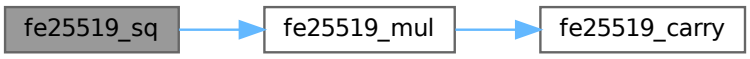
7.142.3.13 fe25519_sq()

```
void fe25519_sq (
             fe25519 * r,
             const fe25519 * a )
```

References fe25519_mul().

Referenced by fe25519_invert(), fe25519_pow(), point_add(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:

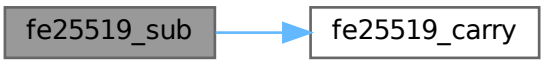
7.142.3.14 fe25519_sub()

```
void fe25519_sub (
             fe25519 * r,
             const fe25519 * a,
             const fe25519 * b )
```

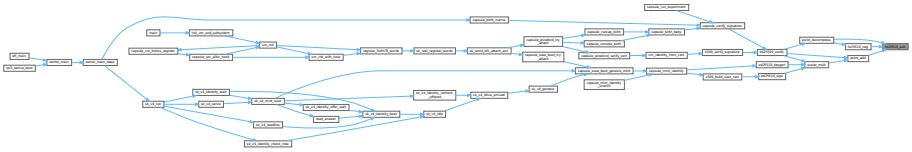
References fe25519_carry(), and fe25519::v.

Referenced by fe25519_neg(), point_add(), and point_decompress().

Here is the call graph for this function:



Here is the caller graph for this function:



7.142.3.15 fe25519_unpack()

```
void fe25519_unpack (
            fe25519 * r,
            const uint8_t in[32] )
```

References MASK51, fe25519::v, and WBITS.

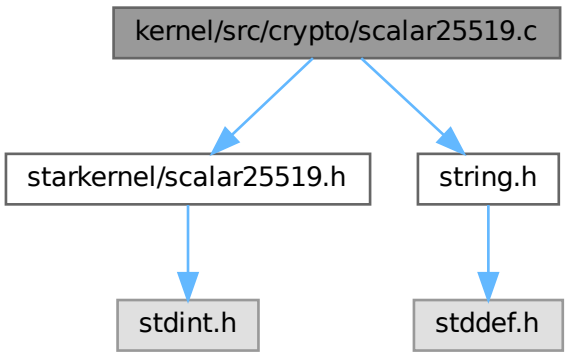
Referenced by base_point(), ed25519_verify(), point_add(), and point_decompress().

Here is the caller graph for this function:

7.143 kernel/src/crypto/scalar25519.c File Reference

```
#include "starkernel/scalar25519.h"
#include <string.h>
```

Include dependency graph for scalar25519.c:



Functions

- static int getbit (const uint8_t *big, int nbytes, int i)
- static void sub_L (uint8_t r[32])
- static int cmp32 (const uint8_t a[32], const uint8_t b[32])
- int scalar_lt_L (const uint8_t s[32])
- void scalar_reduce512 (uint8_t out[32], const uint8_t in[64])
- static void mul256 (uint32_t out_limbs[16], const uint8_t a[32], const uint8_t b[32])
- void scalar_muladd (uint8_t out[32], const uint8_t a[32], const uint8_t b[32], const uint8_t c[32])

Variables

- static const uint8_t L_BYTES [32]

7.143.1 Function Documentation

7.143.1.1 cmp32()

```
static int cmp32 (
            const uint8_t a[32],
            const uint8_t b[32] )  [static]
```

Referenced by scalar_lt_L(), and scalar_reduce512().

Here is the caller graph for this function:

7.143.1.2 getbit()

```
static int getbit (
            const uint8_t * big,
            int nbytes,
            int i )  [static]
```

Referenced by scalar_reduce512().

Here is the caller graph for this function:

7.143.1.3 mul256()

```
static void mul256 (
            uint32_t out_limbs[16],
            const uint8_t a[32],
            const uint8_t b[32] )  [static]
```

Referenced by scalar_muladd().

Here is the caller graph for this function:

7.143.1.4 scalar_lt_L()

```
int scalar_lt_L (
            const uint8_t s[32] )
```

References cmp32(), and L_BYTES.

Referenced by ed25519_verify().

Here is the call graph for this function:

Here is the caller graph for this function:

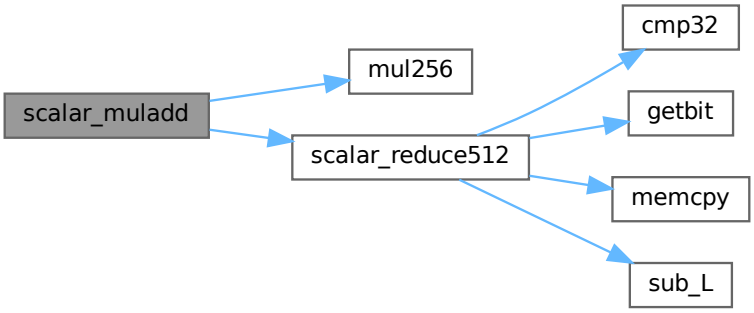
7.143.1.5 scalar_muladd()

```
void scalar_muladd (
            uint8_t out[32],
            const uint8_t a[32],
            const uint8_t b[32],
            const uint8_t c[32] )
```

References mul256(), and scalar_reduce512().

Referenced by ed25519_sign().

Here is the call graph for this function:



Here is the caller graph for this function:

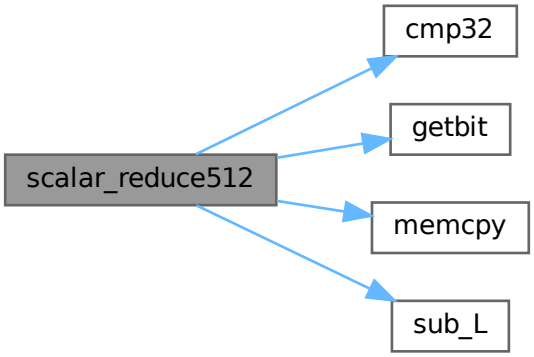
7.143.1.6 scalar_reduce512()

```
void scalar_reduce512 (
            uint8_t out[32],
            const uint8_t in[64] )
```

References cmp32(), getbit(), L_BYTES, memcpy(), and sub_L().

Referenced by ed25519_sign(), ed25519_verify(), and scalar_muladd().

Here is the call graph for this function:



Here is the caller graph for this function:

7.143.1.7 sub_L()

```
static void sub_L (
            uint8_t r[32] )  [static]
```

References L_BYTES.

Referenced by scalar_reduce512().

Here is the caller graph for this function:

7.143.2 Variable Documentation

7.143.2.1 L_BYTES

```
const uint8_t L_BYTES[32]  [static]
```

Initial value:

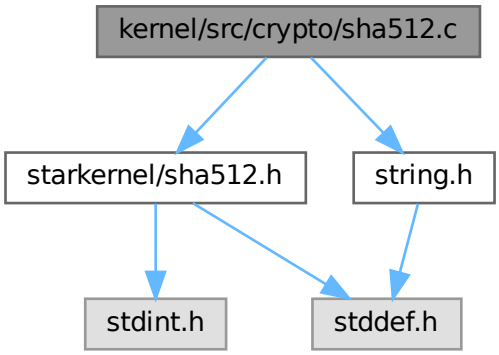
```
= {
    0xed, 0xd3, 0xf5, 0x5c, 0x1a, 0x63, 0x12, 0x58,
    0xd6, 0x9c, 0xf7, 0xa2, 0xde, 0xf9, 0xde, 0x14,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10
}
```

Referenced by scalar_lt_L(), scalar_reduce512(), and sub_L().

7.144 kernel/src/crypto/sha512.c File Reference

```
#include "starkernel/sha512.h"
#include <string.h>
```

Include dependency graph for sha512.c:



Functions

- static uint64_t rotr64 (uint64_t x, int n)
- static void sha512_block (sha512_ctx_t *ctx, const uint8_t block[128])
- void sha512_init (sha512_ctx_t *ctx)
- void sha512_update (sha512_ctx_t *ctx, const uint8_t *data, size_t len)
- void sha512_final (sha512_ctx_t *ctx, uint8_t out[64])
- void sha512 (const uint8_t *data, size_t len, uint8_t out[64])

Variables

- static const uint64_t K [80]

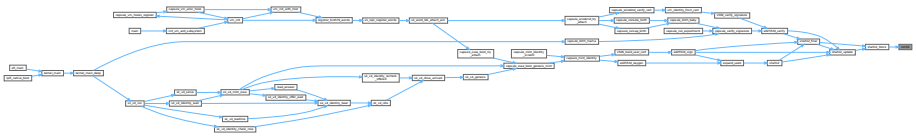
7.144.1 Function Documentation

7.144.1.1 rotr64()

```
static uint64_t rotr64 (
            uint64_t x,
            int n )  [inline], [static]
```

Referenced by sha512_block().

Here is the caller graph for this function:



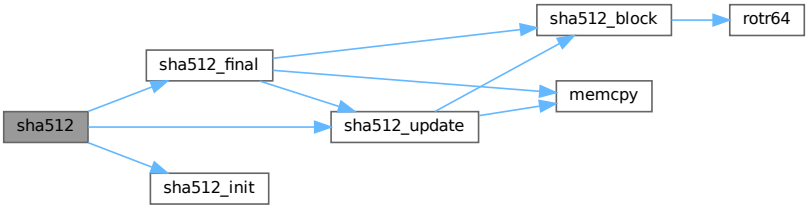
7.144.1.2 sha512()

```
void sha512 (
            const uint8_t * data,
            size_t len,
            uint8_t out[64] )
```

References sha512_final(), sha512_init(), and sha512_update().

Referenced by expand_seed().

Here is the call graph for this function:



Here is the caller graph for this function:

7.144.1.3 sha512_block()

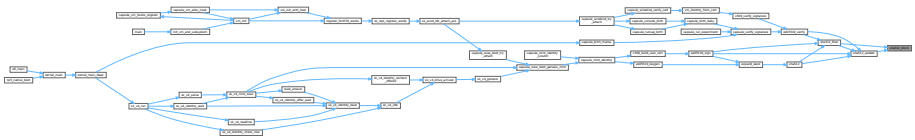
```
static void sha512_block (
            sha512_ctx_t * ctx,
            const uint8_t block[128] )  [static]
```

References g, K, rotr64(), and sha512_ctx_t::state.

Referenced by sha512_final(), and sha512_update().

Here is the call graph for this function:

Here is the caller graph for this function:



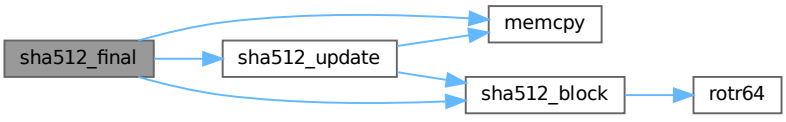
7.144.1.4 sha512_final()

```
void sha512_final (
            sha512_ctx_t * ctx,
            uint8_t out[64] )
```

References sha512_ctx_t::bitlen, sha512_ctx_t::buf, sha512_ctx_t::buf_len, memcpy(), sha512_block(), sha512_update(), sha512_ctx_t::state, and zero.

Referenced by ed25519_sign(), ed25519_verify(), and sha512().

Here is the call graph for this function:



Here is the caller graph for this function:

7.144.1.5 sha512_init()

```
void sha512_init (
            sha512_ctx_t * ctx )
```

References sha512_ctx_t::bitlen, sha512_ctx_t::buf_len, and sha512_ctx_t::state.

Referenced by ed25519_sign(), ed25519_verify(), and sha512().

Here is the caller graph for this function:

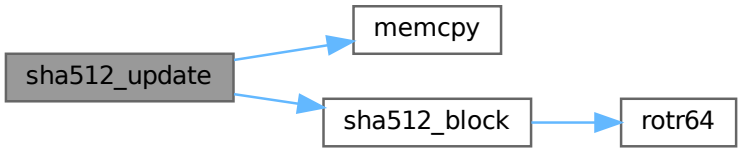
7.144.1.6 sha512_update()

```
void sha512_update (
            sha512_ctx_t * ctx,
            const uint8_t * data,
            size_t len )
```

References sha512_ctx_t::bitlen, sha512_ctx_t::buf, sha512_ctx_t::buf_len, memcpy(), and sha512_block().

Referenced by ed25519_sign(), ed25519_verify(), sha512(), and sha512_final().

Here is the call graph for this function:



Here is the caller graph for this function:

7.144.2 Variable Documentation

7.144.2.1 K

```
const uint64_t K[80]  [static]
```

Initial value:

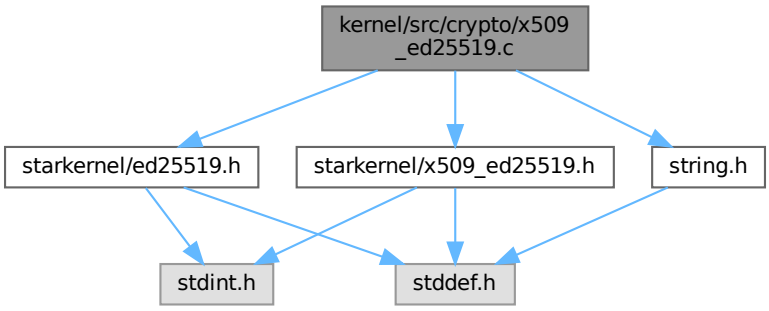
```
= {
    0x428a2f98d728ae22ULL, 0x7137449123ef65cdULL, 0xb5c0fbcfec4d3b2fULL, 0xe9b5dba58189dbbcULL,
    0x3956c25bf348b538ULL, 0x59f111f1b605d019ULL, 0x923f82a4af194f9bULL, 0xab1c5ed5da6d8118ULL,
    0xd807aa98a3030242ULL, 0x12835b0145706fbeULL, 0x243185be4ee4b28cULL, 0x550c7dc3d5ffb4e2ULL,
    0x72be5d74f27b896fULL, 0x80deb1fe3b1696b1ULL, 0x9bdc06a725c71235ULL, 0xc19bf174cf692694ULL,
    0xe49b69c19ef14ad2ULL, 0xefbe4786384f25e3ULL, 0x0fc19dc68b8cd5b5ULL, 0x240ca1cc77ac9c65ULL,
    0x2de92c6f592b0275ULL, 0x4a7484aa6ea6e483ULL, 0x5cb0a9dcbd41fbd4ULL, 0x76f988da831153b5ULL,
    0x983e5152ee66dfabULL, 0xa831c66d2db43210ULL, 0xb00327c898fb213fULL, 0xbf597fc7beef0ee4ULL,
    0xc6e00bf33da88fc2ULL, 0xd5a79147930aa725ULL, 0x06ca6351e003826fULL, 0x142929670a0e6e70ULL,
    0x27b70a8546d22ffcULL, 0x2e1b21385c26c926ULL, 0x4d2c6dfc5ac42aedULL, 0x53380d139d95b3dfULL,
    0x650a73548baf63deULL, 0x766a0abb3c77b2a8ULL, 0x81c2c92e47edaee6ULL, 0x92722c851482353bULL,
    0xa2bfe8a14cf10364ULL, 0xa81a664bbc423001ULL, 0xc24b8b70d0f89791ULL, 0xc76c51a30654be30ULL,
    0xd192e819d6ef5218ULL, 0xd69906245565a910ULL, 0xf40e35855771202aULL, 0x106aa07032bbd1b8ULL,
    0x19a4c116b8d2d0c8ULL, 0x1e376c085141ab53ULL, 0x2748774cdf8eeb99ULL, 0x34b0bcb5e19b48a8ULL,
    0x391c0cb3c5c95a63ULL, 0x4ed8aa4ae3418acbULL, 0x5b9cca4f7763e373ULL, 0x682e6ff3d6b2b8a3ULL,
    0x748f82ee5defb2fcULL, 0x78a5636f43172f60ULL, 0x84c87814a1f0ab72ULL, 0x8cc702081a6439ecULL,
    0x90befffa23631e28ULL, 0xa4506cebde82bde9ULL, 0xbef9a3f7b2c67915ULL, 0xc67178f2e372532bULL,
    0xca273eceea26619cULL, 0xd186b8c721c0c207ULL, 0xeada7dd6cde0eb1eULL, 0xf57d4f7fee6ed178ULL,
    0x06f067aa72176fbaULL, 0x0a637dc5a2c898a6ULL, 0x113f9804bef90daeULL, 0x1b710b35131c471bULL,
    0x28db77f523047d84ULL, 0x32caab7b40c72493ULL, 0x3c9ebe0a15c9bebcULL, 0x431d67c49c100d4cULL,
    0x4cc5d4becb3e42b6ULL, 0x597f299cfc657e2aULL, 0x5fcb6fab3ad6faecULL, 0x6c44198c4a475817ULL
}
```

Referenced by sha512_block().

7.145 kernel/src/crypto/x509_ed25519.c File Reference

```
#include "starkernel/x509_ed25519.h"
#include "starkernel/ed25519.h"
#include <string.h>
```

Include dependency graph for x509_ed25519.c:



Data Structures

- struct der_span_t

Functions

- static int der_next (const uint8_t **cursor, const uint8_t *limit, uint8_t *tag_out, der_span_t *content_out)
- int x509_extract_ed25519_pubkey (const uint8_t *der, size_t der_len, uint8_t pubkey_out[32])
- int x509_verify_signature (const uint8_t *der, size_t der_len, const uint8_t issuer_pubkey[32])
- int x509_extract_serial (const uint8_t *der, size_t der_len, uint8_t *serial_out, size_t serial_out_cap, size_t *serial_len_out)
- static int der_put_tlv (uint8_t **pos, uint8_t *limit, uint8_t tag, const uint8_t *content, size_t content_len)
- static int der_put_ed25519_algid (uint8_t **pos, uint8_t *limit)
- static int der_put_uint_integer (uint8_t **pos, uint8_t *limit, const uint8_t *val, size_t val_len)
- size_t x509_build_user_cert (uint8_t *out, size_t out_cap, const uint8_t subject_pubkey[32], const uint8_t serial[16], const uint8_t issuer_seed[32])

7.145.1 Function Documentation

7.145.1.1 der_next()

```
static int der_next (
            const uint8_t ** cursor,
            const uint8_t * limit,
            uint8_t * tag_out,
            der_span_t * content_out )  [static]
```

References der_span_t::len, limit, and der_span_t::p.

Referenced by x509_extract_ed25519_pubkey(), x509_extract_serial(), and x509_verify_signature().

Here is the caller graph for this function:

7.145.1.2 der_put_ed25519_algid()

```
static int der_put_ed25519_algid (
            uint8_t ** pos,
            uint8_t * limit )  [static]
```

References der_put_tlv(), and limit.

Referenced by x509_build_user_cert().

Here is the call graph for this function:

Here is the caller graph for this function:

7.145.1.3 der_put_tlv()

```
static int der_put_tlv (
            uint8_t ** pos,
            uint8_t * limit,
            uint8_t tag,
            const uint8_t * content,
            size_t content_len )  [static]
```

References limit, and memcpy().

Referenced by der_put_ed25519_algid(), der_put_uint_integer(), and x509_build_user_cert().

Here is the call graph for this function:

Here is the caller graph for this function:

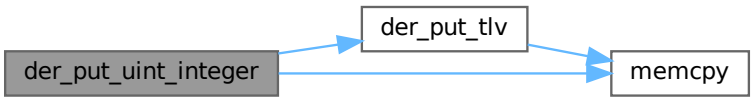
7.145.1.4 der_put_uint_integer()

```
static int der_put_uint_integer (
            uint8_t ** pos,
            uint8_t * limit,
            const uint8_t * val,
            size_t val_len )  [static]
```

References der_put_tlv(), limit, and memcpy().

Referenced by x509_build_user_cert().

Here is the call graph for this function:



Here is the caller graph for this function:

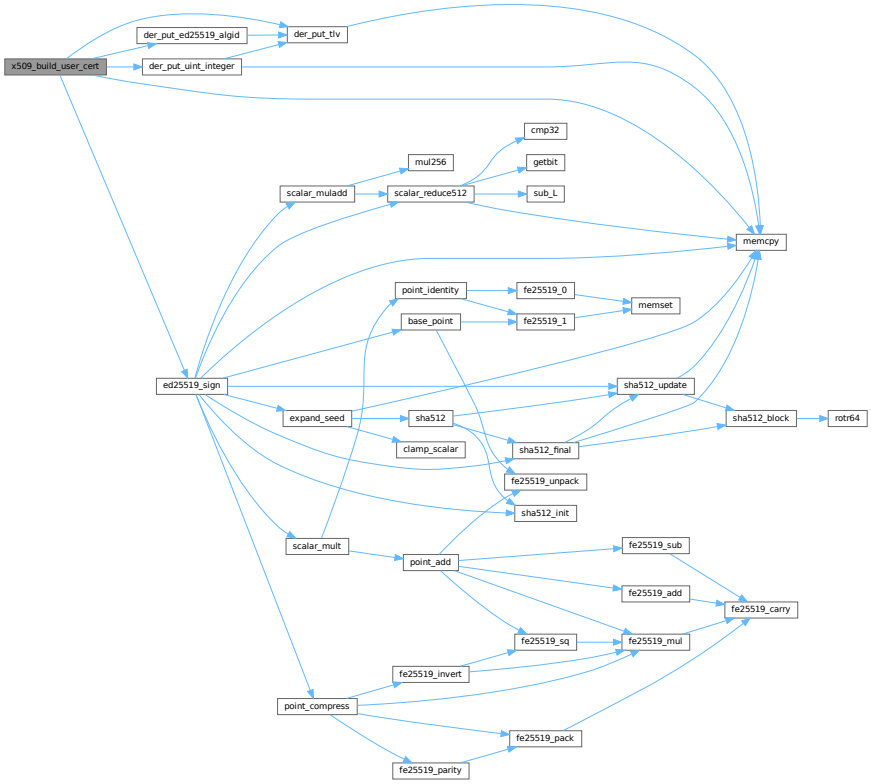
7.145.1.5 x509_build_user_cert()

```
size_t x509_build_user_cert (
            uint8_t * out,
            size_t out_cap,
            const uint8_t subject_pubkey[32],
            const uint8_t serial[16],
            const uint8_t issuer_seed[32] )
```

References der_put_ed25519_algid(), der_put_tlv(), der_put_uint_integer(), ed25519_sign(), and memcpy().

Referenced by capsule_mint_identity().

Here is the call graph for this function:



Here is the caller graph for this function:

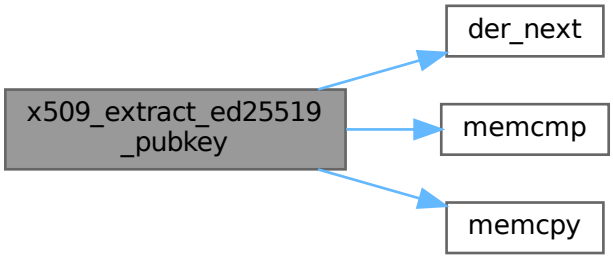
7.145.1.6 x509_extract_ed25519_pubkey()

```
int x509_extract_ed25519_pubkey (
            const uint8_t * der,
            size_t der_len,
            uint8_t pubkey_out[32] )
```

References der_next(), der_span_t::len, memcmp(), memcpy(), and der_span_t::p.

Referenced by find_and_cache_pubkey(), and vm_identity_from_cert().

Here is the call graph for this function:



Here is the caller graph for this function:

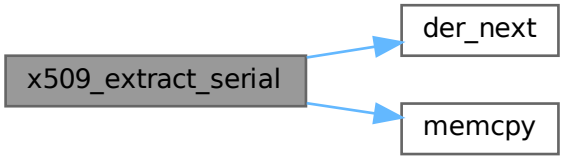
7.145.1.7 x509_extract_serial()

```
int x509_extract_serial (
            const uint8_t * der,
            size_t der_len,
            uint8_t * serial_out,
            size_t serial_out_cap,
            size_t * serial_len_out )
```

References der_next(), der_span_t::len, memcpy(), and der_span_t::p.

Referenced by vm_identity_from_cert().

Here is the call graph for this function:



Here is the caller graph for this function:

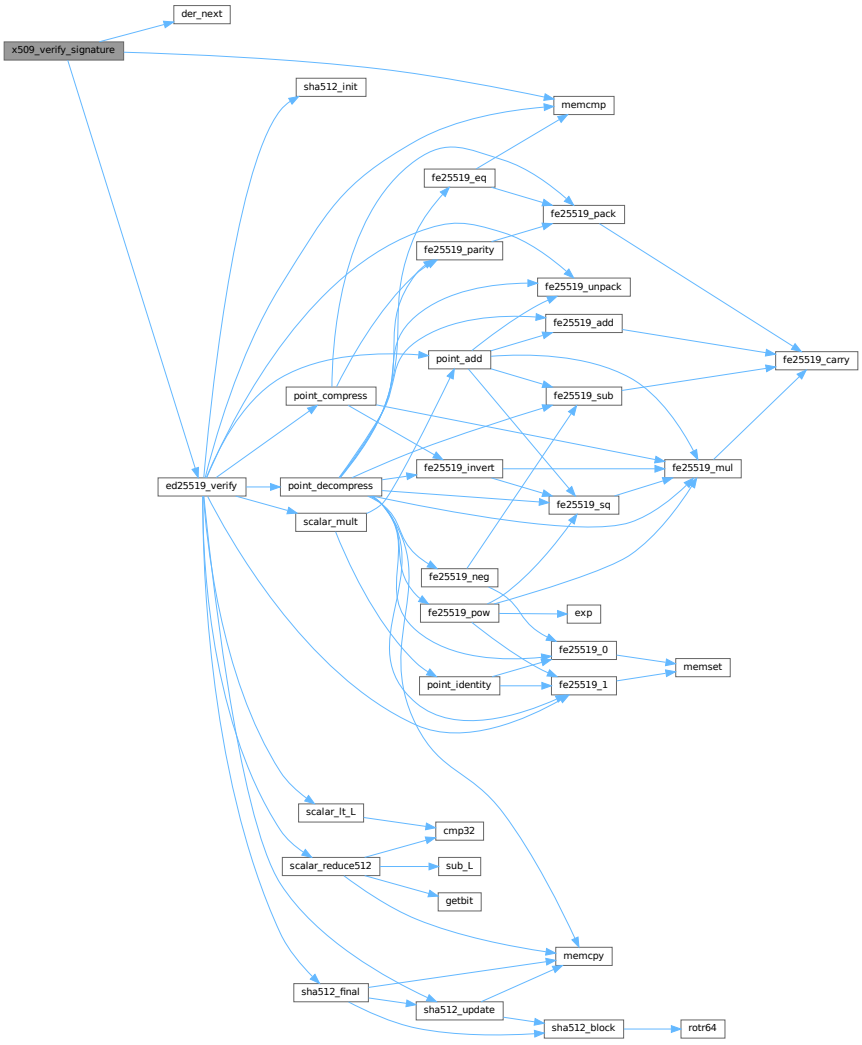
7.145.1.8 x509_verify_signature()

```
int x509_verify_signature (
            const uint8_t * der,
            size_t der_len,
            const uint8_t issuer_pubkey[32] )
```

References der_next(), ed25519_verify(), der_span_t::len, memcmp(), and der_span_t::p.

Referenced by vm_identity_from_cert().

Here is the call graph for this function:

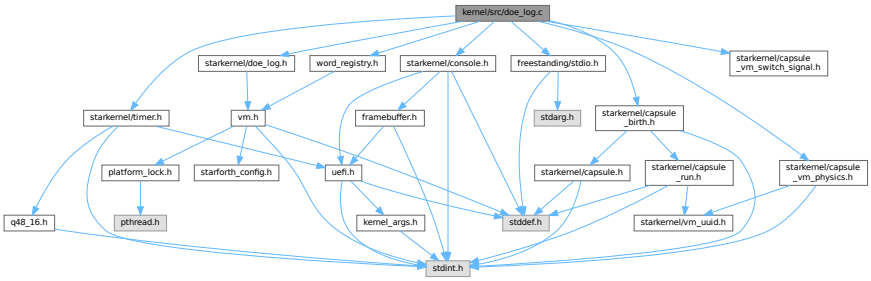


Here is the caller graph for this function:

7.146 kernel/src/doe_log.c File Reference

```
#include "starkernel/doe_log.h"
#include "starkernel/console.h"
#include "starkernel/timer.h"
#include "starkernel/capsule_birth.h"
#include "starkernel/capsule_vm_physics.h"
#include "starkernel/capsule_vm_switch_signal.h"
#include "freestanding/stdio.h"
#include "word_registry.h"
```

Include dependency graph for doe_log.c:



Macros

- #define DOE_PREFIX "\x1b[36m[HADES][DOE]\x1b[0m "
- #define DOE_BUF_SIZE 512

Functions

- static uint64_t doe_log_heat_by_name (const char *name)
- void doe_log_tick_row (VM *vm, const HeartbeatTickSnapshot *snap)

  Emit one CSV row for the current heartbeat tick, unless g_doe_log_enabled is 0 (a no-op then).
- static void doe_word_hb_on (VM *vm)
- static void doe_word_hb_off (VM *vm)
- void register_doe_log_words (VM *vm)

  Registers HB-ON (–) and HB-OFF (–), which set g_doe_log_enabled to 1 and 0 respectively.

Variables

- int g_doe_log_enabled = 0

  doe_log.c — DoE CSV logger for the LithosAnanke kernel
- static const char * doe_header

7.146.1 Macro Definition Documentation

7.146.1.1 DOE_BUF_SIZE

```
#define DOE_BUF_SIZE 512
```

7.146.1.2 DOE_PREFIX

```
#define DOE_PREFIX "\x1b[36m[HADES][DOE ]\x1b[0m "
```

7.146.2 Function Documentation

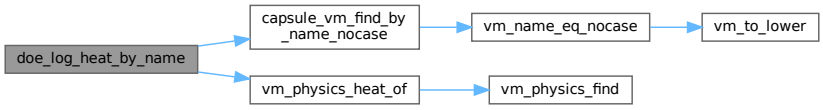
7.146.2.1 doe_log_heat_by_name()

```
static uint64_t doe_log_heat_by_name (
            const char * name )  [static]
```

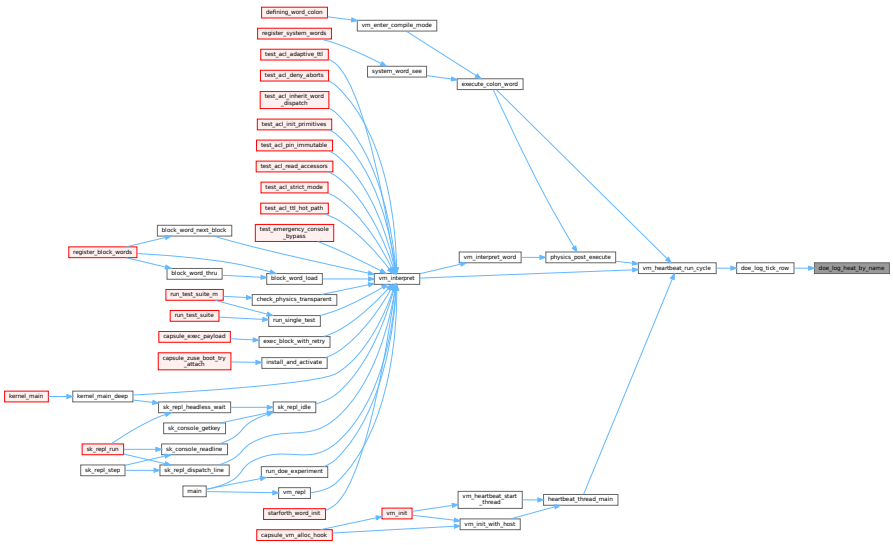
References capsule_vm_find_by_name_nocase(), entry, name, and vm_physics_heat_of().

Referenced by doe_log_tick_row().

Here is the call graph for this function:



Here is the caller graph for this function:



7.146.2.2 doe_log_tick_row()

```
void doe_log_tick_row (
            VM * vm,
            const HeartbeatTickSnapshot * snap )
```

Emit one CSV row for the current heartbeat tick, unless g_doe_log_enabled is 0 (a no-op then).

Pulls APIC TimeTrustState via heartbeat_state(). Prints the column header once, before the first row ever emitted – not re-printed on every HB-ON, so a session toggled off and back on stays one continuous CSV rather than getting a second header embedded partway through.

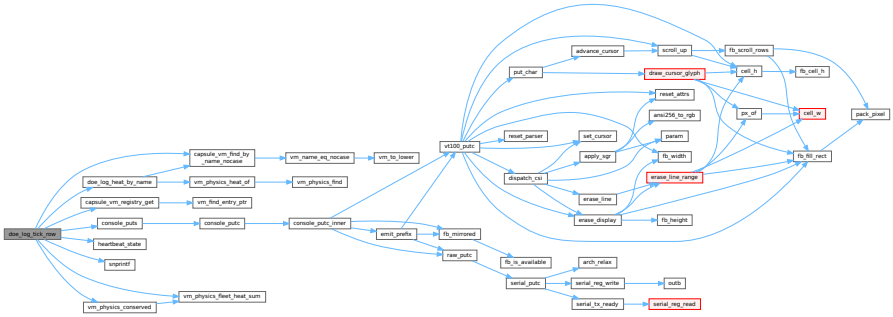
Parameters

vm	VM instance (used for snapshot data)
snap	Populated HeartbeatTickSnapshot from heartbeat_capture_tick_snapshot()

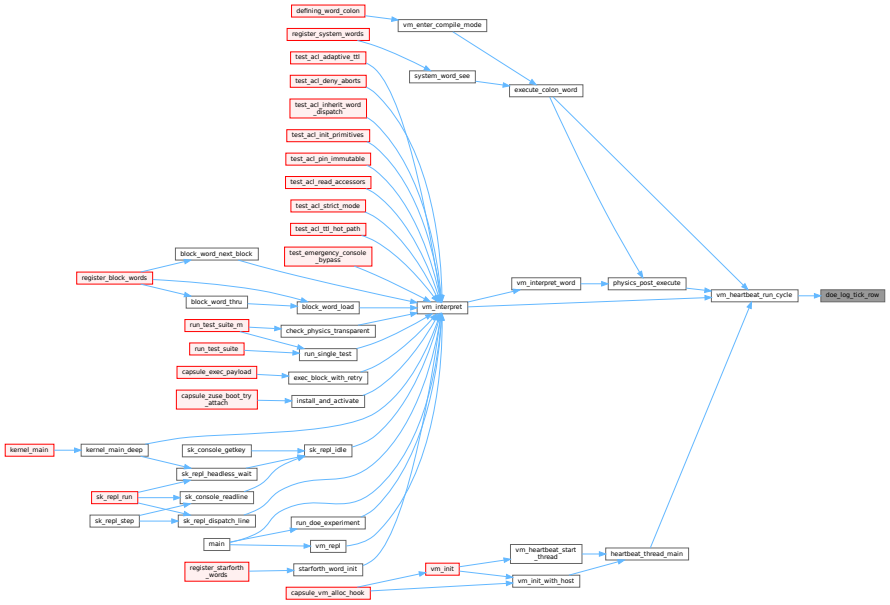
References HeartbeatTickSnapshot::actual_window_size, HeartbeatTickSnapshot::avg_word_heat, HeartbeatTickSnapshot::bucket_, HeartbeatTickSnapshot::cache_hits_delta, capsule_vm_find_by_name_nocase(), capsule_vm_registry_get(), console_puts(), DOE_BUF_SIZE, doe_header, doe_log_heat_by_name(), DOE_PREFIX, HeartbeatTickSnapshot::elapsed_ns, HeartbeatTickSnapshot::estimated_jitter_ns, g_doe_log_enabled, heartbeat_state(), HeartbeatTickSnapshot::hot_word_count, VMRegistryEntry::name, HeartbeatTickSnapshot::predicted_label_hits, snprintf(), HeartbeatTickSnapshot::tick_interval_ns, HeartbeatTickSnapshot::tick_number, time_trust_state::ticks, time_trust_state::trust, time_trust_state::variance, VMRegistryEntry::vm_id, vm_physics_conserved(), vm_physics_fleet_heat_sum(), HeartbeatTickSnapshot::window_width, and HeartbeatTickSnapshot::word_executions_delta.

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.146.2.3 doe_word_hb_off()

```
static void doe_word_hb_off (
            VM * vm )  [static]
```

References g_doe_log_enabled.

Referenced by register_doe_log_words().

Here is the caller graph for this function:

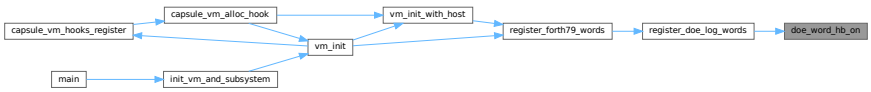
7.146.2.4 doe_word_hb_on()

```
static void doe_word_hb_on (
            VM * vm )  [static]
```

References g_doe_log_enabled.

Referenced by register_doe_log_words().

Here is the caller graph for this function:



7.146.2.5 register_doe_log_words()

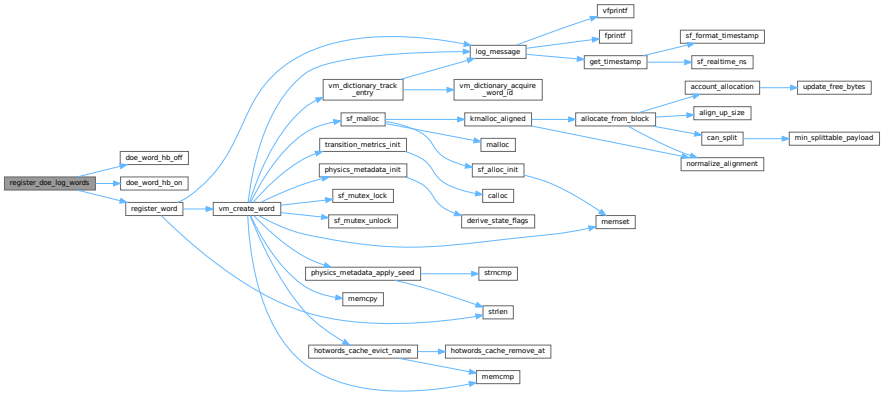
```
void register_doe_log_words (
            VM * vm )
```

Registers HB-ON (–) and HB-OFF (–), which set g_doe_log_enabled to 1 and 0 respectively.

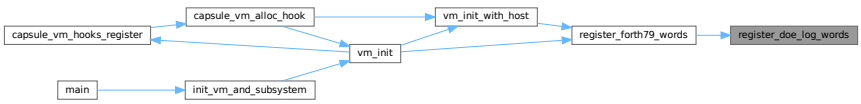
References doe_word_hb_off(), doe_word_hb_on(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.146.3 Variable Documentation

7.146.3.1 doe_header

```
const char* doe_header  [static]
```

Initial value:

```
=
     "\x1b[36m[HADES][DOE ]\x1b[0m "
    "vm_name,vm_id_hex,"
    "tick_number,elapsed_ns,tick_interval_ns,"
    "cache_hits_delta,bucket_hits_delta,word_executions_delta,"
    "hot_word_count,avg_word_heat_q48,"
    "window_width,actual_window_size,predicted_label_hits,jitter_bits,"
    "apic_ticks,time_trust_q48,variance_q48,vm_call_depth_max,"
    "hera_heat_q48,hermes_heat_q48,artemis_heat_q48,"
    "fleet_k_q48,fleet_conserved,"
    "switch_count_cumulative,switch_current_slot,"
    "switch_hera_readiness,switch_hermes_readiness,switch_artemis_readiness,"
    "switch_ticks_since"
```

Referenced by doe_log_tick_row().

7.146.3.2 g_doe_log_enabled

```
int g_doe_log_enabled = 0
```

doe_log.c — DoE CSV logger for the LithosAnanke kernel

starkernel/doe_log.h — DoE (Design of Experiments) CSV logger

Emits CSV rows to the serial log once per heartbeat tick. Each row is prefixed with a cyan [HADES][DOE] tag so it can be extracted cleanly:

grep -aP '[HADES][DOE]' qemu-amd64-∗.log

Columns (in order): 0a. vm_name string — the emitting VM's own registry name (reverse lookup on vm->stadium↩ _vm_id, "?" if unregistered at this tick). Added 2026-09-16 for the multiuser/ multitasking DoE (FABRIC-3.md ?↩ XXXIII): every prior column already described *a* VM's state each row, but nothing said *which* VM emitted it, making concurrent-VM rows indistinguishable by source. 0b. vm_id_hex string — the emitting VM's own stadium_vm_↩ id as "hi-lo" hex, same format PARITY:BIRTH/PARITY:KILL already print. Names get reused across a trial's own KILL+rebirth cycles; this is the one column unique to a single birth instance.

1. tick_number uint32 — monotonic heartbeat counter
2. elapsed_ns uint64 — ns since run start
3. tick_interval_ns uint64 — actual interval from prior tick
4. cache_hits_delta uint32 — hot-words cache hits this tick (TODO: 0)
5. bucket_hits_delta uint32 — bucket hits this tick (TODO: 0)
6. word_executions_delta uint32 — words executed this tick (TODO: 0)
7. hot_word_count uint64 — words with heat >= threshold
8. avg_word_heat_q48 uint64 — mean execution heat as raw Q48.16 integer
9. window_width uint32 — rolling window effective size
10. actual_window_size uint32 — true analysis width: min(total_executions, effective_window_size)
11. predicted_label_hits uint32 — ANOVA early-exit confirmations per tick (L8 validation signal)
12. jitter_bits uint64 — estimated_jitter_ns IEEE 754 raw bits (union-punned)
13. apic_ticks uint64 — APIC timer monotonic tick count
14. time_trust_q48 uint64 — TIME-TRUST in Q48.16 format
15. variance_q48 uint64 — timing variance in Q48.16 format
16. hera_heat_q48 uint64 — VM fleet physics: Hera's execution_heat_q48
17. hermes_heat_q48 uint64 — VM fleet physics: Hermes's execution_heat_q48
18. artemis_heat_q48 uint64 — VM fleet physics: Artemis's execution_heat_q48
19. fleet_k_q48 uint64 — "K": vm_physics_fleet_heat_sum() over ALL live VMs (not just the 3 named Tripod members above – this is the genuine conservation invariant, which should read Q48_ONE (65536) whenever conservation holds). Requested 2026-09-12 alongside the 3x9 factorial analysis in experiments/std79-doe/ – columns 16-18 alone cannot reconstruct this since they omit every identity VM's own heat.
20. fleet_conserved uint32 — vm_physics_conserved() as 0/1: whether |fleet_k_q48 - Q48_ONE| < VM_↩ PHYSICS_EPSILON_Q48 at the moment this row was captured.
21. switch_count_cumulative uint64 — Stage 3 preemptive-switch count, cumulative since boot (capsule_vm_switch_signal.c).

22. switch_current_slot uint32 — which registered switch-signal participant is executing at this tick (0=Hera,1=Hermes, 2=Artemis by current registration order; 0xFFFFFFFF = none/unregistered at this moment).
23. switch_hera_readiness uint32 — Hera's raw readiness accumulator (slot 0).
24. switch_hermes_readiness uint32 — Hermes's raw readiness accumulator (slot 1).
25. switch_artemis_readiness uint32 — Artemis's raw readiness accumulator (slot 2).
26. switch_ticks_since uint32 — ticks elapsed since the last real switch (capsule_vm_switch_signal.c); empirically verifies the fixed 50-tick threshold under real DoE load rather than assuming it.

Columns 21-26 (FABRIC-3.md ?XXVIII Stage 3 follow-on, 2026-09-13): added alongside Stage 3's timer-driven preemptive context switching – read-only exposure of state the switch-signal module already tracked internally for its own decision-making; no new measurement logic, just made observable. Slot readiness columns are hardcoded to the 3 known Tripod slots (same convention as columns 16-18) rather than a dynamic per-slot column set, for the same reason: this is a logging schema for the currently-known fleet, not the switch-signal mechanism itself (which stays slot-count-agnostic, headroom for up to 8 participants – see capsule_vm_switch_signal.h).

Note: avg_word_heat and estimated_jitter_ns are stored as double in the snapshot but the freestanding snprintf has no %f support. avg_word_heat is emitted as raw Q48.16 (multiply by 65536); jitter is union-punned to its IEEE 754 uint64 bit pattern to avoid UB from out-of-range cast.

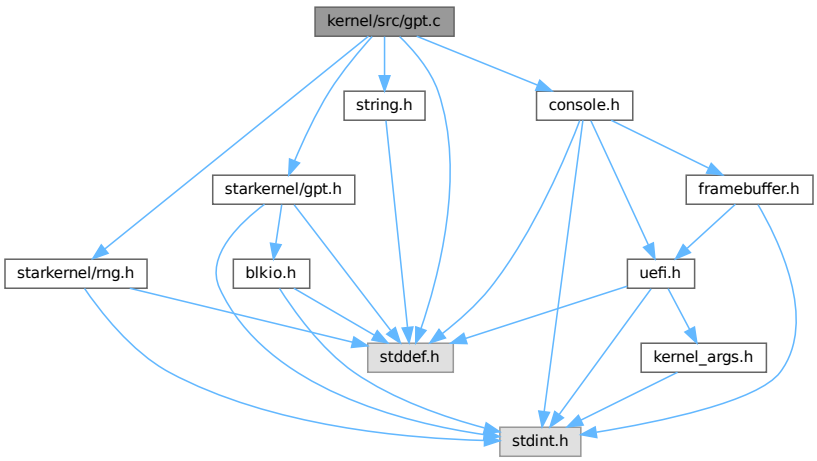
Columns 16-18 (VM-FLEET-ATTRACTOR-DESIGN-20260705.md, Phase 3): looked up by name each tick rather than by a fixed vm_id, since kill/rebirth (e.g. TRIPOD-TEST's K-soak) assigns a fresh vm_id on every rebirth. Fixed to the three known Tripod VMs, not a dynamic per-VM column set – this is a logging schema for a specific known fleet, not the physics mechanism itself (which stays genuinely VM-count-agnostic). Reads 0 for any VM not currently found in the registry (e.g. Hermes momentarily during its kill/rebirth window), which is a legitimate observation, not an error.

Referenced by doe_log_tick_row(), doe_word_hb_off(), and doe_word_hb_on().

7.147 kernel/src/gpt.c File Reference

```
#include "starkernel/gpt.h"
#include "starkernel/rng.h"
#include "console.h"
#include <stddef.h>
#include <string.h>
```

Include dependency graph for gpt.c:



Macros

- #define GPT_SECTOR 512u
- #define GPT_ENTRIES_MAX (32u ∗ 1024u) /∗ the entry array read, at most (128 entries of 128 bytes is 16 KiB) ∗/
- #define GPT_NENT 128u
- #define GPT_ESIZE 128u
- #define GPT_ENT_SECTORS ((GPT_NENT ∗ GPT_ESIZE) / GPT_SECTOR) /∗ 32 ∗/

Functions

- uint32_t gpt_crc32 (const void ∗buf, size_t len)
- static uint32_t le32 (const uint8_t ∗p)
- static uint64_t le64 (const uint8_t ∗p)
- int gpt_read (blkio_dev_t ∗disk, gpt_part_t ∗parts, unsigned max, unsigned ∗count)
- void gpt_disk_guid (uint8_t guid[16])
- int gpt_find (const gpt_part_t ∗parts, unsigned count, const uint8_t type[16])
- static void put_le32 (uint8_t ∗p, uint32_t v)
- static void put_le64 (uint8_t ∗p, uint64_t v)
- static int put_sectors (blkio_dev_t ∗disk, uint64_t lba, const uint8_t ∗src, uint32_t n)
- uint64_t gpt_alternate_lba (void)
- int gpt_grow_last (blkio_dev_t ∗disk, const uint8_t type[16], uint64_t max_part_sectors, uint64_t ∗new_last↩ _lba)
- static int gpt_new_guid (uint8_t g[16])
- int gpt_write_one (blkio_dev_t ∗disk, const uint8_t type[16], uint64_t part_sectors, uint8_t disk_guid[16])
- static int part_open (blkio_dev_t ∗dev, const blkio_params_t ∗p)
- static int part_close (blkio_dev_t ∗dev)
- static int part_read (blkio_dev_t ∗dev, uint32_t fblock, void ∗dst)
- static int part_write (blkio_dev_t ∗dev, uint32_t fblock, const void ∗src)
- static int part_flush (blkio_dev_t ∗dev)
- static int part_info (blkio_dev_t ∗dev, blkio_info_t ∗out)
- int gpt_part_open (blkio_dev_t ∗dev, gpt_part_dev_t ∗st, blkio_dev_t ∗disk, const gpt_part_t ∗p)

Variables

- const uint8_t gpt_type_starship_identity [16]
- const uint8_t gpt_type_starship_artemis [16]
- static uint8_t g_gpt_hdr_block [1024u]
- static uint8_t g_gpt_entries [(32u ∗1024u)]
- static uint8_t g_gpt_disk_guid [16]
- static uint64_t g_gpt_alternate_lba
- static uint8_t g_gpt_rmw [1024u]
- static uint8_t g_gpt_new_entries [128u ∗128u]
- static const blkio_vtable_t g_part_vt = { part_open, part_close, part_read, part_write, part_flush, part_info }

7.147.1 Macro Definition Documentation

7.147.1.1 GPT_ENT_SECTORS

```
#define GPT_ENT_SECTORS ((GPT_NENT * GPT_ESIZE) / GPT_SECTOR) /* 32 */
```

7.147.1.2 GPT_ENTRIES_MAX

```
#define GPT_ENTRIES_MAX (32u * 1024u) /* the entry array read, at most (128 entries of 128
bytes is 16 KiB) */
```

7.147.1.3 GPT_ESIZE

```
#define GPT_ESIZE 128u
```

7.147.1.4 GPT_NENT

```
#define GPT_NENT 128u
```

7.147.1.5 GPT_SECTOR

```
#define GPT_SECTOR 512u
```

7.147.2 Function Documentation

7.147.2.1 gpt_alternate_lba()

```
uint64_t gpt_alternate_lba (
            void  )
```

References g_gpt_alternate_lba.

Referenced by sk_v4_grow_stick().

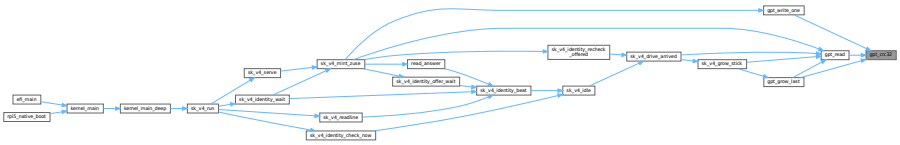
Here is the caller graph for this function:

7.147.2.2 gpt_crc32()

```
uint32_t gpt_crc32 (
            const void * buf,
            size_t len )
```

Referenced by gpt_grow_last(), gpt_read(), and gpt_write_one().

Here is the caller graph for this function:



7.147.2.3 gpt_disk_guid()

```
void gpt_disk_guid (
            uint8_t guid[16] )
```

References g_gpt_disk_guid, and memcpy().

Referenced by sk_v4_is_this_stick().

Here is the call graph for this function:

Here is the caller graph for this function:

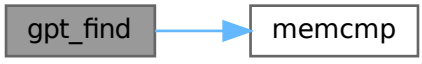
7.147.2.4 gpt_find()

```
int gpt_find (
            const gpt_part_t * parts,
            unsigned count,
            const uint8_t type[16] )
```

References count, and memcmp().

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), sk_v4_is_this_stick(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

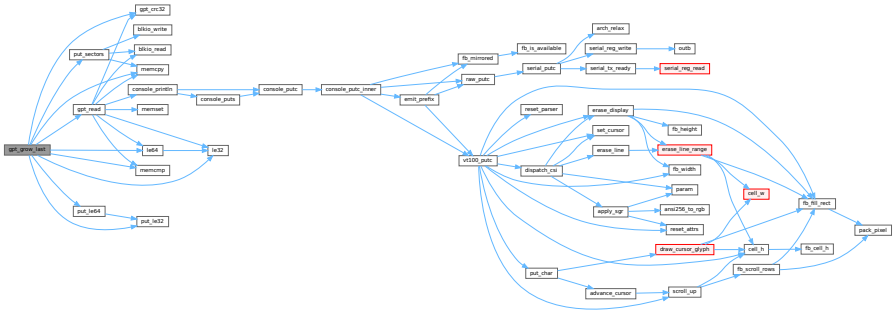
7.147.2.5 gpt_grow_last()

```
int gpt_grow_last (
            blkio_dev_t * disk,
            const uint8_t type[16],
            uint64_t max_part_sectors,
            uint64_t * new_last_lba )
```

References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, count, blkio_vtable_t::flush, g_gpt_entries, g_gpt_hdr_block, GPT_BAD, gpt_crc32(), GPT_EIO, GPT_NONE, GPT_OK, gpt_read(), GPT_SECTOR, le32(), le64(), memcmp(), memcpy(), put_le32(), put_le64(), put_sectors(), blkio_dev::total_blocks, blkio_dev::vt, and zero.

Referenced by sk_v4_grow_stick().

Here is the call graph for this function:



Here is the caller graph for this function:

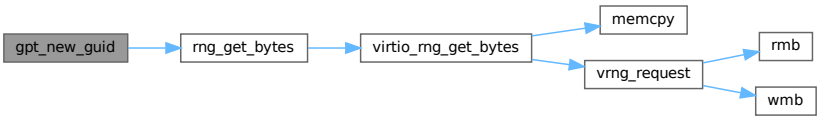
7.147.2.6 gpt_new_guid()

```
static int gpt_new_guid (
            uint8_t g[16] )  [static]
```

References g, and rng_get_bytes().

Referenced by gpt_write_one().

Here is the call graph for this function:



Here is the caller graph for this function:

7.147.2.7 gpt_part_open()

```
int gpt_part_open (
            blkio_dev_t * dev,
            gpt_part_dev_t * st,
            blkio_dev_t * disk,
            const gpt_part_t * p )
```

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, blkio_open(), gpt_part_dev_t::disk, gpt_part_dev_t::fblocks, gpt_part_dev_t::first_fblock, gpt_part_t::first_lba, blkio_params_t::forth_block_size, g_part_vt, gpt_part_t::last_lba, blkio_params_t::opaque, blkio_params_t::total_blocks, and blkio_dev::total_blocks.

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

Here is the call graph for this function:

Here is the caller graph for this function:

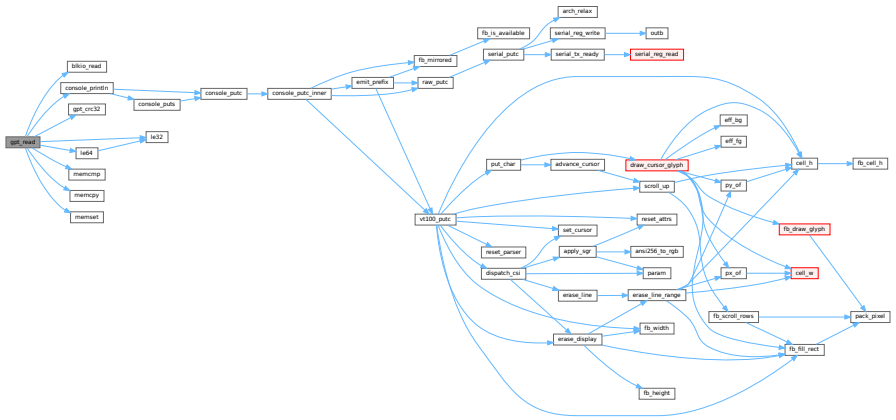
7.147.2.8 gpt_read()

```
int gpt_read (
            blkio_dev_t * disk,
            gpt_part_t * parts,
            unsigned max,
            unsigned * count )
```

References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_read(), console_println(), count, gpt_part_t::first_lba, blkio_dev::forth_block_size, g_gpt_alternate_lba, g_gpt_disk_guid, g_gpt_entries, g_gpt_hdr_block, GPT_BAD, gpt_crc32(), GPT_EIO, GPT_ENTRIES_MAX, GPT_NONE, GPT_OK, GPT_SECTOR, blkio_vtable_t::info, gpt_part_t::last_lba, le32(), le64(), memcmp(), memcpy(), memset(), blkio_info_t::phys_sector_size, blkio_dev::total_blocks, blkio_dev::vt, and zero.

Referenced by gpt_grow_last(), sk_v4_drive_arrived(), sk_v4_grow_stick(), and sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

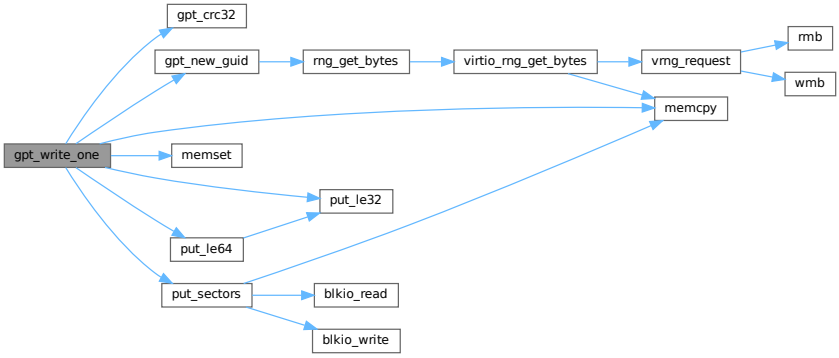
7.147.2.9 gpt_write_one()

```
int gpt_write_one (
            blkio_dev_t * disk,
            const uint8_t type[16],
            uint64_t part_sectors,
            uint8_t disk_guid[16] )
```

References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_vtable_t::flush, blkio_dev::forth_block_size, g_gpt_new_entries, GPT_BAD, gpt_crc32(), GPT_EIO, GPT_ENT_SECTORS, GPT_ESIZE, GPT_NENT, gpt_new_guid(), GPT_OK, GPT_SECTOR, blkio_vtable_t::info, memcpy(), memset(), name, blkio_info_t::phys_sector_size, put_le32(), put_le64(), put_sectors(), blkio_dev::total_blocks, blkio_dev::vt, and zero.

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

7.147.2.10 le32()

```
static uint32_t le32 (
            const uint8_t * p )  [static]
```

Referenced by gpt_grow_last(), gpt_read(), and le64().

Here is the caller graph for this function:

7.147.2.11 le64()

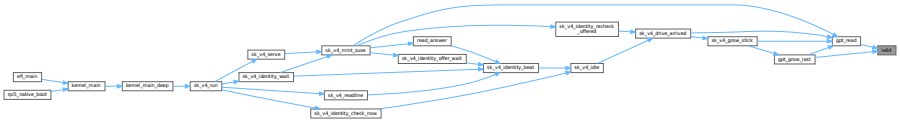
```
static uint64_t le64 (
            const uint8_t * p )  [static]
```

References le32().

Referenced by gpt_grow_last(), and gpt_read().

Here is the call graph for this function:

Here is the caller graph for this function:



7.147.2.12 part_close()

```
static int part_close (
            blkio_dev_t * dev )  [static]
```

References BLKIO_OK.

7.147.2.13 part_flush()

```
static int part_flush (
            blkio_dev_t * dev )  [static]
```

References gpt_part_dev_t::disk, blkio_vtable_t::flush, blkio_dev::state, and blkio_dev::vt.

7.147.2.14 part_info()

```
static int part_info (
            blkio_dev_t * dev,
            blkio_info_t * out )  [static]
```

References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, gpt_part_dev_t::disk, gpt_part_dev_t::fblocks, blkio_vtable_t::info, blkio_info_t::phys_size_bytes, blkio_dev::state, blkio_info_t::total_blocks, and blkio_dev::vt.

7.147.2.15 part_open()

```
static int part_open (
            blkio_dev_t * dev,
            const blkio_params_t * p )  [static]
```

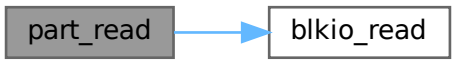
References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, gpt_part_dev_t::fblocks, blkio_dev::forth_block_size, blkio_params_t::opaque, blkio_dev::state, and blkio_dev::total_blocks.

7.147.2.16 part_read()

```
static int part_read (
            blkio_dev_t * dev,
            uint32_t fblock,
            void * dst )  [static]
```

References BLKIO_EINVAL, blkio_read(), gpt_part_dev_t::disk, gpt_part_dev_t::fblocks, gpt_part_dev_t::first_fblock, and blkio_dev::state.

Here is the call graph for this function:

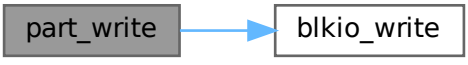


7.147.2.17 part_write()

```
static int part_write (
            blkio_dev_t * dev,
            uint32_t fblock,
            const void * src )  [static]
```

References BLKIO_EINVAL, blkio_write(), gpt_part_dev_t::disk, gpt_part_dev_t::fblocks, gpt_part_dev_t::first_fblock, and blkio_dev::state.

Here is the call graph for this function:



7.147.2.18 put_le32()

```
static void put_le32 (
            uint8_t * p,
            uint32_t v )  [static]
```

Referenced by gpt_grow_last(), gpt_write_one(), and put_le64().

Here is the caller graph for this function:

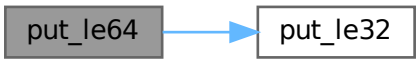
7.147.2.19 put_le64()

```
static void put_le64 (
            uint8_t * p,
            uint64_t v )  [static]
```

References put_le32().

Referenced by gpt_grow_last(), and gpt_write_one().

Here is the call graph for this function:



Here is the caller graph for this function:

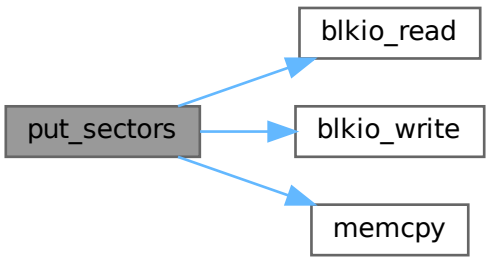
7.147.2.20 put_sectors()

```
static int put_sectors (
            blkio_dev_t * disk,
            uint64_t lba,
            const uint8_t * src,
            uint32_t n )  [static]
```

References BLKIO_OK, blkio_read(), blkio_write(), g_gpt_rmw, GPT_SECTOR, and memcpy().

Referenced by gpt_grow_last(), and gpt_write_one().

Here is the call graph for this function:



Here is the caller graph for this function:

7.147.3 Variable Documentation

7.147.3.1 g_gpt_alternate_lba

```
uint64_t g_gpt_alternate_lba  [static]
```

Referenced by gpt_alternate_lba(), and gpt_read().

7.147.3.2 g_gpt_disk_guid

```
uint8_t g_gpt_disk_guid[16]  [static]
```

Referenced by gpt_disk_guid(), and gpt_read().

7.147.3.3 g_gpt_entries

```
uint8_t g_gpt_entries[(32u *1024u)]  [static]
```

Referenced by gpt_grow_last(), and gpt_read().

7.147.3.4 g_gpt_hdr_block

```
uint8_t g_gpt_hdr_block[1024u]  [static]
```

Referenced by gpt_grow_last(), and gpt_read().

7.147.3.5 g_gpt_new_entries

```
uint8_t g_gpt_new_entries[128u * 128u]  [static]
```

Referenced by gpt_write_one().

7.147.3.6 g_gpt_rmw

```
uint8_t g_gpt_rmw[1024u]  [static]
```

Referenced by put_sectors().

7.147.3.7 g_part_vt

```
const blkio_vtable_t g_part_vt = { part_open, part_close, part_read, part_write, part_flush,
part_info }  [static]
```

Referenced by gpt_part_open().

7.147.3.8 gpt_type_starship_artemis

```
const uint8_t gpt_type_starship_artemis[16]
```

Initial value:

```
= {
    0x86, 0xB5, 0xD7, 0xBC, 0x8E, 0x1C, 0xEF, 0x4E,
    0x86, 0xA3, 0x27, 0x3A, 0x8D, 0x8E, 0xE6, 0x24
}
```

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), and sk_v4_is_this_stick().

7.147.3.9 gpt_type_starship_identity

```
const uint8_t gpt_type_starship_identity[16]
```

Initial value:

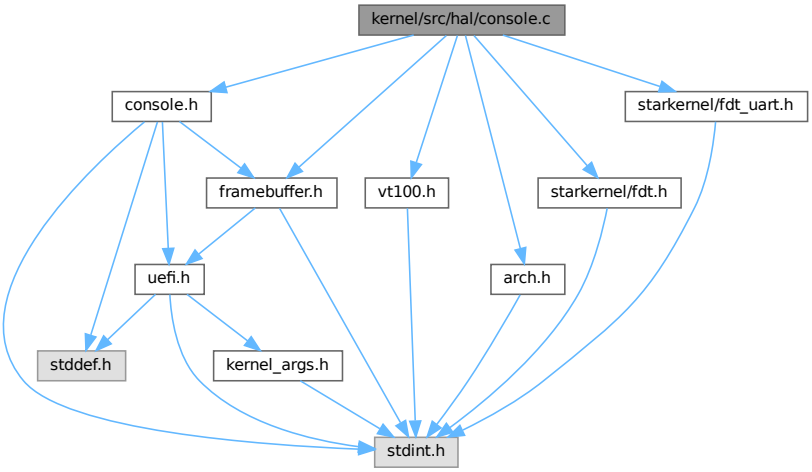
```
= {
    0xE1, 0xBF, 0xBE, 0x23, 0x17, 0x48, 0xF3, 0x4C,
    0xA5, 0x37, 0x90, 0x69, 0xBF, 0x15, 0x97, 0x88
}
```

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), sk_v4_is_this_stick(), sk_v4_mint_zuse(), and sk_v4_real_zuse_arrived().

7.148 kernel/src/hal/console.c File Reference

```
#include "console.h"
#include "framebuffer.h"
#include "vt100.h"
#include "arch.h"
#include "starkernel/fdt.h"
#include "starkernel/fdt_uart.h"
```

Include dependency graph for console.c:



Macros

- #define SERIAL_TX_MAX_SPIN 4000000u
- #define UART16550_LSR 5u
- #define UART16550_LSR_DR (1u << 0)
- #define UART16550_LSR_THRE (1u << 5)
- #define PL011_FR 6u
- #define PL011_FR_RXFE (1u << 4)
- #define PL011_FR_TXFF (1u << 5)
- #define CONSOLE_VM_NAME_BUF 64

Enumerations

- enum { SERIAL_NONE = 0 , SERIAL_PORT16550 , SERIAL_MMIO16550 , SERIAL_PL011 }

  console.c - Serial console implementation (UART 16550) Supports amd64 via I/O ports

Functions

- static uint32_t serial_reg_read (unsigned n)
- static void serial_reg_write (unsigned n, uint32_t v)
- static int serial_tx_ready (void)
- static int serial_rx_ready (void)

- static void serial_putc (char c)
- void console_attach_serial (const void *dtb)

  console.h - Serial console + framebuffer VT100 interface for StarKernel
- void console_init (void)

  Initialize serial console x86: programs the 16550 at COM1, if console_attach_serial() found one.
- void console_describe_serial (void)

  Print one line saying which serial line the console found, or that there is none.
- static int fb_mirrored (void)
- void console_fb_mirror (int on)
- void console_fb_clear (void)
- void console_set_vm_name (const char *name)

  Set the active VM name shown as [Name] prefix on each output line.
- const char * console_get_vm_name (void)
- void console_set_user_prefix_provider (console_user_prefix_fn fn)
- void console_save_vm_name (char *out, size_t cap)

  console_save_vm_name - Copy the current active-VM name into the caller's own buffer, for a later console_set_vm_name() restore.
- static void raw_putc (char c)
- static void emit_prefix (void)
- static void console_putc_inner (char c)

  Write a single character to serial console.
- void console_putc (char c)

  Write a single character to serial console.
- void console_puts (const char *s)

  Write a null-terminated string to serial console.
- uint64_t console_tx_count (void)

  console_tx_count - Monotonic count of console_putc() calls delivered to either output (serial and/or framebuffer).
- void console_ensure_line_start (void)

  console_ensure_line_start - Make sure the next real output starts at the beginning of a fresh output line, closing off whatever is currently mid-line (e.g.
- void console_cancel_deferred_line_start (void)

  console_cancel_deferred_line_start - Discard a pending deferred newline from console_ensure_line_start() without ever emitting it.
- void console_println (const char *s)

  Write a string with newline to serial console.
- int console_poll (void)

  Check if data is available to read.
- void console_fb_init (const FramebufferInfo *info, FbPixelFormat fmt)

  Initialize the framebuffer VT100 terminal and attach it to the console.
- void console_fb_enable_ttf (void)

  FABRIC-0.md item 4.4j: switch the framebuffer console's glyph backend from font_8x16.c to TTF-TEXT's rasterizer.
- void console_fb_scroll_back (uint32_t n)

  FABRIC-0.md item 4.4q: thin wrappers over vt100_scroll_back()/ vt100_scroll_fwd() – see those functions' doc comments for the full contract.
- void console_fb_scroll_fwd (uint32_t n)
- void console_fb_toggle_graphics (void)

  FABRIC-0.md item 4.4y-revised: thin wrapper over vt100_toggle_graphics() – see that function's doc comment for the full contract (the Alt+TAB graphics/text state machine).
- void console_fb_draw_cursor (void)

  Thin wrapper over vt100_draw_cursor() – see that function's doc comment for the full contract (a static block cursor at the terminal's current position).
- void console_fb_erase_cursor (void)

  Thin wrapper over vt100_erase_cursor().
- int console_getc (void)

  Read a single character from serial console (non-blocking) Returns -1 if no character available.

Variables

- static int g_serial_kind
- static uintptr_t g_serial_base
- static unsigned g_serial_shift
- static unsigned g_serial_width
- static int g_serial_from_dt
- static char g_active_vm_name_buf [64]
- static const char * g_active_vm_name = (void *)0
- static int g_line_start = 1
- static int g_fb_mirror = 1
- static uint64_t g_console_tx_count
- static console_user_prefix_fn g_user_prefix_fn = (void *)0
- static int g_pending_line_close = 0

7.148.1 Macro Definition Documentation

7.148.1.1 CONSOLE_VM_NAME_BUF

```
#define CONSOLE_VM_NAME_BUF 64
```

7.148.1.2 PL011_FR

```
#define PL011_FR 6u
```

7.148.1.3 PL011_FR_RXFE

```
#define PL011_FR_RXFE (1u << 4)
```

7.148.1.4 PL011_FR_TXFF

```
#define PL011_FR_TXFF (1u << 5)
```

7.148.1.5 SERIAL_TX_MAX_SPIN

```
#define SERIAL_TX_MAX_SPIN 4000000u
```

7.148.1.6 UART16550_LSR

```
#define UART16550_LSR 5u
```

7.148.1.7 UART16550_LSR_DR

```
#define UART16550_LSR_DR (1u << 0)
```

7.148.1.8 UART16550_LSR_THRE

```
#define UART16550_LSR_THRE (1u << 5)
```

7.148.2 Enumeration Type Documentation

7.148.2.1 anonymous enum

```
anonymous enum
```

console.c - Serial console implementation (UART 16550) Supports amd64 via I/O ports

Enumerator

SERIAL_NONE	
SERIAL_PORT16550	
SERIAL_MMIO16550	
SERIAL_PL011	

7.148.3 Function Documentation

7.148.3.1 console_attach_serial()

```
void console_attach_serial (
            const void * dtb )
```

console.h - Serial console + framebuffer VT100 interface for StarKernel

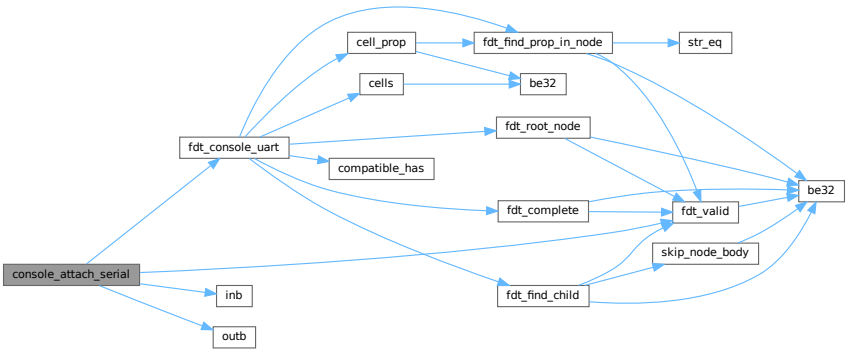
Output policy:

- Serial UART is always active (initialized by console_init).
- When console_fb_init() has been called and the framebuffer is ready, every character is also rendered through the VT100 terminal on screen.
- Both outputs are always live simultaneously; serial cannot be disabled. Find the board's serial console line. Must be called before anything is printed: until it is, and on a board with no such line, the console writes to no serial port and reads from none. On a PC it is COM1, if a UART answers there; elsewhere it is the UART the devicetree names as the console (/chosen/stdout-path); with no devicetree at all it is the UART of QEMU's virt machine. `dtb` may be NULL.

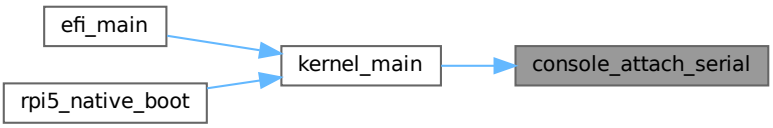
References fdt_uart_t::base, fdt_console_uart(), FDT_UART_PL011, fdt_valid(), g_serial_base, g_serial_from_dt, g_serial_kind, g_serial_shift, g_serial_width, inb(), fdt_uart_t::io_width, fdt_uart_t::kind, outb(), fdt_uart_t::reg_shift, SERIAL_MMIO16550, SERIAL_NONE, SERIAL_PL011, and SERIAL_PORT16550.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.2 console_cancel_deferred_line_start()

```
void console_cancel_deferred_line_start (
            void  )
```

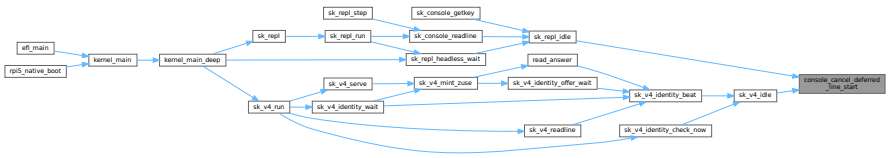
console_cancel_deferred_line_start - Discard a pending deferred newline from console_ensure_line_start() without ever emitting it.

No-op if no newline is currently deferred. Callers that speculatively deferred a line break before checking whether they actually have anything to print (sk_repl_idle()'s idle-beat check being the motivating case, FABRIC-2.md ?I.9) call this when the check comes back negative, so the console is left exactly as it was – no stray newline, no phantom blank line.

References g_pending_line_close.

Referenced by sk_repl_idle(), and sk_v4_idle().

Here is the caller graph for this function:



7.148.3.3 console_describe_serial()

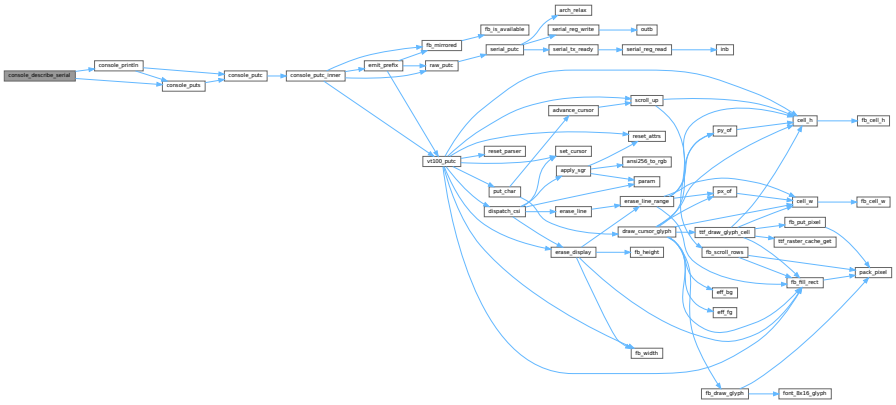
```
void console_describe_serial (
            void  )
```

Print one line saying which serial line the console found, or that there is none.

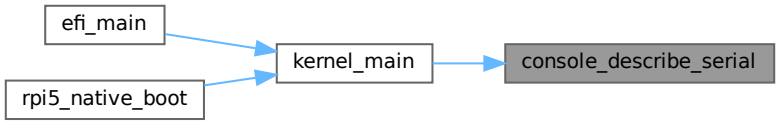
References console_println(), console_puts(), g_serial_base, g_serial_from_dt, g_serial_kind, SERIAL_NONE, SERIAL_PL011, and SERIAL_PORT16550.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.4 console_ensure_line_start()

```
void console_ensure_line_start (
            void  )
```

console_ensure_line_start - Make sure the *next real output* starts at the beginning of a fresh output line, closing off whatever is currently mid-line (e.g.

a dangling prompt). No-op if already at line start.

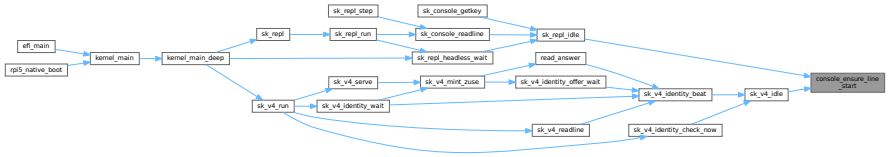
FABRIC-2.md ?I.9 fix, 2026-09-05: the newline is DEFERRED, not emitted immediately – it only actually reaches the console on the next real console_putc() call, and is silently dropped (never emitted at all) if

console_cancel_deferred_line_start() is called first instead. Before this fix, an immediate, unconditional newline here meant any caller invoking this function "just in case" (sk_repl_idle() being the one real caller) would visibly snap a bare, unfinished prompt line to a fresh blank line even when nothing was actually about to be printed – indistinguishable from Enter having already been pressed at that prompt. Deferring means a caller that turns out to have nothing to print can cancel cleanly, with zero visible effect, while a caller that does print gets the correct "close the old line first" behavior for free, still counted by console_tx_count() as real output (unlike the old always-silent inner form) since it is realized through the normal console_putc() path.

References g_line_start, and g_pending_line_close.

Referenced by sk_repl_idle(), and sk_v4_idle().

Here is the caller graph for this function:



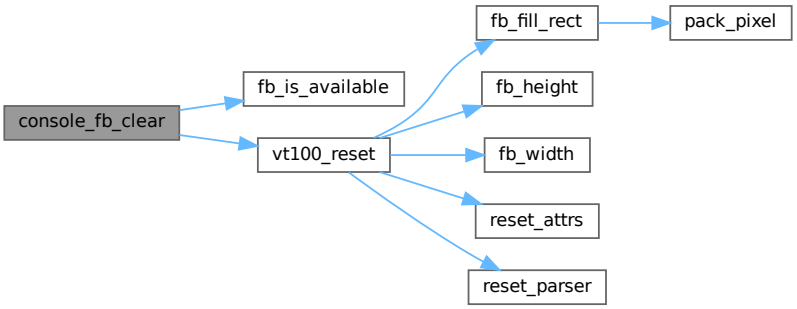
7.148.3.5 console_fb_clear()

```
void console_fb_clear (
            void  )
```

References fb_is_available(), and vt100_reset().

Referenced by sk_v4_drive_left(), sk_v4_fixed_screen(), and sk_v4_session_end().

Here is the call graph for this function:



Here is the caller graph for this function:

7.148.3.6 console_fb_draw_cursor()

```
void console_fb_draw_cursor (
            void  )
```

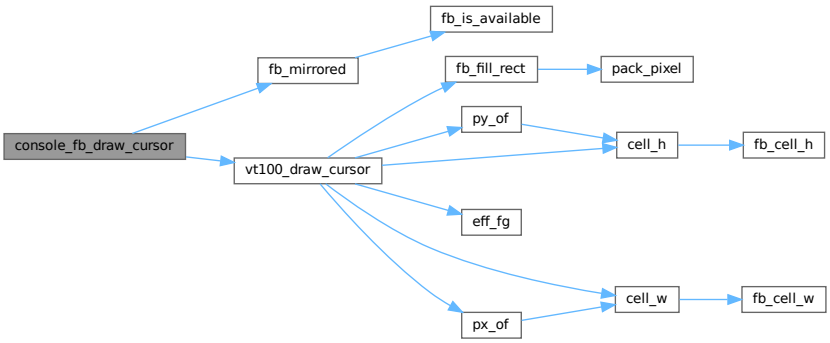
Thin wrapper over vt100_draw_cursor() – see that function's doc comment for the full contract (a static block cursor at the terminal's current position).

No-op if the framebuffer console was never initialized.

References fb_mirrored(), and vt100_draw_cursor().

Referenced by sk_console_readline(), sk_cursor_show(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:

7.148.3.7 console_fb_enable_ttf()

```
void console_fb_enable_ttf (
            void  )
```

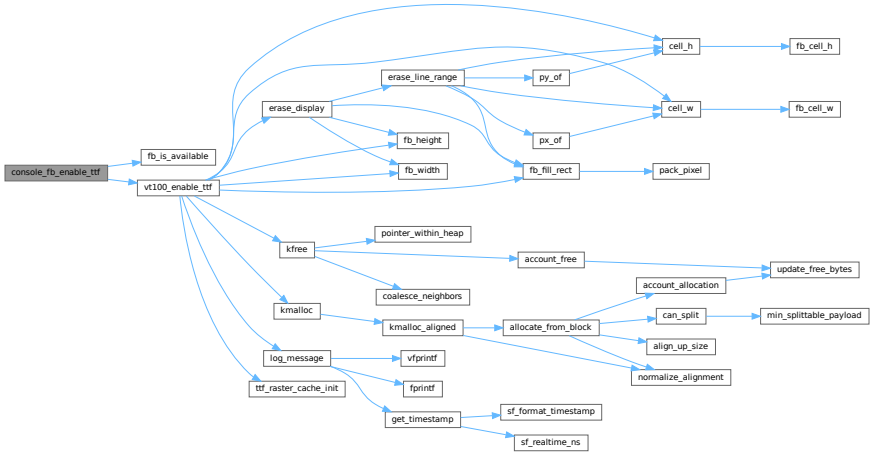
FABRIC-0.md item 4.4j: switch the framebuffer console's glyph backend from font_8x16.c to TTF-TEXT's rasterizer.

Thin wrapper over vt100_enable_ttf() – see that function's doc comment for the full contract (lazy font load, cell-geometry/cols/rows recompute, screen clear, one-shot). No-op if the framebuffer console was never initialized (console_fb_init() not called, or it no-op'd on a NULL framebuffer).

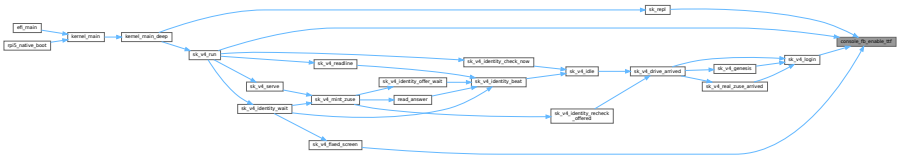
References fb_is_available(), and vt100_enable_ttf().

Referenced by sk_repl(), sk_v4_fixed_screen(), sk_v4_login(), and sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.8 console_fb_erase_cursor()

```
void console_fb_erase_cursor (
            void  )
```

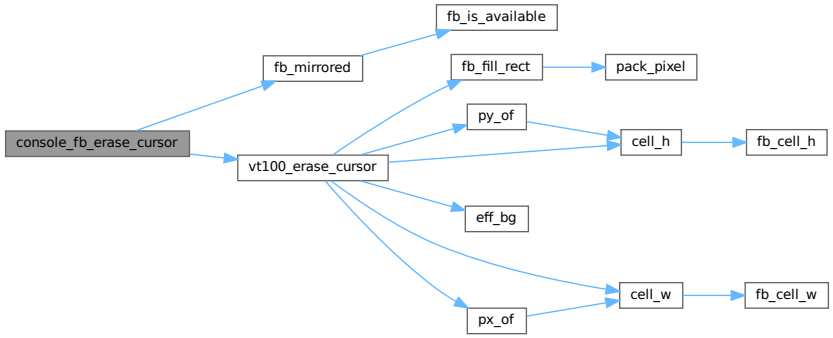
Thin wrapper over vt100_erase_cursor().

No-op if the framebuffer console was never initialized.

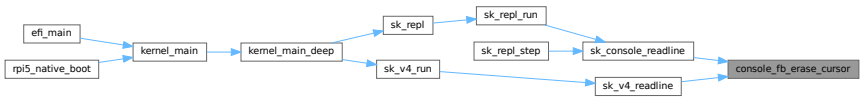
References fb_mirrored(), and vt100_erase_cursor().

Referenced by sk_console_readline(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.9 console_fb_init()

```
void console_fb_init (
            const FramebufferInfo * info,
            FbPixelFormat fmt )
```

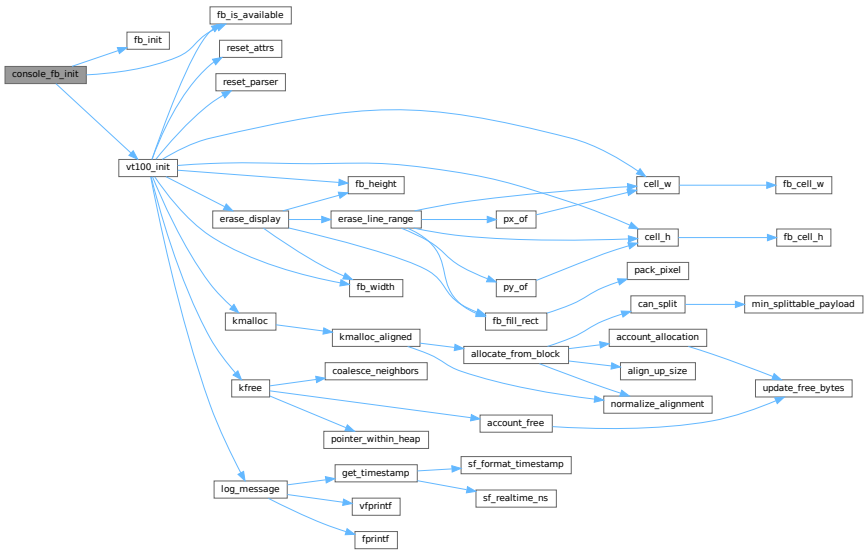
Initialize the framebuffer VT100 terminal and attach it to the console.

Initialize the framebuffer VT100 terminal.

Safe to call with info == NULL or when the framebuffer base is zero (no-op). After this call every console_putc / console_puts output is mirrored to the screen; the serial UART continues to function concurrently.

References FramebufferInfo::base, fb_init(), fb_is_available(), and vt100_init().

Here is the call graph for this function:



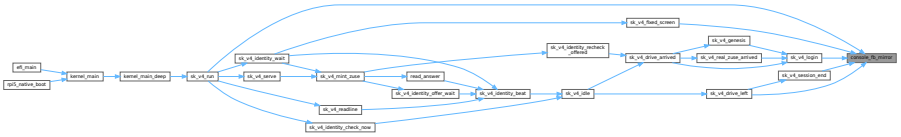
7.148.3.10 console_fb_mirror()

```
void console_fb_mirror (
            int on )
```

References g_fb_mirror.

Referenced by sk_v4_drive_left(), sk_v4_fixed_screen(), sk_v4_login(), sk_v4_run(), and sk_v4_session_end().

Here is the caller graph for this function:



7.148.3.11 console_fb_scroll_back()

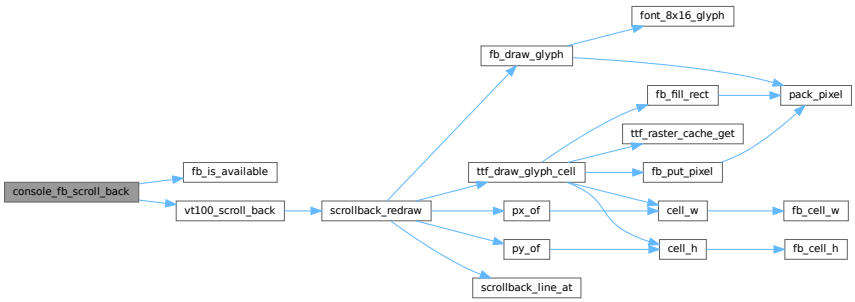
```
void console_fb_scroll_back (
            uint32_t n )
```

FABRIC-0.md item 4.4q: thin wrappers over vt100_scroll_back()/ vt100_scroll_fwd() – see those functions' doc comments for the full contract.

No-op if the framebuffer console was never initialized.

References fb_is_available(), and vt100_scroll_back().

Here is the call graph for this function:

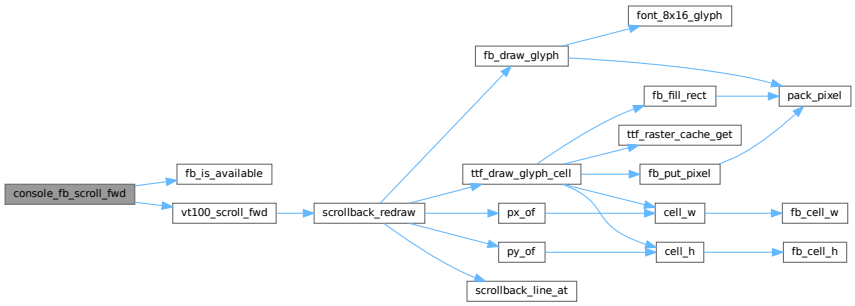


7.148.3.12 console_fb_scroll_fwd()

```
void console_fb_scroll_fwd (
            uint32_t n )
```

References fb_is_available(), and vt100_scroll_fwd().

Here is the call graph for this function:



7.148.3.13 console_fb_toggle_graphics()

```
void console_fb_toggle_graphics (
            void  )
```

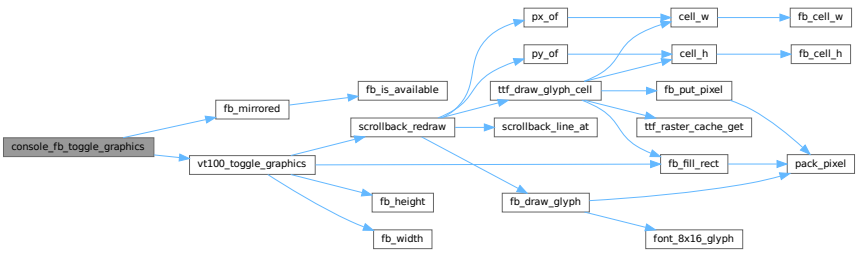
FABRIC-0.md item 4.4y-revised: thin wrapper over vt100_toggle_graphics() – see that function's doc comment for the full contract (the Alt+TAB graphics/text state machine).

No-op if the framebuffer console was never initialized.

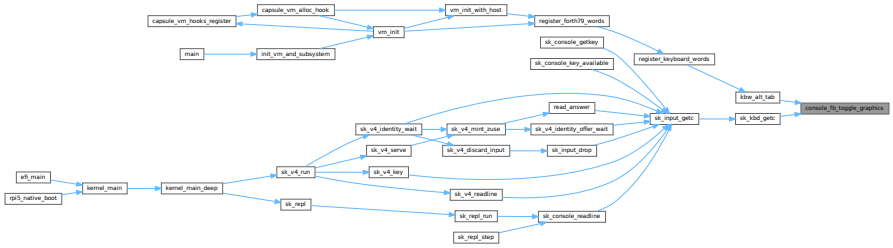
References fb_mirrored(), and vt100_toggle_graphics().

Referenced by kbw_alt_tab(), and sk_kbd_getc().

Here is the call graph for this function:



Here is the caller graph for this function:



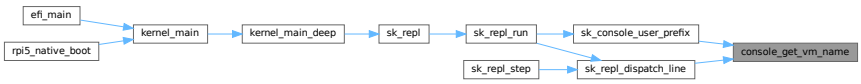
7.148.3.14 console_get_vm_name()

```
const char * console_get_vm_name (
            void  )
```

References g_active_vm_name.

Referenced by sk_console_user_prefix(), and sk_repl_dispatch_line().

Here is the caller graph for this function:



7.148.3.15 console_getc()

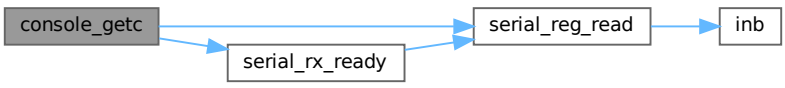
```
int console_getc (
            void  )
```

Read a single character from serial console (non-blocking) Returns -1 if no character available.

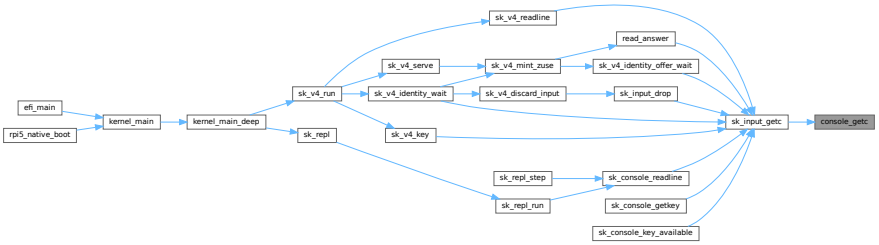
References serial_reg_read(), and serial_rx_ready().

Referenced by sk_input_getc().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.16 console_init()

```
void console_init (
            void  )
```

Initialize serial console x86: programs the 16550 at COM1, if console_attach_serial() found one.

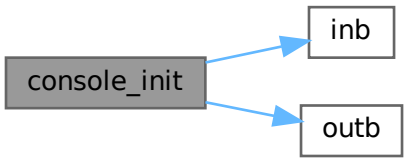
Initialize serial console (UART).

aarch64, riscv64: the line was set up by the firmware; nothing to do.

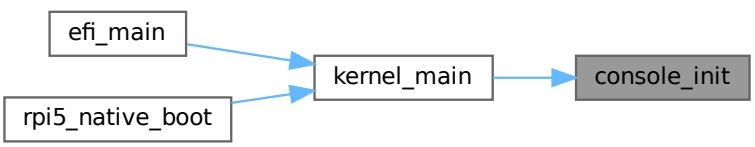
References g_serial_kind, inb(), outb(), SERIAL_NONE, and SERIAL_PORT16550.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.17 console_poll()

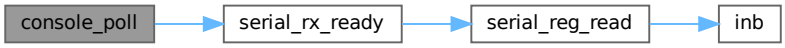
```
int console_poll (
            void  )
```

Check if data is available to read.

Check if character is available for reading.

References serial_rx_ready().

Here is the call graph for this function:



7.148.3.18 console_println()

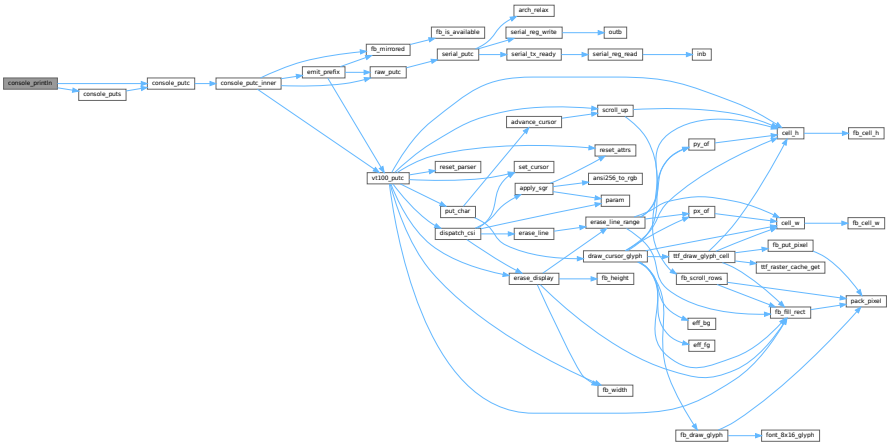
```
void console_println (
            const char * s )
```

Write a string with newline to serial console.

References console_putc(), and console_puts().

Referenced by aarch64_exception_handler(), aarch64_psci_conduit_init(), apic_init(), apic_timer_init(), apic_timer_start(), arch_early_init(), arch_interrupts_init(), blkio_usb_open_msc(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_wirebind_eject(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), capsule_wirebind_unclean_detac capsule_zuse_boot_logout(), capsule_zuse_boot_try_attach(), console_describe_serial(), converge_tsc_bare_metal(), decode_pf_error(), enable_interrupt_route(), failed(), gpt_read(), i8042_init(), init_timer_vm_mode(), ioapic_init(), isr_common_handler(), kernel_main(), kernel_main_deep(), pci_init(), plic_init(), print_banner(), print_boot_info(), print_heap_stats(), print_nl(), print_pmm_stats(), print_uint(), riscv64_exception_handler(), rng_init(), say_size(), sk_repl(), sk_repl_dispatch_line(), sk_repl_run(), sk_repl_step(), sk_v4_drive_arrived(), sk_v4_fixed_screen(), sk_v4_genesis(), sk_v4_grow_stick(), sk_v4_identity_boot(), sk_v4_login(), sk_v4_mint_zuse(), sk_v4_real_zuse_arrived(), sk_v4_run(), sk_v4_serve(), sk_v4_session_end(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), timer_fatal(), timer_init(), vblk_init_device(), vinput_init_device(), virtio_blk_find_artemis(), virtio_input_find_keyboard(), virtio_rng_init(), vm_bootstrap_root_vocabulary(), vm_init_with_host(), vm_physics_status(), vmm_init(), vmm_map_page(), vocab_sync_vm_vars(), vocabulary_create_vocabulary_direct(), vrng_init_device(), and xhci_poll_one().

Here is the call graph for this function:



7.148.3.19 console_putc()

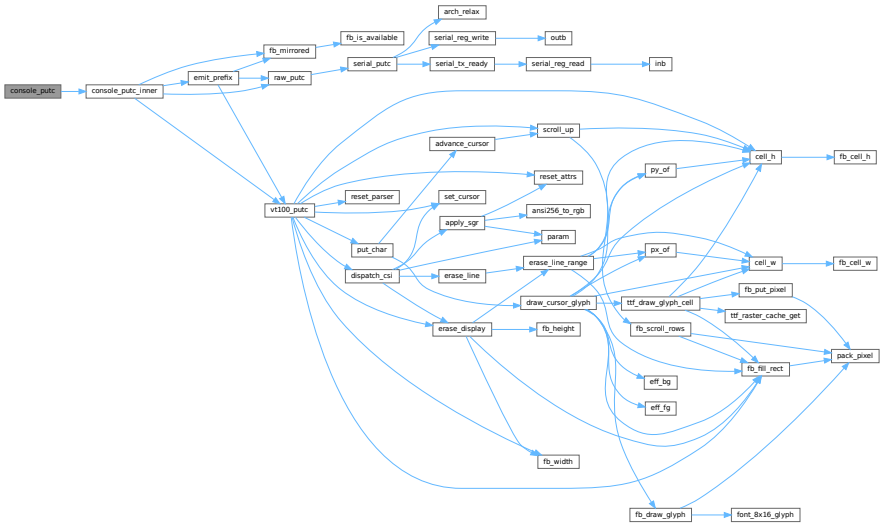
```
void console_putc (
            char c )
```

Write a single character to serial console.

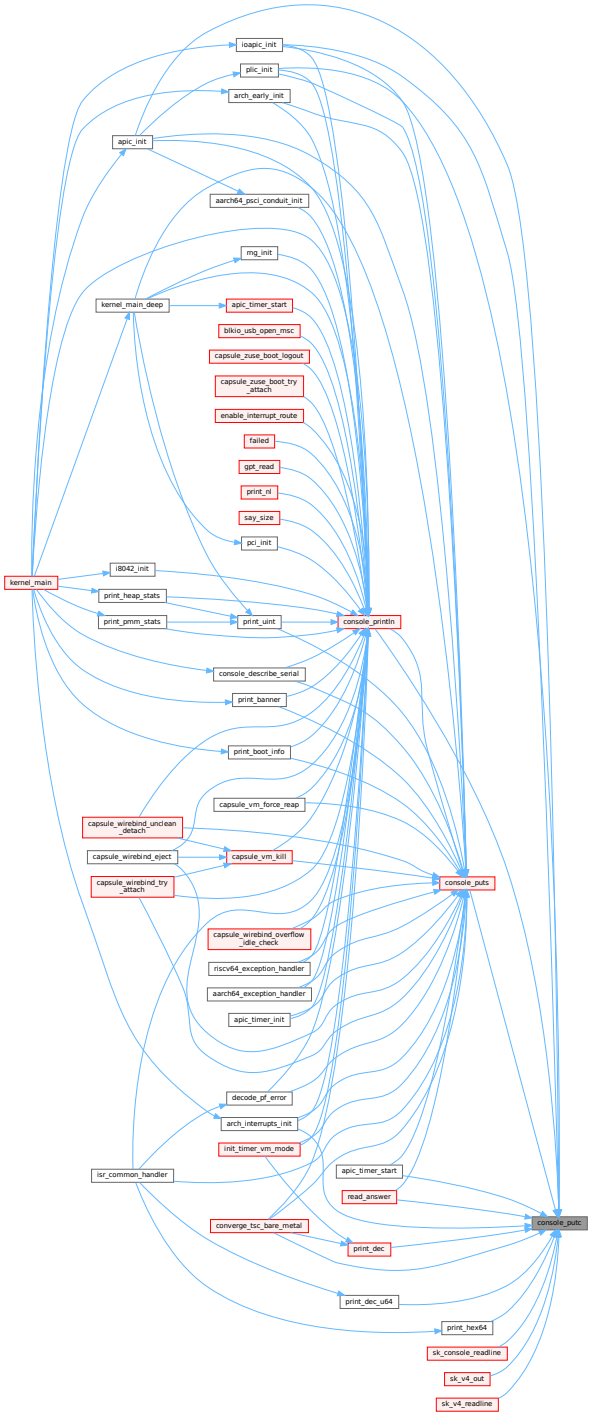
References console_putc_inner(), g_console_tx_count, and g_pending_line_close.

Referenced by apic_init(), apic_timer_start(), arch_interrupts_init(), console_println(), console_puts(), converge_tsc_bare_metal(), ioapic_init(), plic_init(), print_dec(), print_dec_u64(), print_hex64(), read_answer(), sk_console_readline(), sk_v4_out(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.20 console_putc_inner()

```
static void console_putc_inner (
            char c )  [static]
```

Write a single character to serial console.

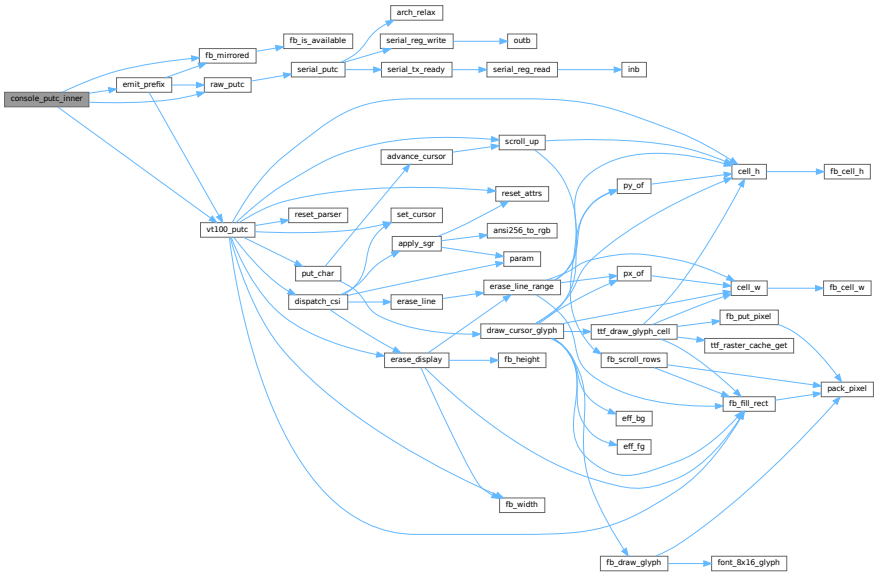
Emits "[VMName] " at the start of each new line when a VM name is set. Also mirrors output to the framebuffer VT100 terminal when available. Serial output is ALWAYS active regardless of framebuffer state.

console_putc_inner(): the same body minus the tx-counter increment – used internally for the newline console_putc() itself realizes out of g_pending_line_close below (see console_ensure_line_start()), so that one real character write emits at most one counted "start" (the '
' plus the character together read as a single unit of real output, not two, to console_tx_count()'s callers).

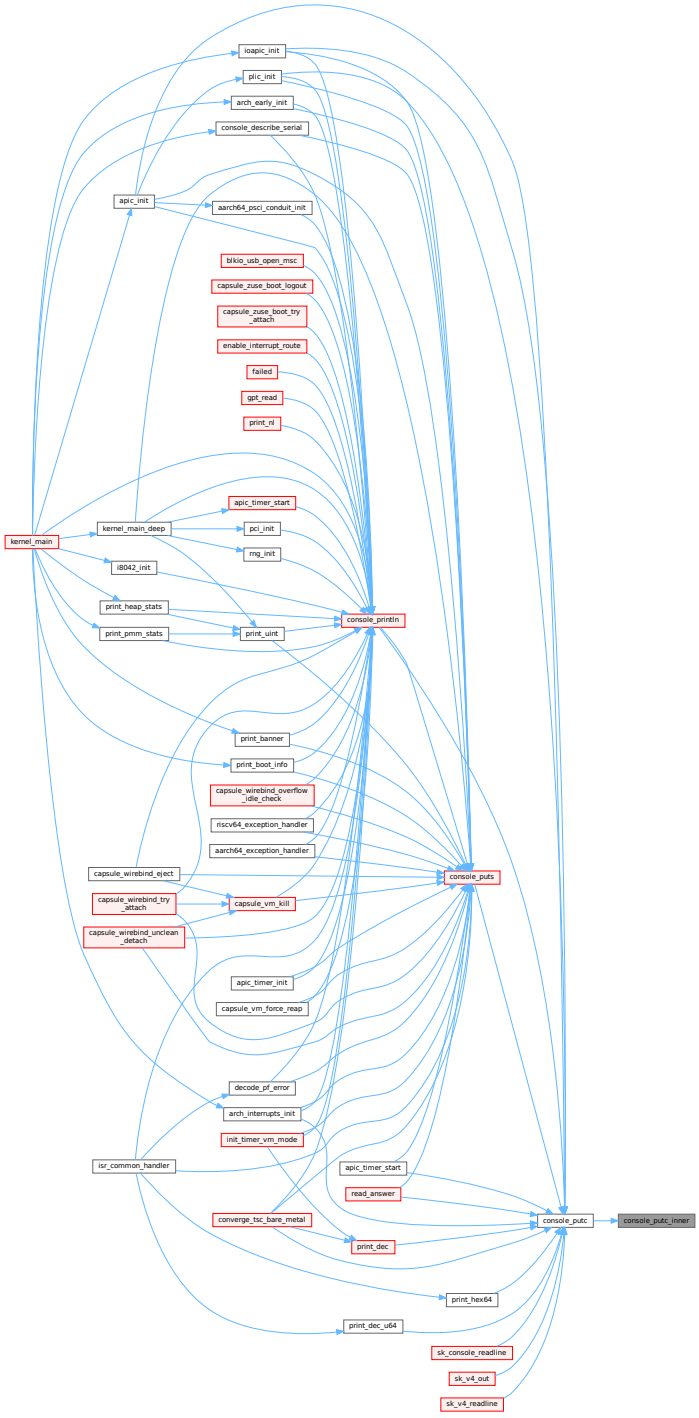
References emit_prefix(), fb_mirrored(), g_active_vm_name, g_line_start, raw_putc(), and vt100_putc().

Referenced by console_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.21 console_puts()

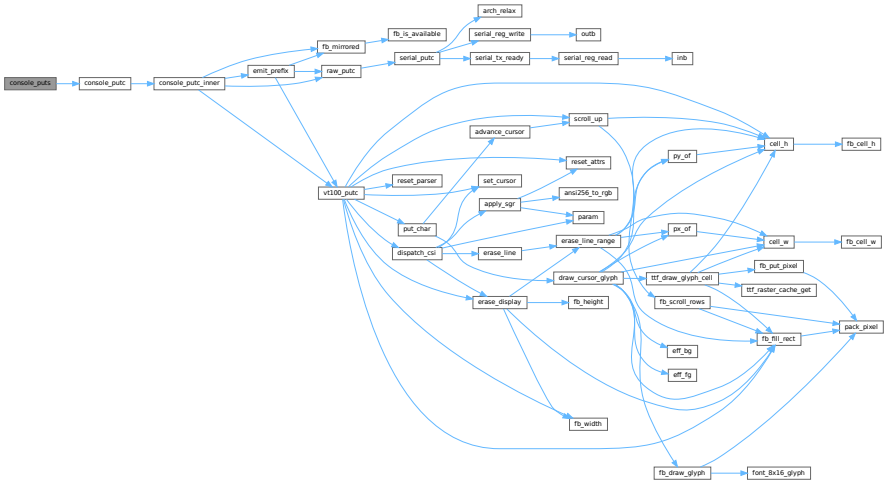
```
void console_puts (
            const char * s )
```

Write a null-terminated string to serial console.

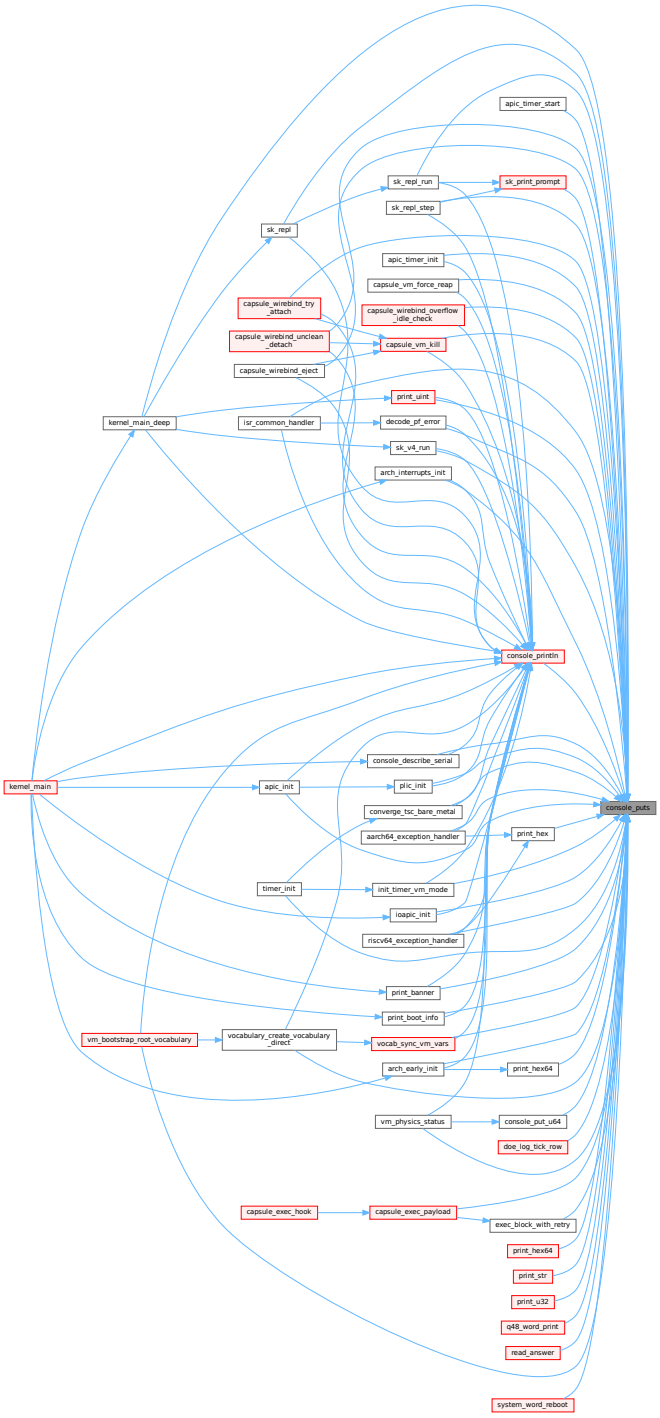
References console_putc().

Referenced by aarch64_exception_handler(), apic_init(), apic_timer_init(), apic_timer_start(), arch_early_init(), arch_interrupts_init(), capsule_exec_payload(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_wirebind_eject(), capsule_wirebind_overflow_idle_check(), capsule_wirebind_try_attach(), capsule_wirebind_unclean_detach(), console_describe_serial(), console_println(), console_put_u64(), converge_tsc_bare_metal(), decode_pf_error(), doe_log_tick_row(), exec_block_with_retry(), init_timer_vm_mode(), ioapic_init(), isr_common_handler(), kernel_main_deep(), plic_init(), print_banner(), print_boot_info(), print_hex(), print_hex64(), print_hex64(), print_str(), print_u32(), print_uint(), q48_word_print(), read_answer(), riscv64_exception_handler(), sk_print_prompt(), sk_repl(), sk_repl_run(), sk_repl_step(), sk_v4_run(), system_word_reboot(), timer_init(), vm_bootstrap_root_vocabulary(), vm_physics_status(), vocab_sync_vm_vars(), and vocabulary_create_vocabulary_direct().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.22 console_save_vm_name()

```
void console_save_vm_name (
            char * out,
            size_t cap )
```

console_save_vm_name - Copy the current active-VM name into the caller's own buffer, for a later console_set_vm_name() restore.

console_get_vm_name() alone is NOT safe for save-then-restore: it returns a pointer into the single internal buffer console_set_vm_name() copies into, so an intervening console_set_vm_name() call (the normal "switch, do work, switch back" pattern every BIRTH/VM-EXEC/CONNECT-∗ call site uses) overwrites the very bytes the saved pointer points at before the restore ever runs – found live 2026-08-28, the restore silently no-ops. Copies at most cap-1 bytes plus a NUL terminator; writes "" if there was no active name (NULL) to save.

Parameters

out	Caller-owned buffer.
cap	Its size in bytes.

References g_active_vm_name.

7.148.3.23 console_set_user_prefix_provider()

```
void console_set_user_prefix_provider (
            console_user_prefix_fn fn )
```

References g_user_prefix_fn.

Referenced by sk_repl_run(), and sk_v4_login().

Here is the caller graph for this function:

7.148.3.24 console_set_vm_name()

```
void console_set_vm_name (
            const char * name )
```

Set the active VM name shown as [Name] prefix on each output line.

Pass NULL to suppress the prefix (kernel-only output before any VM). Copies into internal storage (FABRIC-2.md Phase F, 2026-08-28) – the caller's own pointer does not need to remain valid afterward.

References CONSOLE_VM_NAME_BUF, g_active_vm_name, g_active_vm_name_buf, and name.

Referenced by capsule_vm_registry_init(), and sk_v4_login().

Here is the caller graph for this function:

7.148.3.25 console_tx_count()

```
uint64_t console_tx_count (
            void  )
```

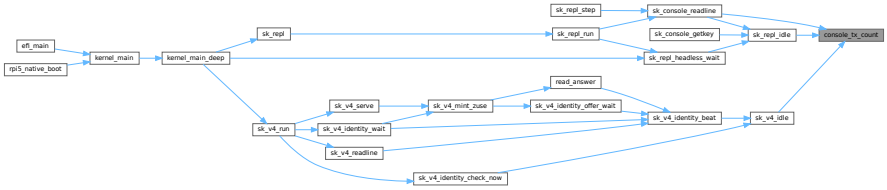
console_tx_count - Monotonic count of console_putc() calls delivered to either output (serial and/or framebuffer).

In use by the REPL to detect that an idle bottom half (heartbeat, USB attach/detach) wrote to the console while the top-level prompt was showing, so it can re-anchor the prompt afterward. Never decreases.

References g_console_tx_count.

Referenced by sk_console_readline(), sk_repl_idle(), and sk_v4_idle().

Here is the caller graph for this function:



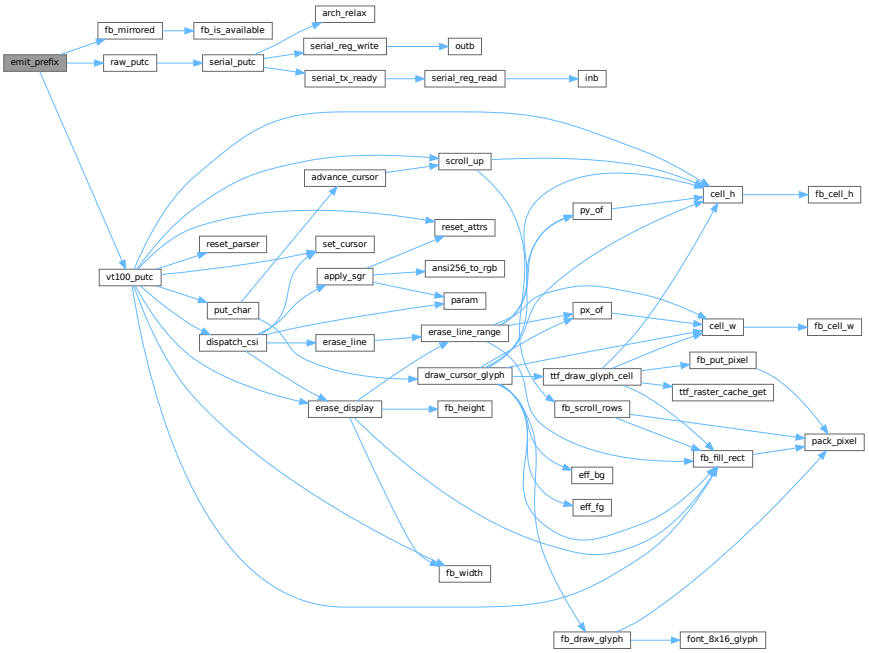
7.148.3.26 emit_prefix()

```
static void emit_prefix (
            void  )  [static]
```

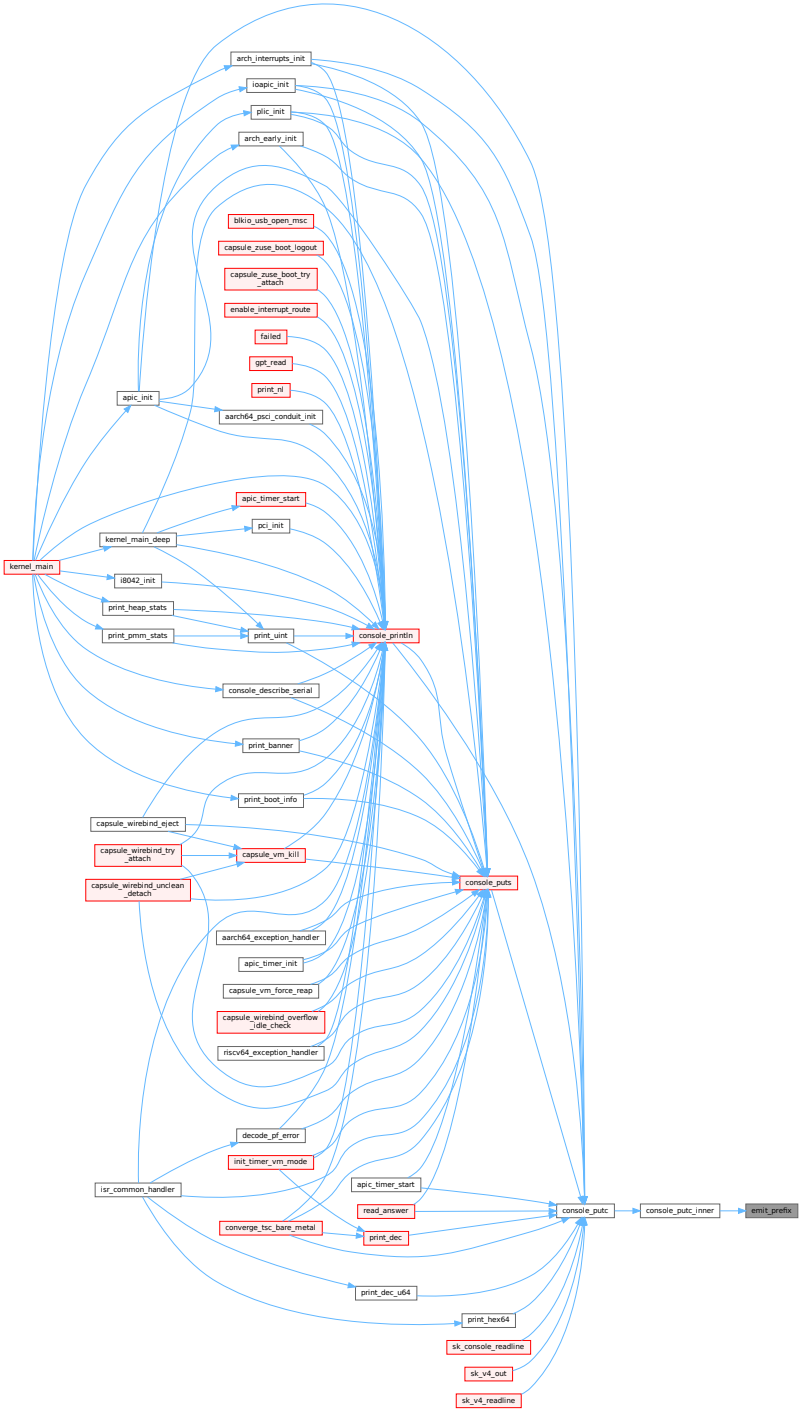
References fb_mirrored(), g_active_vm_name, g_user_prefix_fn, raw_putc(), and vt100_putc().

Referenced by console_putc_inner().

Here is the call graph for this function:



Here is the caller graph for this function:



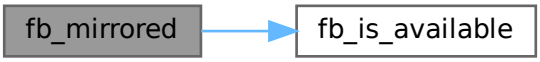
7.148.3.27 fb_mirrored()

```
static int fb_mirrored (
            void  )  [static]
```

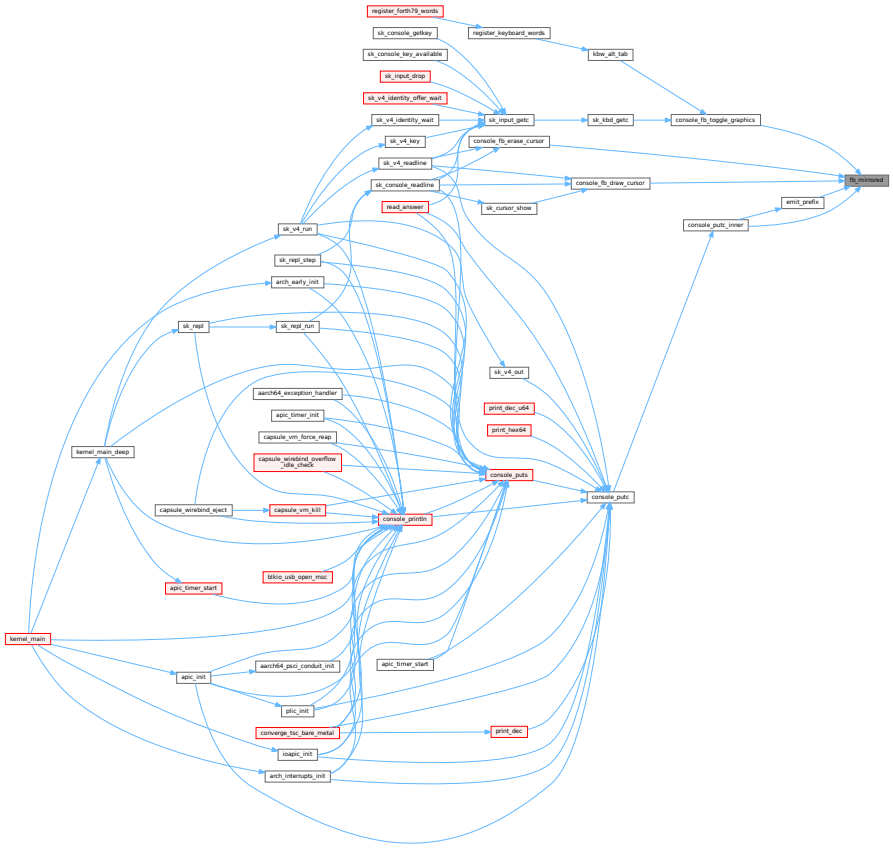
References fb_is_available(), and g_fb_mirror.

Referenced by console_fb_draw_cursor(), console_fb_erase_cursor(), console_fb_toggle_graphics(), console_putc_inner(), and emit_prefix().

Here is the call graph for this function:



Here is the caller graph for this function:



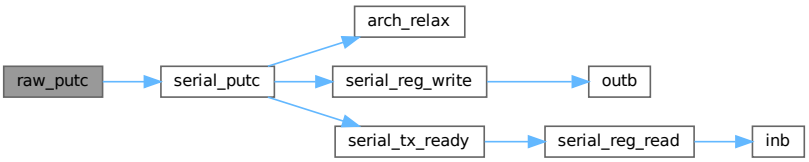
7.148.3.28 raw_putc()

```
static void raw_putc (
            char c )  [static]
```

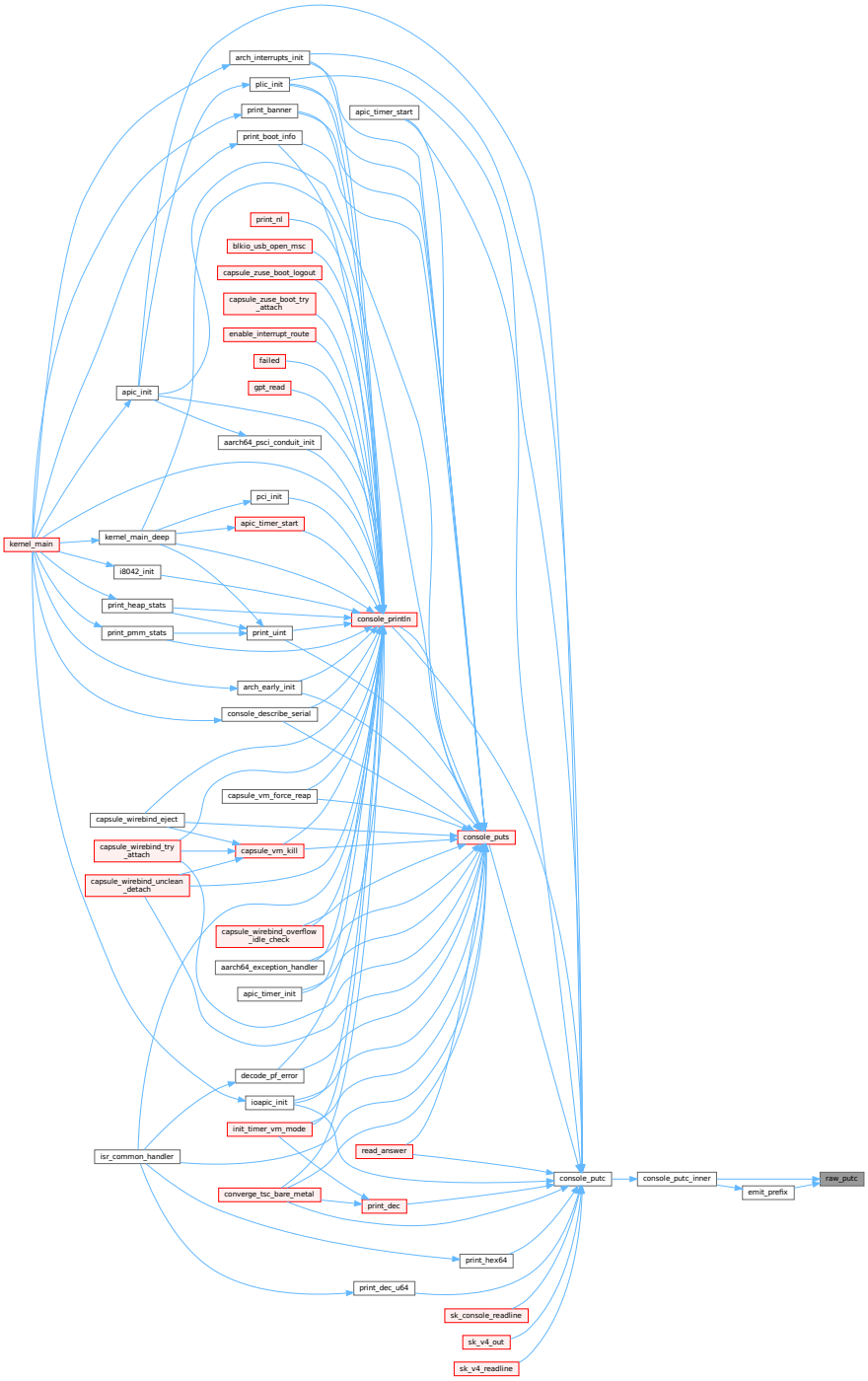
References serial_putc().

Referenced by console_putc_inner(), and emit_prefix().

Here is the call graph for this function:



Here is the caller graph for this function:



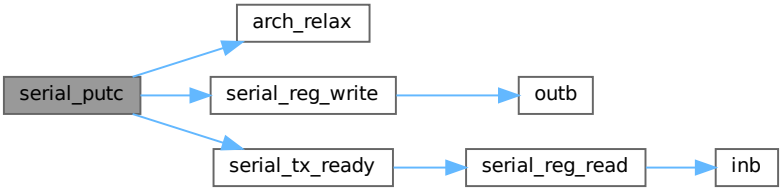
7.148.3.29 serial_putc()

```
static void serial_putc (
            char c )  [static]
```

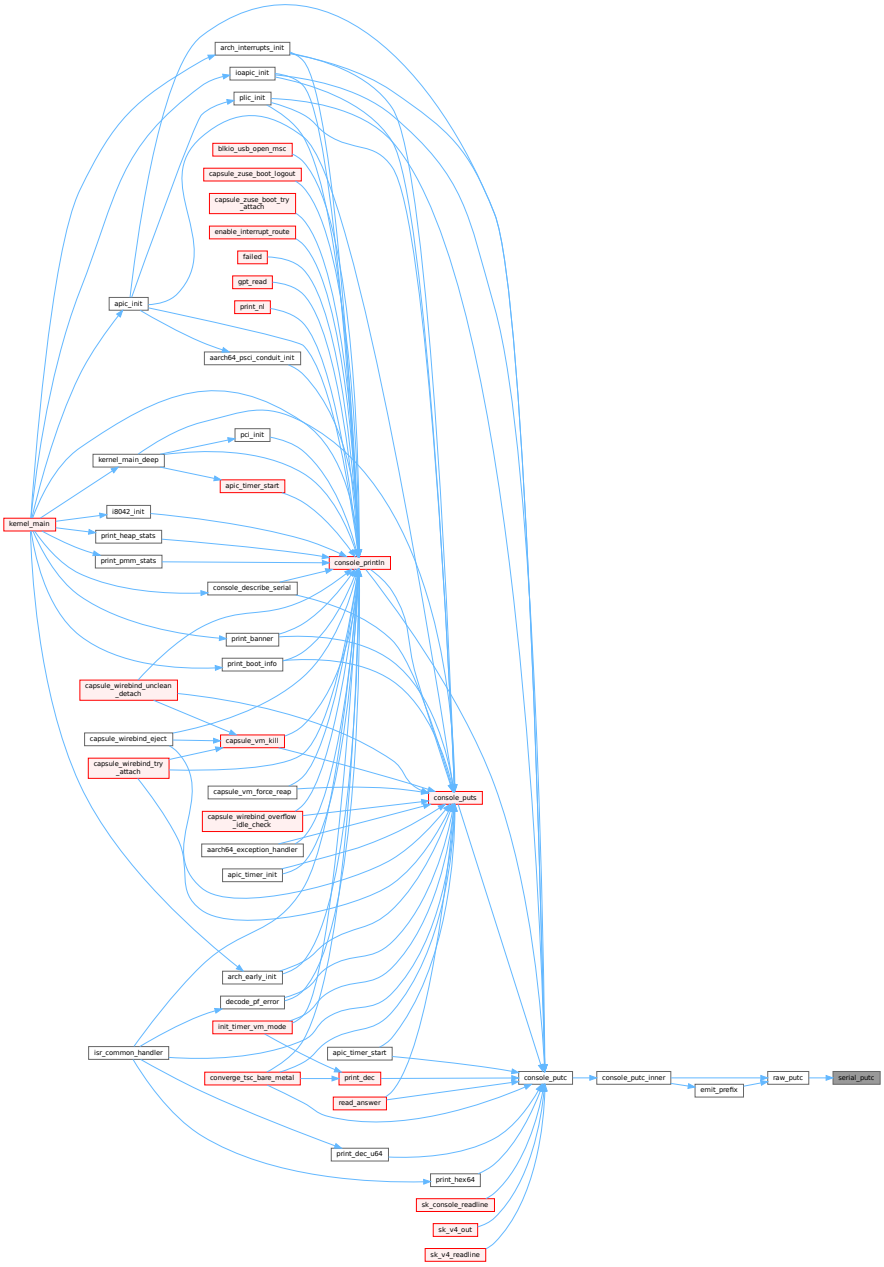
References arch_relax(), g_serial_kind, SERIAL_NONE, serial_reg_write(), SERIAL_TX_MAX_SPIN, and serial_tx_ready().

Referenced by raw_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.3.30 serial_reg_read()

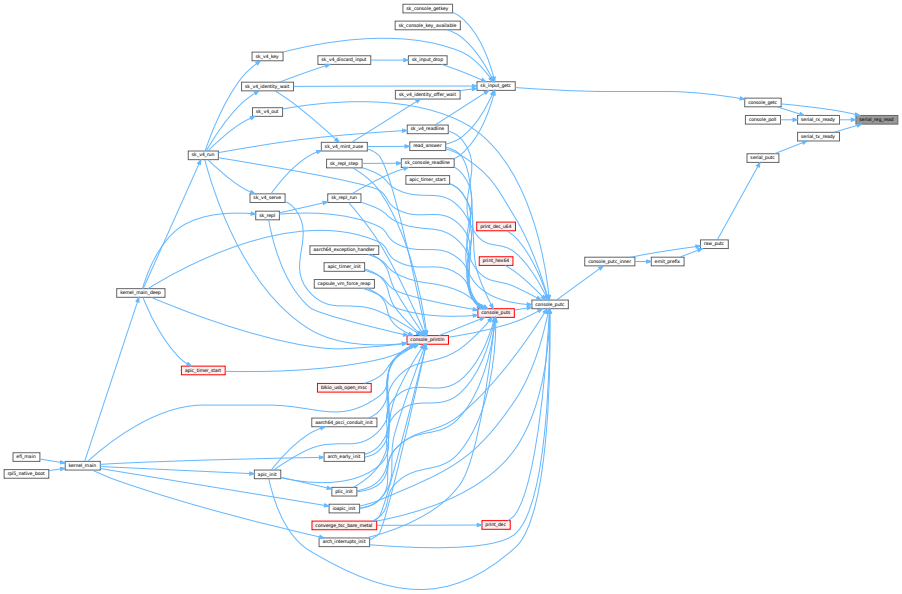
```
static uint32_t serial_reg_read (
            unsigned n )  [static]
```

References g_serial_base, g_serial_kind, g_serial_shift, g_serial_width, inb(), and SERIAL_PORT16550.

Referenced by console_getc(), serial_rx_ready(), and serial_tx_ready().

Here is the call graph for this function:

Here is the caller graph for this function:



7.148.3.31 serial_reg_write()

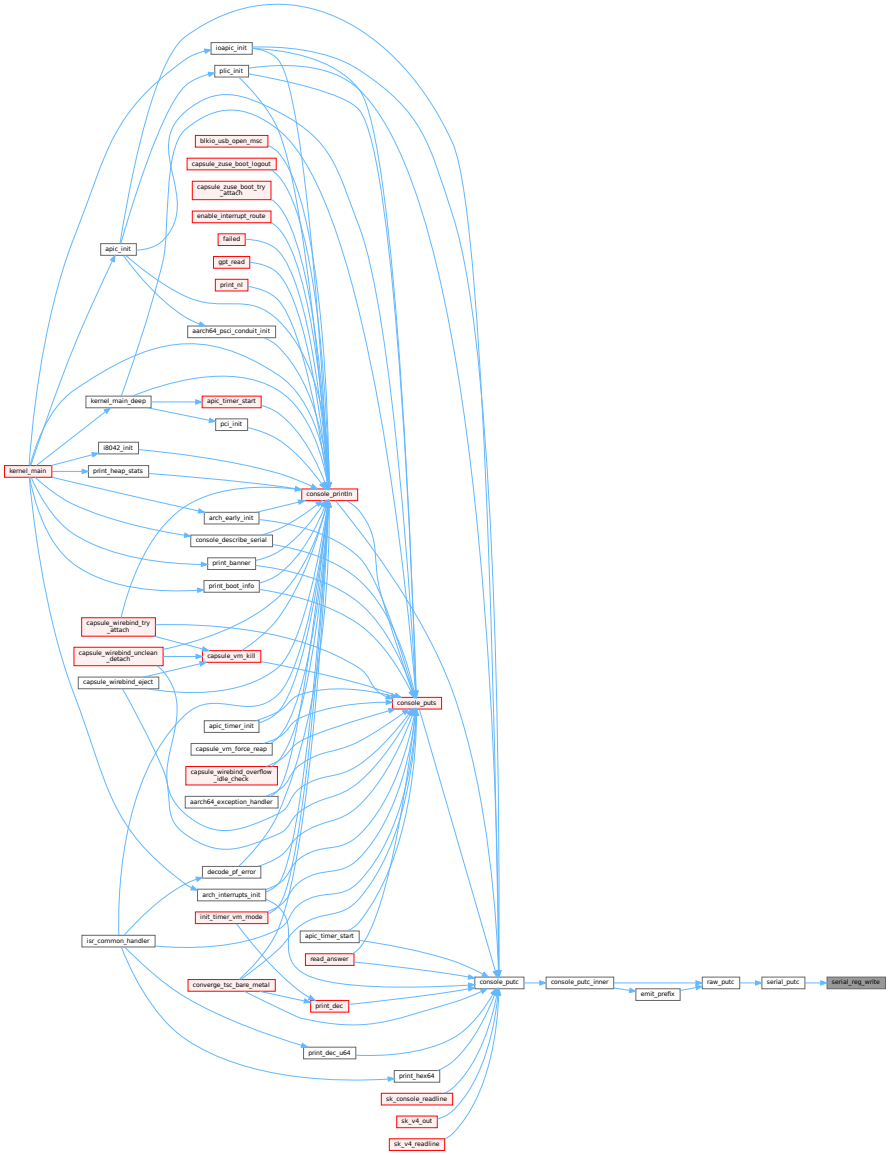
```
static void serial_reg_write (
            unsigned n,
            uint32_t v )  [static]
```

References g_serial_base, g_serial_kind, g_serial_shift, g_serial_width, outb(), and SERIAL_PORT16550.

Referenced by serial_putc().

Here is the call graph for this function:

Here is the caller graph for this function:



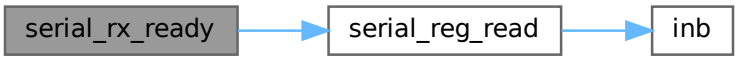
7.148.3.32 serial_rx_ready()

```
static int serial_rx_ready (
            void  )  [static]
```

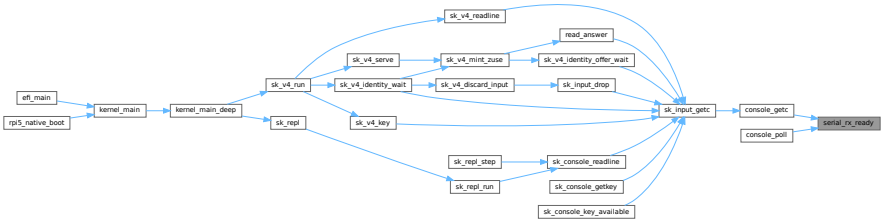
References g_serial_kind, PL011_FR, PL011_FR_RXFE, SERIAL_NONE, SERIAL_PL011, serial_reg_read(), UART16550_LSR, and UART16550_LSR_DR.

Referenced by console_getc(), and console_poll().

Here is the call graph for this function:



Here is the caller graph for this function:



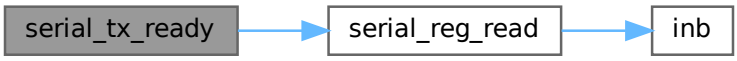
7.148.3.33 serial_tx_ready()

```
static int serial_tx_ready (
            void  )  [static]
```

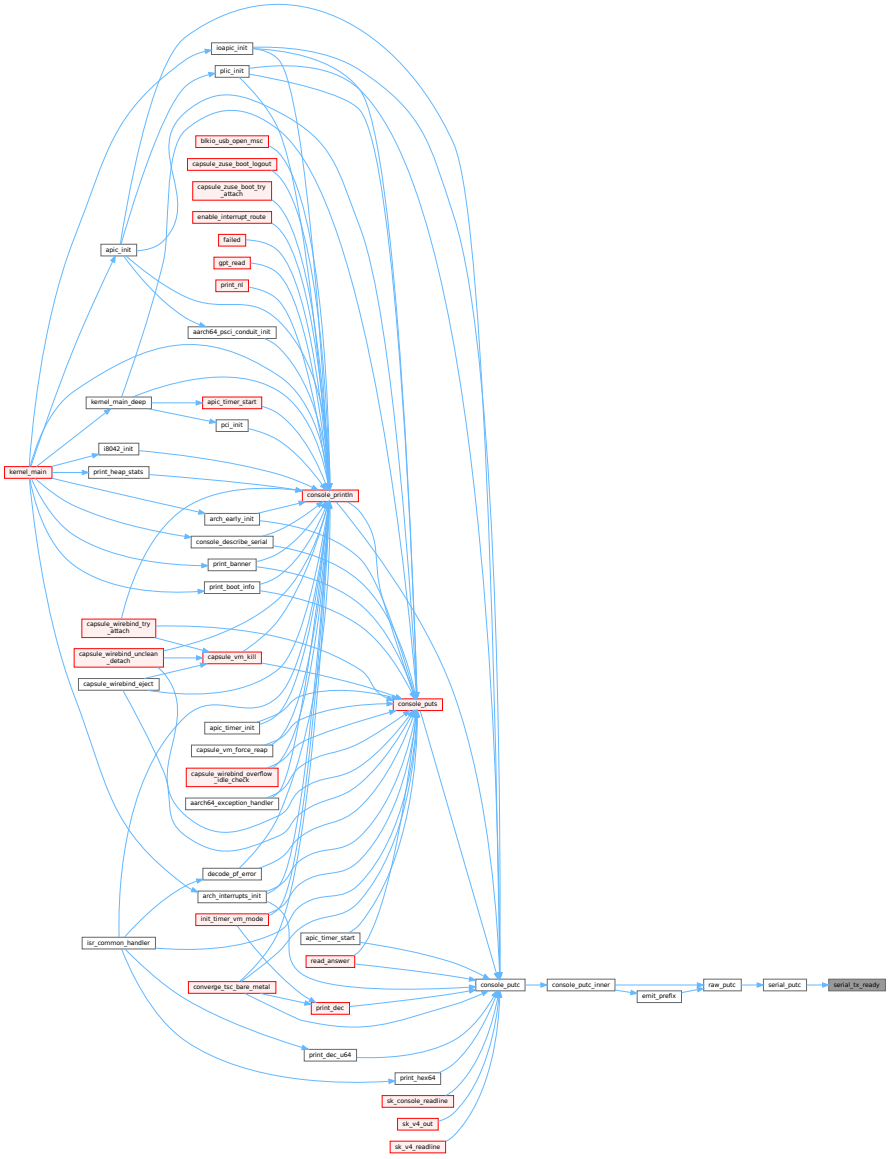
References g_serial_kind, PL011_FR, PL011_FR_TXFF, SERIAL_PL011, serial_reg_read(), UART16550_LSR, and UART16550_LSR_THRE.

Referenced by serial_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



7.148.4 Variable Documentation

7.148.4.1 g_active_vm_name

```
const char* g_active_vm_name = (void *)0  [static]
```

Referenced by console_get_vm_name(), console_putc_inner(), console_save_vm_name(), console_set_vm_name(), and emit_prefix().

7.148.4.2 g_active_vm_name_buf

```
char g_active_vm_name_buf[64]  [static]
```

Referenced by console_set_vm_name().

7.148.4.3 g_console_tx_count

```
uint64_t g_console_tx_count  [static]
```

Referenced by console_putc(), and console_tx_count().

7.148.4.4 g_fb_mirror

```
int g_fb_mirror = 1  [static]
```

Referenced by console_fb_mirror(), and fb_mirrored().

7.148.4.5 g_line_start

```
int g_line_start = 1  [static]
```

Referenced by console_ensure_line_start(), and console_putc_inner().

7.148.4.6 g_pending_line_close

```
int g_pending_line_close = 0  [static]
```

Referenced by console_cancel_deferred_line_start(), console_ensure_line_start(), and console_putc().

7.148.4.7 g_serial_base

```
uintptr_t g_serial_base  [static]
```

Referenced by console_attach_serial(), console_describe_serial(), serial_reg_read(), and serial_reg_write().

7.148.4.8 g_serial_from_dt

```
int g_serial_from_dt  [static]
```

Referenced by console_attach_serial(), and console_describe_serial().

7.148.4.9 g_serial_kind

```
int g_serial_kind  [static]
```

Referenced by console_attach_serial(), console_describe_serial(), console_init(), serial_putc(), serial_reg_read(), serial_reg_write(), serial_rx_ready(), and serial_tx_ready().

7.148.4.10 g_serial_shift

```
unsigned g_serial_shift  [static]
```

Referenced by console_attach_serial(), serial_reg_read(), and serial_reg_write().

7.148.4.11 g_serial_width

```
unsigned g_serial_width  [static]
```

Referenced by console_attach_serial(), serial_reg_read(), and serial_reg_write().

7.148.4.12 g_user_prefix_fn

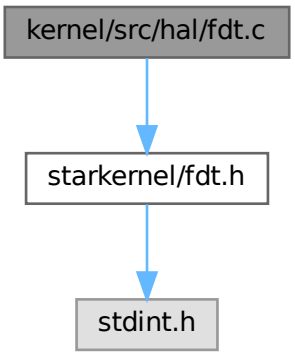
```
console_user_prefix_fn g_user_prefix_fn = (void *)0  [static]
```

Referenced by console_set_user_prefix_provider(), and emit_prefix().

7.149 kernel/src/hal/fdt.c File Reference

```
#include "starkernel/fdt.h"
```

Include dependency graph for fdt.c:



Data Structures

- struct fdt_header_t

Macros

- #define FDT_BEGIN_NODE 0x00000001u

  fdt.c - Minimal flattened-devicetree reader
- #define FDT_END_NODE 0x00000002u
- #define FDT_PROP 0x00000003u
- #define FDT_NOP 0x00000004u
- #define FDT_END 0x00000009u
- #define FDT_MAGIC 0xd00dfeedu
- #define FDT_PATH_MAX 12u

Functions

- static uint32_t be32 (const void *p)

  Byte-swap a big-endian 32-bit field from the blob.
- static int str_eq (const char *a, const char *b)

  Freestanding string compare; returns 1 when equal.
- int fdt_valid (const void *fdt)

  fdt.h - Minimal flattened-devicetree reader
- const void * fdt_find_prop (const void *fdt, const char *name, uint32_t *len_out)

  Find the first property with `name` anywhere in the tree.
- int fdt_prop_u32 (const void *fdt, const char *name, uint32_t *out)

  Read a single-cell (32-bit) property by name.
- static int compat_list_contains (const char *list, uint32_t len, const char *want)

  Test whether NUL-separated string list `list` (length `len`) contains `want` as one of its entries.
- const void * fdt_find_node_by_compatible (const void *fdt, const char *compatible)

  Find the first node whose "compatible" property matches `compatible`.
- const void * fdt_find_node_by_device_type (const void *fdt, const char *type)

  Find the first node whose "device_type" property equals `type`.
- static int node_name_eq (const char *node_name, const char *want)

  Test whether a node-name string (blob text, possibly followed by an `@unit-address`) matches `want` up to that boundary.
- const void * fdt_find_node_by_name (const void *fdt, const char *name)

  Find the first node whose own name matches `name`.
- static const unsigned char * skip_node_body (const unsigned char *p, const unsigned char *end)

  Advance `p` past one complete node body (properties, then every descendant, recursively via a depth counter) starting just inside it (i.e.
- const void * fdt_next_child_node (const void *fdt, const void *parent, const void *prev_child)

  Iterate the direct children of one node.
- int fdt_complete (const void *fdt)

  Is the structure block whole? `fdt_valid()` checks the header; this checks also that the structure block ends with its end token, as a blob that has not been cut short does.
- const void * fdt_root_node (const void *fdt)

  The root node.
- const void * fdt_find_child (const void *fdt, const void *parent, const char *name, uint32_t name_len)

  Find a direct child of a node by its whole name.
- const void * fdt_find_prop_in_node (const void *fdt, const void *node, const char *name, uint32_t *len_out)

  Find a property by name, scoped to one node.
- static uint32_t node_cells (const void *fdt, const void *node, const char *name, uint32_t otherwise)
- static uint64_t read_cells_at (const unsigned char *p, uint32_t n)
- static int node_enabled (const void *fdt, const void *node)
- static uint32_t path_to (const void *fdt, const void *node, const char *want, const void **path, uint32_t depth)
- int fdt_reg_address (const void *fdt, const char *compatible, uint32_t index, uint64_t *out)

  The CPU's address for entry `index` of the `reg` of the first enabled node compatible with `compatible` (2026-10-14).

7.149.1 Macro Definition Documentation

7.149.1.1 FDT_BEGIN_NODE

```
#define FDT_BEGIN_NODE 0x00000001u
```

fdt.c - Minimal flattened-devicetree reader

Devicetree Specification v0.4 ?5. Everything in the blob is big-endian; the kernel targets are little-endian, so every field goes through be32().

7.149.1.2 FDT_END

```
#define FDT_END 0x00000009u
```

7.149.1.3 FDT_END_NODE

```
#define FDT_END_NODE 0x00000002u
```

7.149.1.4 FDT_MAGIC

```
#define FDT_MAGIC 0xd00dfeedu
```

7.149.1.5 FDT_NOP

```
#define FDT_NOP 0x00000004u
```

7.149.1.6 FDT_PATH_MAX

```
#define FDT_PATH_MAX 12u
```

7.149.1.7 FDT_PROP

```
#define FDT_PROP 0x00000003u
```

7.149.2 Function Documentation

7.149.2.1 be32()

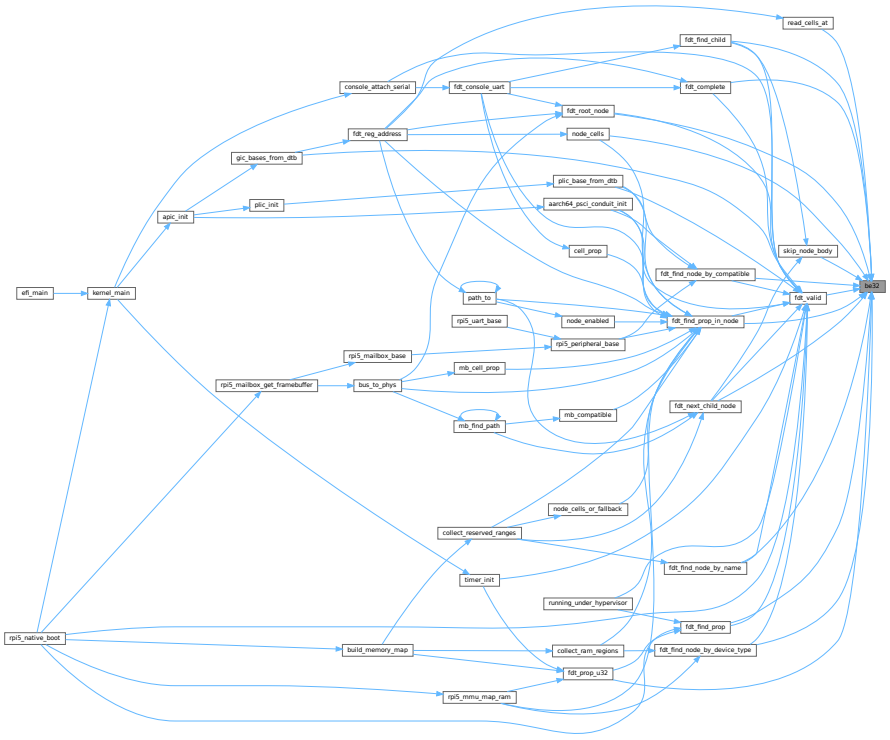
```
static uint32_t be32 (
            const void * p )  [static]
```

Byte-swap a big-endian 32-bit field from the blob.

Read bytewise rather than as a `uint32_t` load: structure-block tokens are only guaranteed 4-byte aligned relative to the block start, and property values carry no alignment guarantee at all.

Referenced by fdt_complete(), fdt_find_child(), fdt_find_node_by_compatible(), fdt_find_node_by_device_type(), fdt_find_node_by_name(), fdt_find_prop(), fdt_find_prop_in_node(), fdt_next_child_node(), fdt_prop_u32(), fdt_root_node(), fdt_valid(), node_cells(), read_cells_at(), and skip_node_body().

Here is the caller graph for this function:



7.149.2.2 compat_list_contains()

```
static int compat_list_contains (
            const char * list,
            uint32_t len,
            const char * want )  [static]
```

Test whether NUL-separated string list `list` (length `len`) contains `want` as one of its entries.

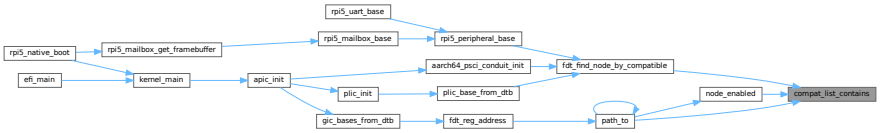
"compatible" properties are formatted as one or more NUL-terminated strings concatenated (DT spec ?2.3.1) – each entry's own embedded NUL is real property data, not synthesized, so str_eq() (which walks until either side's NUL) terminates correctly at each entry's real boundary.

References str_eq().

Referenced by fdt_find_node_by_compatible(), node_enabled(), and path_to().

Here is the call graph for this function:

Here is the caller graph for this function:



7.149.2.3 fdt_complete()

```
int fdt_complete (
            const void * fdt )
```

Is the structure block whole? `fdt_valid()` checks the header; this checks also that the structure block ends with its end token, as a blob that has not been cut short does.

A reader that takes a property's absence to mean its default (a UART's "reg-shift") must not be handed half a tree.

Returns

1 if `fdt` is valid and its structure block ends with FDT_END.

References be32(), FDT_END, fdt_valid(), fdt_header_t::off_dt_struct, and fdt_header_t::size_dt_struct.

Referenced by fdt_console_uart(), and fdt_reg_address().

Here is the call graph for this function:

Here is the caller graph for this function:

7.149.2.4 fdt_find_child()

```
const void * fdt_find_child (
            const void * fdt,
            const void * parent,
            const char * name,
            uint32_t name_len )
```

Find a direct child of a node by its whole name.

The name is matched exactly, unit address included ("serial@10000000" is not "serial"), and is given with its length so that one component of a path can be passed without copying it. Only `parent`'s own children are looked at, never their descendants.

Parameters

fdt	Blob; checked with `fdt_valid()`.
parent	Node handle; NULL is safe and returns NULL.
name	The child's name; need not be NUL-terminated.
name_len	Its length in bytes.

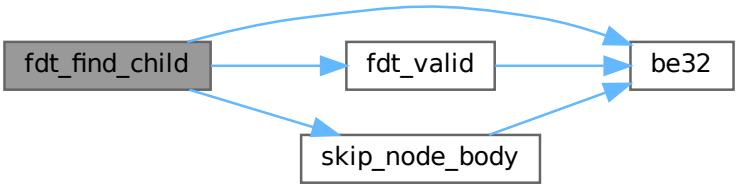
Returns

Handle to the child, or NULL if `parent` has none of that name.

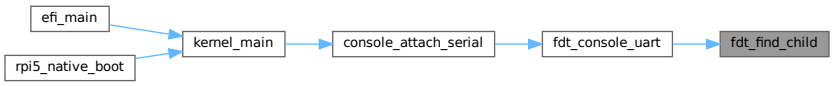
References be32(), FDT_BEGIN_NODE, FDT_NOP, FDT_PROP, fdt_valid(), name, name_len, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and skip_node_body().

Referenced by fdt_console_uart().

Here is the call graph for this function:



Here is the caller graph for this function:



7.149.2.5 fdt_find_node_by_compatible()

```
const void * fdt_find_node_by_compatible (
             const void * fdt,
             const char * compatible )
```

Find the first node whose "compatible" property matches `compatible`.

"compatible" is a NUL-separated list of strings (DT spec ?2.3.1) — matches if `compatible` equals any one entry in the list, not just the whole property verbatim. Scans the whole tree in document order; the first matching node wins if more than one exists.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
compatible	Compatible string to match, NUL-terminated.

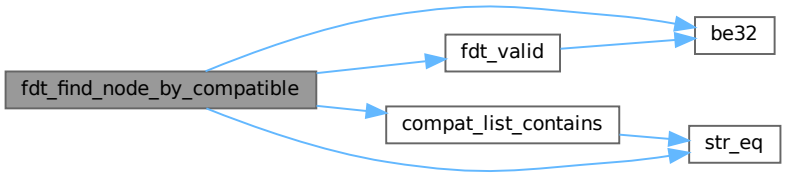
Returns

An opaque handle to the matched node, for use with `fdt_find_prop_in_node()` only (not a raw offset or a pointer to anything else meaningful) — or NULL if no node matches.

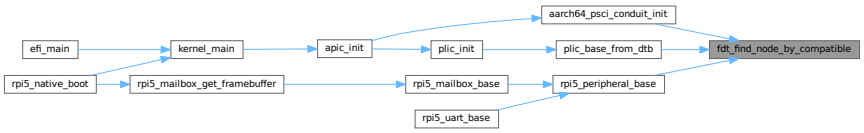
References base, be32(), compat_list_contains(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, FDT_PROP, fdt_valid(), fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and str_eq().

Referenced by aarch64_psci_conduit_init(), plic_base_from_dtb(), and rpi5_peripheral_base().

Here is the call graph for this function:



Here is the caller graph for this function:



7.149.2.6 fdt_find_node_by_device_type()

```
const void * fdt_find_node_by_device_type (
            const void * fdt,
            const char * type )
```

Find the first node whose "device_type" property equals `type`.

Some standard nodes (`/memory` per DT spec ?3.4) are identified by `device_type`, not `compatible` — unlike `compatible`, `device_type` is a single NUL-terminated string, not a list, so this matches the whole property value rather than scanning entries within it. Scans the whole tree in document order; the first matching node wins if more than one exists.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
type	device_type value to match, NUL-terminated.

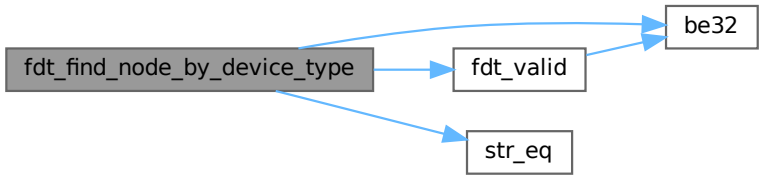
Returns

An opaque handle to the matched node, for use with `fdt_find_prop_in_node()` only — or NULL if no node matches.

References base, be32(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, FDT_PROP, fdt_valid(), fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and str_eq().

Referenced by collect_ram_regions(), and rpi5_mmu_map_ram().

Here is the call graph for this function:



Here is the caller graph for this function:

7.149.2.7 fdt_find_node_by_name()

```
const void * fdt_find_node_by_name (
            const void * fdt,
            const char * name )
```

Find the first node whose own name matches `name`.

Node names follow the DT spec ?2.2.1 `name[@unit-address]` convention — matches if `name` equals the node's name up to (not including) an `@` suffix, or the whole name if there is none. For a singleton node with no unit address (`/reserved-memory` being the concrete case this was added for), this is an exact match. Scans the whole tree in document order; the first matching node wins if more than one exists.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
name	Node name to match, NUL-terminated, no `@` suffix.

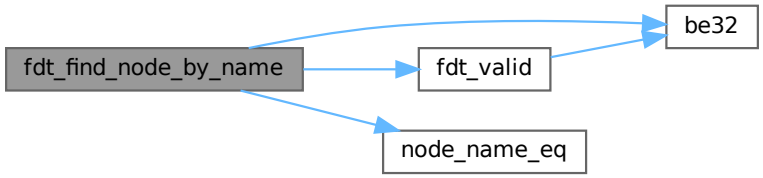
Returns

An opaque handle to the matched node, for use with `fdt_find_prop_in_node()` / `fdt_next_child_node()` only — or NULL if no node matches.

References base, be32(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, FDT_PROP, fdt_valid(), name, node_name_eq(), fdt_header_t::off_dt_struct, and fdt_header_t::size_dt_struct.

Referenced by collect_reserved_ranges().

Here is the call graph for this function:



Here is the caller graph for this function:

7.149.2.8 fdt_find_prop()

```
const void * fdt_find_prop (
             const void * fdt,
             const char * name,
             uint32_t * len_out )
```

Find the first property with `name` anywhere in the tree.

Scans the structure block in document order and returns the first match regardless of which node it belongs to. That is sufficient for properties which are uniform across a machine (timebase-frequency being the case this was written for) and is *not* sufficient for anything node-scoped.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
name	Property name, NUL-terminated.
len_out	Receives the property length in bytes; may be NULL.

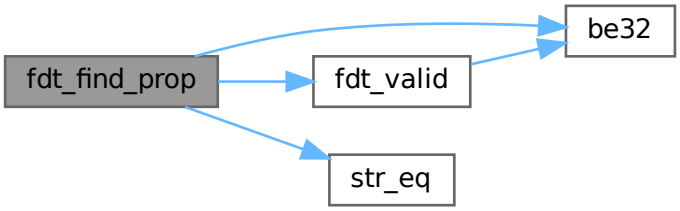
Returns

Pointer to the property value inside `fdt`, or NULL if not found. The value is big-endian as stored in the blob.

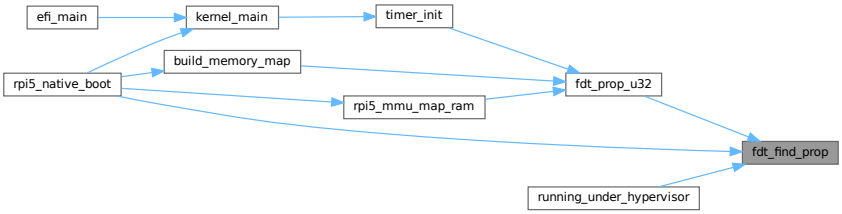
References base, be32(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, FDT_PROP, fdt_valid(), name, fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and str_eq().

Referenced by fdt_prop_u32(), rpi5_native_boot(), and running_under_hypervisor().

Here is the call graph for this function:



Here is the caller graph for this function:



7.149.2.9 fdt_find_prop_in_node()

```
const void * fdt_find_prop_in_node (
            const void * fdt,
            const void * node,
            const char * name,
            uint32_t * len_out )
```

Find a property by name, scoped to one node.

Like `fdt_find_prop()`, but scans only `node`'s own direct properties (as returned by `fdt_find_node_by_compatible()` — stops at the first child node or the end of `node`'s property list, never descends into children, never continues into a sibling. This is the difference that matters for a property name like "reg", which is not unique across the tree the way "timebase-frequency" (the whole reason `fdt_find_prop()` was originally sufficient) happens to be.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
node	Handle from `fdt_find_node_by_compatible()`; NULL is safe and returns NULL (propagates a failed node lookup without a separate caller-side check).
name	Property name, NUL-terminated.
len_out	Receives the property length in bytes; may be NULL.

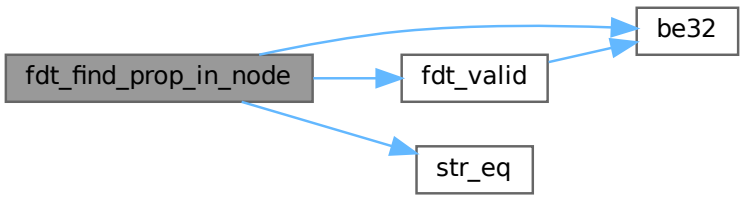
Returns

Pointer to the property value inside `fdt`, or NULL if not found (or if `node` is NULL). The value is big-endian as stored in the blob.

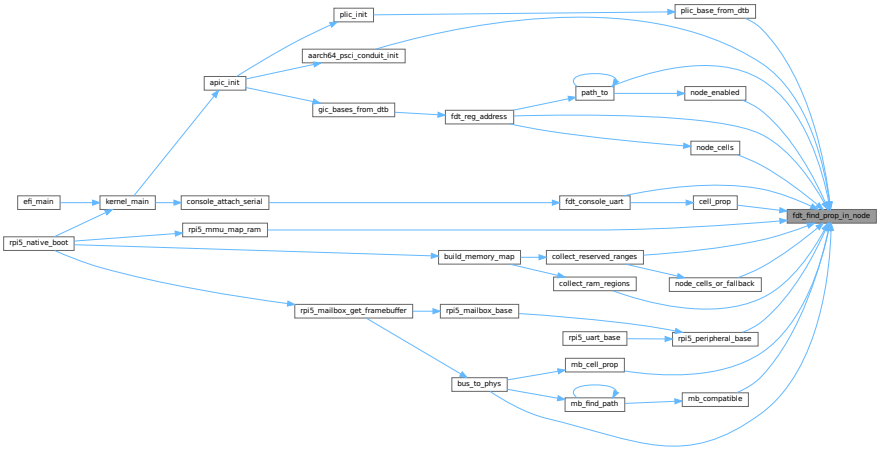
References be32(), FDT_NOP, FDT_PROP, fdt_valid(), name, fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and str_eq().

Referenced by aarch64_psci_conduit_init(), bus_to_phys(), cell_prop(), collect_ram_regions(), collect_reserved_ranges(), fdt_console_uart(), fdt_reg_address(), mb_cell_prop(), mb_compatible(), node_cells(), node_cells_or_fallback(), node_enabled(), path_to(), plic_base_from_dtb(), rpi5_mmu_map_ram(), and rpi5_peripheral_base().

Here is the call graph for this function:



Here is the caller graph for this function:



7.149.2.10 fdt_next_child_node()

```
const void * fdt_next_child_node (
            const void * fdt,
            const void * parent,
            const void * prev_child )
```

Iterate the direct children of one node.

Pass `prev_child` as NULL to get the first child; pass a previous result back in to get the next one. Stops (returns NULL) once there are no more children. Skips over each child's own descendants correctly (so a child with grandchildren doesn't confuse the scan), but does not itself descend into them — only direct children of `parent` are ever returned.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
parent	Handle from one of the `fdt_find_node_by_*()` functions.
prev_child	NULL for the first child, or a handle previously returned by this function for `parent` to continue from.

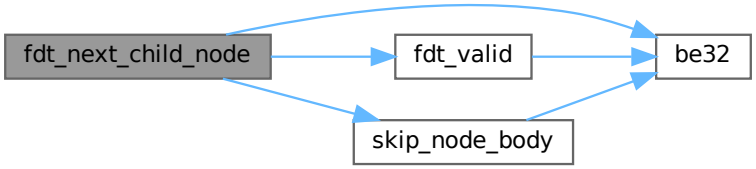
Returns

Handle to the next direct child, for use with `fdt_find_prop_in_node()` / `fdt_next_child_node()` only — or NULL once `parent`'s children are exhausted.

References be32(), FDT_BEGIN_NODE, FDT_NOP, FDT_PROP, fdt_valid(), fdt_header_t::off_dt_struct, fdt_header_t::size_dt_struct, and skip_node_body().

Referenced by collect_reserved_ranges(), mb_find_path(), and path_to().

Here is the call graph for this function:



Here is the caller graph for this function:

7.149.2.11 fdt_prop_u32()

```
int fdt_prop_u32 (
            const void * fdt,
            const char * name,
            uint32_t * out )
```

Read a single-cell (32-bit) property by name.

Convenience over `fdt_find_prop()` that also handles the big-endian conversion. Fails if the property is absent or not exactly 4 bytes.

Parameters

fdt	Blob, already checked with `fdt_valid()`.
name	Property name, NUL-terminated.
out	Receives the host-order value on success; untouched on failure.

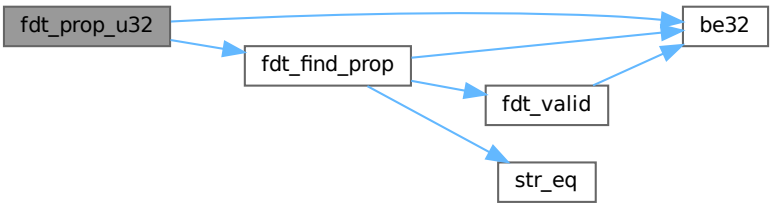
Returns

1 on success, 0 on failure.

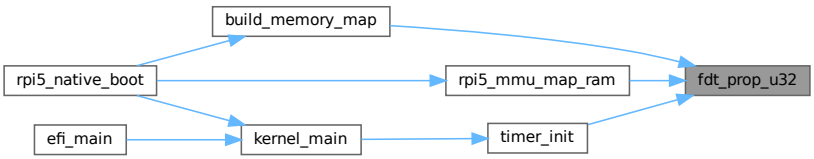
References be32(), fdt_find_prop(), and name.

Referenced by build_memory_map(), rpi5_mmu_map_ram(), and timer_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.149.2.12 fdt_reg_address()

```
int fdt_reg_address (
            const void * fdt,
            const char * compatible,
            uint32_t index,
            uint64_t * out )
```

The CPU's address for entry `index` of the `reg` of the first enabled node compatible with `compatible` (2026-10-14).

The entry is read in its parent's #address-cells/#size-cells and carried up through every bus's `ranges` to the root – an empty `ranges` keeps the address, a bus with no `ranges` has no way up. The Pi 5's GIC, for one, sits at 0x7fff9000 on /soc, which is 0x107fff9000 to the CPU.

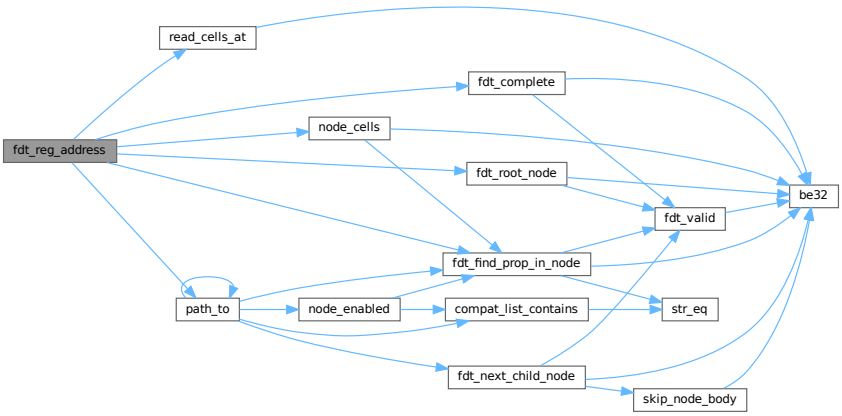
Returns

1 and the address in `out`, or 0 if there is no such node, entry or way up.

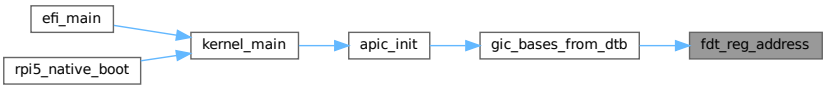
References entry, fdt_complete(), fdt_find_prop_in_node(), FDT_PATH_MAX, fdt_root_node(), node_cells(), path_to(), and read_cells_at().

Referenced by gic_bases_from_dtb().

Here is the call graph for this function:



Here is the caller graph for this function:



7.149.2.13 fdt_root_node()

```
const void * fdt_root_node (
            const void * fdt )
```

The root node.

Parameters

fdt	Blob; checked with `fdt_valid()`.

Returns

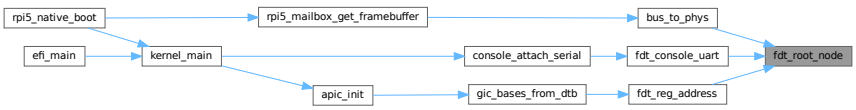
Handle to the root node, for `fdt_find_child()` and `fdt_find_prop_in_node()`, or NULL.

References be32(), FDT_BEGIN_NODE, FDT_NOP, fdt_valid(), fdt_header_t::off_dt_struct, and fdt_header_t::size_dt_struct.

Referenced by bus_to_phys(), fdt_console_uart(), and fdt_reg_address().

Here is the call graph for this function:

Here is the caller graph for this function:



7.149.2.14 fdt_valid()

```
int fdt_valid (
            const void * fdt )
```

fdt.h - Minimal flattened-devicetree reader

Just enough of the Devicetree Specification v0.4 ?5 to pull values out of the blob the UEFI firmware publishes under EFI_DTB_TABLE_GUID, or that a native (non-UEFI) boot entry passes directly. Read-only, no allocation, no tree construction — it walks the structure block each call, which is fine for the handful of boot-time lookups the kernel needs.

Deliberately not a general devicetree library. Added for punch-list item 0.3 (riscv64 timebase-frequency); extended (FABRIC-3.md ?IV.3/?V.3, 2026-09-04) with node-scoped lookup, for exactly the case this header originally flagged as a future need (item 0.6's aarch64 GIC) plus its real, concrete consumers as of this pass: the Raspberry Pi 5's UART/ mailbox register addresses (native boot, no ACPI) and the Milk-V Mars's real PLIC base address (currently hardcoded to QEMU-virt's own value, `arch/riscv64/plic.c`'s own doc comment already warned this isn't assumed stable across configurations).

Test whether `fdt` points at a valid flattened devicetree.

Checks the 0xd00dfeed magic and that the structure and strings blocks lie inside totalsize. Does not validate the token stream.

Parameters

fdt	Candidate blob; NULL is safe and returns 0.

Returns

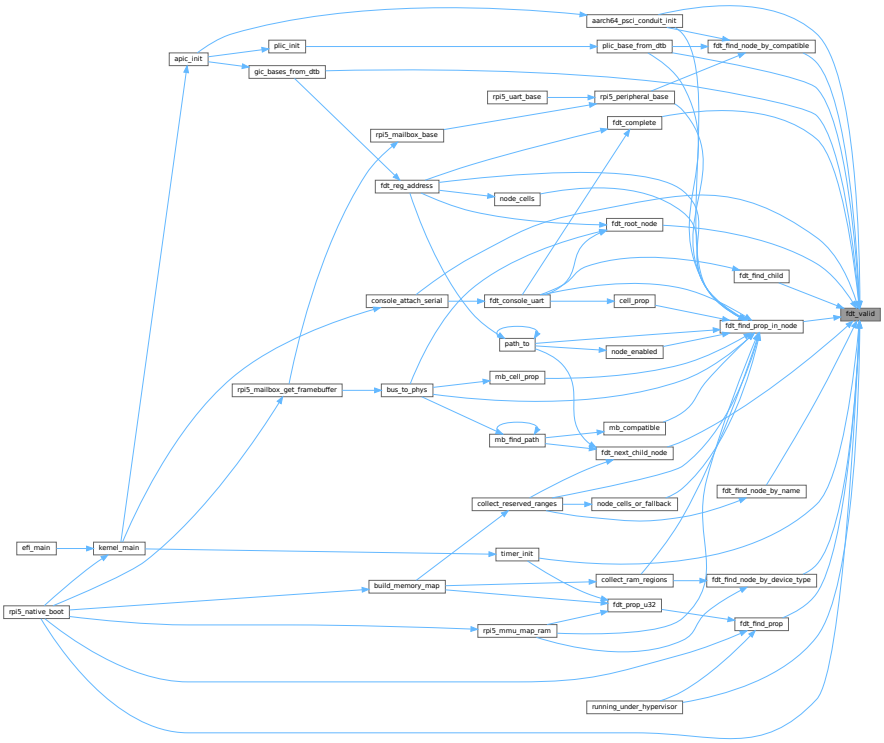
1 if the header is usable, 0 otherwise.

References be32(), FDT_MAGIC, fdt_header_t::magic, fdt_header_t::off_dt_strings, fdt_header_t::off_dt_struct, fdt_header_t::size_dt_strings, fdt_header_t::size_dt_struct, and fdt_header_t::totalsize.

Referenced by aarch64_psci_conduit_init(), console_attach_serial(), fdt_complete(), fdt_find_child(), fdt_find_node_by_compatible(), fdt_find_node_by_device_type(), fdt_find_node_by_name(), fdt_find_prop(), fdt_find_prop_in_node(), fdt_next_child_node(), fdt_root_node(), gic_bases_from_dtb(), plic_base_from_dtb(), rpi5_native_boot(), running_under_hypervisor(), and timer_init().

Here is the call graph for this function:

Here is the caller graph for this function:



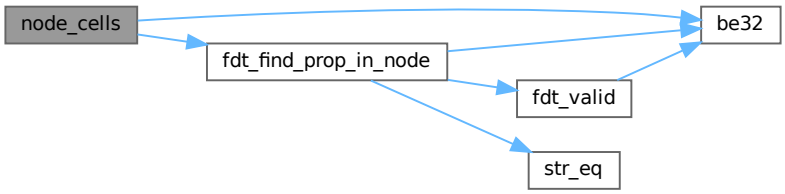
7.149.2.15 node_cells()

```
static uint32_t node_cells (
            const void * fdt,
            const void * node,
            const char * name,
            uint32_t otherwise )  [static]
```

References be32(), fdt_find_prop_in_node(), and name.

Referenced by fdt_reg_address().

Here is the call graph for this function:



Here is the caller graph for this function:

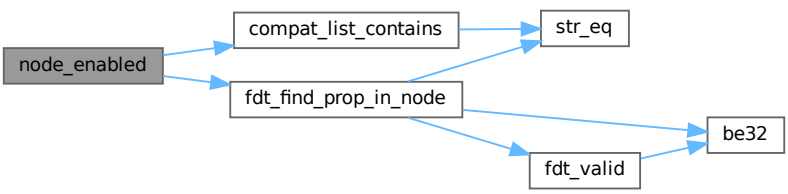
7.149.2.16 node_enabled()

```
static int node_enabled (
            const void * fdt,
            const void * node )  [static]
```

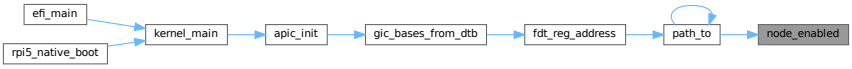
References compat_list_contains(), and fdt_find_prop_in_node().

Referenced by path_to().

Here is the call graph for this function:



Here is the caller graph for this function:



7.149.2.17 node_name_eq()

```
static int node_name_eq (
            const char * node_name,
            const char * want )  [static]
```

Test whether a node-name string (blob text, possibly followed by an `@unit-address`) matches `want` up to that boundary.

Referenced by fdt_find_node_by_name().

Here is the caller graph for this function:

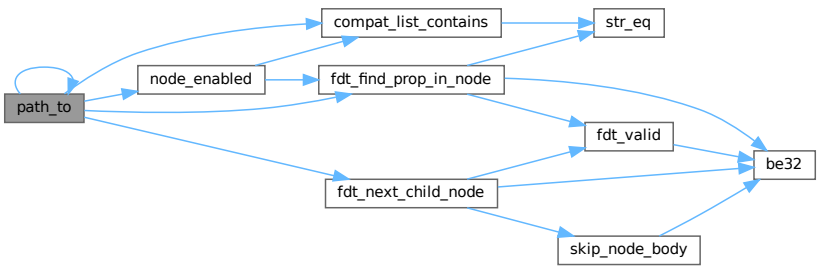
7.149.2.18 path_to()

```
static uint32_t path_to (
            const void * fdt,
            const void * node,
            const char * want,
            const void ** path,
            uint32_t depth )  [static]
```

References compat_list_contains(), fdt_find_prop_in_node(), fdt_next_child_node(), FDT_PATH_MAX, node_enabled(), and path_to().

Referenced by fdt_reg_address(), and path_to().

Here is the call graph for this function:



Here is the caller graph for this function:

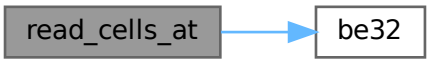
7.149.2.19 read_cells_at()

```
static uint64_t read_cells_at (
            const unsigned char * p,
            uint32_t n )  [static]
```

References be32().

Referenced by fdt_reg_address().

Here is the call graph for this function:



Here is the caller graph for this function:

7.149.2.20 skip_node_body()

```
static const unsigned char * skip_node_body (
            const unsigned char * p,
            const unsigned char * end )  [static]
```

Advance p past one complete node body (properties, then every descendant, recursively via a depth counter) starting just inside it (i.e.

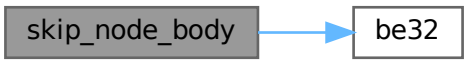
p is a node handle, right after its own FDT_BEGIN_NODE name), stopping just after the matching FDT_END_↩ NODE.

Internal to `fdt_next_child_node()` — a node handle's own children may have further descendants even though this reader's public node-lookup functions never return anything but direct-child handles, so finding the *next sibling* after a child still requires skipping past whatever that child itself contains.

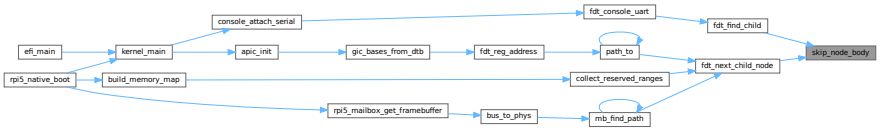
References be32(), FDT_BEGIN_NODE, FDT_END_NODE, FDT_NOP, and FDT_PROP.

Referenced by fdt_find_child(), and fdt_next_child_node().

Here is the call graph for this function:



Here is the caller graph for this function:



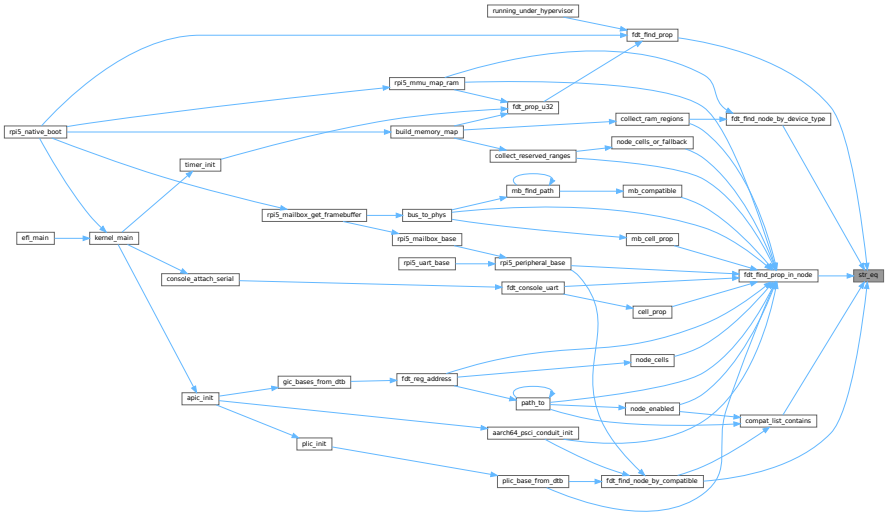
7.149.2.21 str_eq()

```
static int str_eq (
            const char * a,
            const char * b )  [static]
```

Freestanding string compare; returns 1 when equal.

Referenced by compat_list_contains(), fdt_find_node_by_compatible(), fdt_find_node_by_device_type(), fdt_find_prop(), and fdt_find_prop_in_node().

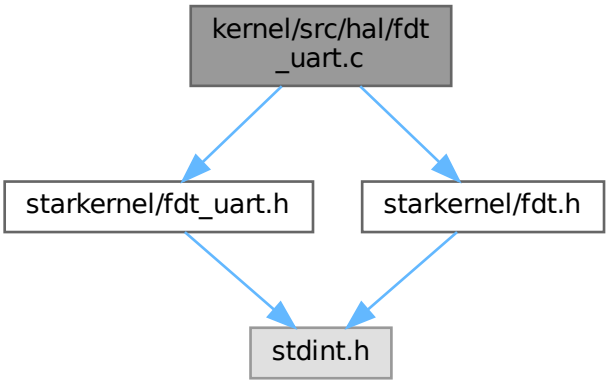
Here is the caller graph for this function:



7.150 kernel/src/hal/fdt_uart.c File Reference

```
#include "starkernel/fdt_uart.h"
#include "starkernel/fdt.h"
```

Include dependency graph for fdt_uart.c:



Macros

- #define PATH_DEPTH 8 /∗ nodes from the root to the UART, the root counted ∗/

  fdt_uart.c - the console's serial line, as a board's devicetree names it.

Functions

- static uint32_t be32 (const unsigned char ∗b)
- static uint32_t cell_prop (const void ∗fdt, const void ∗node, const char ∗name, uint32_t otherwise)
- static uint64_t cells (const unsigned char ∗p, uint32_t n)
- static int compatible_has (const char ∗list, uint32_t len, const char ∗want)
- int fdt_console_uart (const void ∗fdt, fdt_uart_t ∗out)

  Find the console's serial line in a devicetree.

7.150.1 Macro Definition Documentation

7.150.1.1 PATH_DEPTH

```
#define PATH_DEPTH 8 /* nodes from the root to the UART, the root counted */
```

fdt_uart.c - the console's serial line, as a board's devicetree names it.

See fdt_uart.h.

The node is reached a component of its path at a time from the root, so that each bus on the way is in hand: a node's "reg" is an address on its parent's bus, and is carried up to the CPU's own addresses through the "ranges" of every bus above it (DT spec 2.3.8). A bus with an empty "ranges" passes addresses on unchanged; one with none at all has no mapping, and the line is not found.

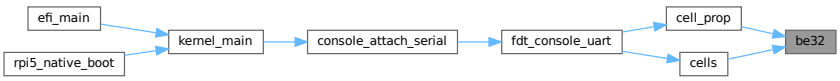
7.150.2 Function Documentation

7.150.2.1 be32()

```
static uint32_t be32 (
            const unsigned char * b )  [static]
```

Referenced by cell_prop(), and cells().

Here is the caller graph for this function:



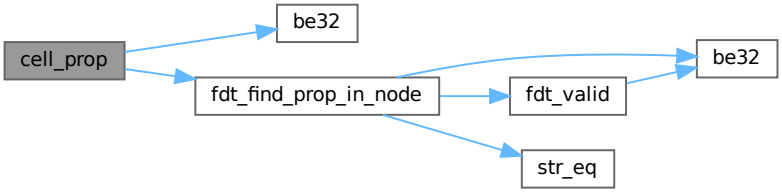
7.150.2.2 cell_prop()

```
static uint32_t cell_prop (
            const void * fdt,
            const void * node,
            const char * name,
            uint32_t otherwise )  [static]
```

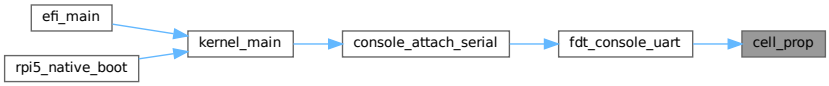
References be32(), fdt_find_prop_in_node(), and name.

Referenced by fdt_console_uart().

Here is the call graph for this function:



Here is the caller graph for this function:



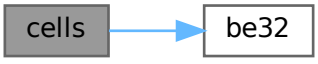
7.150.2.3 cells()

```
static uint64_t cells (
            const unsigned char * p,
            uint32_t n )  [static]
```

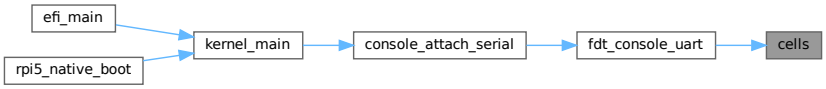
References be32().

Referenced by fdt_console_uart().

Here is the call graph for this function:



Here is the caller graph for this function:

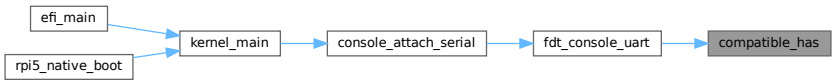


7.150.2.4 compatible_has()

```
static int compatible_has (
            const char * list,
            uint32_t len,
            const char * want )  [static]
```

Referenced by fdt_console_uart().

Here is the caller graph for this function:



7.150.2.5 fdt_console_uart()

```
int fdt_console_uart (
            const void * fdt,
            fdt_uart_t * out )
```

Find the console's serial line in a devicetree.

Parameters

fdt	Blob; NULL and what is not a devicetree are safe.
out	Receives the line; its kind is FDT_UART_NONE when 0 is returned.

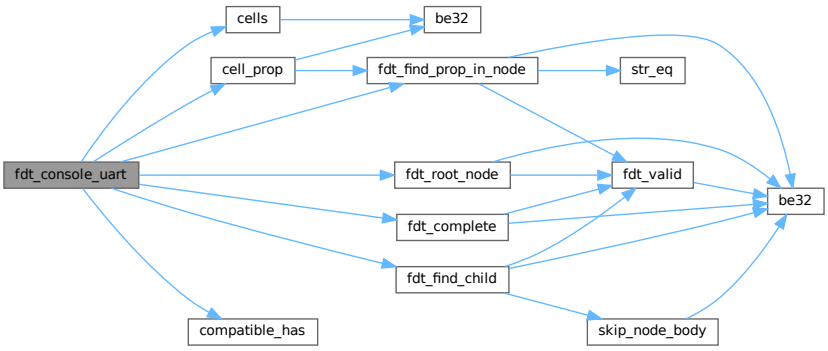
Returns

1 if the devicetree names a console UART that was found, placed and is of a kind listed above; 0 otherwise.

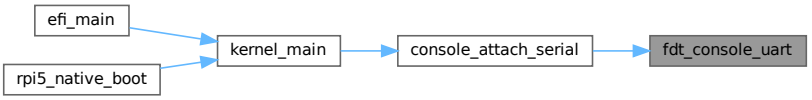
References fdt_uart_t::base, cell_prop(), cells(), compatible_has(), entry, fdt_complete(), fdt_find_child(), fdt_find_prop_in_node(), fdt_root_node(), FDT_UART_16550, FDT_UART_NONE, FDT_UART_PL011, fdt_uart_t::io_width, fdt_uart_t::kind, PATH_DEPTH, and fdt_uart_t::reg_shift.

Referenced by console_attach_serial().

Here is the call graph for this function:

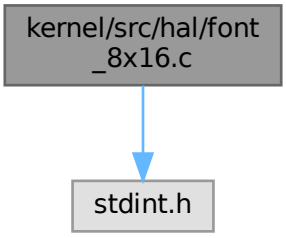


Here is the caller graph for this function:



7.151 kernel/src/hal/font_8x16.c File Reference

`#include <stdint.h>`
Include dependency graph for font_8x16.c:



Functions

- const uint8_t * font_8x16_glyph (uint8_t ch)

  Return pointer to the 16-byte glyph bitmap for character ch.

Variables

- const uint8_t font_8x16_data [256 ∗16]

  font_8x16.c — Embedded IBM CP437 VGA 8× 16 bitmap font (256 glyphs × 16 bytes)

7.151.1 Function Documentation

7.151.1.1 font_8x16_glyph()

```
const uint8_t * font_8x16_glyph (
            uint8_t ch )
```

Return pointer to the 16-byte glyph bitmap for character ch.

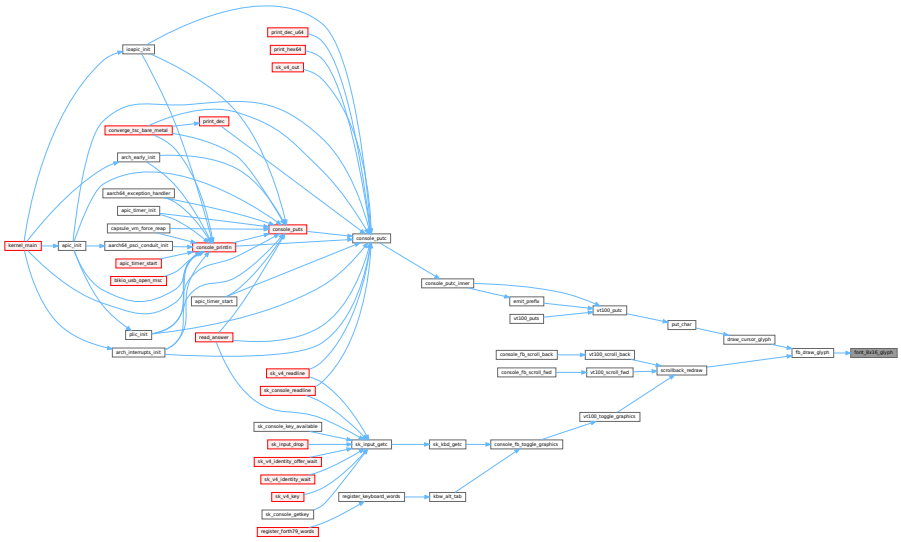
framebuffer.c — UEFI GOP framebuffer pixel driver

Caller must not modify the returned data.

References font_8x16_data.

Referenced by fb_draw_glyph().

Here is the caller graph for this function:



7.151.2 Variable Documentation

7.151.2.1 font_8x16_data

```
const uint8_t font_8x16_data[256 ∗16]
```

font_8x16.c — Embedded IBM CP437 VGA 8×16 bitmap font (256 glyphs × 16 bytes)

Format: one byte per scanline, MSB = leftmost pixel. Glyph for character N starts at byte N∗16.

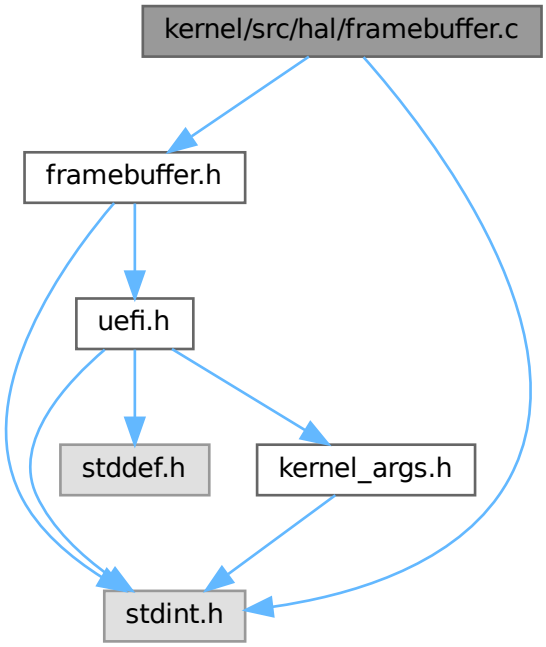
Derived from the IBM PC BIOS ROM font, which is in the public domain.

Referenced by font_8x16_glyph().

7.152 kernel/src/hal/framebuffer.c File Reference

```
#include "framebuffer.h"
#include <stdint.h>
```

Include dependency graph for framebuffer.c:



Data Structures

- struct FbState

Functions

- const uint8_t * font_8x16_glyph (uint8_t ch)

  framebuffer.c — UEFI GOP framebuffer pixel driver
- const uint32_t FB_ANSI_PALETTE[16] __attribute__ ((visibility("hidden")))
- static uint32_t pack_pixel (uint32_t rgb)
- void fb_init (const FramebufferInfo *info, FbPixelFormat fmt)

  Initialize the framebuffer from UEFI bootloader GOP data.
- int fb_is_available (void)

  Returns 1 if the framebuffer has been successfully initialized.
- uint32_t fb_width (void)
- uint32_t fb_height (void)
- uint32_t fb_cell_w (void)

  Effective character cell dimensions in pixels (accounts for scale).
- uint32_t fb_cell_h (void)

- void fb_put_pixel (uint32_t x, uint32_t y, uint32_t rgb)

  Write a single pixel.
- void fb_fill_rect (uint32_t x, uint32_t y, uint32_t w, uint32_t h, uint32_t rgb)

  Fill a rectangle with a solid color.
- void fb_draw_glyph (uint32_t px, uint32_t py, uint8_t ch, uint32_t fg, uint32_t bg)

  Draw one 8×16 glyph at pixel position (px, py).
- void fb_draw_orientation_test (void)

  fb_draw_orientation_test — one-time boot diagnostic (FABRIC-0.md item 4.3.1).
- void fb_scroll_rows (uint32_t pixel_rows, uint32_t bg)

  Scroll the framebuffer up by `pixel_rows` pixel rows.
- void fb_scroll_rect (uint32_t x, uint32_t y, uint32_t w, uint32_t h, uint32_t pixel_rows, uint32_t bg)

  fb_scroll_rect — box-confined scroll (FABRIC-0.md item 4.4t).

Variables

- static FbState g_fb

7.152.1 Function Documentation

7.152.1.1 __attribute__()

```
const uint32_t FB_ANSI_PALETTE[16] __attribute__ (
            (visibility("hidden"))  )
```

References FB_RGB.

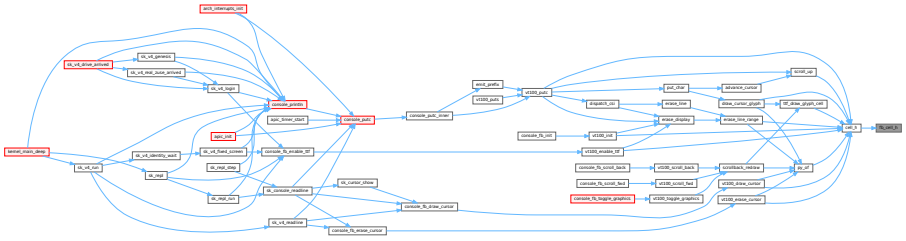
7.152.1.2 fb_cell_h()

```
uint32_t fb_cell_h (
            void  )
```

References g_fb, and FbState::scale.

Referenced by cell_h().

Here is the caller graph for this function:



7.152.1.3 fb_cell_w()

```
uint32_t fb_cell_w (
            void  )
```

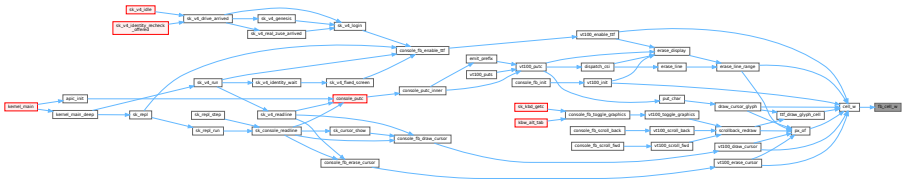
Effective character cell dimensions in pixels (accounts for scale).

Effective character cell width/height in pixels (8/16 × scale factor).

References g_fb, and FbState::scale.

Referenced by cell_w().

Here is the caller graph for this function:



7.152.1.4 fb_draw_glyph()

```
void fb_draw_glyph (
            uint32_t px,
            uint32_t py,
            uint8_t ch,
            uint32_t fg,
            uint32_t bg )
```

Draw one 8×16 glyph at pixel position (px, py).

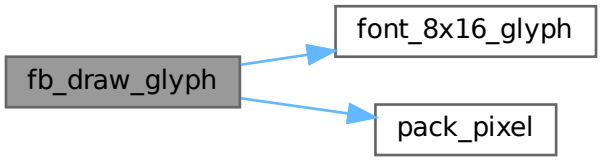
Draw one 8×16 character glyph at pixel position (px, py).

Scales by g_fb.scale so the same font works at any resolution.

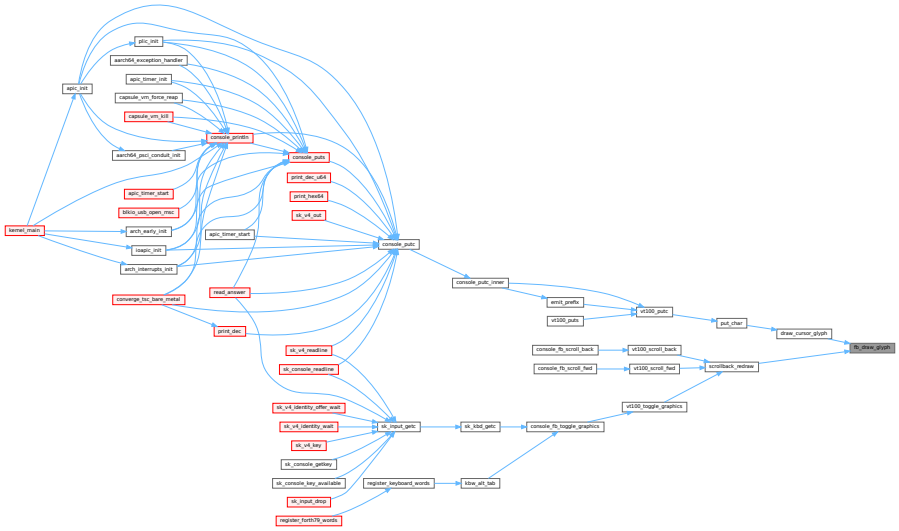
References FbState::base, font_8x16_glyph(), g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::scale, FbState::stride, and FbState::width.

Referenced by draw_cursor_glyph(), and scrollback_redraw().

Here is the call graph for this function:



Here is the caller graph for this function:



7.152.1.5 fb_draw_orientation_test()

```
void fb_draw_orientation_test (
            void  )
```

fb_draw_orientation_test — one-time boot diagnostic (FABRIC-0.md item 4.3.1).

One-time boot diagnostic (FABRIC-0.md item 4.3.1): fills each raster corner with a distinct solid color so a screendump reveals orientation.

Fills each raster corner with a distinct solid color so a screendump immediately reveals whether the image is flipped or rotated. Not part of the Console drawing fabric itself – diagnostic-only, analogous to item 4.2's self-test scaffolding.

Corner colors (raster/top-left-origin coordinates, pre-Cartesian-flip): top-left RED top-right GREEN bottom-left BLUE bottom-right YELLOW

References fb_fill_rect(), FB_RGB, g_fb, FbState::height, FbState::ready, and FbState::width.

Here is the call graph for this function:

7.152.1.6 fb_fill_rect()

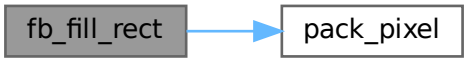
```
void fb_fill_rect (
            uint32_t x,
            uint32_t y,
            uint32_t w,
            uint32_t h,
            uint32_t rgb )
```

Fill a rectangle with a solid color.

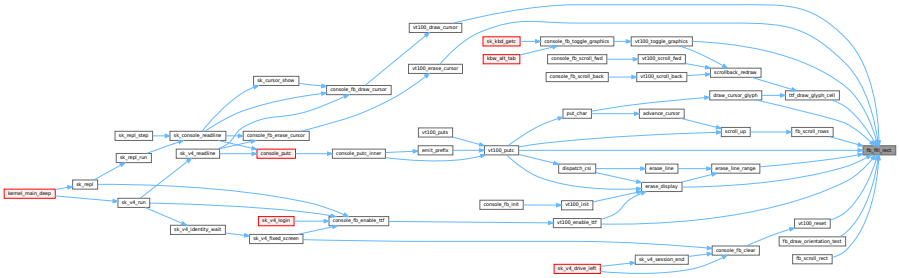
References FbState::base, g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::stride, and FbState::width.

Referenced by draw_cursor_glyph(), erase_display(), erase_line_range(), fb_draw_orientation_test(), fb_scroll_rect(), fb_scroll_rows(), ttf_draw_glyph_cell(), vt100_draw_cursor(), vt100_enable_ttf(), vt100_erase_cursor(), vt100_putc(), vt100_reset(), and vt100_toggle_graphics().

Here is the call graph for this function:



Here is the caller graph for this function:



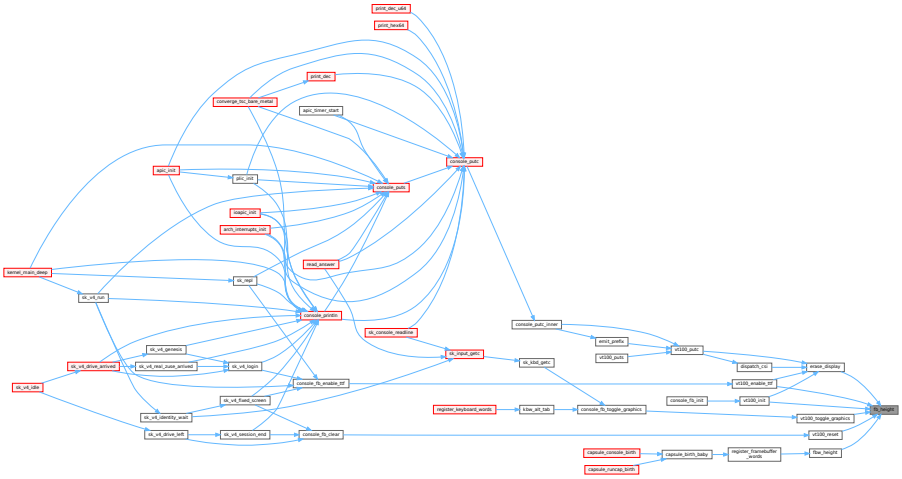
7.152.1.7 fb_height()

```
uint32_t fb_height (
            void  )
```

References g_fb, and FbState::height.

Referenced by erase_display(), fbw_height(), vt100_enable_ttf(), vt100_init(), vt100_reset(), and vt100_toggle_graphics().

Here is the caller graph for this function:



7.152.1.8 fb_init()

```
void fb_init (
            const FramebufferInfo * info,
            FbPixelFormat fmt )
```

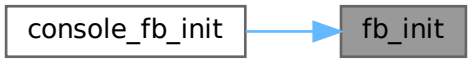
Initialize the framebuffer from UEFI bootloader GOP data.

Must be called before any other fb_∗ function. fmt: pixel layout reported by GOP (default FB_PIXEL_BGRX32).

References FramebufferInfo::base, FbState::base, FbState::fmt, g_fb, FramebufferInfo::height, FbState::height, FramebufferInfo::pixels_per_scanline, FbState::ready, FbState::scale, FbState::stride, FramebufferInfo::width, and FbState::width.

Referenced by console_fb_init().

Here is the caller graph for this function:



7.152.1.9 fb_is_available()

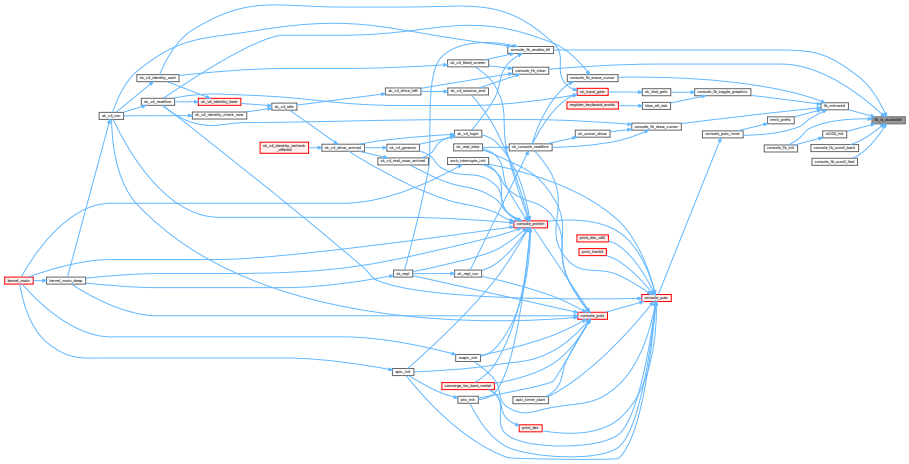
```
int fb_is_available (
            void  )
```

Returns 1 if the framebuffer has been successfully initialized.

References g_fb, and FbState::ready.

Referenced by console_fb_clear(), console_fb_enable_ttf(), console_fb_init(), console_fb_scroll_back(), console_fb_scroll_fwd(), fb_mirrored(), and vt100_init().

Here is the caller graph for this function:



7.152.1.10 fb_put_pixel()

```
void fb_put_pixel (
            uint32_t x,
            uint32_t y,
            uint32_t rgb )
```

Write a single pixel.

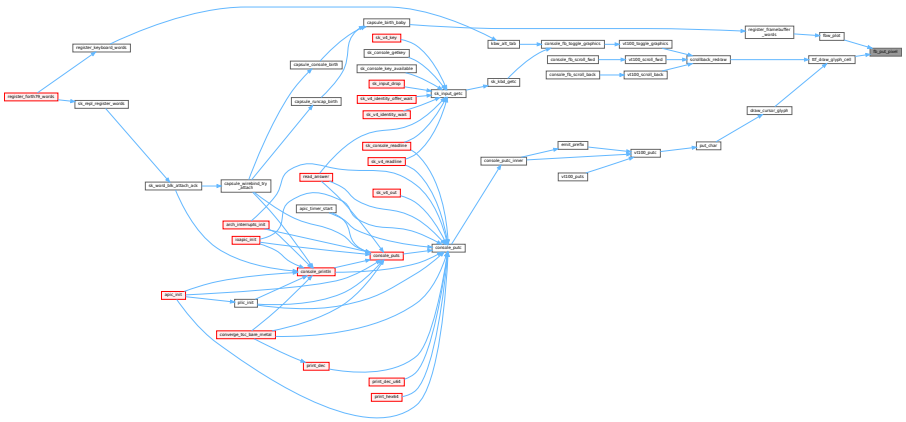
Out-of-bounds writes are silently ignored.

References FbState::base, g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::stride, and FbState::width.

Referenced by fbw_plot(), and ttf_draw_glyph_cell().

Here is the call graph for this function:

Here is the caller graph for this function:



7.152.1.11 fb_scroll_rect()

```
void fb_scroll_rect (
            uint32_t x,
            uint32_t y,
            uint32_t w,
            uint32_t h,
            uint32_t pixel_rows,
            uint32_t bg )
```

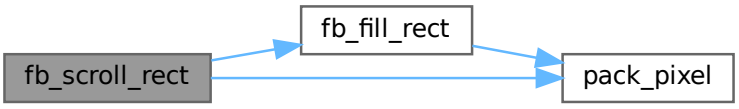
fb_scroll_rect — box-confined scroll (FABRIC-0.md item 4.4t).

Scroll a sub-rectangle of the framebuffer up by `pixel_rows` pixel rows (FABRIC-0.md item 4.4t: box-confined REPL scrolling).

Same non-volatile bulk-copy rationale as fb_scroll_rows() above, but bounded to a caller-supplied rectangle instead of the whole framebuffer – needed once REPL text is confined to the 4.4o CANVAS box, since a whole-framebuffer scroll would drag pixels from outside the box (e.g. a future 4.4r border) into the text area. Takes pixel_rows directly rather than char_rows ∗ a hardcoded cell height, since TTF mode's cell height (24px) differs from bitmap mode's (16px ∗ scale).

References FbState::base, fb_fill_rect(), g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::stride, and FbState::width.

Here is the call graph for this function:



7.152.1.12 fb_scroll_rows()

```
void fb_scroll_rows (
            uint32_t pixel_rows,
            uint32_t bg )
```

Scroll the framebuffer up by `pixel_rows` pixel rows.

Scroll the whole framebuffer up by `pixel_rows` pixel rows.

The vacated rows at the bottom are filled with bg.

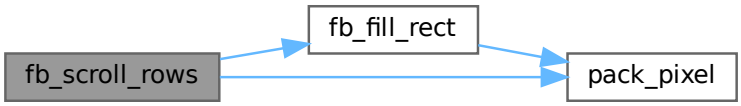
Takes a pixel-row count directly rather than a character-row count scaled by a hardcoded 16px cell height: the caller (vt100.c's scroll_up()) knows the actual active glyph cell height (16 × scale for bitmap mode, 24 for TTF mode), and a mismatch between the height this function scrolls by and the height the text model actually draws rows at compounds by the gap on every single scroll – a few scrolls barely show it, but enough scrolls (or scrolling quickly) accumulates visible pixel overlap between rows.

Copies through non-volatile pointers (FABRIC-0.md item 4.4g performance fix, 2026-08-11): the GOP framebuffer is mapped write-back, not cache-disabled MMIO (vmm.c:350-363 – "QEMU's VGA emulation is coherent and UC-mapping causes #GP"), so it behaves as ordinary RAM and does not need per-word volatile access. `g_fb.base` stays `volatile` for single-pixel writes elsewhere in this file (draw calls that must not be reordered relative to other framebuffer state), but that qualifier forced this copy into one un-batchable 4-byte access per element with no compiler vectorization – catastrophic once boot output exceeds one screen (~10^10 volatile accesses for an 18,000-line boot transcript under TCG). Casting away volatile here for the bulk copy/fill lets the compiler batch these loops normally.

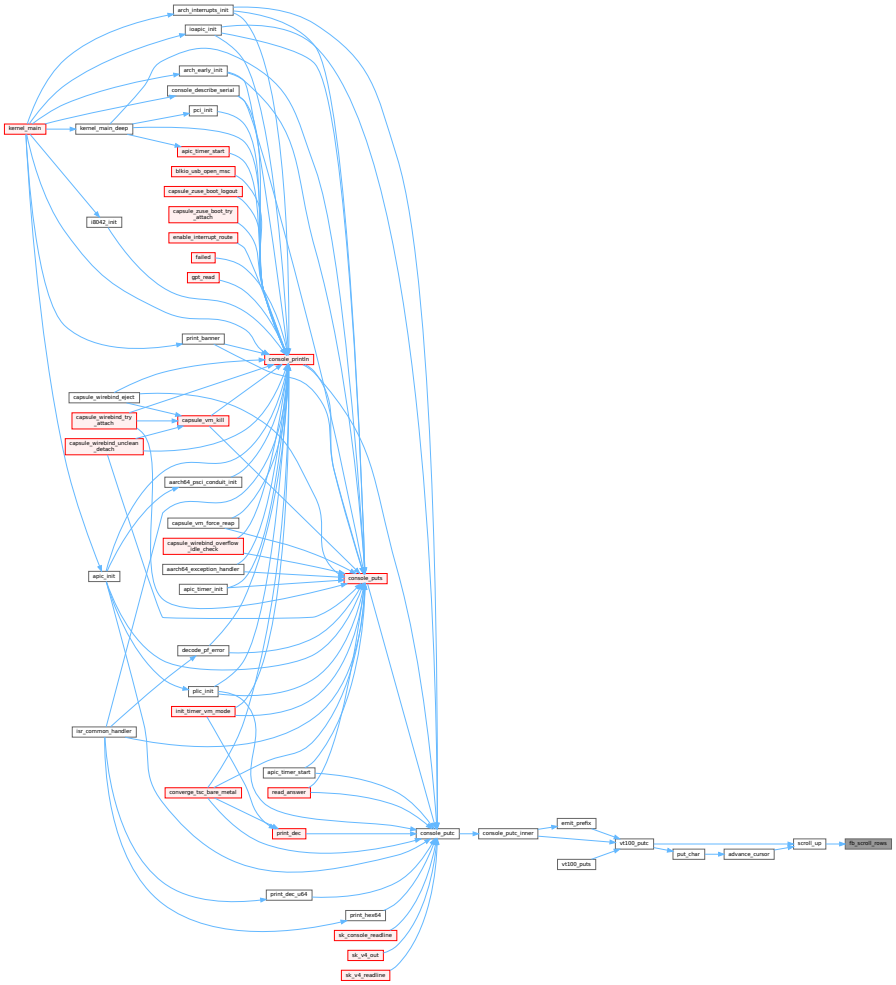
References FbState::base, fb_fill_rect(), g_fb, FbState::height, pack_pixel(), FbState::ready, FbState::stride, and FbState::width.

Referenced by scroll_up().

Here is the call graph for this function:



Here is the caller graph for this function:



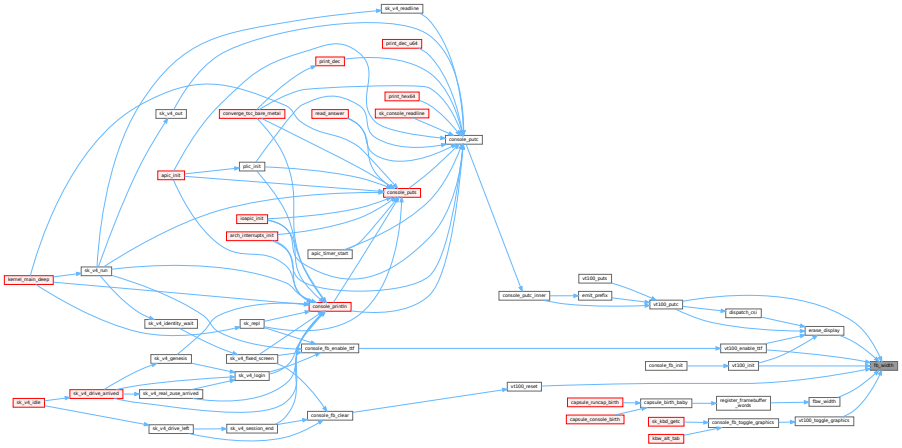
7.152.1.13 fb_width()

```
uint32_t fb_width (
            void  )
```

References g_fb, and FbState::width.

Referenced by erase_display(), fbw_width(), vt100_enable_ttf(), vt100_init(), vt100_putc(), vt100_reset(), and vt100_toggle_graphics().

Here is the caller graph for this function:



7.152.1.14 font_8x16_glyph()

```
const uint8_t * font_8x16_glyph (
            uint8_t ch )  [extern]
```

framebuffer.c — UEFI GOP framebuffer pixel driver

Supports BGRX32 and RGBX32 pixel formats. Pixel scale factor allows 1×/2×/4× rendering for hi-res displays.

Hi-res mode: 1080p (1920×1080) @ scale=1 → 240 cols × 67 rows (8×16 cells) 1080p (1920×1080) @ scale=2 → 120 cols × 33 rows (16×32 cells) 4K (3840×2160) @ scale=2 → 240 cols × 67 rows (16×32 cells) 4K (3840×2160) @ scale=4 → 120 cols × 33 rows (32×64 cells)

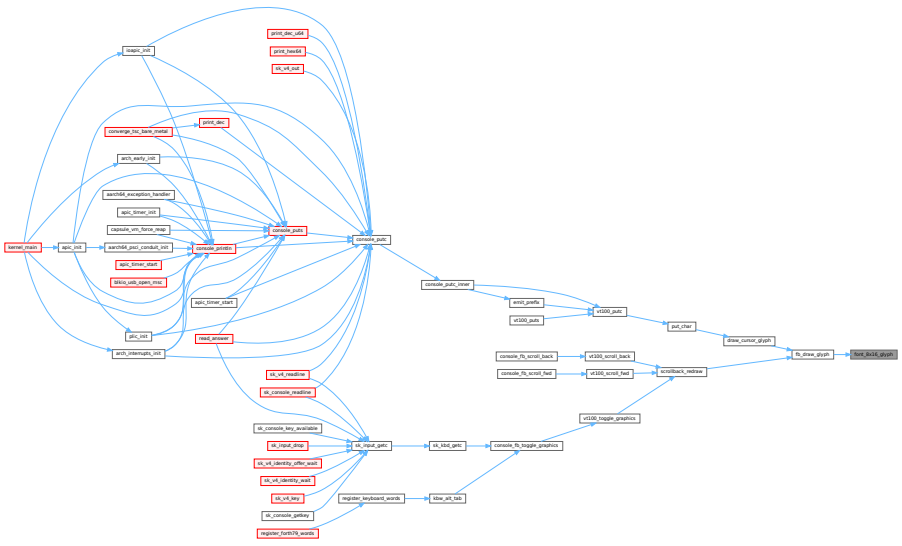
framebuffer.c — UEFI GOP framebuffer pixel driver

Caller must not modify the returned data.

References font_8x16_data.

Referenced by fb_draw_glyph().

Here is the caller graph for this function:



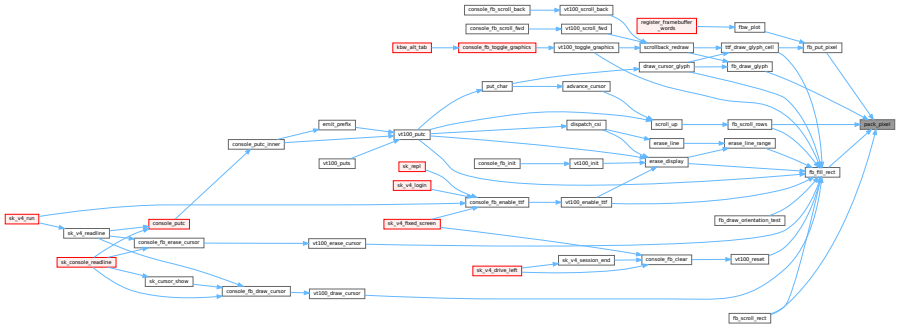
7.152.1.15 pack_pixel()

```
static uint32_t pack_pixel (
            uint32_t rgb )  [static]
```

References FB_B, FB_G, FB_PIXEL_RGBX32, FB_R, FbState::fmt, g, and g_fb.

Referenced by fb_draw_glyph(), fb_fill_rect(), fb_put_pixel(), fb_scroll_rect(), and fb_scroll_rows().

Here is the caller graph for this function:



7.152.2 Variable Documentation

7.152.2.1 g_fb

```
FbState g_fb  [static]
```

Referenced by fb_cell_h(), fb_cell_w(), fb_draw_glyph(), fb_draw_orientation_test(), fb_fill_rect(), fb_height(), fb_init(), fb_is_available(), fb_put_pixel(), fb_scroll_rect(), fb_scroll_rows(), fb_width(), and pack_pixel().

7.153 kernel/src/hal/hal.c File Reference

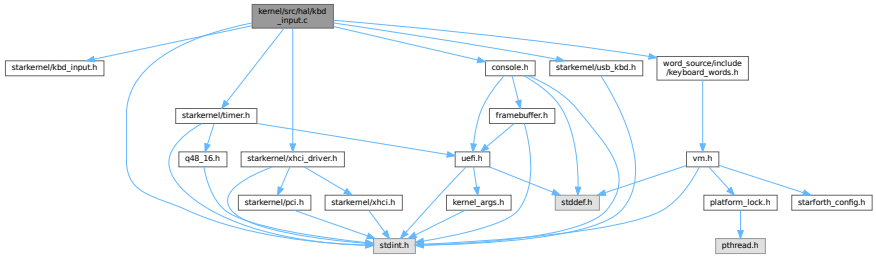
7.154 kernel/src/hal/host_services.c File Reference

7.155 kernel/src/hal/kbd_input.c File Reference

```
#include "starkernel/kbd_input.h"
#include "console.h"
#include "starkernel/timer.h"
#include "starkernel/xhci_driver.h"
#include "starkernel/usb_kbd.h"
#include "word_source/include/keyboard_words.h"
```

```
#include <stdint.h>
```

Include dependency graph for kbd_input.c:



Macros

- #define SK_KBD_TABLE_SIZE 98u /∗ highest keycode used below is KEY_RIGHTCTRL=97 ∗/
- #define SK_KEY_BACKSPACE 14u
- #define SK_KEY_TAB 15u
- #define SK_KEY_ENTER 28u
- #define SK_KEY_LEFTSHIFT 42u
- #define SK_KEY_RIGHTSHIFT 54u
- #define SK_KEY_LEFTALT 56u
- #define SK_KEY_RIGHTALT 100u
- #define SK_KEY_CAPSLOCK 58u
- #define SK_KEY_NUMLOCK 69u
- #define SK_KEY_KPENTER 96u
- #define SK_KEY_KPSLASH 98u
- #define SK_KEY_102ND 86u

Functions

- static void sk_kbd_show_locks (void)
- static int sk_kbd_keypad (uint16_t keycode)
- static int sk_kbd_getc (void)
- int sk_input_getc (void)
- void sk_input_drop (void)

Variables

- static const char sk_kbd_unshifted [98u]
- static const char sk_kbd_shifted [98u]
- static int g_kbd_shift_down
- static int g_kbd_alt_down
- static int g_kbd_caps_lock
- static int g_kbd_num_lock = 1
- static uint64_t g_kbd_poll_tick

7.155.1 Macro Definition Documentation

7.155.1.1 SK_KBD_TABLE_SIZE

```
#define SK_KBD_TABLE_SIZE 98u /* highest keycode used below is KEY_RIGHTCTRL=97 */
```

7.155.1.2 SK_KEY_102ND

```
#define SK_KEY_102ND 86u
```

7.155.1.3 SK_KEY_BACKSPACE

```
#define SK_KEY_BACKSPACE 14u
```

7.155.1.4 SK_KEY_CAPSLOCK

```
#define SK_KEY_CAPSLOCK 58u
```

7.155.1.5 SK_KEY_ENTER

```
#define SK_KEY_ENTER 28u
```

7.155.1.6 SK_KEY_KPENTER

```
#define SK_KEY_KPENTER 96u
```

7.155.1.7 SK_KEY_KPSLASH

```
#define SK_KEY_KPSLASH 98u
```

7.155.1.8 SK_KEY_LEFTALT

```
#define SK_KEY_LEFTALT 56u
```

7.155.1.9 SK_KEY_LEFTSHIFT

```
#define SK_KEY_LEFTSHIFT 42u
```

7.155.1.10 SK_KEY_NUMLOCK

```
#define SK_KEY_NUMLOCK 69u
```

7.155.1.11 SK_KEY_RIGHTALT

```
#define SK_KEY_RIGHTALT 100u
```

7.155.1.12 SK_KEY_RIGHTSHIFT

```
#define SK_KEY_RIGHTSHIFT 54u
```

7.155.1.13 SK_KEY_TAB

```
#define SK_KEY_TAB 15u
```

7.155.2 Function Documentation

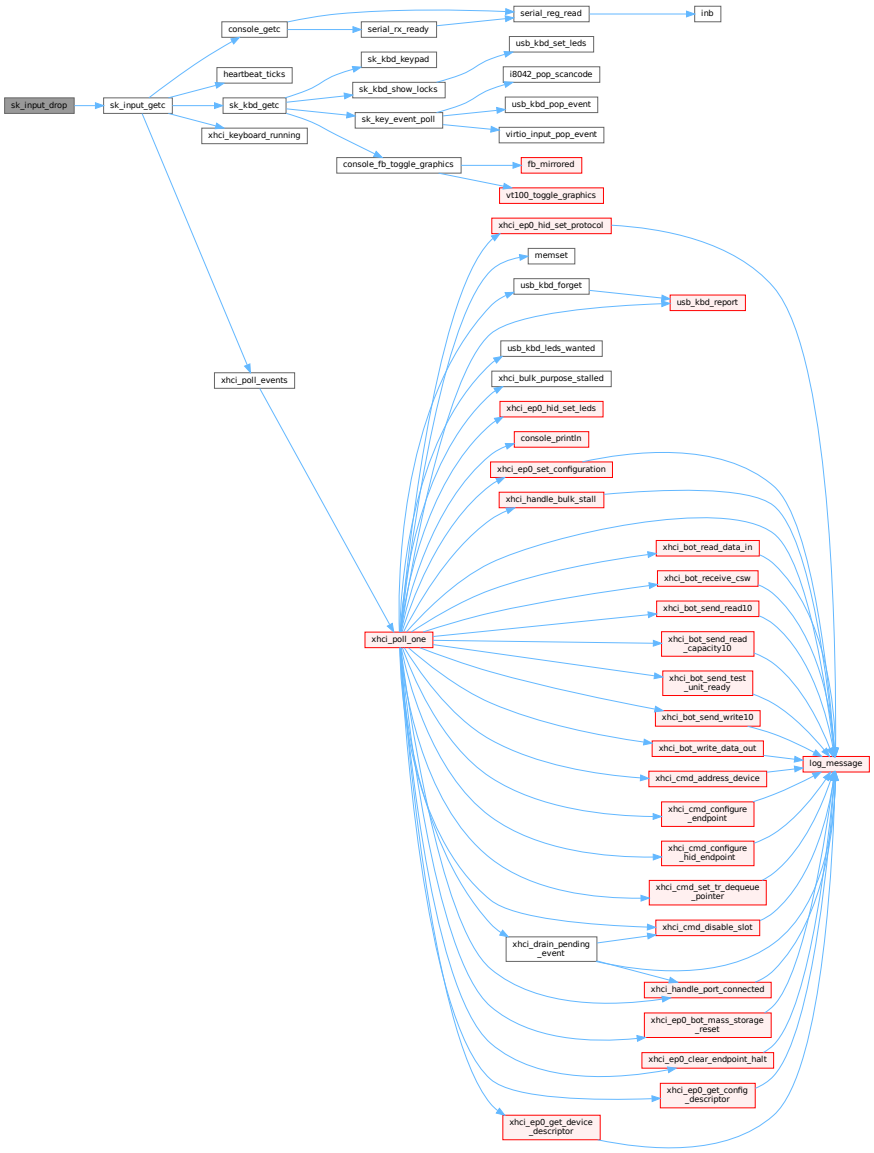
7.155.2.1 sk_input_drop()

```
void sk_input_drop (
            void  )
```

References sk_input_getc().

Referenced by sk_v4_discard_input().

Here is the call graph for this function:



Here is the caller graph for this function:

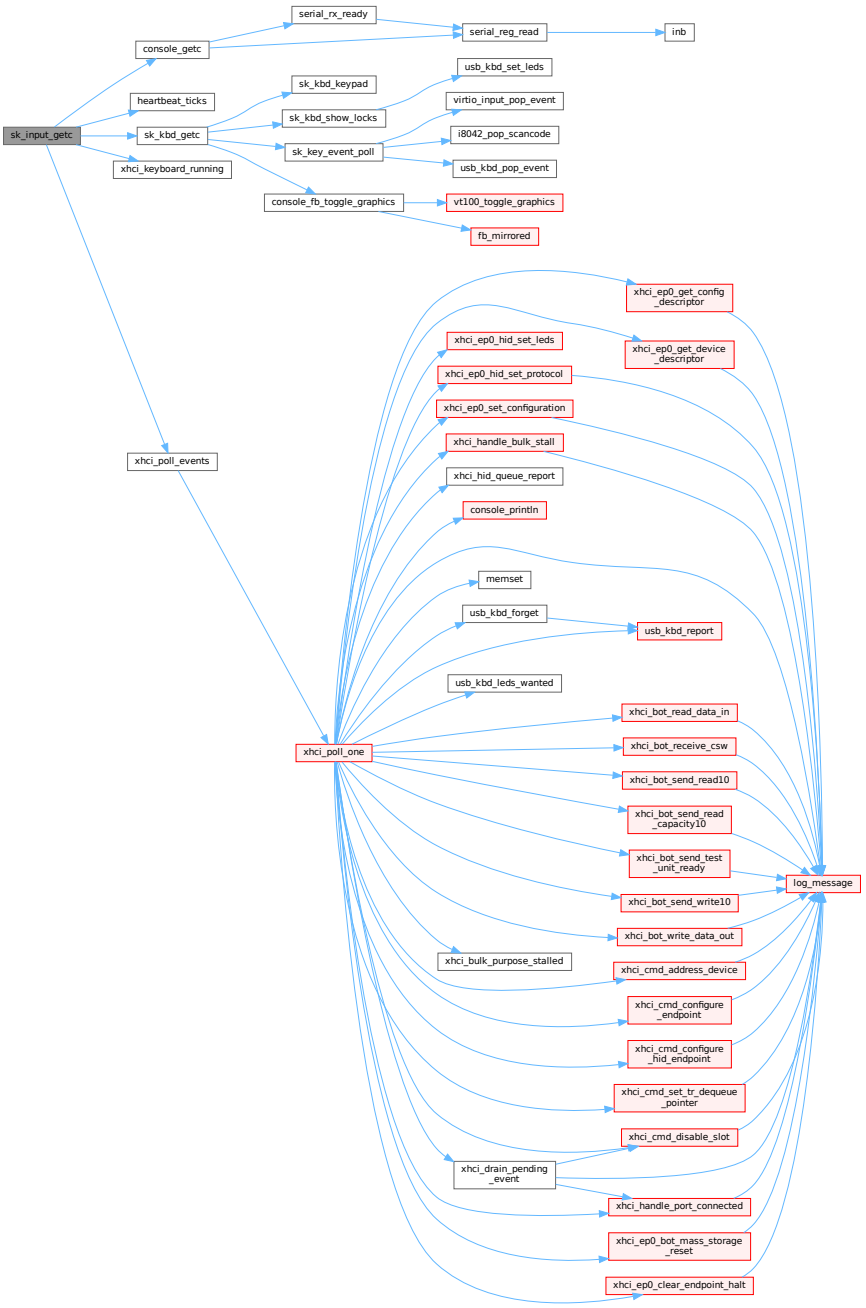
7.155.2.2 sk_input_getc()

```
int sk_input_getc (
            void  )
```

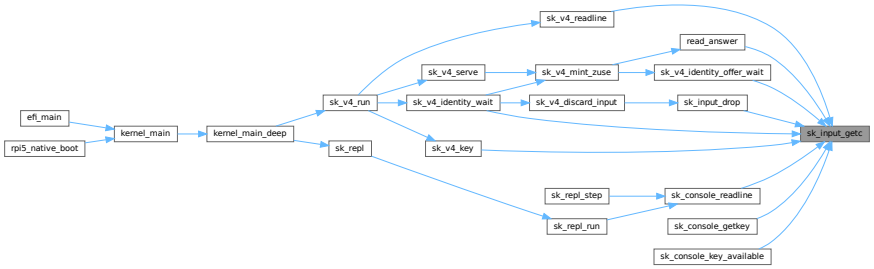
References console_getc(), g_kbd_poll_tick, heartbeat_ticks(), sk_kbd_getc(), xhci_keyboard_running(), and xhci_poll_events().

Referenced by read_answer(), sk_console_getkey(), sk_console_key_available(), sk_console_readline(), sk_input_drop(), sk_v4_identity_offer_wait(), sk_v4_identity_wait(), sk_v4_key(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:



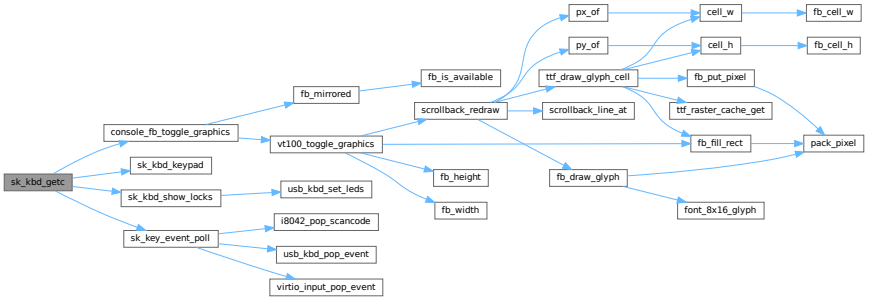
7.155.2.3 sk_kbd_getc()

```
static int sk_kbd_getc (
            void  )  [static]
```

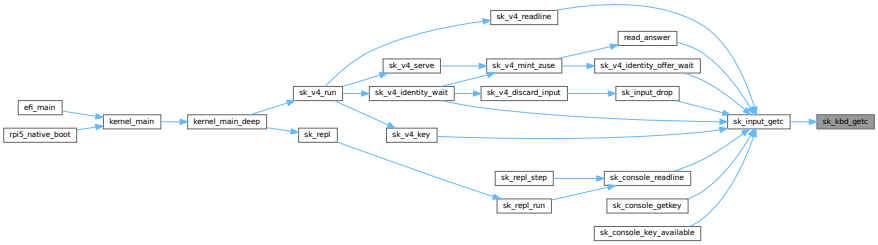
References console_fb_toggle_graphics(), g_kbd_alt_down, g_kbd_caps_lock, g_kbd_num_lock, g_kbd_shift_down, keycode, pressed, sk_kbd_keypad(), sk_kbd_shifted, sk_kbd_show_locks(), SK_KBD_TABLE_SIZE, sk_kbd_unshifted, SK_KEY_102ND, SK_KEY_BACKSPACE, SK_KEY_CAPSLOCK, SK_KEY_ENTER, sk_key_event_poll(), SK_KEY_LEFTALT, SK_KEY_LEFTSHIFT, SK_KEY_NUMLOCK, SK_KEY_RIGHTALT, SK_KEY_RIGHTSHIFT, and SK_KEY_TAB.

Referenced by sk_input_getc().

Here is the call graph for this function:



Here is the caller graph for this function:



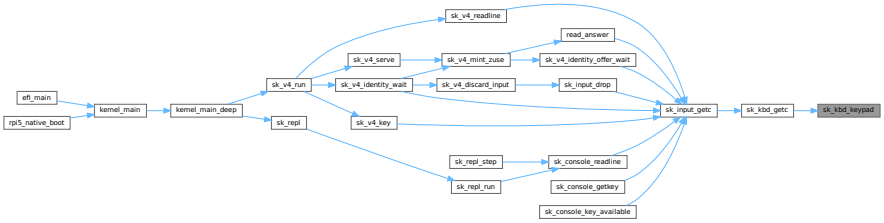
7.155.2.4 sk_kbd_keypad()

```
static int sk_kbd_keypad (
            uint16_t keycode )  [static]
```

References g_kbd_num_lock, keycode, SK_KEY_KPENTER, and SK_KEY_KPSLASH.

Referenced by sk_kbd_getc().

Here is the caller graph for this function:



7.155.2.5 sk_kbd_show_locks()

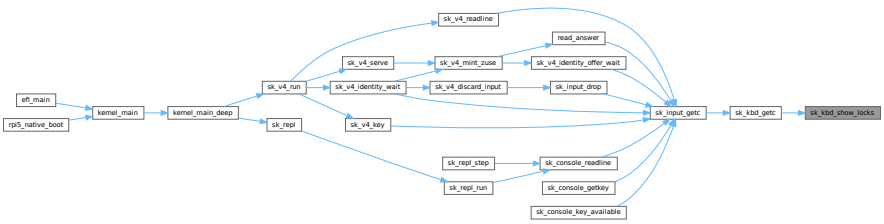
```
static void sk_kbd_show_locks (
            void  )  [static]
```

References g_kbd_caps_lock, g_kbd_num_lock, USB_KBD_LED_CAPS, USB_KBD_LED_NUM, and usb_kbd_set_leds().

Referenced by sk_kbd_getc().

Here is the call graph for this function:

Here is the caller graph for this function:



7.155.3 Variable Documentation

7.155.3.1 g_kbd_alt_down

```
int g_kbd_alt_down  [static]
```

Referenced by sk_kbd_getc().

7.155.3.2 g_kbd_caps_lock

```
int g_kbd_caps_lock  [static]
```

Referenced by sk_kbd_getc(), and sk_kbd_show_locks().

7.155.3.3 g_kbd_num_lock

```
int g_kbd_num_lock = 1  [static]
```

Referenced by sk_kbd_getc(), sk_kbd_keypad(), and sk_kbd_show_locks().

7.155.3.4 g_kbd_poll_tick

```
uint64_t g_kbd_poll_tick  [static]
```

Referenced by sk_input_getc().

7.155.3.5 g_kbd_shift_down

```
int g_kbd_shift_down  [static]
```

Referenced by sk_kbd_getc().

7.155.3.6 sk_kbd_shifted

```
const char sk_kbd_shifted[98u]  [static]
```

Initial value:

```
= {
    [2]='!',[3]='@',[4]='#',[5]='$',[6]='%',[7]='^',[8]='&',[9]='*',[10]='(',[11]=')',
    [12]='_',[13]='+',
    [16]='Q',[17]='W',[18]='E',[19]='R',[20]='T',[21]='Y',[22]='U',[23]='I',[24]='O',[25]='P',
    [26]='{',[27]='}',
    [30]='A',[31]='S',[32]='D',[33]='F',[34]='G',[35]='H',[36]='J',[37]='K',[38]='L',
    [39]=':',[40]='"',[41]='~',[43]='|',
    [44]='Z',[45]='X',[46]='C',[47]='V',[48]='B',[49]='N',[50]='M',
    [51]='<',[52]='>',[53]='?',
    [57]=' ',
}
```

Referenced by sk_kbd_getc().

7.155.3.7 sk_kbd_unshifted

```
const char sk_kbd_unshifted[98u]  [static]
```

Initial value:

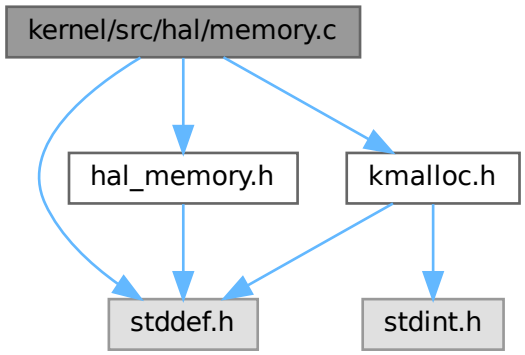
```
= {
    [2]='1',[3]='2',[4]='3',[5]='4',[6]='5',[7]='6',[8]='7',[9]='8',[10]='9',[11]='0',
    [12]='-',[13]='=',
    [16]='q',[17]='w',[18]='e',[19]='r',[20]='t',[21]='y',[22]='u',[23]='i',[24]='o',[25]='p',
    [26]='[',[27]=']',
    [30]='a',[31]='s',[32]='d',[33]='f',[34]='g',[35]='h',[36]='j',[37]='k',[38]='l',
    [39]=';',[40]='\"',[41]='`',[43]='\\',
    [44]='z',[45]='x',[46]='c',[47]='v',[48]='b',[49]='n',[50]='m',
    [51]=',',[52]='.',[53]='/',
    [57]=' ',
}
```

Referenced by sk_kbd_getc().

7.156 kernel/src/hal/memory.c File Reference

```
#include <stddef.h>
#include "hal_memory.h"
#include "kmalloc.h"
```

Include dependency graph for memory.c:



Functions

- void * hal_mem_alloc (size_t size)

  memory.c - HAL memory wrappers
- void * hal_mem_alloc_aligned (size_t size, size_t align)
- void hal_mem_free (void *ptr)

7.156.1 Function Documentation

7.156.1.1 hal_mem_alloc()

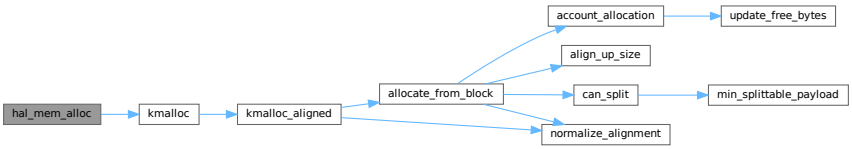
```
void * hal_mem_alloc (
            size_t size )
```

memory.c - HAL memory wrappers

hal_memory.h - HAL memory allocation wrappers

References kmalloc().

Here is the call graph for this function:

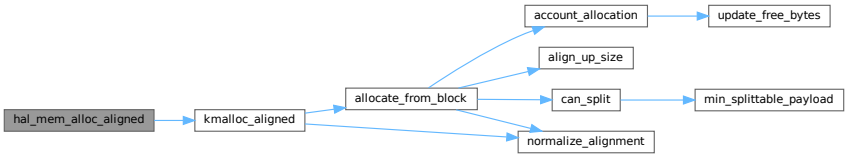


7.156.1.2 hal_mem_alloc_aligned()

```
void * hal_mem_alloc_aligned (
            size_t size,
            size_t align )
```

References kmalloc_aligned().

Here is the call graph for this function:

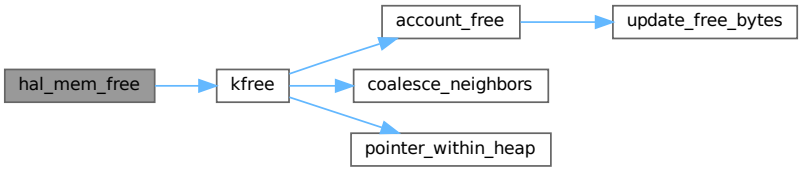


7.156.1.3 hal_mem_free()

```
void hal_mem_free (
            void * ptr )
```

References kfree().

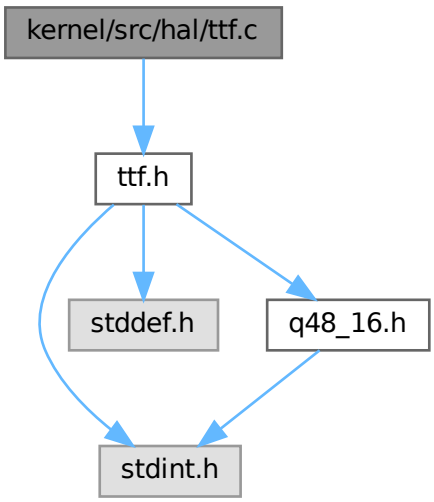
Here is the call graph for this function:



7.157 kernel/src/hal/ttf.c File Reference

```
#include "ttf.h"
```

Include dependency graph for ttf.c:



Data Structures

- struct ttf_flat_point_t

Macros

- #define TAG(a, b, c, d) (((uint32_t)(a) << 24) | ((uint32_t)(b) << 16) | ((uint32_t)(c) << 8) | (uint32_t)(d))

  ttf.c — TrueType font parser core (FABRIC-0.md items 4.3.7, 4.3.7a)
- #define TTF_FLAG_ON_CURVE 0x01
- #define TTF_FLAG_X_SHORT 0x02
- #define TTF_FLAG_Y_SHORT 0x04
- #define TTF_FLAG_REPEAT 0x08
- #define TTF_FLAG_X_SAME_OR_POS 0x10
- #define TTF_FLAG_Y_SAME_OR_POS 0x20
- #define TTF_MAX_SIMPLE_POINTS 512
- #define TTF_COMP_ARGS_ARE_WORDS 0x0001
- #define TTF_COMP_ARGS_ARE_XY_VALUES 0x0002
- #define TTF_COMP_WE_HAVE_A_SCALE 0x0008
- #define TTF_COMP_MORE_COMPONENTS 0x0020
- #define TTF_COMP_WE_HAVE_XY_SCALE 0x0040
- #define TTF_COMP_WE_HAVE_2X2 0x0080
- #define TTF_F2DOT14_ONE 16384 /* 1.0 in F2Dot14 */
- #define TTF_QUAD_SEGMENTS
- #define TTF_RASTER_RAW_MAX_POINTS 256
- #define TTF_RASTER_RAW_MAX_CONTOURS 16
- #define TTF_RASTER_FLAT_MAX_POINTS (TTF_RASTER_RAW_MAX_POINTS * TTF_QUAD_SEGMENTS)
- #define TTF_RASTER_FLAT_MAX_CONTOURS TTF_RASTER_RAW_MAX_CONTOURS

Functions

- static int fits (uint32_t size, uint32_t off, uint32_t len)
- static int in_bounds (const ttf_font_t *f, uint32_t off, uint32_t len)
- static uint16_t rd_u16 (const uint8_t *d, uint32_t off)
- static int16_t rd_i16 (const uint8_t *d, uint32_t off)
- static uint32_t rd_u32 (const uint8_t *d, uint32_t off)
- static int find_table (const uint8_t *data, uint32_t size, uint16_t num_tables, uint32_t tag, uint32_t *tbl_off, uint32_t *tbl_len)
- int ttf_parse (const uint8_t *data, uint32_t size, ttf_font_t *out)

  ttf_parse - Locate and validate the sfnt directory and the head/maxp/ loca/glyf tables (cmap is optional; ttf_codepoint_to_glyph() fails cleanly if absent).
- uint32_t ttf_codepoint_to_glyph (const ttf_font_t *font, uint32_t codepoint)

  ttf_codepoint_to_glyph - Resolve a Unicode codepoint via the font's format-4 cmap subtable.
- int ttf_glyph_header (const ttf_font_t *font, uint32_t glyph_index, ttf_glyph_header_t *out)

  ttf_glyph_header - Read a glyph's outline header (contour count, bounding box) via loca + glyf.
- uint16_t ttf_glyph_advance_width (const ttf_font_t *font, uint32_t glyph_index)

  ttf_glyph_advance_width - Horizontal advance width via hmtx, in raw font design units (NOT scaled by unitsPerEm – caller's job, e.g.
- static uint8_t rd_u8 (const uint8_t *d, uint32_t off)
- static int8_t rd_i8 (const uint8_t *d, uint32_t off)
- static q48_16_t q48_from_i32 (int32_t v)
- static int decode_simple_glyph (const ttf_font_t *font, const ttf_glyph_header_t *hdr, ttf_outline_t *out)
- static int decode_glyph_r (const ttf_font_t *font, uint32_t glyph_index, ttf_outline_t *out, int depth)
- static int decode_composite_glyph (const ttf_font_t *font, const ttf_glyph_header_t *hdr, ttf_outline_t *out, int depth)
- int ttf_glyph_outline (const ttf_font_t *font, uint32_t glyph_index, ttf_outline_t *out)

  ttf_glyph_outline - Extract a glyph's outline (simple or composite, recursively resolving composite components) into caller-supplied buffers.

- static q48_16_t q48_smul (q48_16_t a, q48_16_t b)
- static q48_16_t q48_mid (q48_16_t a, q48_16_t b)
- static void raw_midpoint (const ttf_point_t *a, const ttf_point_t *b, ttf_point_t *out)
- static void to_pixel_space (q48_16_t fx, q48_16_t fy, q48_16_t scale, q48_16_t origin_x, q48_16_t origin_y, q48_16_t *px, q48_16_t *py)
- static void flat_push (ttf_flat_point_t *pts, uint32_t *count, uint32_t max, q48_16_t x, q48_16_t y, int *err)
- static void flatten_quad (ttf_flat_point_t *pts, uint32_t *count, uint32_t max, q48_16_t p0x, q48_16_t p0y, q48_16_t p1x, q48_16_t p1y, q48_16_t p2x, q48_16_t p2y, int *err)
- static void flatten_contour (const ttf_point_t *raw, uint32_t begin, uint32_t end, q48_16_t scale, q48_16_t origin_x, q48_16_t origin_y, ttf_flat_point_t *flat_pts, uint32_t *flat_count, uint32_t flat_max, int *err)
- int ttf_rasterize_glyph (const ttf_font_t *font, uint32_t glyph_index, q48_16_t scale, q48_16_t origin_x, q48_16_t origin_y, uint8_t fill_value, ttf_bitmap_t *out)

  ttf_rasterize_glyph - Flatten a glyph's outline (quadratic Bezier contours, fixed segment count per curve — see ttf.c's file header comment) and fill it into `out` using the even-odd rule (FABRIC-0.md item 4.3.7c; see that item's completion note for why even-odd rather than nonzero winding — correct for the v1 glyph repertoire's non-self- intersecting nested contours, not necessarily for an arbitrary font).
- void ttf_raster_cache_init (ttf_raster_cache_t *cache, ttf_raster_cache_slot_t *slots, uint32_t slot_count)

  ttf_raster_cache_init - Bind a caller-supplied slot array to `cache` and mark every slot empty.
- int ttf_raster_cache_get (ttf_raster_cache_t *cache, const ttf_font_t *font, uint32_t codepoint, uint32_t size_px, ttf_bitmap_t *out, int *was_hit)

  ttf_raster_cache_get - Look up (font, codepoint, size_px); on a miss, rasterize and insert (evicting round-robin if every slot is full).

7.157.1 Macro Definition Documentation

7.157.1.1 TAG

```
#define TAG(
              a,
              b,
              c,
              d )  (((uint32_t)(a) << 24) | ((uint32_t)(b) << 16) | ((uint32_t)(c) << 8) |
(uint32_t)(d))
```

ttf.c — TrueType font parser core (FABRIC-0.md items 4.3.7, 4.3.7a)

sfnt directory + head/maxp/loca/glyf/cmap(format 4) table parsing, plus simple- and composite-glyph outline extraction. All multi-byte fields in a TTF are big-endian; this file reads them by hand (no libc byteswap dependency) with bounds checks against the buffer length on every access, since the buffer is untrusted input.

Composite-glyph transforms: this file applies (dx,dy) TRANSLATION only. A composite component carrying a non-identity scale/rotation/skew (the WE_HAVE_A_SCALE/WE_HAVE_AN_X_AND_Y_SCALE/WE_HAVE_A_TWO_BY_TWO flags with a non-1.0/non-zero F2Dot14 value) is rejected with TTF_ERR_UNSUPPORTED rather than silently mis-rendered. Reason: applying such a transform needs a signed fixed-point multiply, and the shared q48_mul()/q48_div() in src/starkernel/math/q48_16.c are unsigned-only (q48_div saturates via an unsigned overflow check; q48_mul does a plain unsigned widen-multiply) — confirmed by reading that file, not assumed. Per this repo's rule against modifying a shared/tested module to "fix" it without being asked, that gap is reported (see FABRIC-0.md's 4.3.7a completion note), not patched here. No glyph in the v1 repertoire (?27.6.4) needs a non-identity composite transform — checked against fonts/JetBrainsMono-Regular.ttf before writing this, not assumed.

7.157.1.2 TTF_COMP_ARGS_ARE_WORDS

```
#define TTF_COMP_ARGS_ARE_WORDS 0x0001
```

7.157.1.3 TTF_COMP_ARGS_ARE_XY_VALUES

```
#define TTF_COMP_ARGS_ARE_XY_VALUES 0x0002
```

7.157.1.4 TTF_COMP_MORE_COMPONENTS

```
#define TTF_COMP_MORE_COMPONENTS 0x0020
```

7.157.1.5 TTF_COMP_WE_HAVE_2X2

```
#define TTF_COMP_WE_HAVE_2X2 0x0080
```

7.157.1.6 TTF_COMP_WE_HAVE_A_SCALE

```
#define TTF_COMP_WE_HAVE_A_SCALE 0x0008
```

7.157.1.7 TTF_COMP_WE_HAVE_XY_SCALE

```
#define TTF_COMP_WE_HAVE_XY_SCALE 0x0040
```

7.157.1.8 TTF_F2DOT14_ONE

```
#define TTF_F2DOT14_ONE 16384 /* 1.0 in F2Dot14 */
```

7.157.1.9 TTF_FLAG_ON_CURVE

```
#define TTF_FLAG_ON_CURVE 0x01
```

7.157.1.10 TTF_FLAG_REPEAT

```
#define TTF_FLAG_REPEAT 0x08
```

7.157.1.11 TTF_FLAG_X_SAME_OR_POS

```
#define TTF_FLAG_X_SAME_OR_POS 0x10
```

7.157.1.12 TTF_FLAG_X_SHORT

```
#define TTF_FLAG_X_SHORT 0x02
```

7.157.1.13 TTF_FLAG_Y_SAME_OR_POS

```
#define TTF_FLAG_Y_SAME_OR_POS 0x20
```

7.157.1.14 TTF_FLAG_Y_SHORT

```
#define TTF_FLAG_Y_SHORT 0x04
```

7.157.1.15 TTF_MAX_SIMPLE_POINTS

```
#define TTF_MAX_SIMPLE_POINTS 512
```

7.157.1.16 TTF_QUAD_SEGMENTS

```
#define TTF_QUAD_SEGMENTS
```

Value:

```
8 /* fixed subdivision count per quadratic curve --
* same fixed-segment-count approach as
* capsules/fabric.4th's CIRCLE/ELLIPSE (36 segs)
* and ARC (18 segs); no adaptive tessellation. */
```

7.157.1.17 TTF_RASTER_FLAT_MAX_CONTOURS

```
#define TTF_RASTER_FLAT_MAX_CONTOURS TTF_RASTER_RAW_MAX_CONTOURS
```

7.157.1.18 TTF_RASTER_FLAT_MAX_POINTS

```
#define TTF_RASTER_FLAT_MAX_POINTS (TTF_RASTER_RAW_MAX_POINTS * TTF_QUAD_SEGMENTS)
```

7.157.1.19 TTF_RASTER_RAW_MAX_CONTOURS

```
#define TTF_RASTER_RAW_MAX_CONTOURS 16
```

7.157.1.20 TTF_RASTER_RAW_MAX_POINTS

```
#define TTF_RASTER_RAW_MAX_POINTS 256
```

7.157.2 Function Documentation

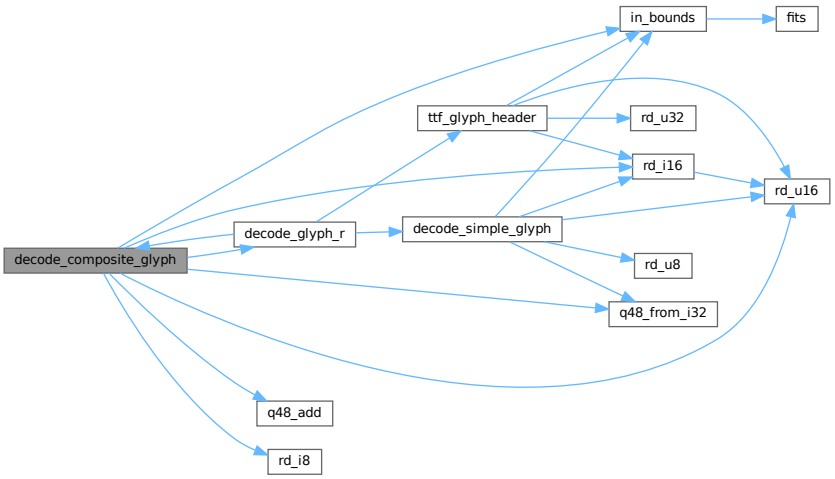
7.157.2.1 decode_composite_glyph()

```
static int decode_composite_glyph (
            const ttf_font_t * font,
            const ttf_glyph_header_t * hdr,
            ttf_outline_t * out,
            int depth )  [static]
```

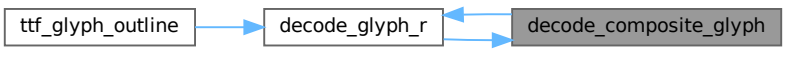
References ttf_font_t::data, decode_glyph_r(), flags, ttf_glyph_header_t::glyf_offset, in_bounds(), ttf_outline_t::point_count, ttf_outline_t::points, q48_add(), q48_from_i32(), rd_i16(), rd_i8(), rd_u16(), TTF_COMP_ARGS_ARE_WORDS, TTF_COMP_ARGS_ARE_XY_VALUES, TTF_COMP_MORE_COMPONENTS, TTF_COMP_WE_HAVE_2X2, TTF_COMP_WE_HAVE_A_SCALE, TTF_COMP_WE_HAVE_XY_SCALE, TTF_ERR_OUT_OF_BOUNDS, TTF_ERR_UNSUPPORTED, TTF_F2DOT14_ONE, TTF_OK, ttf_point_t::x, and ttf_point_t::y.

Referenced by decode_glyph_r().

Here is the call graph for this function:



Here is the caller graph for this function:



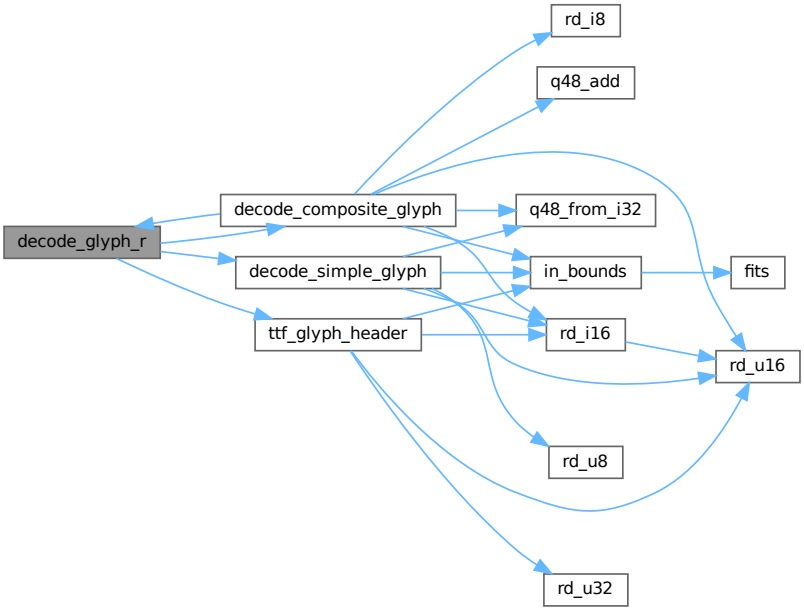
7.157.2.2 decode_glyph_r()

```
static int decode_glyph_r (
            const ttf_font_t * font,
            uint32_t glyph_index,
            ttf_outline_t * out,
            int depth )  [static]
```

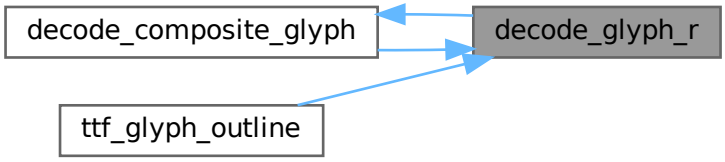
References decode_composite_glyph(), decode_simple_glyph(), ttf_glyph_header_t::glyf_length, ttf_glyph_header_t::num_contours, TTF_ERR_TOO_DEEP, ttf_glyph_header(), TTF_MAX_COMPOSITE_DEPTH, and TTF_OK.

Referenced by decode_composite_glyph(), and ttf_glyph_outline().

Here is the call graph for this function:



Here is the caller graph for this function:



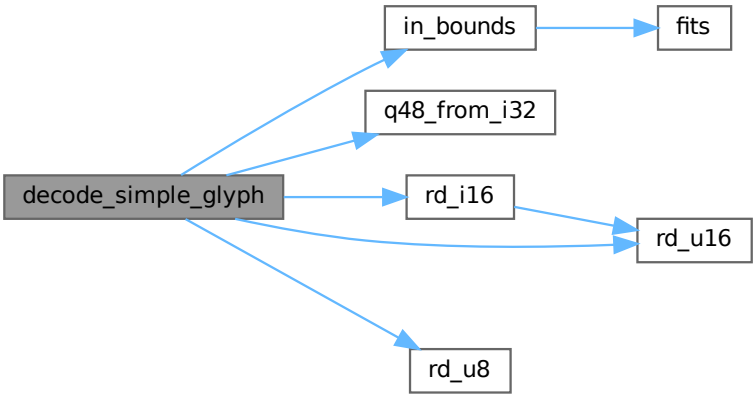
7.157.2.3 decode_simple_glyph()

```
static int decode_simple_glyph (
            const ttf_font_t * font,
            const ttf_glyph_header_t * hdr,
            ttf_outline_t * out )  [static]
```

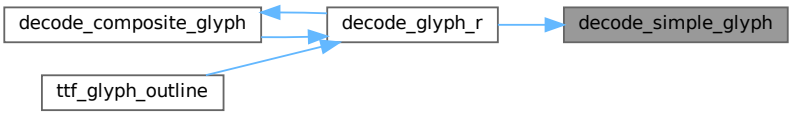
References ttf_outline_t::contour_count, ttf_outline_t::contour_ends, ttf_font_t::data, flags, ttf_glyph_header_t::glyf_offset, in_bounds(), ttf_outline_t::max_contours, ttf_outline_t::max_points, ttf_glyph_header_t::num_contours, ttf_point_t::on_curve, ttf_outline_t::point_count, ttf_outline_t::points, q48_from_i32(), rd_i16(), rd_u16(), rd_u8(), TTF_ERR_OUT_OF_BOUNDS, TTF_ERR_TOO_MANY_CONTOURS, TTF_ERR_TOO_MANY_POINTS, TTF_FLAG_ON_CURVE, TTF_FLAG_REPEAT, TTF_FLAG_X_SAME_OR_POS, TTF_FLAG_X_SHORT, TTF_FLAG_Y_SAME_OR_POS, TTF_FLAG_Y_SHORT, TTF_MAX_SIMPLE_POINTS, TTF_OK, ttf_point_t::x, and ttf_point_t::y.

Referenced by decode_glyph_r().

Here is the call graph for this function:



Here is the caller graph for this function:



7.157.2.4 find_table()

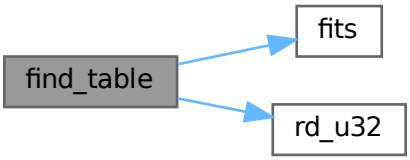
```
static int find_table (
            const uint8_t * data,
```

```
            uint32_t size,
            uint16_t num_tables,
            uint32_t tag,
            uint32_t * tbl_off,
            uint32_t * tbl_len )  [static]
```

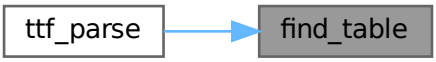
References fits(), rd_u32(), TTF_ERR_OUT_OF_BOUNDS, TTF_ERR_TABLE_MISSING, and TTF_OK.

Referenced by ttf_parse().

Here is the call graph for this function:



Here is the caller graph for this function:

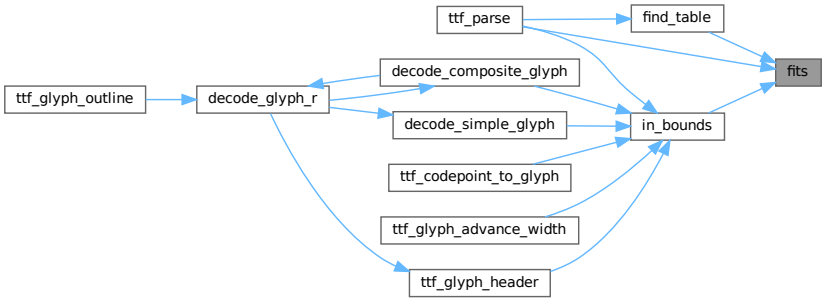


7.157.2.5 fits()

```
static int fits (
            uint32_t size,
            uint32_t off,
            uint32_t len )  [static]
```

Referenced by find_table(), in_bounds(), and ttf_parse().

Here is the caller graph for this function:



7.157.2.6 flat_push()

```
static void flat_push (
            ttf_flat_point_t * pts,
            uint32_t * count,
            uint32_t max,
            q48_16_t x,
            q48_16_t y,
            int * err )  [static]
```

Referenced by flatten_contour(), and flatten_quad().

Here is the caller graph for this function:

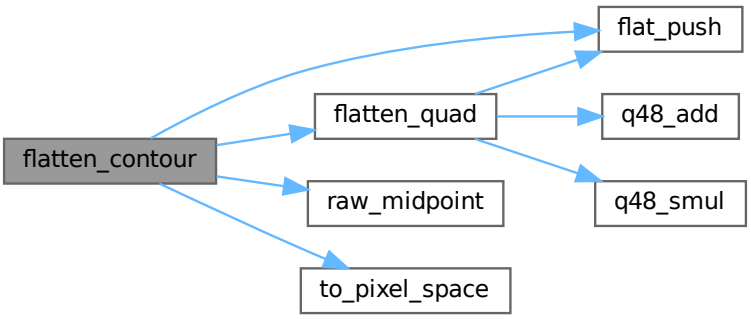
7.157.2.7 flatten_contour()

```
static void flatten_contour (
            const ttf_point_t * raw,
            uint32_t begin,
            uint32_t end,
            q48_16_t scale,
            q48_16_t origin_x,
            q48_16_t origin_y,
            ttf_flat_point_t * flat_pts,
            uint32_t * flat_count,
```

```
            uint32_t flat_max,
            int * err )  [static]
```

References flat_push(), flatten_quad(), ttf_point_t::on_curve, raw_midpoint(), to_pixel_space(), TTF_OK, ttf_point_t::x, and ttf_point_t::y.

Here is the call graph for this function:



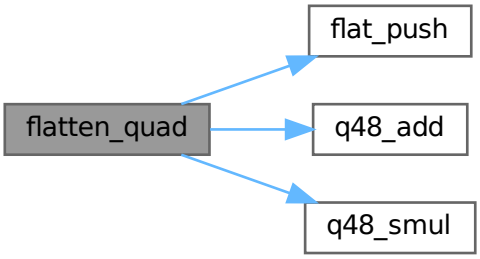
7.157.2.8 flatten_quad()

```
static void flatten_quad (
            ttf_flat_point_t * pts,
            uint32_t * count,
            uint32_t max,
            q48_16_t p0x,
            q48_16_t p0y,
            q48_16_t p1x,
            q48_16_t p1y,
            q48_16_t p2x,
            q48_16_t p2y,
            int * err )  [static]
```

References count, flat_push(), q48_add(), Q48_ONE, q48_smul(), and TTF_QUAD_SEGMENTS.

Referenced by flatten_contour().

Here is the call graph for this function:



Here is the caller graph for this function:

7.157.2.9 in_bounds()

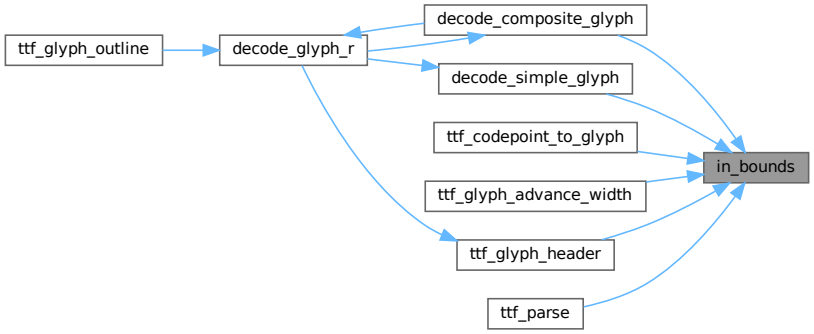
```
static int in_bounds (
            const ttf_font_t * f,
            uint32_t off,
            uint32_t len )  [static]
```

References fits(), and ttf_font_t::size.

Referenced by decode_composite_glyph(), decode_simple_glyph(), ttf_codepoint_to_glyph(), ttf_glyph_advance_width(), ttf_glyph_header(), and ttf_parse().

Here is the call graph for this function:

Here is the caller graph for this function:

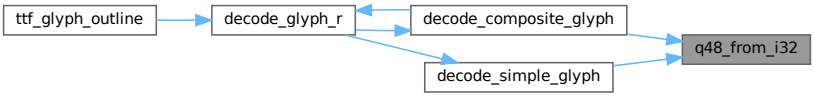


7.157.2.10 q48_from_i32()

```
static q48_16_t q48_from_i32 (
            int32_t v )  [static]
```

Referenced by decode_composite_glyph(), and decode_simple_glyph().

Here is the caller graph for this function:



7.157.2.11 q48_mid()

```
static q48_16_t q48_mid (
            q48_16_t a,
            q48_16_t b )  [static]
```

7.157.2.12 q48_smul()

```
static q48_16_t q48_smul (
            q48_16_t a,
            q48_16_t b )  [static]
```

Referenced by flatten_quad().

Here is the caller graph for this function:

7.157.2.13 raw_midpoint()

```
static void raw_midpoint (
            const ttf_point_t * a,
            const ttf_point_t * b,
            ttf_point_t * out )  [static]
```

Referenced by flatten_contour().

Here is the caller graph for this function:

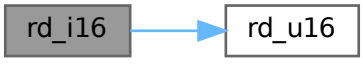
7.157.2.14 rd_i16()

```
static int16_t rd_i16 (
            const uint8_t * d,
            uint32_t off )  [static]
```

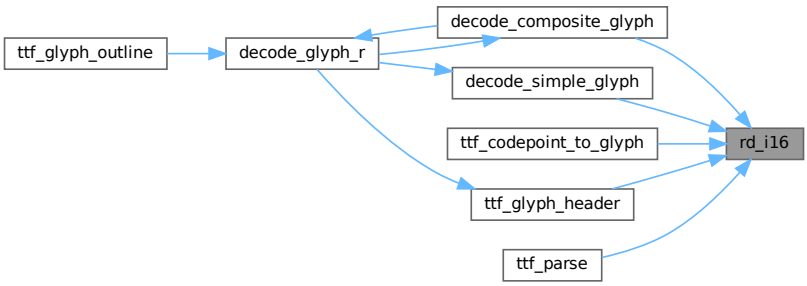
References rd_u16().

Referenced by decode_composite_glyph(), decode_simple_glyph(), ttf_codepoint_to_glyph(), ttf_glyph_header(), and ttf_parse().

Here is the call graph for this function:



Here is the caller graph for this function:



7.157.2.15 rd_i8()

```
static int8_t rd_i8 (
            const uint8_t * d,
            uint32_t off )  [static]
```

Referenced by decode_composite_glyph().

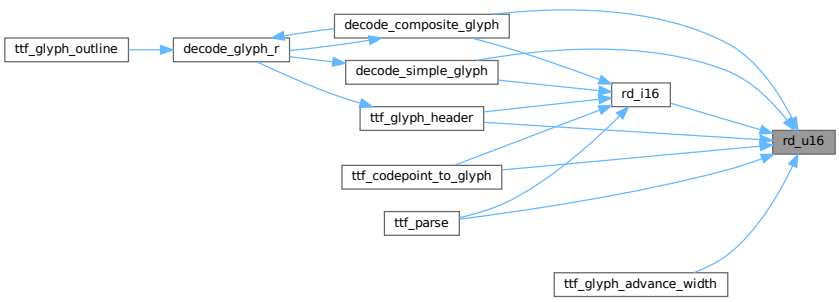
Here is the caller graph for this function:

7.157.2.16 rd_u16()

```
static uint16_t rd_u16 (
            const uint8_t * d,
            uint32_t off )  [static]
```

Referenced by decode_composite_glyph(), decode_simple_glyph(), rd_i16(), ttf_codepoint_to_glyph(), ttf_glyph_advance_width(), ttf_glyph_header(), and ttf_parse().

Here is the caller graph for this function:

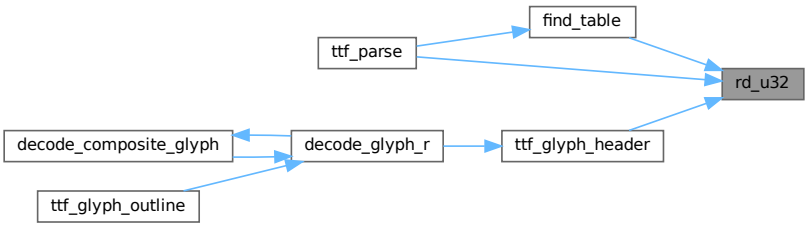


7.157.2.17 rd_u32()

```
static uint32_t rd_u32 (
            const uint8_t * d,
            uint32_t off )  [static]
```

Referenced by find_table(), ttf_glyph_header(), and ttf_parse().

Here is the caller graph for this function:

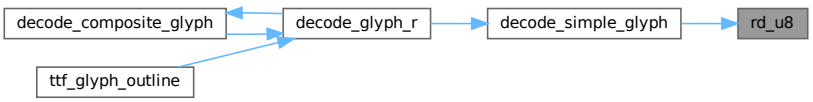


7.157.2.18 rd_u8()

```
static uint8_t rd_u8 (
            const uint8_t * d,
            uint32_t off )  [static]
```

Referenced by decode_simple_glyph().

Here is the caller graph for this function:

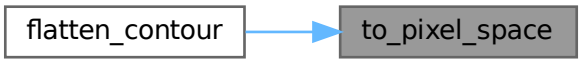


7.157.2.19 to_pixel_space()

```
static void to_pixel_space (
            q48_16_t fx,
            q48_16_t fy,
            q48_16_t scale,
            q48_16_t origin_x,
            q48_16_t origin_y,
            q48_16_t * px,
            q48_16_t * py )  [static]
```

Referenced by flatten_contour().

Here is the caller graph for this function:



7.157.2.20 ttf_codepoint_to_glyph()

```
uint32_t ttf_codepoint_to_glyph (
            const ttf_font_t * font,
            uint32_t codepoint )
```

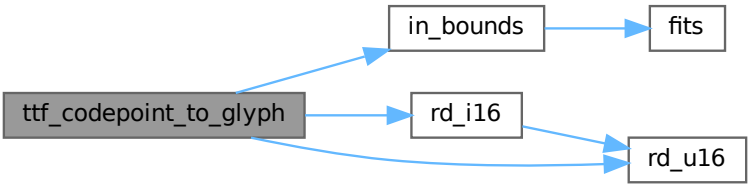
ttf_codepoint_to_glyph - Resolve a Unicode codepoint via the font's format-4 cmap subtable.

Returns

glyph index, or TTF_GLYPH_MISSING if unmapped or no cmap

References ttf_font_t::cmap_subtable_off, ttf_font_t::data, ttf_font_t::has_cmap, in_bounds(), rd_i16(), rd_u16(), and TTF_GLYPH_MISSING.

Here is the call graph for this function:



7.157.2.21 ttf_glyph_advance_width()

```
uint16_t ttf_glyph_advance_width (
            const ttf_font_t * font,
            uint32_t glyph_index )
```

ttf_glyph_advance_width - Horizontal advance width via hmtx, in raw font design units (NOT scaled by unitsPerEm – caller's job, e.g.

`q48_mul(q48_from_u64(width), scale)`; always non-negative, so the shared q48_mul is fine here unlike the signed cases documented in ttf.c's rasterizer). Glyphs past hmtx's numberOfHMetrics entries share the last entry's width, per spec.

Parameters

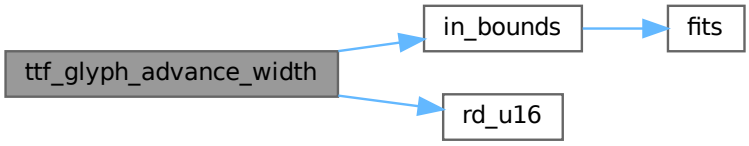
glyph_index	As returned by ttf_codepoint_to_glyph()

Returns

advance width, or 0 if glyph_index is out of range

References ttf_font_t::data, ttf_font_t::hmtx_off, in_bounds(), ttf_font_t::num_h_metrics, and rd_u16().

Here is the call graph for this function:



7.157.2.22 ttf_glyph_header()

```
int ttf_glyph_header (
            const ttf_font_t * font,
            uint32_t glyph_index,
            ttf_glyph_header_t * out )
```

ttf_glyph_header - Read a glyph's outline header (contour count, bounding box) via loca + glyf.

Does not extract contour points.

Parameters

glyph_index	As returned by ttf_codepoint_to_glyph()
out	Header to populate

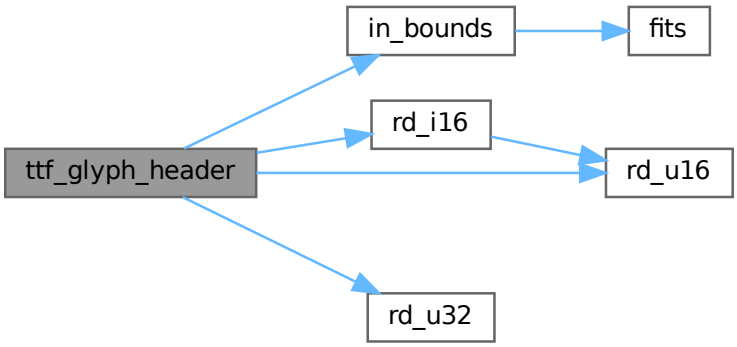
Returns

TTF_OK, or a TTF_ERR_∗ code

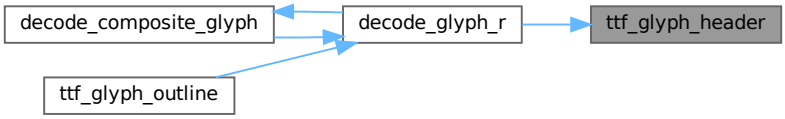
References ttf_font_t::data, ttf_glyph_header_t::glyf_length, ttf_font_t::glyf_off, ttf_glyph_header_t::glyf_offset, in_bounds(), ttf_font_t::index_to_loc_format, ttf_font_t::loca_off, ttf_glyph_header_t::num_contours, ttf_font_t::num_glyphs, rd_i16(), rd_u16(), rd_u32(), TTF_ERR_BAD_GLYPH_INDEX, TTF_ERR_BAD_TABLE, TTF_ERR_OUT_OF_BOUNDS, TTF_OK, ttf_glyph_header_t::x_max, ttf_glyph_header_t::x_min, ttf_glyph_header_t::y_max, and ttf_glyph_header_t::y_min.

Referenced by decode_glyph_r().

Here is the call graph for this function:



Here is the caller graph for this function:



7.157.2.23 ttf_glyph_outline()

```
int ttf_glyph_outline (
            const ttf_font_t * font,
            uint32_t glyph_index,
            ttf_outline_t * out )
```

ttf_glyph_outline - Extract a glyph's outline (simple or composite, recursively resolving composite components) into caller-supplied buffers.

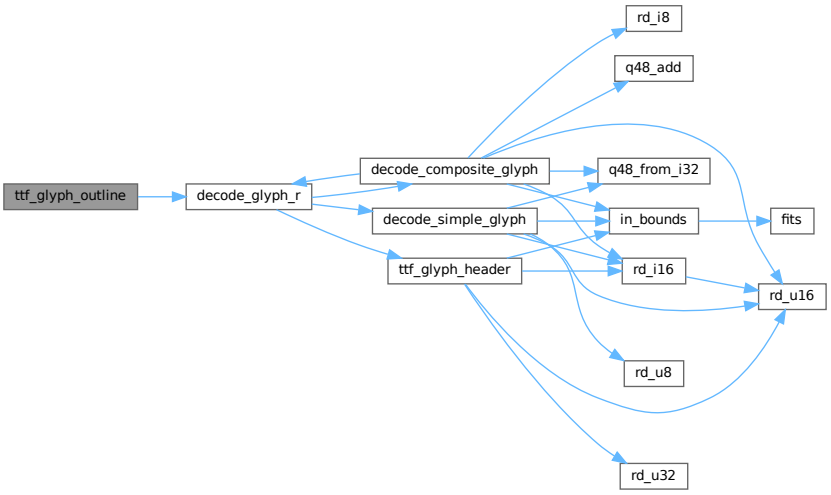
Composite components with a non-identity transform (any scale/rotation/ skew, i.e. anything but a pure (dx,dy) translation) or with point-matched (rather than xy-offset) placement args return TTF_ERR_UNSUPPORTED rather than silently producing a wrong outline — see ttf.c's file header comment for why, and check that limitation before relying on this for an arbitrary font.

Returns

TTF_OK, or a TTF_ERR_∗ code (including TTF_ERR_TOO_MANY_POINTS/ _CONTOURS if a caller buffer is too small, and TTF_ERR_TOO_DEEP if composite nesting exceeds TTF_MAX_COMPOSITE_DEPTH)

References ttf_outline_t::contour_count, decode_glyph_r(), ttf_outline_t::point_count, and TTF_ERR_BAD_TABLE.

Here is the call graph for this function:



7.157.2.24 ttf_parse()

```
int ttf_parse (
            const uint8_t * data,
            uint32_t size,
            ttf_font_t * out )
```

ttf_parse - Locate and validate the sfnt directory and the head/maxp/ loca/glyf tables (cmap is optional; ttf_codepoint_to_glyph() fails cleanly if absent).

Does not copy `data` — `out` borrows it.

Parameters

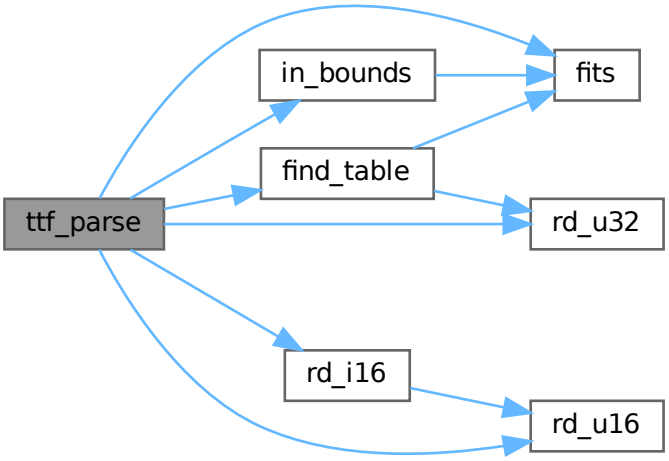
data	Whole .ttf file contents
size	Length of data in bytes
out	Parsed handle to populate

Returns

TTF_OK, or a TTF_ERR_∗ code

References ttf_font_t::cmap_subtable_off, ttf_font_t::data, find_table(), fits(), ttf_font_t::glyf_len, ttf_font_t::glyf_off, ttf_font_t::has_cmap, ttf_font_t::head_off, ttf_font_t::hmtx_off, in_bounds(), ttf_font_t::index_to_loc_format, ttf_font_t::loca_len, ttf_font_t::loca_off, ttf_font_t::maxp_off, ttf_font_t::num_glyphs, ttf_font_t::num_h_metrics, rd_i16(), rd_u16(), rd_u32(), ttf_font_t::size, TAG, TTF_ERR_BAD_SFNT, TTF_ERR_BAD_TABLE, TTF_ERR_OUT_OF_BOUNDS, TTF_OK, and ttf_font_t::units_per_em.

Here is the call graph for this function:



7.157.2.25 ttf_raster_cache_get()

```
int ttf_raster_cache_get (
            ttf_raster_cache_t * cache,
            const ttf_font_t * font,
            uint32_t codepoint,
            uint32_t size_px,
            ttf_bitmap_t * out,
            int * was_hit )
```

ttf_raster_cache_get - Look up (font, codepoint, size_px); on a miss, rasterize and insert (evicting round-robin if every slot is full).

`out` borrows the winning slot's buffer directly – valid until that slot is evicted by a later call.

Parameters

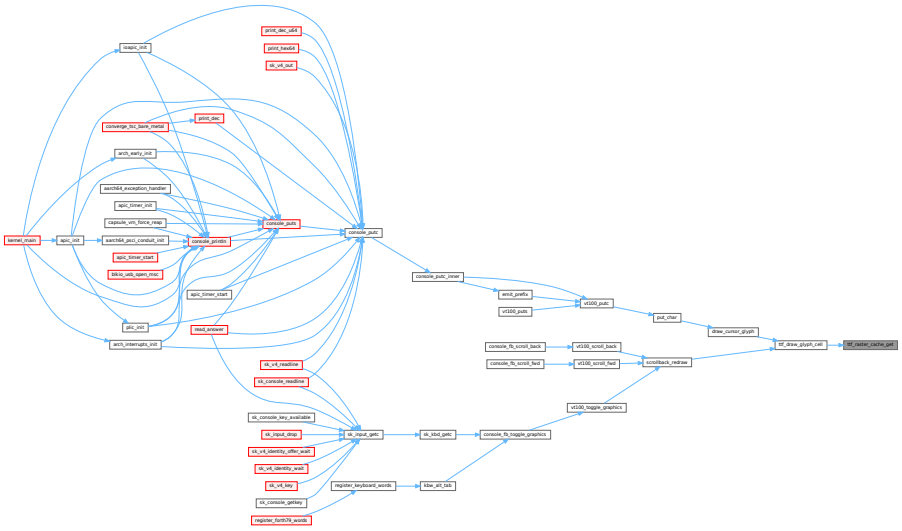
was_hit	If non-NULL, set to 1 on a cache hit, 0 if this call rasterized and inserted

Returns

TTF_OK, TTF_ERR_UNSUPPORTED if size_px > TTF_CACHE_MAX_SIZE_PX, or a ttf_rasterize_glyph() TTF_ERR_* code on a miss that failed to rasterize

Referenced by ttf_draw_glyph_cell().

Here is the caller graph for this function:



7.157.2.26 ttf_raster_cache_init()

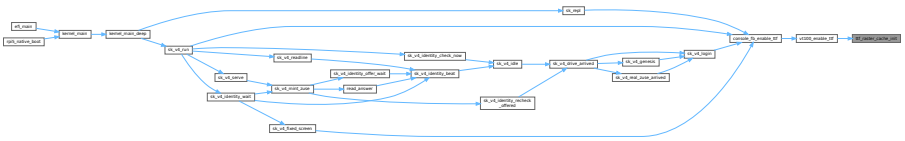
```
void ttf_raster_cache_init (
            ttf_raster_cache_t * cache,
            ttf_raster_cache_slot_t * slots,
            uint32_t slot_count )
```

ttf_raster_cache_init - Bind a caller-supplied slot array to `cache` and mark every slot empty.

References ttf_raster_cache_t::evict_next, ttf_raster_cache_t::slot_count, ttf_raster_cache_t::slots, and slots.

Referenced by vt100_enable_ttf().

Here is the caller graph for this function:



7.157.2.27 ttf_rasterize_glyph()

```
int ttf_rasterize_glyph (
            const ttf_font_t * font,
            uint32_t glyph_index,
            q48_16_t scale,
            q48_16_t origin_x,
            q48_16_t origin_y,
```

```
        uint8_t fill_value,
        ttf_bitmap_t * out )
```

ttf_rasterize_glyph - Flatten a glyph's outline (quadratic Bezier contours, fixed segment count per curve — see ttf.c's file header comment) and fill it into `out` using the even-odd rule (FABRIC-0.md item 4.3.7c; see that item's completion note for why even-odd rather than nonzero winding — correct for the v1 glyph repertoire's non-self-intersecting nested contours, not necessarily for an arbitrary font).

No antialiasing (explicitly deferred, per 4.3.7c's own "done when" clause). Never allocates — flattening uses a fixed-size local buffer bounded by TTF_RASTER_MAX_POINTS/TTF_RASTER_MAX_CONTOURS.

Parameters

scale	Font-design-units-to-pixels scale, Q48.16, e.g. q48_div(q48_from_u64(size_px), q48_from_u64(font->units_per_em))
origin_x	Pixel-space X (Q48.16) of the glyph's (0,0) font origin within `out`
origin_y	Pixel-space Y (Q48.16) of the glyph's baseline (font y=0) within `out`; font Y-up is flipped to raster Y-down internally

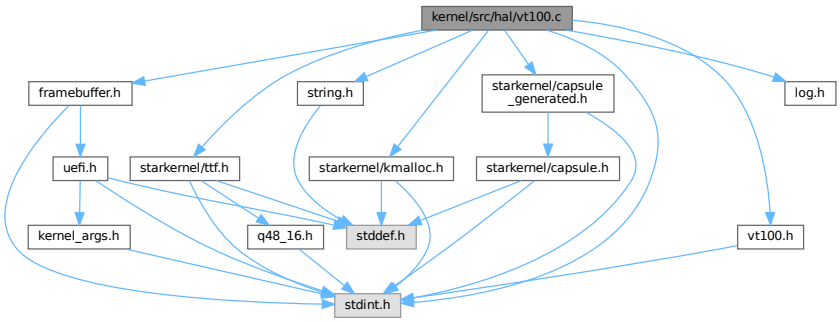
Returns

TTF_OK, or a TTF_ERR_* code (including TTF_ERR_TOO_MANY_POINTS/ _CONTOURS if the flattened outline exceeds the local buffer)

7.158 kernel/src/hal/vt100.c File Reference

```
#include "vt100.h"
#include "framebuffer.h"
#include "starkernel/ttf.h"
#include "starkernel/capsule_generated.h"
#include "starkernel/kmalloc.h"
#include "log.h"
#include <stdint.h>
#include <string.h>
```

Include dependency graph for vt100.c:



Data Structures

- struct VT100State

Macros

- #define VT100_MAX_PARAMS 16

  vt100.c — Full-color ANSI / VT100 terminal state machine
- #define UNDERLINE_ROW 14u
- #define VT100_TTF_SIZE_PX 20u
- #define VT100_TTF_CELL_W_PX 12u /∗ 600/1000 ∗ VT100_TTF_SIZE_PX ∗/
- #define VT100_TTF_CELL_H_PX 24u /∗ provisional: size + 4px leading/descender ∗/
- #define VT100_TTF_CACHE_SLOTS 96u /∗ covers all 95 printable ASCII, no thrash ∗/
- #define VT100_TTF_FONT_CAPSULE "fonts:JetBrainsMono-Regular.ttf"
- #define VT100_SCROLLBACK_MAX_LINES 1000u
- #define VT100_BOOT_SCROLLBACK_MAX_LINES 4096u

Enumerations

- enum vt_glyph_mode_t { VT_GLYPH_BITMAP = 0 , VT_GLYPH_TTF = 1 }
- enum ParseState {
  ST_NORMAL = 0 , ST_ESC , ST_CSI , ST_ESC_HASH ,
  ST_ESC_CHAR }

Functions

- static void put_char (uint8_t ch)
- static void advance_cursor (void)
- static void scroll_up (uint32_t lines)
- static void erase_display (int mode)
- static void erase_line (int mode)
- static void set_cursor (uint32_t row, uint32_t col)
- static void apply_sgr (void)
- static uint32_t cell_w (void)
- static uint32_t cell_h (void)
- static uint32_t px_of (uint32_t col)
- static uint32_t py_of (uint32_t row)
- static uint32_t eff_fg (void)
- static uint32_t eff_bg (void)
- static void ttf_draw_glyph_cell (uint32_t cell_px, uint32_t cell_py, uint8_t ch, uint32_t fg, uint32_t bg)
- static void draw_cursor_glyph (uint8_t ch)
- void vt100_draw_cursor (void)

  Draw a solid block cursor at the terminal's current position.
- void vt100_erase_cursor (void)

  Erases whatever vt100_draw_cursor() last drew, restoring the cell to plain background.
- static void reset_attrs (void)
- static void reset_parser (void)
- static int32_t param (uint32_t n, int32_t def)
- void vt100_init (void)

  Initialize the VT100 terminal.
- void vt100_enable_ttf (void)

  FABRIC-0.md item 4.4j: switch the glyph-draw backend from font_8x16.c to TTF-TEXT's rasterizer for everything drawn from this call onward – boot/POST output before this call stays font_8x16.c, unaffected.
- static const char ∗ scrollback_line_at (uint32_t idx)
- static void scrollback_redraw (void)
- void vt100_reset (void)
- void vt100_toggle_graphics (void)

FABRIC-0.md item 4.4y-revised: toggle the full-screen vt100 terminal between visible (normal operation) and hidden (graphics mode – the terminal stops drawing, letting direct framebuffer/TTF-TEXT calls show through undisturbed).

- void vt100_scroll_back (uint32_t n)

  FABRIC-0.md item 4.4q: move the REPL scrollback view back/forward by `n` lines and redraw.
- void vt100_scroll_fwd (uint32_t n)
- uint32_t vt100_cols (void)

  Terminal width in character columns.
- uint32_t vt100_rows (void)

  Terminal height in character rows.
- static void erase_line_range (uint32_t row, uint32_t c0, uint32_t c1)
- static uint32_t ansi256_to_rgb (uint32_t idx)
- static void dispatch_csi (char final)
- void vt100_putc (char raw)

  Feed one byte into the terminal (handles escape sequences transparently).
- void vt100_puts (const char *s)

  Feed a null-terminated string.

Variables

- static vt_glyph_mode_t g_glyph_mode = VT_GLYPH_BITMAP
- static ttf_font_t g_ttf_font
- static int g_ttf_ready = 0
- static ttf_raster_cache_t g_ttf_cache
- static int g_terminal_visible = 1
- static char * g_shadow = (void *)0
- static char * g_ring = (void *)0
- static uint32_t g_line_stride = 0
- static uint32_t g_ring_count = 0
- static uint32_t g_ring_head = 0
- static uint32_t g_ring_cap = 0
- static uint32_t g_scroll_offset = 0
- static char * g_boot_shadow = (void *)0
- static char * g_boot_ring = (void *)0
- static uint32_t g_boot_line_stride = 0
- static uint32_t g_boot_ring_count = 0
- static uint32_t g_boot_ring_head = 0
- static uint32_t g_boot_ring_cap = 0
- static uint32_t g_boot_rows = 0
- static VT100State g_vt

7.158.1 Macro Definition Documentation

7.158.1.1 UNDERLINE_ROW

```
#define UNDERLINE_ROW 14u
```

7.158.1.2 VT100_BOOT_SCROLLBACK_MAX_LINES

```
#define VT100_BOOT_SCROLLBACK_MAX_LINES 4096u
```

7.158.1.3 VT100_MAX_PARAMS

```
#define VT100_MAX_PARAMS 16
```

vt100.c — Full-color ANSI / VT100 terminal state machine

Parser states ————————- NORMAL plain text characters ESC received 0x1B, waiting for next byte CSI received ESC [, accumulating parameter digits / intermediates ESC_HASH received ESC # (line attributes — we ignore, eat next byte) ESC_CHAR received ESC (or ESC) (charset designation — ignored)

Supported CSI sequences ———————————— CUU A cursor up CUD B cursor down CUF C cursor forward CUB D cursor back CUP H cursor position (row ; col) ED J erase display (0=below 1=above 2=all) EL K erase line (0=right 1=left 2=all) SGR m select graphic rendition (see header) SCP s save cursor position RCP u restore cursor position

ESC sequences (2-char) ESC 7 save cursor (DEC) ESC 8 restore cursor (DEC) ESC M reverse index (scroll down) ESC c full reset

7.158.1.4 VT100_SCROLLBACK_MAX_LINES

```
#define VT100_SCROLLBACK_MAX_LINES 1000u
```

7.158.1.5 VT100_TTF_CACHE_SLOTS

```
#define VT100_TTF_CACHE_SLOTS 96u /* covers all 95 printable ASCII, no thrash */
```

7.158.1.6 VT100_TTF_CELL_H_PX

```
#define VT100_TTF_CELL_H_PX 24u /* provisional:  size + 4px leading/descender */
```

7.158.1.7 VT100_TTF_CELL_W_PX

```
#define VT100_TTF_CELL_W_PX 12u /* 600/1000 * VT100_TTF_SIZE_PX */
```

7.158.1.8 VT100_TTF_FONT_CAPSULE

```
#define VT100_TTF_FONT_CAPSULE "fonts:JetBrainsMono-Regular.ttf"
```

7.158.1.9 VT100_TTF_SIZE_PX

```
#define VT100_TTF_SIZE_PX 20u
```

7.158.2 Enumeration Type Documentation

7.158.2.1 ParseState

```
enum ParseState
```

Enumerator

ST_NORMAL	
ST_ESC	
ST_CSI	
ST_ESC_HASH	
ST_ESC_CHAR	

7.158.2.2 vt_glyph_mode_t

```
enum vt_glyph_mode_t
```

Enumerator

VT_GLYPH_BITMAP	
VT_GLYPH_TTF	

7.158.3 Function Documentation

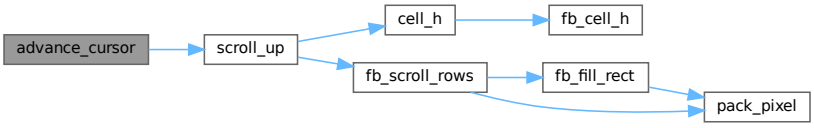
7.158.3.1 advance_cursor()

```
static void advance_cursor (
            void  )  [static]
```

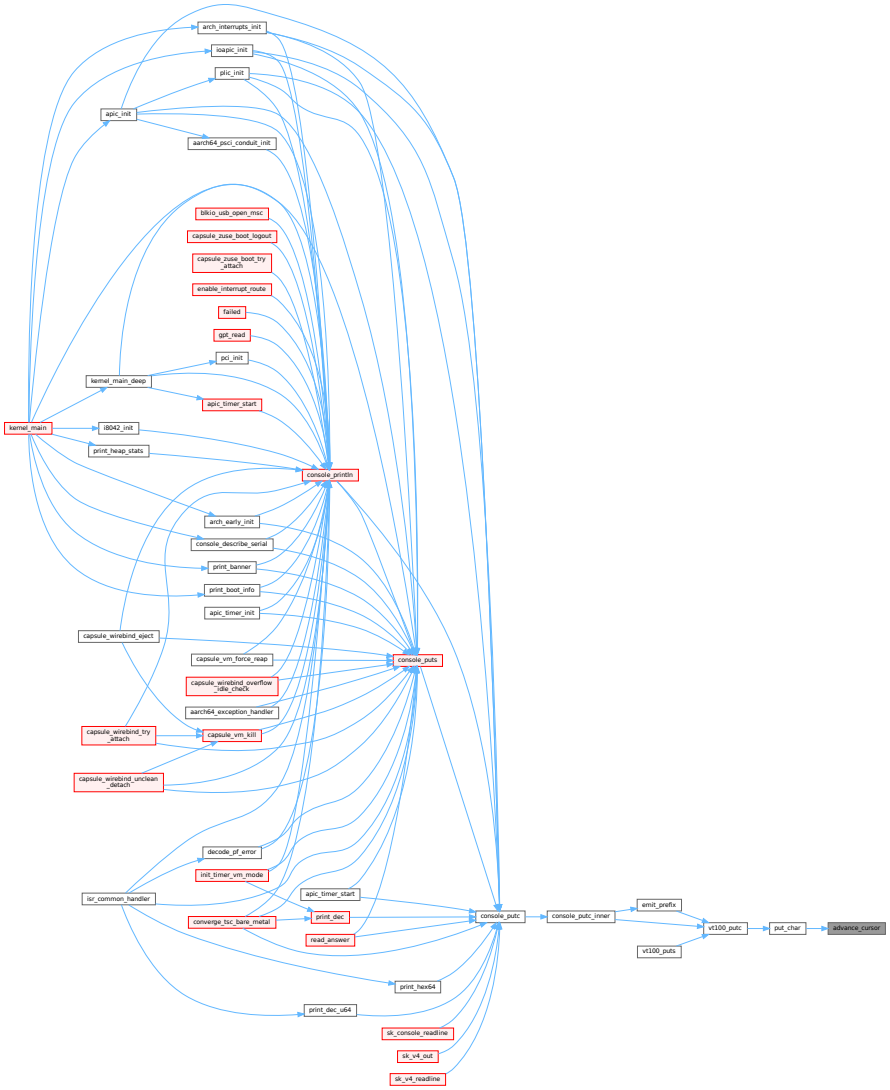
References VT100State::cols, VT100State::cx, VT100State::cy, g_vt, VT100State::rows, and scroll_up().

Referenced by put_char().

Here is the call graph for this function:



Here is the caller graph for this function:



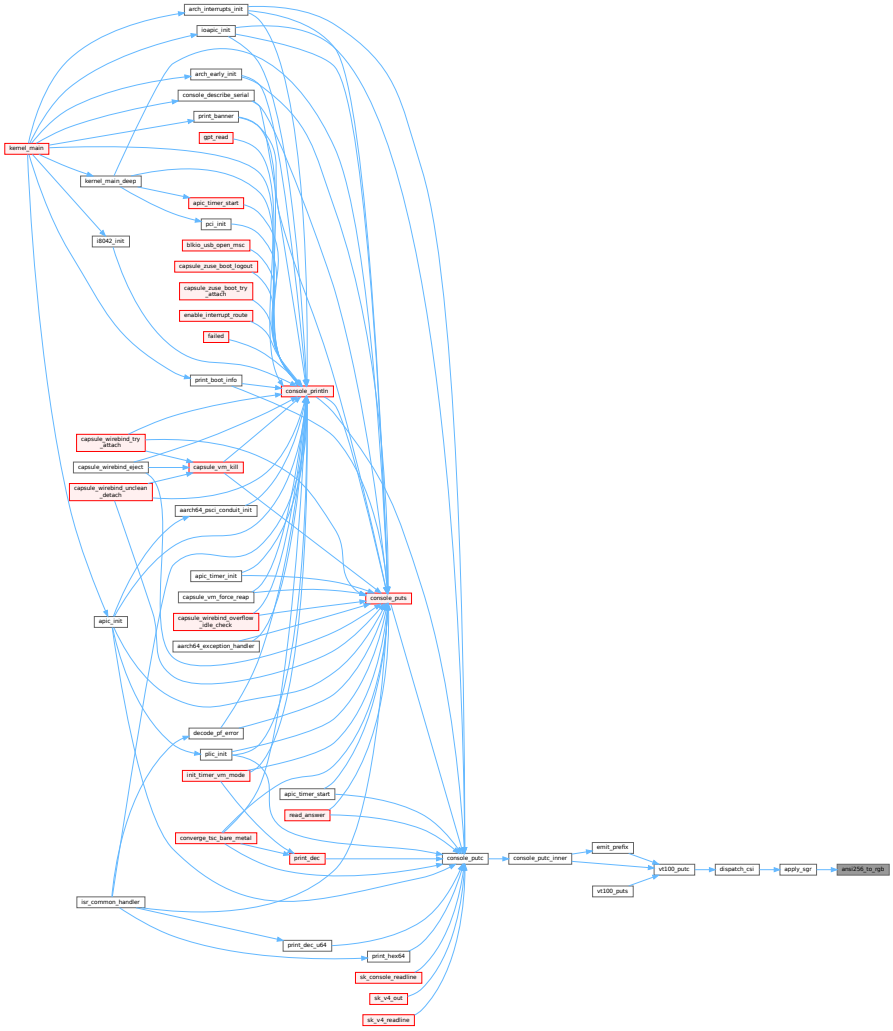
7.158.3.2 ansi256_to_rgb()

```
static uint32_t ansi256_to_rgb (
            uint32_t idx )  [static]
```

References FB_RGB, and g.

Referenced by apply_sgr().

Here is the caller graph for this function:



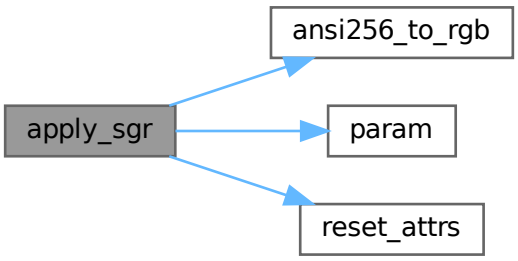
7.158.3.3 apply_sgr()

```
static void apply_sgr (
            void  )  [static]
```

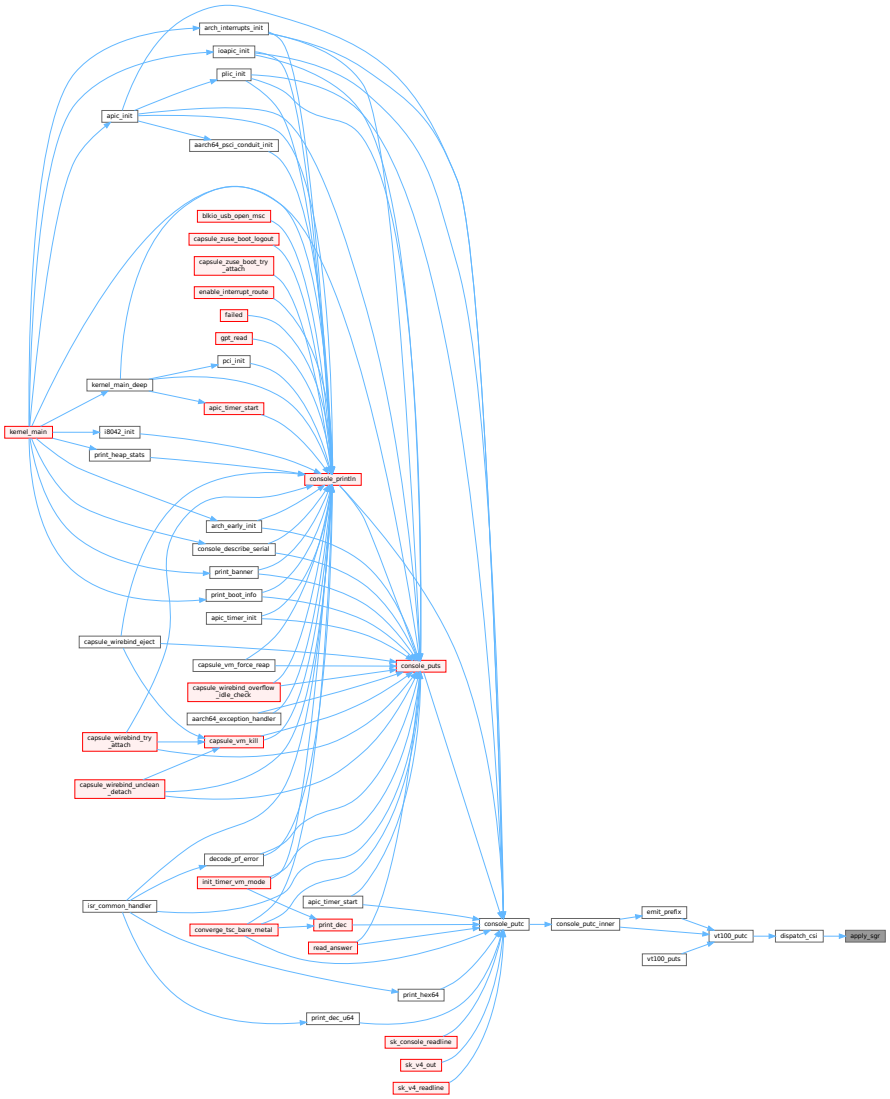
References ansi256_to_rgb(), VT100State::bg, VT100State::bold, VT100State::def_bg, VT100State::def_fg, FB_RGB, VT100State::fg, g, g_vt, param(), VT100State::param_count, reset_attrs(), VT100State::reverse, and VT100State::underline.

Referenced by dispatch_csi().

Here is the call graph for this function:



Here is the caller graph for this function:



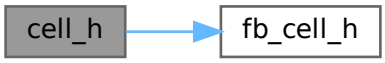
7.158.3.4 cell_h()

```
static uint32_t cell_h (
            void  )  [static]
```

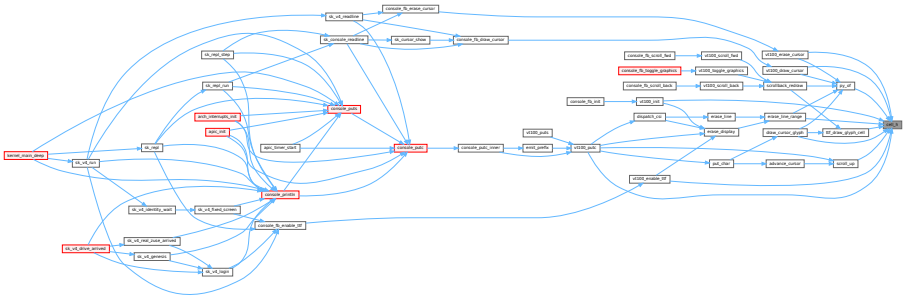
References fb_cell_h(), g_glyph_mode, VT100_TTF_CELL_H_PX, and VT_GLYPH_TTF.

Referenced by draw_cursor_glyph(), erase_line_range(), py_of(), scroll_up(), ttf_draw_glyph_cell(), vt100_draw_cursor(), vt100_enable_ttf(), vt100_erase_cursor(), vt100_init(), and vt100_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



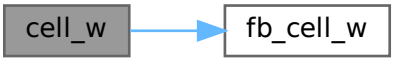
7.158.3.5 cell_w()

```
static uint32_t cell_w (
            void  )  [static]
```

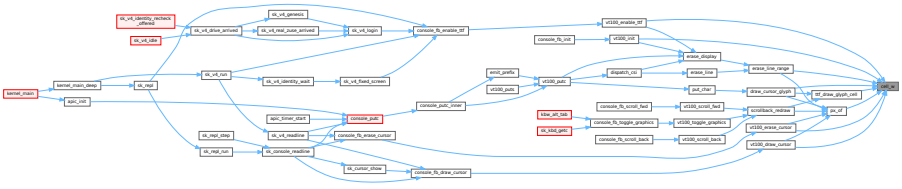
References fb_cell_w(), g_glyph_mode, VT100_TTF_CELL_W_PX, and VT_GLYPH_TTF.

Referenced by draw_cursor_glyph(), erase_line_range(), px_of(), ttf_draw_glyph_cell(), vt100_draw_cursor(), vt100_enable_ttf(), vt100_erase_cursor(), and vt100_init().

Here is the call graph for this function:



Here is the caller graph for this function:



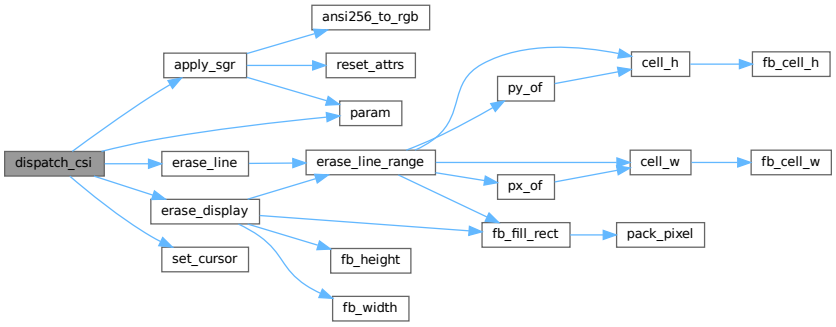
7.158.3.6 dispatch_csi()

```
static void dispatch_csi (
            char final )  [static]
```

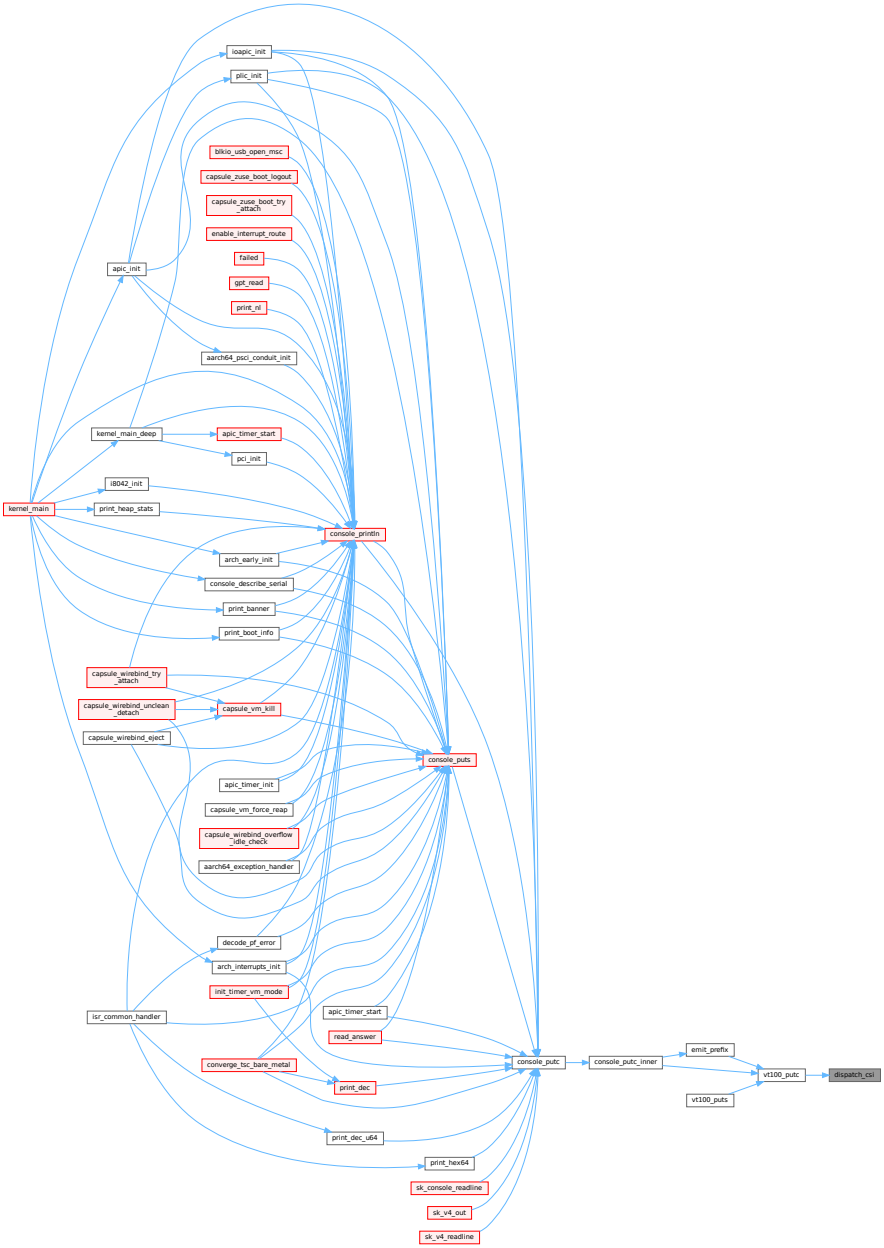
References apply_sgr(), VT100State::cols, VT100State::cx, VT100State::cy, erase_display(), erase_line(), g_vt, param(), VT100State::rows, VT100State::saved_cx, VT100State::saved_cy, and set_cursor().

Referenced by vt100_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



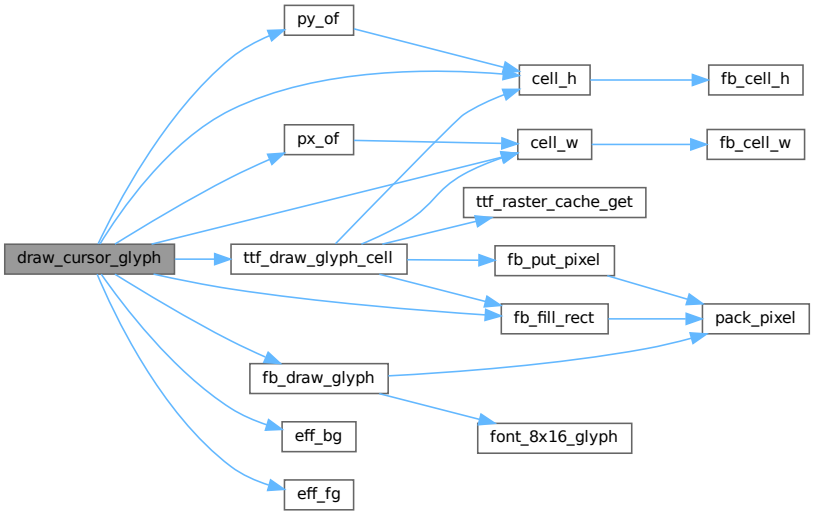
7.158.3.7 draw_cursor_glyph()

```
static void draw_cursor_glyph (
            uint8_t ch )  [static]
```

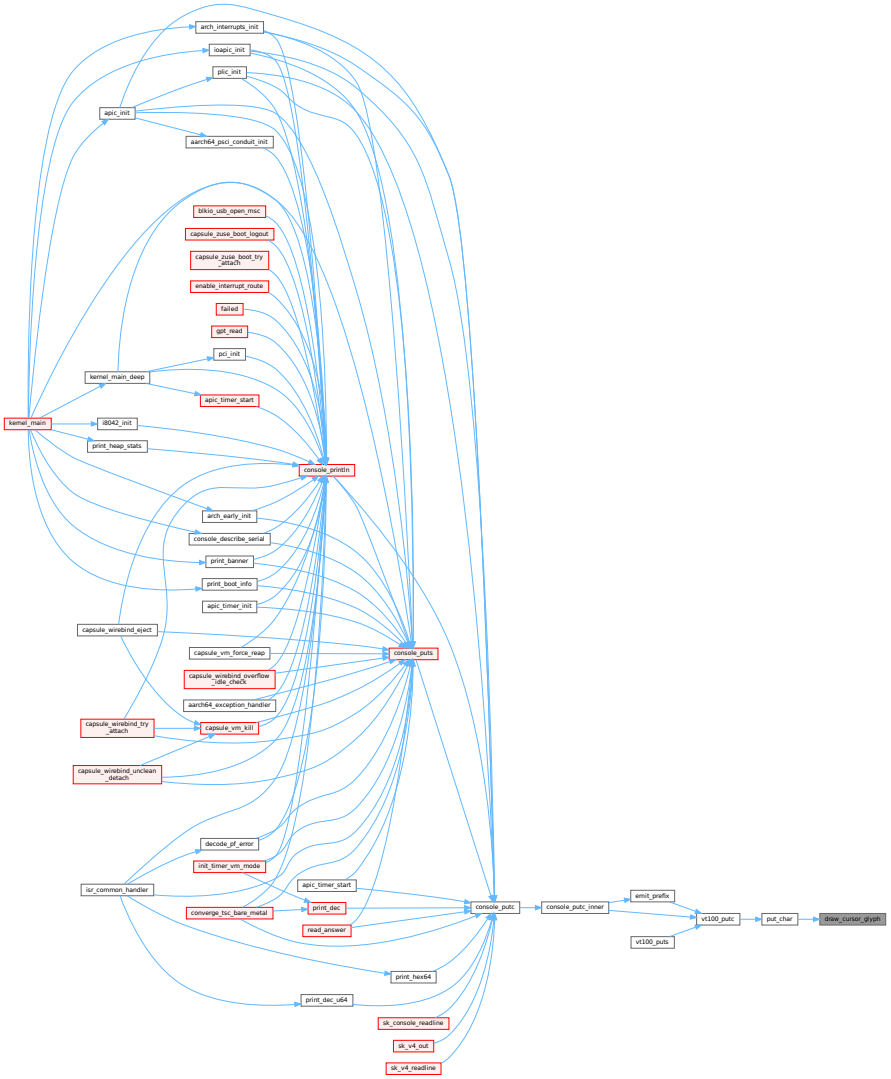
References VT100State::bold, cell_h(), cell_w(), VT100State::cols, VT100State::cx, VT100State::cy, eff_bg(), eff_fg(), FB_B, fb_draw_glyph(), fb_fill_rect(), FB_G, FB_R, FB_RGB, g, g_glyph_mode, g_line_stride, g_shadow, g_terminal_visible, g_vt, px_of(), py_of(), VT100State::rows, ttf_draw_glyph_cell(), VT100State::underline, UNDERLINE_ROW, VT100_TTF_SIZE_PX, and VT_GLYPH_TTF.

Referenced by put_char().

Here is the call graph for this function:



Here is the caller graph for this function:



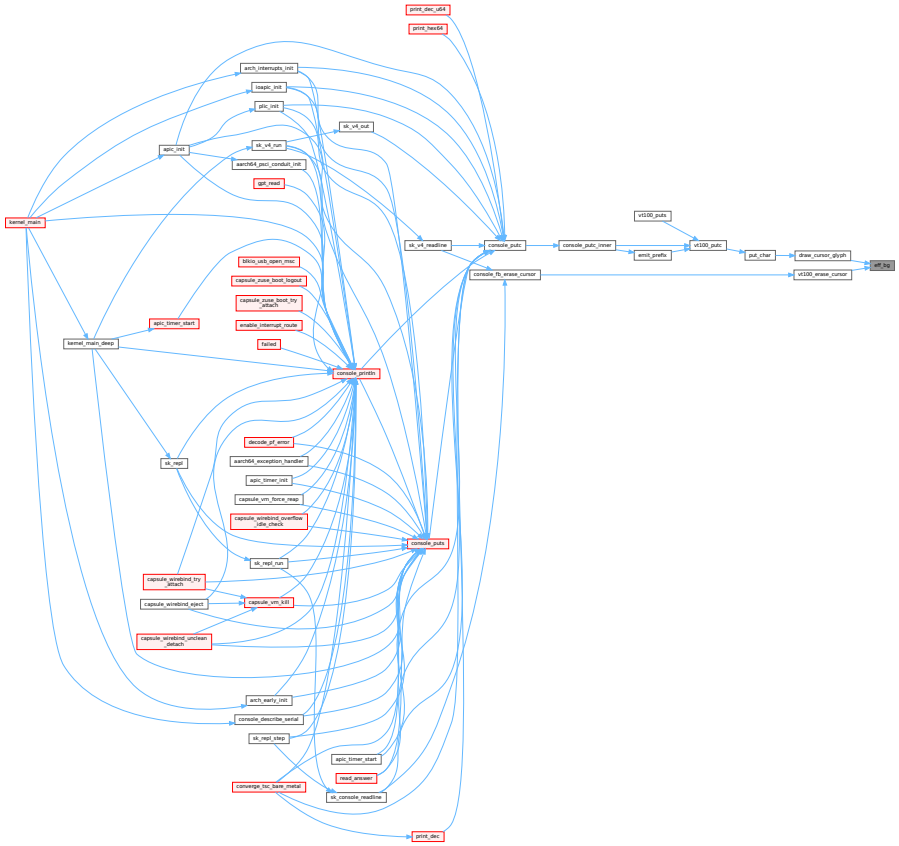
7.158.3.8 eff_bg()

```
static uint32_t eff_bg (
            void  )  [static]
```

References VT100State::bg, VT100State::fg, g_vt, and VT100State::reverse.

Referenced by draw_cursor_glyph(), and vt100_erase_cursor().

Here is the caller graph for this function:



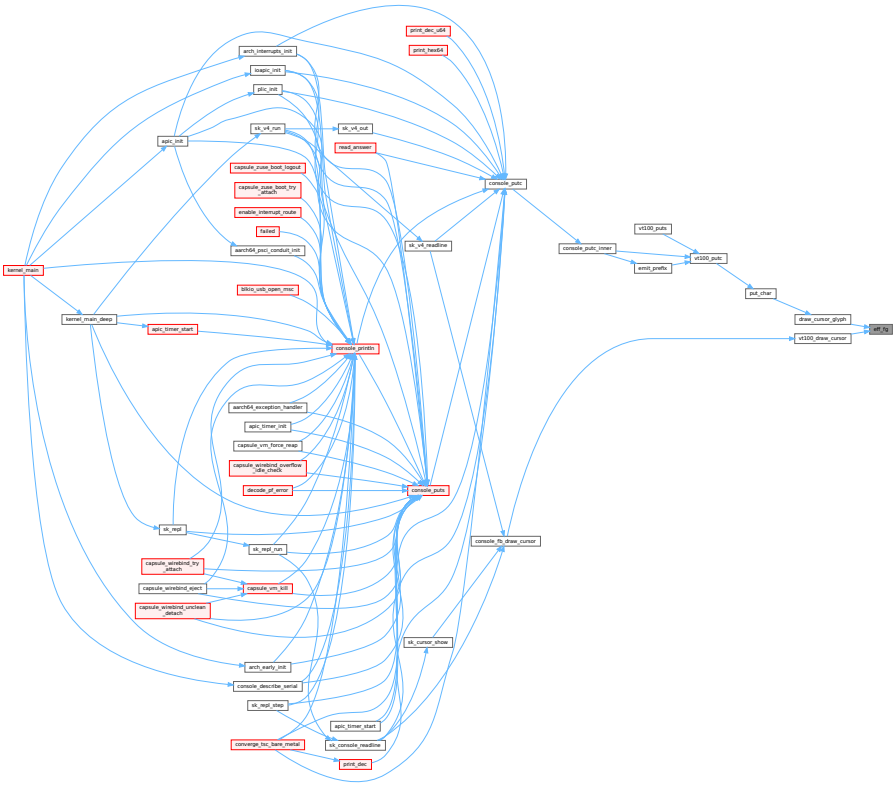
7.158.3.9 eff_fg()

```
static uint32_t eff_fg (
            void  )  [static]
```

References VT100State::bg, VT100State::fg, g_vt, and VT100State::reverse.

Referenced by draw_cursor_glyph(), and vt100_draw_cursor().

Here is the caller graph for this function:



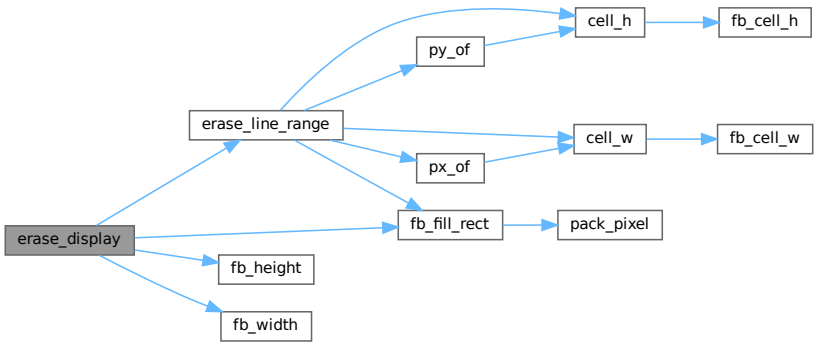
7.158.3.10 erase_display()

```
static void erase_display (
            int mode )  [static]
```

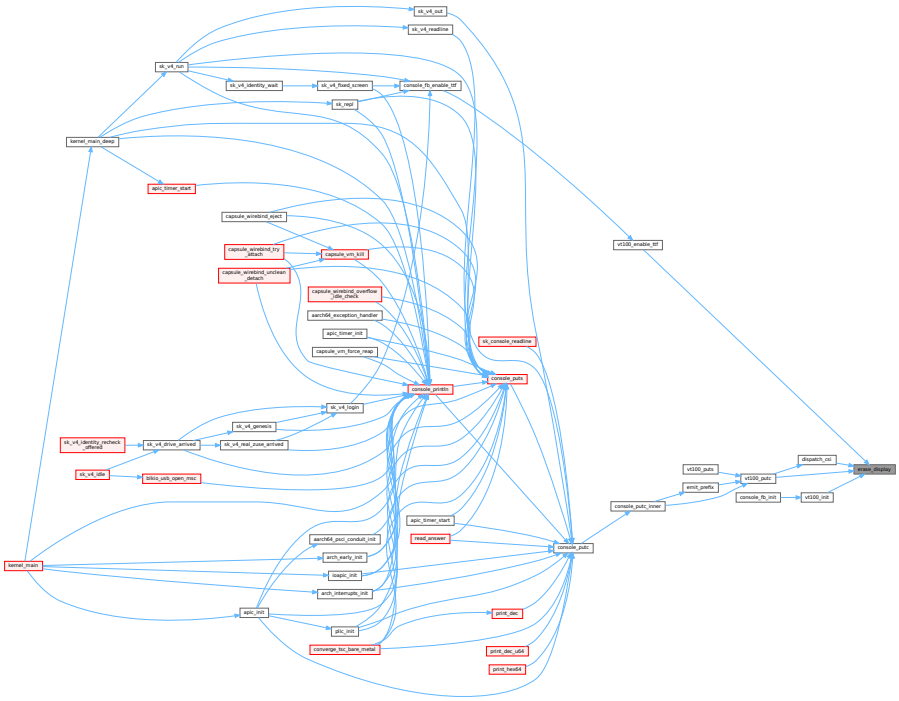
References VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, erase_line_range(), fb_fill_rect(), fb_height(), fb_width(), g_terminal_visible, g_vt, and VT100State::rows.

Referenced by dispatch_csi(), vt100_enable_ttf(), vt100_init(), and vt100_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



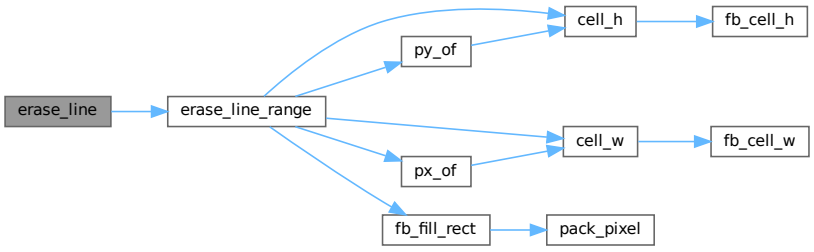
7.158.3.11 erase_line()

```
static void erase_line (
            int mode )  [static]
```

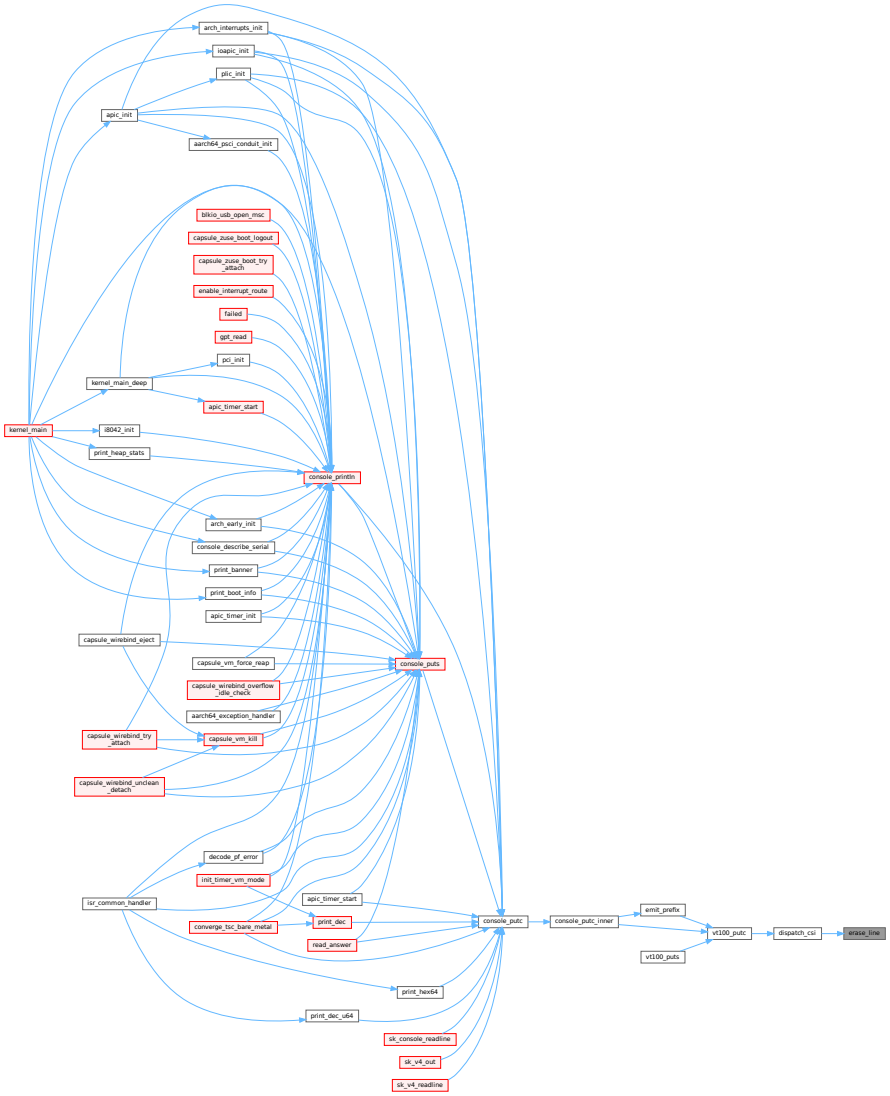
References VT100State::cols, VT100State::cx, VT100State::cy, erase_line_range(), and g_vt.

Referenced by dispatch_csi().

Here is the call graph for this function:



Here is the caller graph for this function:



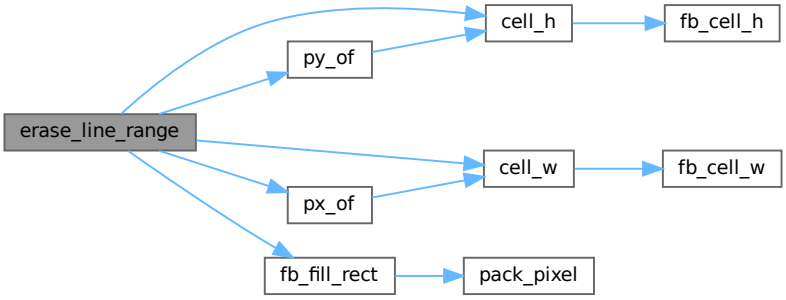
7.158.3.12 erase_line_range()

```
static void erase_line_range (
            uint32_t row,
            uint32_t c0,
            uint32_t c1 )  [static]
```

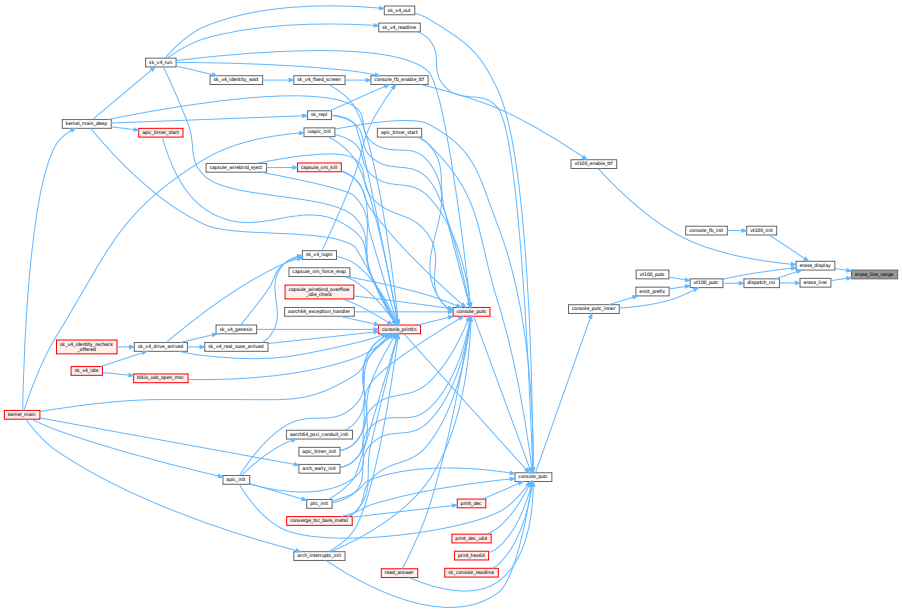
References VT100State::bg, cell_h(), cell_w(), fb_fill_rect(), g_terminal_visible, g_vt, px_of(), and py_of().

Referenced by erase_display(), and erase_line().

Here is the call graph for this function:



Here is the caller graph for this function:



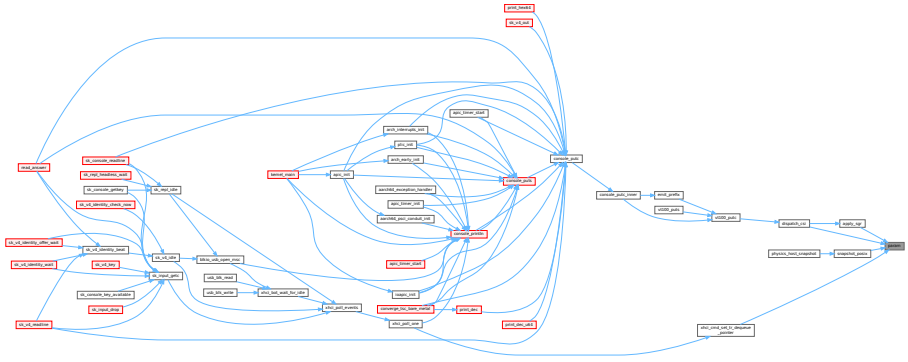
7.158.3.13 param()

```
static int32_t param (
            uint32_t n,
            int32_t def )  [static]
```

References g_vt, VT100State::param_count, and VT100State::params.

Referenced by apply_sgr(), dispatch_csi(), snapshot_posix(), and xhci_cmd_set_tr_dequeue_pointer().

Here is the caller graph for this function:



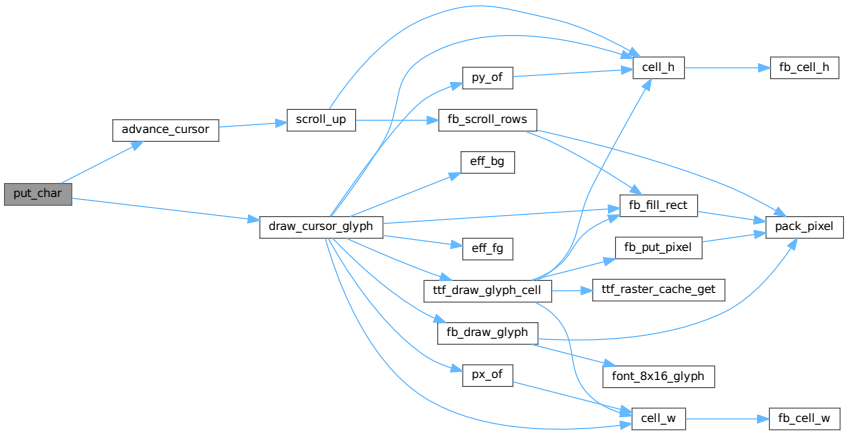
7.158.3.14 put_char()

```
static void put_char (
            uint8_t ch )  [static]
```

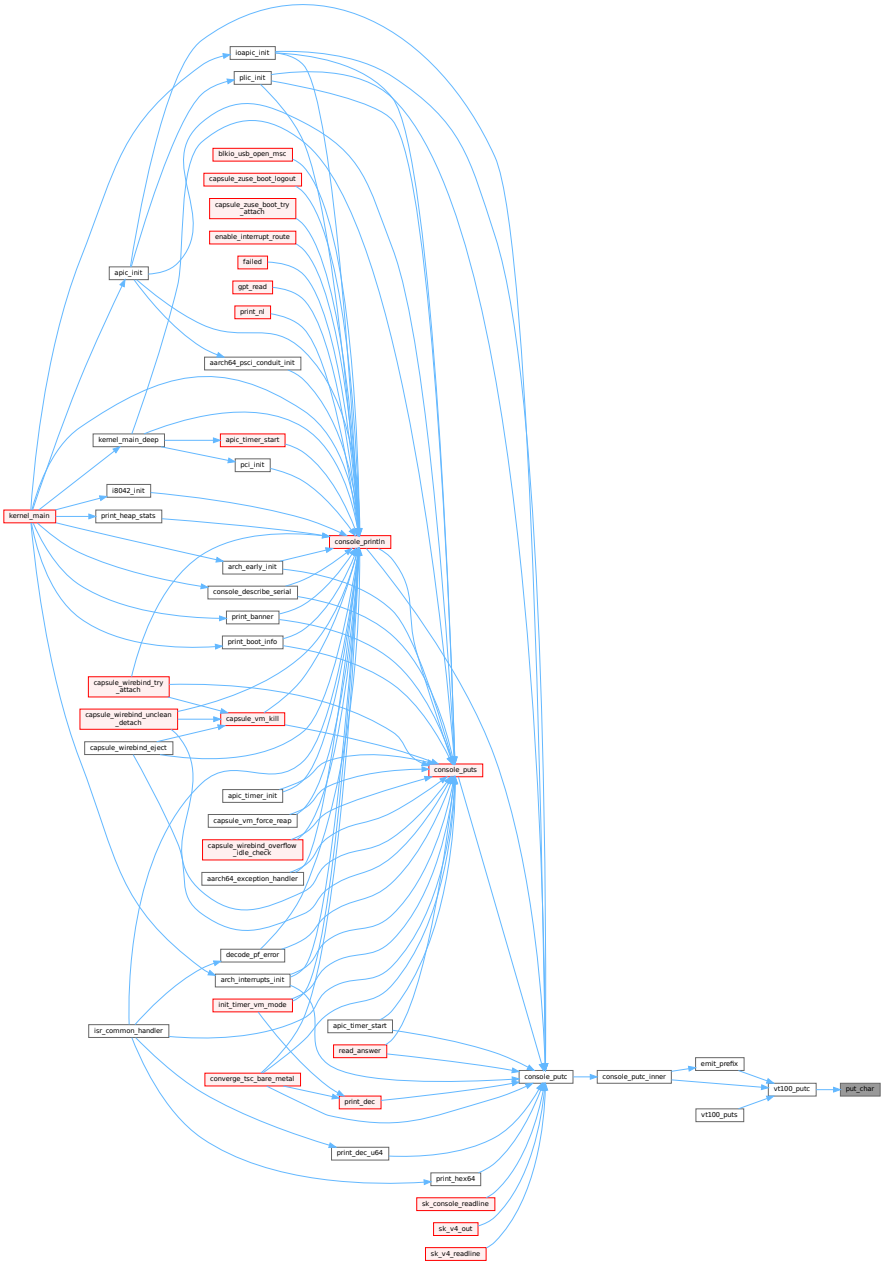
References advance_cursor(), and draw_cursor_glyph().

Referenced by vt100_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



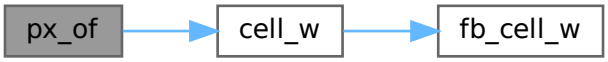
7.158.3.15 px_of()

```
static uint32_t px_of (
            uint32_t col )  [static]
```

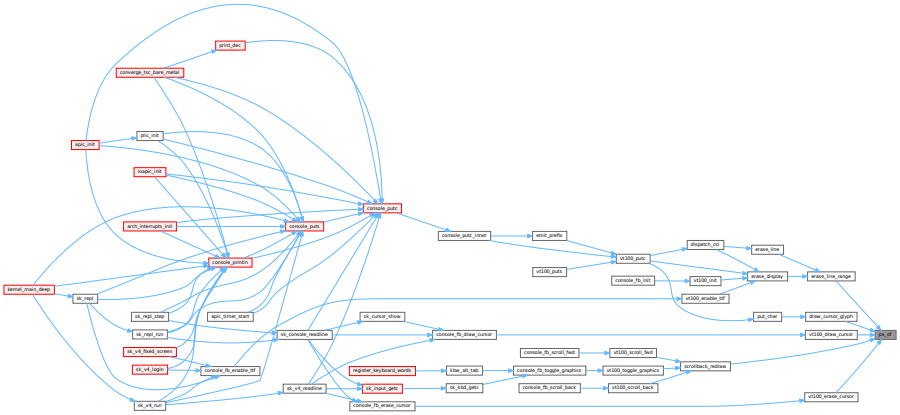
References cell_w().

Referenced by draw_cursor_glyph(), erase_line_range(), scrollback_redraw(), vt100_draw_cursor(), and vt100_erase_cursor().

Here is the call graph for this function:



Here is the caller graph for this function:



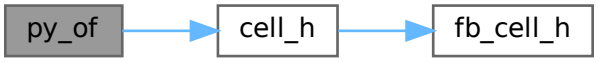
7.158.3.16 py_of()

```
static uint32_t py_of (
            uint32_t row )  [static]
```

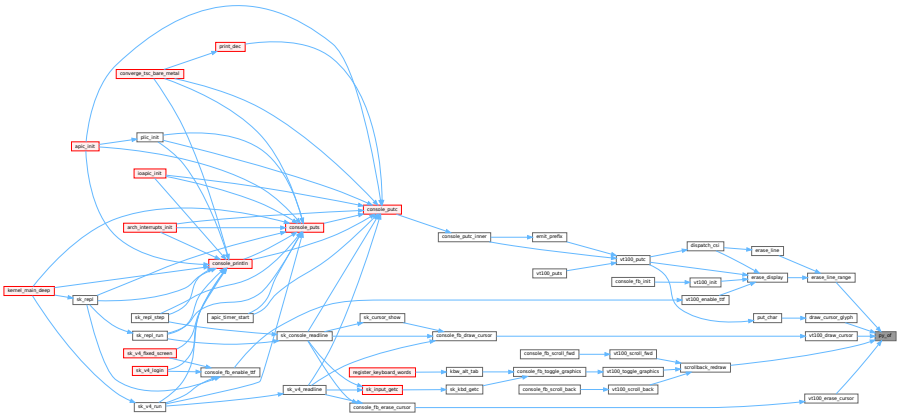
References cell_h().

Referenced by draw_cursor_glyph(), erase_line_range(), scrollback_redraw(), vt100_draw_cursor(), and vt100_erase_cursor().

Here is the call graph for this function:



Here is the caller graph for this function:



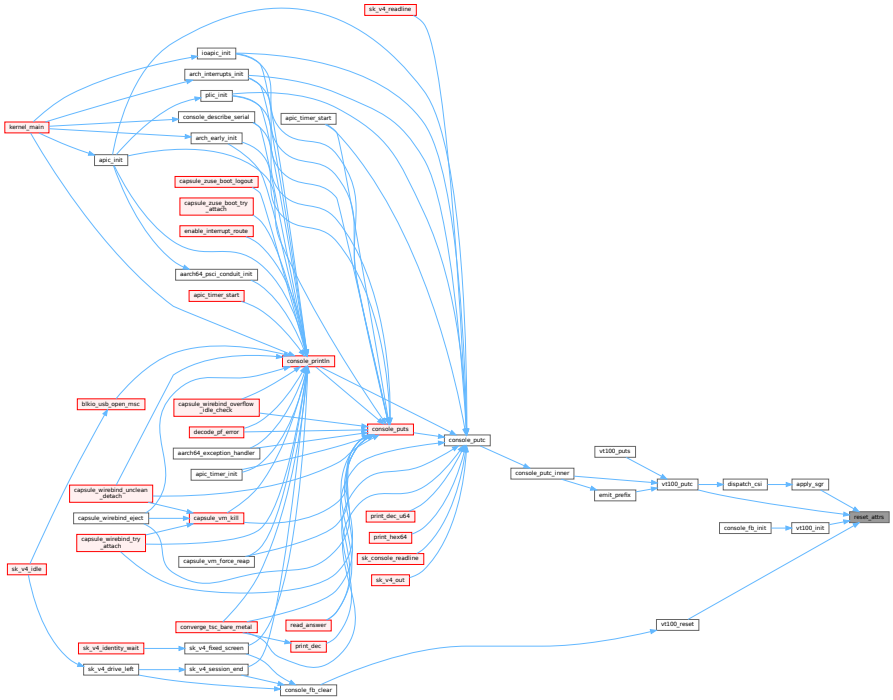
7.158.3.17 reset_attrs()

```
static void reset_attrs (
            void  )  [static]
```

References VT100State::bg, VT100State::bold, VT100State::def_bg, VT100State::def_fg, VT100State::fg, g_vt, VT100State::reverse, and VT100State::underline.

Referenced by apply_sgr(), vt100_init(), vt100_putc(), and vt100_reset().

Here is the caller graph for this function:



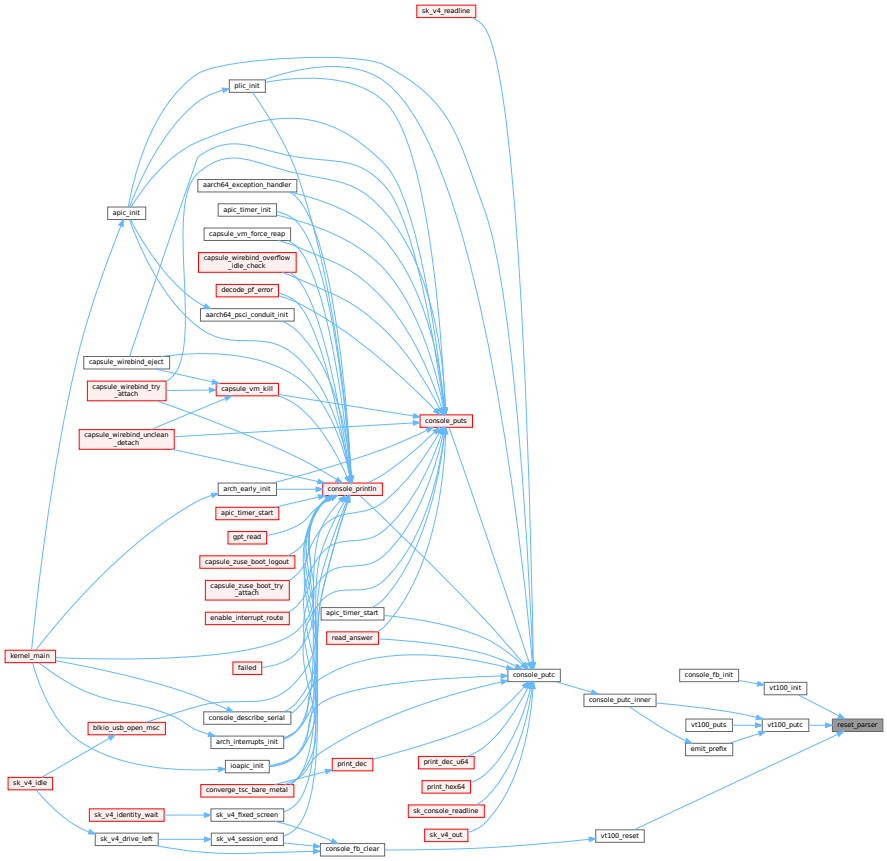
7.158.3.18 reset_parser()

```
static void reset_parser (
            void  )  [static]
```

References g_vt, VT100State::param_count, VT100State::param_has_value, VT100State::params, ST_NORMAL, VT100State::state, and VT100_MAX_PARAMS.

Referenced by vt100_init(), vt100_putc(), and vt100_reset().

Here is the caller graph for this function:



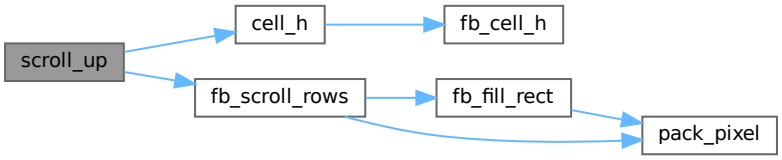
7.158.3.19 scroll_up()

```
static void scroll_up (
            uint32_t lines )  [static]
```

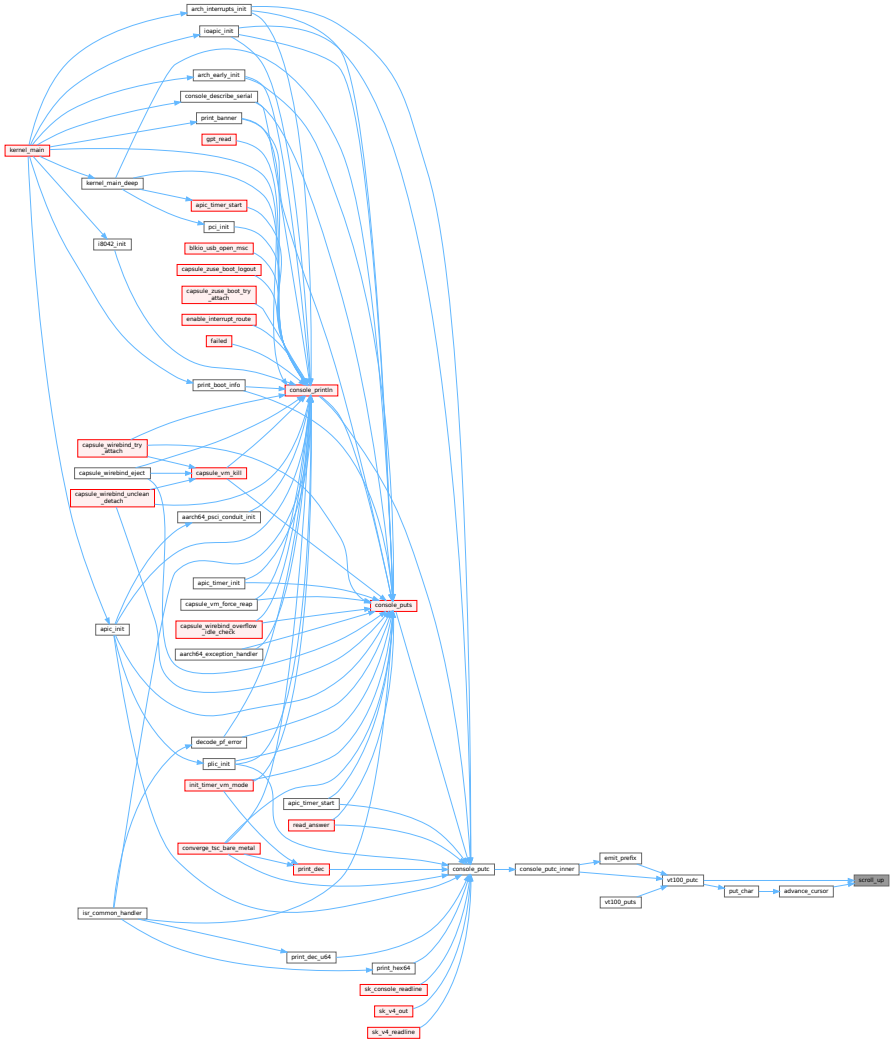
References cell_h(), VT100State::cols, VT100State::def_bg, fb_scroll_rows(), g_line_stride, g_ring, g_ring_cap, g_ring_count, g_ring_head, g_shadow, g_terminal_visible, g_vt, and VT100State::rows.

Referenced by advance_cursor(), and vt100_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



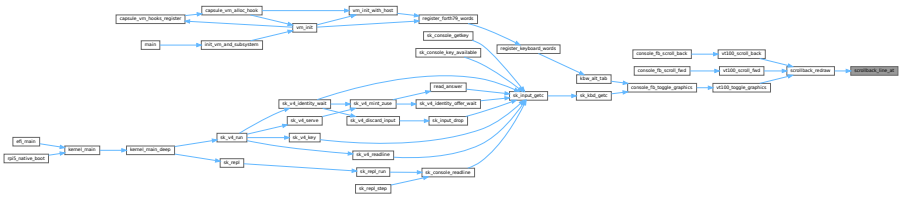
7.158.3.20 scrollback_line_at()

```
static const char * scrollback_line_at (
            uint32_t idx )  [static]
```

References g_boot_line_stride, g_boot_ring, g_boot_ring_cap, g_boot_ring_count, g_boot_ring_head, g_boot_rows, g_boot_shadow, g_line_stride, g_ring, g_ring_cap, g_ring_count, g_ring_head, g_shadow, g_vt, and VT100State::rows.

Referenced by scrollback_redraw().

Here is the caller graph for this function:



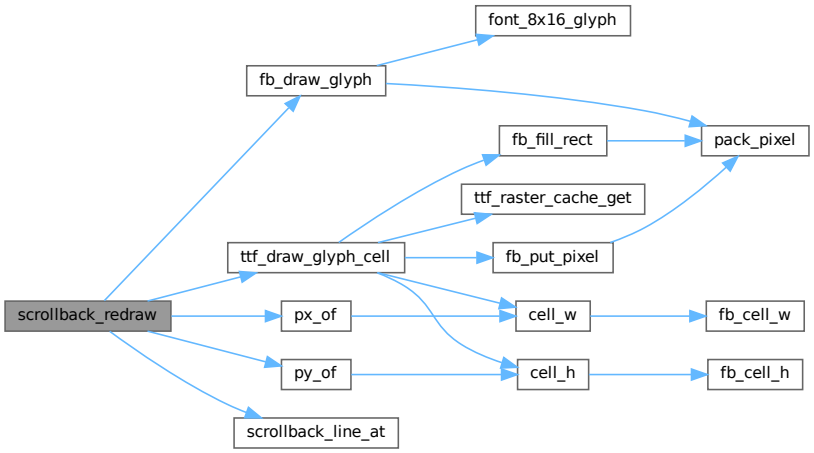
7.158.3.21 scrollback_redraw()

```
static void scrollback_redraw (
            void  )  [static]
```

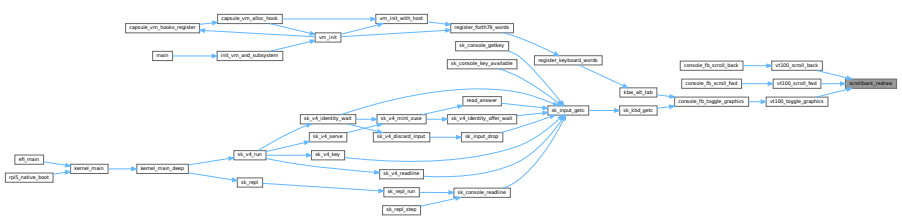
References VT100State::cols, VT100State::def_bg, VT100State::def_fg, fb_draw_glyph(), g_boot_ring_count, g_boot_rows, g_glyph_mode, g_ring_count, g_scroll_offset, g_vt, px_of(), py_of(), VT100State::rows, scrollback_line_at(), ttf_draw_glyph_cell(), and VT_GLYPH_TTF.

Referenced by vt100_scroll_back(), vt100_scroll_fwd(), and vt100_toggle_graphics().

Here is the call graph for this function:



Here is the caller graph for this function:



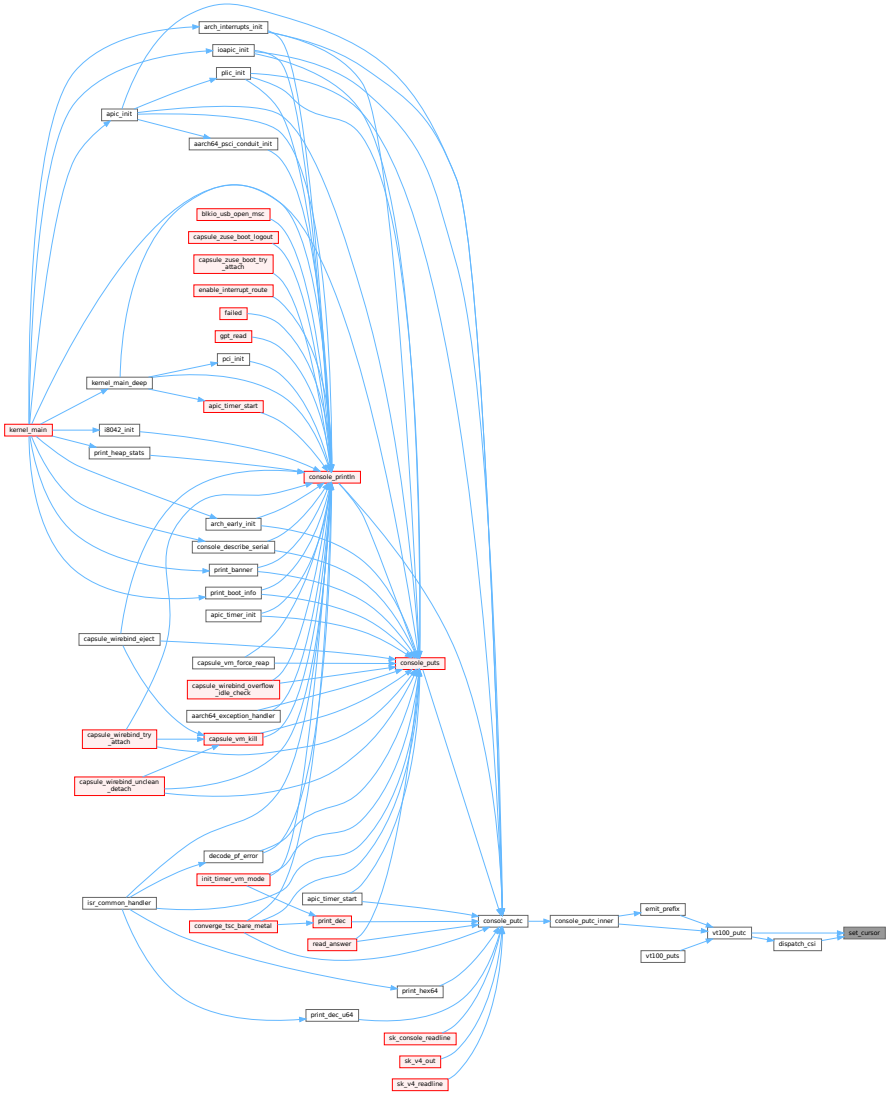
7.158.3.22 set_cursor()

```
static void set_cursor (
            uint32_t row,
            uint32_t col )  [static]
```

References VT100State::cols, VT100State::cx, VT100State::cy, g_vt, and VT100State::rows.

Referenced by dispatch_csi(), and vt100_putc().

Here is the caller graph for this function:



7.158.3.23 ttf_draw_glyph_cell()

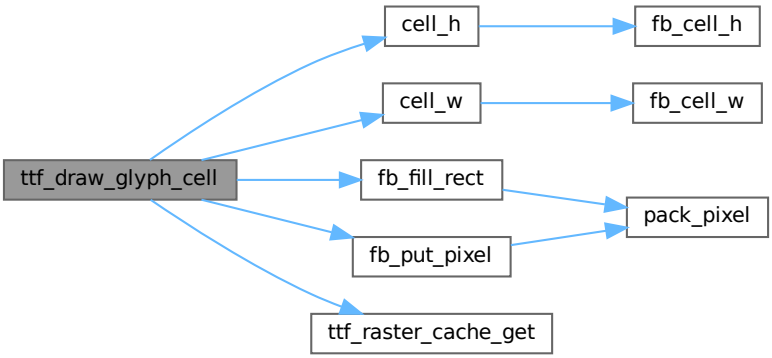
```
static void ttf_draw_glyph_cell (
            uint32_t cell_px,
            uint32_t cell_py,
```

```
    uint8_t ch,
    uint32_t fg,
    uint32_t bg )  [static]
```

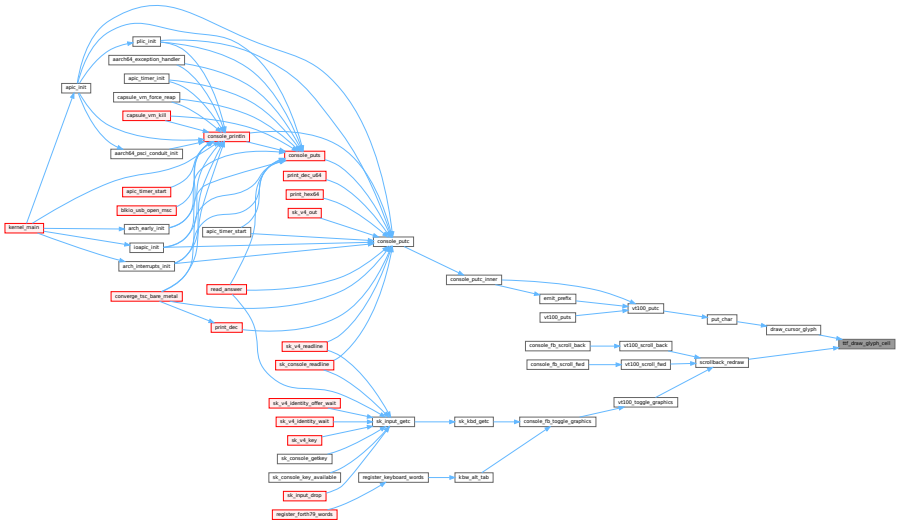
References cell_h(), cell_w(), fb_fill_rect(), fb_put_pixel(), g_ttf_cache, g_ttf_font, g_ttf_ready, ttf_bitmap_t::height, ttf_bitmap_t::pixels, TTF_CACHE_MARGIN, TTF_OK, ttf_raster_cache_get(), VT100_TTF_SIZE_PX, and ttf_bitmap_t::width.

Referenced by draw_cursor_glyph(), and scrollback_redraw().

Here is the call graph for this function:



Here is the caller graph for this function:



7.158.3.24 vt100_cols()

```
uint32_t vt100_cols (
            void  )
```

Terminal width in character columns.

References VT100State::cols, and g_vt.

7.158.3.25 vt100_draw_cursor()

```
void vt100_draw_cursor (
            void  )
```

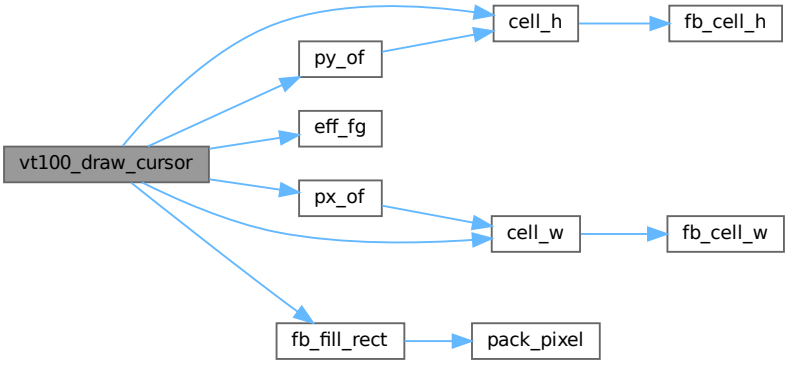
Draw a solid block cursor at the terminal's current position.

Static, not blinking. Idempotent – safe to call after every keystroke, since whatever gets typed next naturally overwrites it. No-op before TTF mode is active or while in graphics mode (vt100_toggle_graphics()).

References cell_h(), cell_w(), VT100State::cx, VT100State::cy, eff_fg(), fb_fill_rect(), g_glyph_mode, g_terminal_visible, g_vt, px_of(), py_of(), and VT_GLYPH_TTF.

Referenced by console_fb_draw_cursor().

Here is the call graph for this function:



Here is the caller graph for this function:

7.158.3.26 vt100_enable_ttf()

```
void vt100_enable_ttf (
            void  )
```

FABRIC-0.md item 4.4j: switch the glyph-draw backend from font_8x16.c to TTF-TEXT's rasterizer for everything drawn from this call onward – boot/POST output before this call stays font_8x16.c, unaffected.

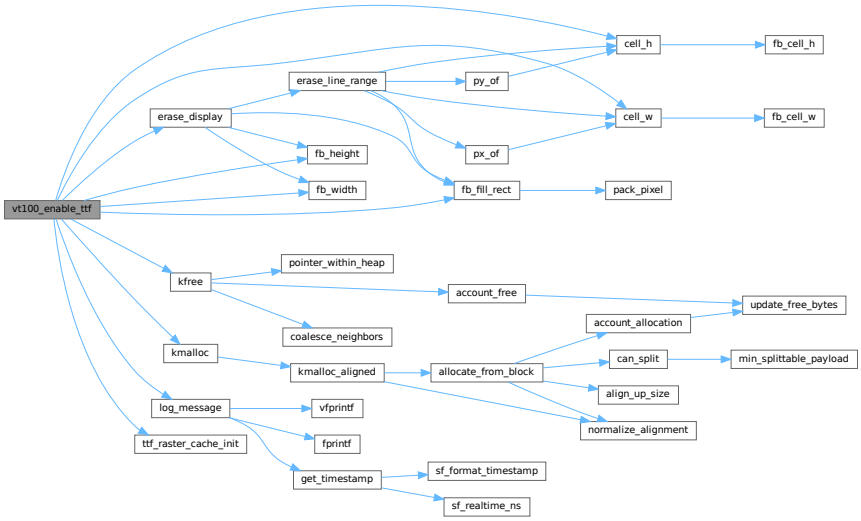
Lazily loads the font capsule and its raster cache on first call (no-op on later calls). Recomputes cols/rows for the new cell size and clears the screen, since the two glyph backends use different cell dimensions. TTF point size/cell

dimensions decided final by FABRIC-0.md item 4.4m (20px text, 96px REPL strip). No-op if the font capsule fails to load (stays on font_8x16.c; logged, not fatal).

References cell_h(), cell_w(), VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, erase_display(), fb_fill_rect(), fb_height(), fb_width(), g_boot_line_stride, g_boot_ring, g_boot_ring_cap, g_boot_ring_count, g_boot_ring_head, g_boot_rows, g_boot_shadow, g_glyph_mode, g_line_stride, g_ring, g_ring_cap, g_ring_count, g_ring_head, g_scroll_offset, g_shadow, g_ttf_cache, g_ttf_font, g_ttf_ready, g_vt, VT100State::initialized, kfree(), kmalloc(), LOG_ERROR, log_message(), VT100State::rows, slots, TTF_OK, ttf_raster_cache_init(), VT100_SCROLLBACK_MAX_LINES, VT100_TTF_CACHE_SLOTS, VT100_TTF_FONT_CAPSULE, and VT_GLYPH_TTF.

Referenced by console_fb_enable_ttf().

Here is the call graph for this function:



Here is the caller graph for this function:

7.158.3.27 vt100_erase_cursor()

```
void vt100_erase_cursor (
            void  )
```

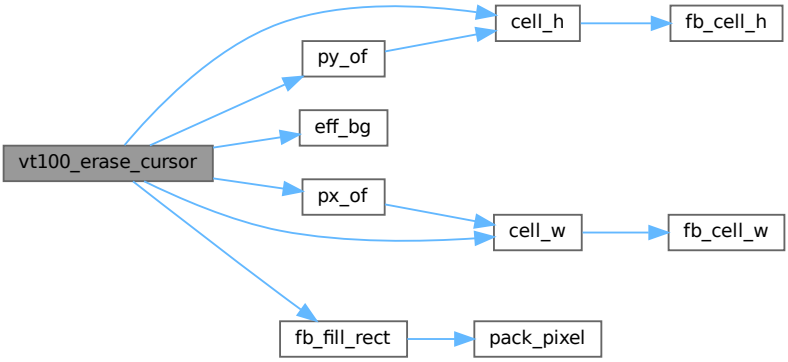
Erases whatever vt100_draw_cursor() last drew, restoring the cell to plain background.

Call before moving away from a cursor-drawn cell without also drawing a character there (e.g. Enter/newline). No-op before TTF mode is active or while in graphics mode.

References cell_h(), cell_w(), VT100State::cx, VT100State::cy, eff_bg(), fb_fill_rect(), g_glyph_mode, g_terminal_visible, g_vt, px_of(), py_of(), and VT_GLYPH_TTF.

Referenced by console_fb_erase_cursor().

Here is the call graph for this function:



Here is the caller graph for this function:

7.158.3.28 vt100_init()

```
void vt100_init (
            void  )
```

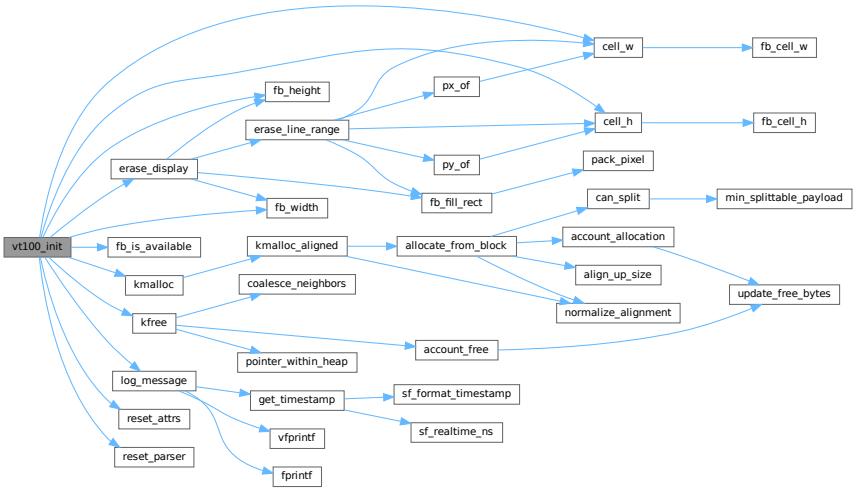
Initialize the VT100 terminal.

Must be called after fb_init(). Clears the screen and homes the cursor.

References cell_h(), cell_w(), VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, VT100State::def_fg, erase_display(), fb_height(), fb_is_available(), fb_width(), g_line_stride, g_ring, g_ring_cap, g_ring_count, g_ring_head, g_shadow, g_vt, VT100State::initialized, kfree(), kmalloc(), LOG_ERROR, log_message(), reset_attrs(), reset_parser(), VT100State::rows, VT100State::saved_cx, VT100State::saved_cy, VT100_BOOT_SCROLLBACK_MAX_LINES, VT100_DEFAULT_BG, and VT100_DEFAULT_FG.

Referenced by console_fb_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.158.3.29 vt100_putc()

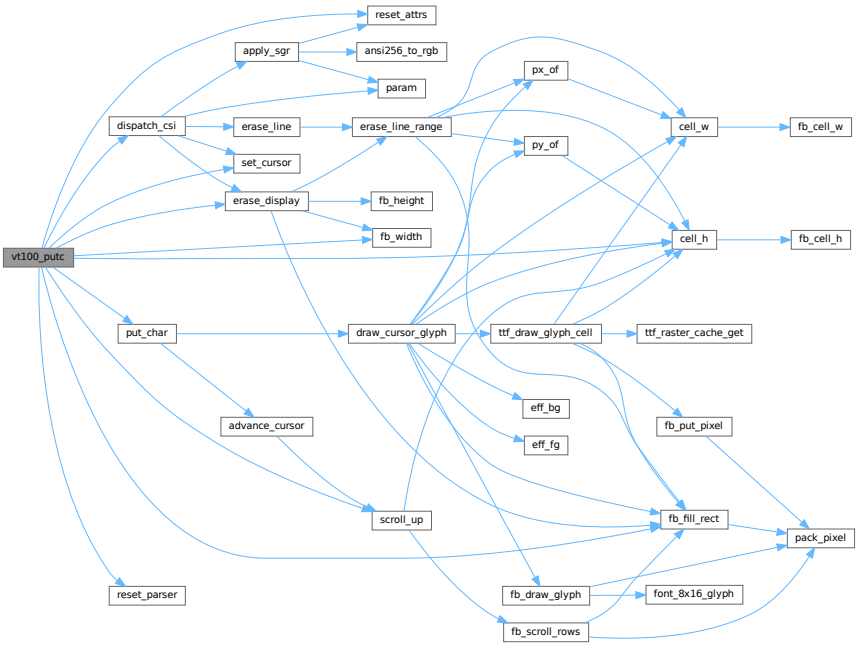
```
void vt100_putc (
            char raw )
```

Feed one byte into the terminal (handles escape sequences transparently).

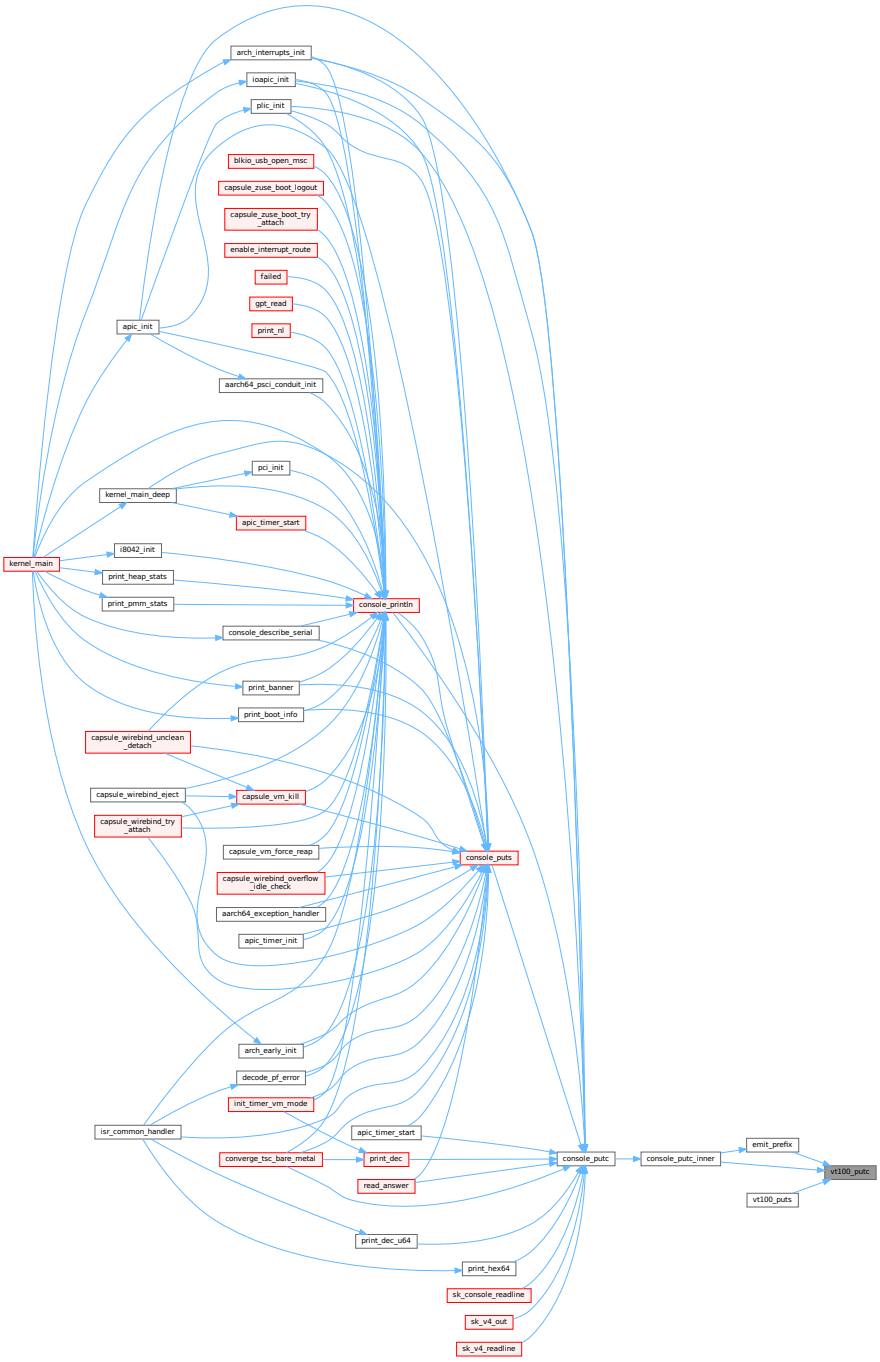
References cell_h(), VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, dispatch_csi(), erase_display(), fb_fill_rect(), fb_width(), g_vt, VT100State::initialized, VT100State::param_count, VT100State::param_has_value, VT100State::params, put_char(), reset_attrs(), reset_parser(), VT100State::rows, VT100State::saved_cx, VT100State::saved_cy, scroll_up(), set_cursor(), ST_CSI, ST_ESC, ST_ESC_CHAR, ST_ESC_HASH, ST_NORMAL, VT100State::state, and VT100_MAX_PARAMS.

Referenced by console_putc_inner(), emit_prefix(), and vt100_puts().

Here is the call graph for this function:



Here is the caller graph for this function:



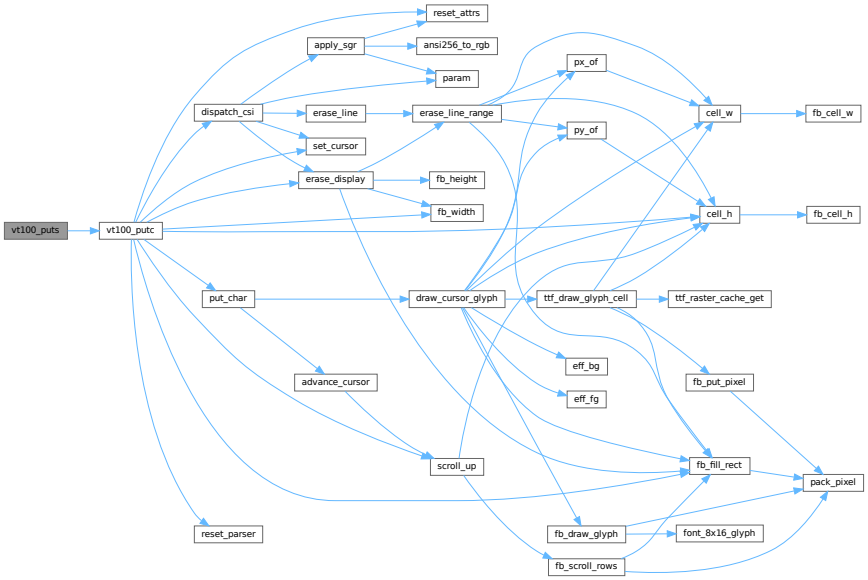
7.158.3.30 vt100_puts()

```
void vt100_puts (
            const char * s )
```

Feed a null-terminated string.

References vt100_putc().

Here is the call graph for this function:



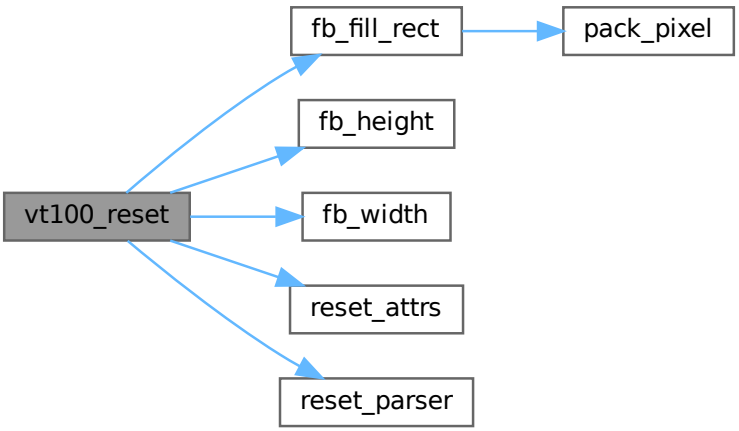
7.158.3.31 vt100_reset()

```
void vt100_reset (
            void  )
```

References VT100State::cols, VT100State::cx, VT100State::cy, VT100State::def_bg, fb_fill_rect(), fb_height(), fb_width(), g_boot_ring_count, g_boot_rows, g_line_stride, g_ring_count, g_ring_head, g_scroll_offset, g_shadow, g_terminal_visible, g_vt, VT100State::initialized, reset_attrs(), reset_parser(), VT100State::rows, VT100State::saved_cx, and VT100State::saved_cy.

Referenced by console_fb_clear().

Here is the call graph for this function:



Here is the caller graph for this function:

7.158.3.32 vt100_rows()

```
uint32_t vt100_rows (
            void  )
```

Terminal height in character rows.

References g_vt, and VT100State::rows.

7.158.3.33 vt100_scroll_back()

```
void vt100_scroll_back (
            uint32_t n )
```

FABRIC-0.md item 4.4q: move the REPL scrollback view back/forward by `n` lines and redraw.

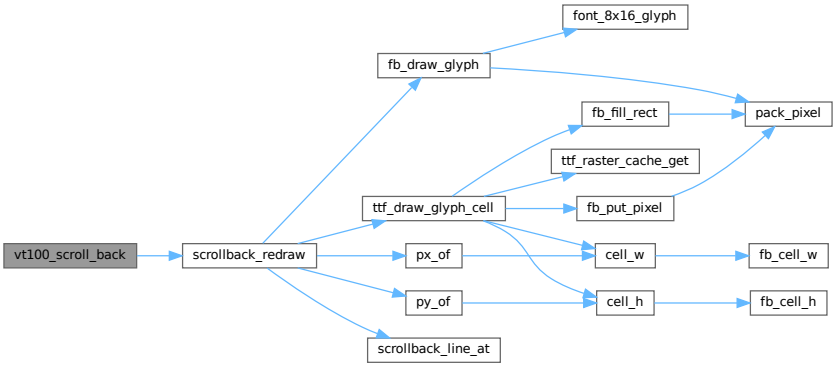
Offset 0 (the default, and where every call eventually returns to) is the live view – the same content already on screen. Clamped at both ends: back cannot pass the oldest stored line, forward cannot pass the live view. No-op (including no redraw) if the scrollback buffers failed to allocate at vt100_enable_ttf() time, or before TTF mode is active at all.

Known limitation: redraw uses the terminal's current default fg/bg, not each line's original SGR color state at the time it was printed (colors aren't stored per-cell) – text content is recovered exactly, color is not.

References g_boot_ring_count, g_boot_rows, g_ring, g_ring_count, g_scroll_offset, g_shadow, and scrollback_redraw().

Referenced by console_fb_scroll_back().

Here is the call graph for this function:



Here is the caller graph for this function:

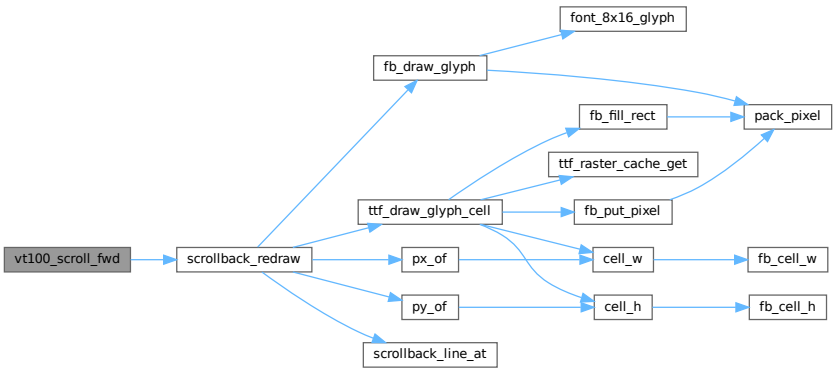
7.158.3.34 vt100_scroll_fwd()

```
void vt100_scroll_fwd (
            uint32_t n )
```

References g_ring, g_scroll_offset, g_shadow, and scrollback_redraw().

Referenced by console_fb_scroll_fwd().

Here is the call graph for this function:



Here is the caller graph for this function:

7.158.3.35 vt100_toggle_graphics()

```
void vt100_toggle_graphics (
            void  )
```

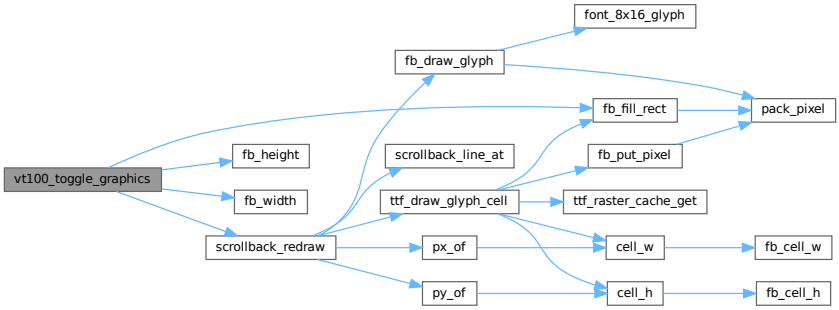
FABRIC-0.md item 4.4y-revised: toggle the full-screen vt100 terminal between visible (normal operation) and hidden (graphics mode – the terminal stops drawing, letting direct framebuffer/TTF-TEXT calls show through undisturbed).

Toggling back to visible does a full redraw of the terminal's current on-screen content, restoring it exactly as it was. No-op before TTF mode is active.

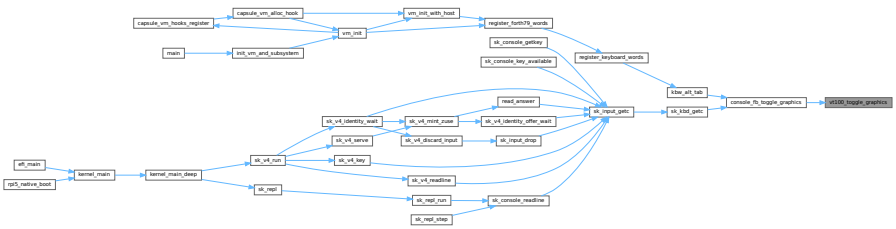
References VT100State::def_bg, fb_fill_rect(), fb_height(), fb_width(), g_glyph_mode, g_terminal_visible, g_ttf_ready, g_vt, scrollback_redraw(), and VT_GLYPH_TTF.

Referenced by console_fb_toggle_graphics().

Here is the call graph for this function:



Here is the caller graph for this function:



7.158.4 Variable Documentation

7.158.4.1 g_boot_line_stride

```
uint32_t g_boot_line_stride = 0  [static]
```

Referenced by scrollback_line_at(), and vt100_enable_ttf().

7.158.4.2 g_boot_ring

```
char* g_boot_ring = (void *)0  [static]
```

Referenced by scrollback_line_at(), and vt100_enable_ttf().

7.158.4.3 g_boot_ring_cap

```
uint32_t g_boot_ring_cap = 0  [static]
```

Referenced by scrollback_line_at(), and vt100_enable_ttf().

7.158.4.4 g_boot_ring_count

```
uint32_t g_boot_ring_count = 0  [static]
```

Referenced by scrollback_line_at(), scrollback_redraw(), vt100_enable_ttf(), vt100_reset(), and vt100_scroll_back().

7.158.4.5 g_boot_ring_head

```
uint32_t g_boot_ring_head = 0  [static]
```

Referenced by scrollback_line_at(), and vt100_enable_ttf().

7.158.4.6 g_boot_rows

```
uint32_t g_boot_rows = 0  [static]
```

Referenced by scrollback_line_at(), scrollback_redraw(), vt100_enable_ttf(), vt100_reset(), and vt100_scroll_back().

7.158.4.7 g_boot_shadow

```
char* g_boot_shadow = (void *)0  [static]
```

Referenced by scrollback_line_at(), and vt100_enable_ttf().

7.158.4.8 g_glyph_mode

```
vt_glyph_mode_t g_glyph_mode = VT_GLYPH_BITMAP  [static]
```

Referenced by cell_h(), cell_w(), draw_cursor_glyph(), scrollback_redraw(), vt100_draw_cursor(), vt100_enable_ttf(), vt100_erase_cursor(), and vt100_toggle_graphics().

7.158.4.9 g_line_stride

```
uint32_t g_line_stride = 0  [static]
```

Referenced by draw_cursor_glyph(), scroll_up(), scrollback_line_at(), vt100_enable_ttf(), vt100_init(), and vt100_reset().

7.158.4.10 g_ring

```
char* g_ring = (void *)0  [static]
```

Referenced by scroll_up(), scrollback_line_at(), vt100_enable_ttf(), vt100_init(), vt100_scroll_back(), and vt100_scroll_fwd().

7.158.4.11 g_ring_cap

```
uint32_t g_ring_cap = 0  [static]
```

Referenced by scroll_up(), scrollback_line_at(), vt100_enable_ttf(), and vt100_init().

7.158.4.12 g_ring_count

```
uint32_t g_ring_count = 0  [static]
```

Referenced by scroll_up(), scrollback_line_at(), scrollback_redraw(), vt100_enable_ttf(), vt100_init(), vt100_reset(), and vt100_scroll_back().

7.158.4.13 g_ring_head

```
uint32_t g_ring_head = 0  [static]
```

Referenced by scroll_up(), scrollback_line_at(), vt100_enable_ttf(), vt100_init(), and vt100_reset().

7.158.4.14 g_scroll_offset

```
uint32_t g_scroll_offset = 0  [static]
```

Referenced by scrollback_redraw(), vt100_enable_ttf(), vt100_reset(), vt100_scroll_back(), and vt100_scroll_fwd().

7.158.4.15 g_shadow

```
char* g_shadow = (void *)0  [static]
```

Referenced by draw_cursor_glyph(), scroll_up(), scrollback_line_at(), vt100_enable_ttf(), vt100_init(), vt100_reset(), vt100_scroll_back(), and vt100_scroll_fwd().

7.158.4.16 g_terminal_visible

```
int g_terminal_visible = 1  [static]
```

Referenced by draw_cursor_glyph(), erase_display(), erase_line_range(), scroll_up(), vt100_draw_cursor(), vt100_erase_cursor(), vt100_reset(), and vt100_toggle_graphics().

7.158.4.17 g_ttf_cache

```
ttf_raster_cache_t g_ttf_cache  [static]
```

Referenced by ttf_draw_glyph_cell(), and vt100_enable_ttf().

7.158.4.18 g_ttf_font

```
ttf_font_t g_ttf_font  [static]
```

Referenced by ttf_draw_glyph_cell(), and vt100_enable_ttf().

7.158.4.19 g_ttf_ready

```
int g_ttf_ready = 0  [static]
```

Referenced by ttf_draw_glyph_cell(), vt100_enable_ttf(), and vt100_toggle_graphics().

7.158.4.20 g_vt

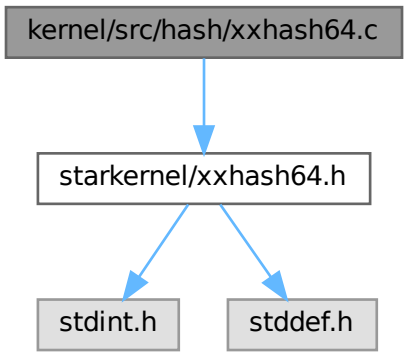
```
VT100State g_vt  [static]
```

Referenced by advance_cursor(), apply_sgr(), dispatch_csi(), draw_cursor_glyph(), eff_bg(), eff_fg(), erase_display(), erase_line(), erase_line_range(), param(), reset_attrs(), reset_parser(), scroll_up(), scrollback_line_at(), scrollback_redraw(), set_cursor(), vt100_cols(), vt100_draw_cursor(), vt100_enable_ttf(), vt100_erase_cursor(), vt100_init(), vt100_putc(), vt100_reset(), vt100_rows(), and vt100_toggle_graphics().

7.159 kernel/src/hash/xxhash64.c File Reference

```
#include "starkernel/xxhash64.h"
```

Include dependency graph for xxhash64.c:



Macros

- #define XXHASH64_PRIME1 0x9E3779B185EBCA87ULL
  xxhash64.c - Freestanding xxHash64 Implementation
- #define XXHASH64_PRIME2 0xC2B2AE3D27D4EB4FULL
- #define XXHASH64_PRIME3 0x165667B19E3779F9ULL
- #define XXHASH64_PRIME4 0x85EBCA77C2B2AE63ULL
- #define XXHASH64_PRIME5 0x27D4EB2F165667C5ULL

Functions

- static uint64_t xxh64_rotl (uint64_t x, int r)
  Rotate left 64-bit.
- static uint64_t xxh64_read64 (const uint8_t *p)
  Read 64-bit little-endian (unaligned safe)
- static uint32_t xxh64_read32 (const uint8_t *p)
  Read 32-bit little-endian (unaligned safe)
- static uint64_t xxh64_round (uint64_t acc, uint64_t input)
  Round function.
- static uint64_t xxh64_merge_round (uint64_t acc, uint64_t val)
  Merge accumulator.
- static uint64_t xxh64_avalanche (uint64_t h)
  Avalanche finalization.
- uint64_t xxhash64 (const void *data, size_t len, uint64_t seed)
  xxhash64 - Compute xxHash64 of a buffer
- void xxhash64_reset (XXHash64State *state, uint64_t seed)
  xxhash64_reset - Initialize streaming state
- void xxhash64_update (XXHash64State *state, const void *data, size_t len)
  xxhash64_update - Feed data into streaming hash
- uint64_t xxhash64_digest (const XXHash64State *state)
  xxhash64_digest - Finalize and get hash value

7.159.1 Macro Definition Documentation

7.159.1.1 XXHASH64_PRIME1

```
#define XXHASH64_PRIME1 0x9E3779B185EBCA87ULL
```

xxhash64.c - Freestanding xxHash64 Implementation

Based on xxHash by Yann Collet (BSD-2-Clause). Adapted for freestanding kernel use - no libc dependency.

xxHash is an extremely fast non-cryptographic hash algorithm. https://github.com/Cyan4973/xx↩Hash

7.159.1.2 XXHASH64_PRIME2

```
#define XXHASH64_PRIME2 0xC2B2AE3D27D4EB4FULL
```

7.159.1.3 XXHASH64_PRIME3

```
#define XXHASH64_PRIME3 0x165667B19E3779F9ULL
```

7.159.1.4 XXHASH64_PRIME4

```
#define XXHASH64_PRIME4 0x85EBCA77C2B2AE63ULL
```

7.159.1.5 XXHASH64_PRIME5

```
#define XXHASH64_PRIME5 0x27D4EB2F165667C5ULL
```

7.159.2 Function Documentation

7.159.2.1 xxh64_avalanche()

```
static uint64_t xxh64_avalanche (
            uint64_t h )  [inline], [static]
```

Avalanche finalization.

References XXHASH64_PRIME2, and XXHASH64_PRIME3.

Referenced by xxhash64(), and xxhash64_digest().

Here is the caller graph for this function:

7.159.2.2 xxh64_merge_round()

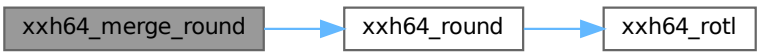
```
static uint64_t xxh64_merge_round (
            uint64_t acc,
            uint64_t val )  [inline], [static]
```

Merge accumulator.

References xxh64_round(), XXHASH64_PRIME1, and XXHASH64_PRIME4.

Referenced by xxhash64(), and xxhash64_digest().

Here is the call graph for this function:



Here is the caller graph for this function:

7.159.2.3 xxh64_read32()

```
static uint32_t xxh64_read32 (
            const uint8_t * p )  [inline], [static]
```

Read 32-bit little-endian (unaligned safe)

Referenced by xxhash64(), and xxhash64_digest().

Here is the caller graph for this function:

7.159.2.4 xxh64_read64()

```
static uint64_t xxh64_read64 (
            const uint8_t * p )  [inline], [static]
```

Read 64-bit little-endian (unaligned safe)

Referenced by xxhash64(), xxhash64_digest(), and xxhash64_update().

Here is the caller graph for this function:

7.159.2.5 xxh64_rotl()

```
static uint64_t xxh64_rotl (
            uint64_t x,
            int r )  [inline], [static]
```

Rotate left 64-bit.

Referenced by xxh64_round(), xxhash64(), and xxhash64_digest().

Here is the caller graph for this function:

7.159.2.6 xxh64_round()

```
static uint64_t xxh64_round (
            uint64_t acc,
            uint64_t input )  [inline], [static]
```

Round function.

References xxh64_rotl(), XXHASH64_PRIME1, and XXHASH64_PRIME2.

Referenced by xxh64_merge_round(), xxhash64(), xxhash64_digest(), and xxhash64_update().

Here is the call graph for this function:

Here is the caller graph for this function:

7.159.2.7 xxhash64()

```
uint64_t xxhash64 (
            const void * data,
            size_t len,
            uint64_t seed )
```

xxhash64 - Compute xxHash64 of a buffer

Parameters

data	Input data
len	Length in bytes
seed	Hash seed (use XXHASH64_CAPSULE_SEED for capsules)

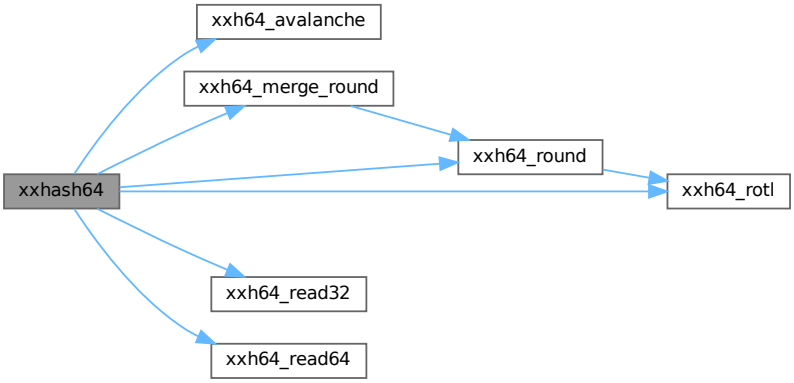
Returns

64-bit hash value

References limit, v1, v2, xxh64_avalanche(), xxh64_merge_round(), xxh64_read32(), xxh64_read64(), xxh64_rotl(), xxh64_round(), XXHASH64_PRIME1, XXHASH64_PRIME2, XXHASH64_PRIME3, XXHASH64_PRIME4, and XXHASH64_PRIME5.

Referenced by xxhash64_capsule().

Here is the call graph for this function:



Here is the caller graph for this function:

7.159.2.8 xxhash64_digest()

```
uint64_t xxhash64_digest (
            const XXHash64State * state )
```

xxhash64_digest - Finalize and get hash value

Parameters

state	Streaming state

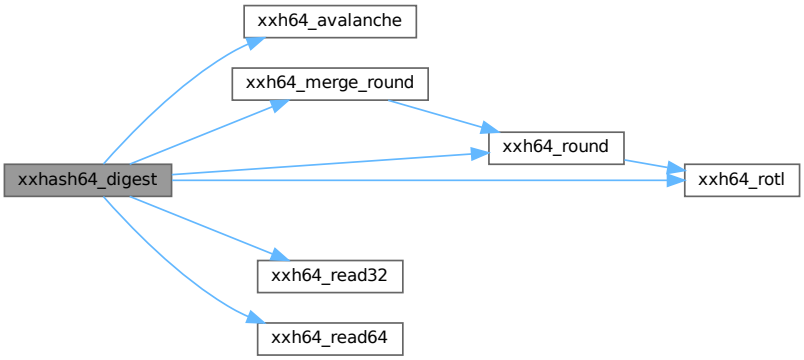
Returns

64-bit hash value

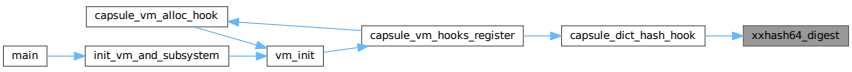
References XXHash64State::buffer, XXHash64State::buffer_size, XXHash64State::seed, XXHash64State::total_len, XXHash64State::v1, XXHash64State::v2, XXHash64State::v3, XXHash64State::v4, xxh64_avalanche(), xxh64_merge_round(), xxh64_read32(), xxh64_read64(), xxh64_rotl(), xxh64_round(), XXHASH64_PRIME1, XXHASH64_PRIME2, XXHASH64_PRIME3, XXHASH64_PRIME4, and XXHASH64_PRIME5.

Referenced by capsule_dict_hash_hook().

Here is the call graph for this function:



Here is the caller graph for this function:



7.159.2.9 xxhash64_reset()

```
void xxhash64_reset (
            XXHash64State * state,
            uint64_t seed )
```

xxhash64_reset - Initialize streaming state

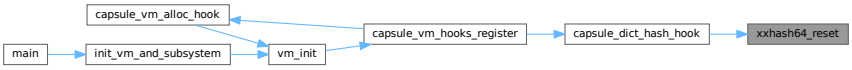
Parameters

state	State to initialize
seed	Hash seed

References XXHash64State::buffer_size, XXHash64State::seed, XXHash64State::total_len, XXHash64State::v1, XXHash64State::v2, XXHash64State::v3, XXHash64State::v4, XXHASH64_PRIME1, and XXHASH64_PRIME2.

Referenced by capsule_dict_hash_hook().

Here is the caller graph for this function:



7.159.2.10 xxhash64_update()

```
void xxhash64_update (
            XXHash64State * state,
            const void * data,
            size_t len )
```

xxhash64_update - Feed data into streaming hash

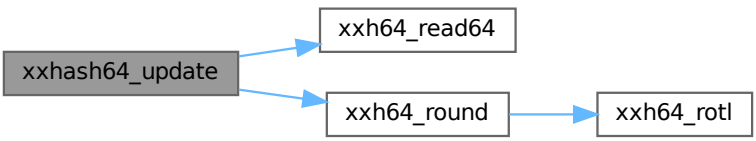
Parameters

state	Streaming state
data	Input data
len	Length in bytes

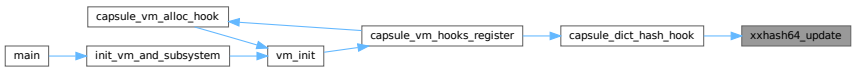
References XXHash64State::buffer, XXHash64State::buffer_size, limit, XXHash64State::total_len, XXHash64State::v1, XXHash64State::v2, XXHash64State::v3, XXHash64State::v4, xxh64_read64(), and xxh64_round().

Referenced by capsule_dict_hash_hook().

Here is the call graph for this function:



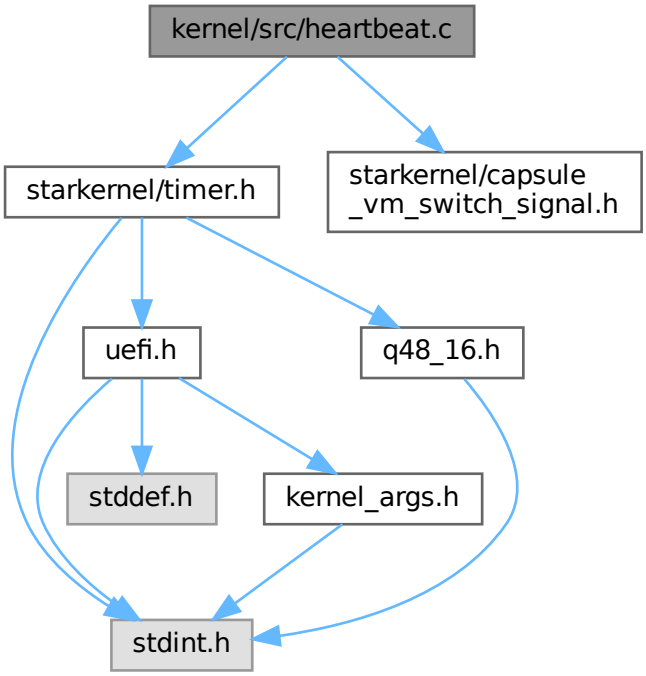
Here is the caller graph for this function:



7.160 kernel/src/heartbeat.c File Reference

```
#include "starkernel/timer.h"
#include "starkernel/capsule_vm_switch_signal.h"
```

Include dependency graph for heartbeat.c:



Macros

- #define HEARTBEAT_HAS_VARIANCE 0

  heartbeat.c - Shared M5 heartbeat / TIME-TRUST engine (punch-list item 0.8)
- #define HEARTBEAT_BASE_PERIOD_NS 10000000ULL

Functions

- void heartbeat_init (uint64_t tsc_hz, uint64_t tick_hz)

  Initialise the M5 heartbeat / TIME-TRUST subsystem.
- void heartbeat_tick (void)

  Top half.
- void heartbeat_service (void)

  Bottom half (punch-list item 0.8).
- uint64_t heartbeat_ticks (void)

  Get current TIME-TICKS (monotonic heartbeat count).
- time_trust_t heartbeat_trust (void)

  Get current TIME-TRUST (Q48.16 confidence metric).
- const TimeTrustState * heartbeat_state (void)

  Get pointer to full heartbeat state (for diagnostics).
- void heartbeat_set_adaptive_period_ns (uint64_t ns)

  Set the adaptive re-arm period (FABRIC-0.md ?26).
- uint64_t heartbeat_next_period_ns (void)

  Read the period the next hardware re-arm should use.

Variables

- static TimeTrustState g_heartbeat
- static volatile uint64_t g_pending_counter
- static volatile int g_pending_valid
- static volatile uint64_t g_adaptive_period_ns = 10000000ULL

7.160.1 Macro Definition Documentation

7.160.1.1 HEARTBEAT_BASE_PERIOD_NS

```
#define HEARTBEAT_BASE_PERIOD_NS 10000000ULL
```

7.160.1.2 HEARTBEAT_HAS_VARIANCE

```
#define HEARTBEAT_HAS_VARIANCE 0
```

heartbeat.c - Shared M5 heartbeat / TIME-TRUST engine (punch-list item 0.8)

One implementation of heartbeat_init()/heartbeat_ticks()/heartbeat_trust()/ heartbeat_state(), shared by all three architectures. Each architecture's timer.c contributes only heartbeat_read_counter() — the one thing that is genuinely per-ISA (rdtsc / rdtime / CNTPCT_EL0).

Top half / bottom half split (FABRIC-0.md ?25.1 item 0.8, per the GAP-A1 ruling in ?16.4/?18.4): heartbeat_tick() is called from interrupt context and does nothing but read the counter, bump TIME-TICKS, and latch a pending sample. heartbeat_service() runs on the mainline (the REPL idle loop) and does the window/variance/trust work. Neither one feeds patron state — the engine stays on the virtual tick per ?18.4, unchanged by this file.

Adaptive re-arm period (FABRIC-0.md ?26, ruled 2026-08-03): Loop #7 (vm_runtime.c) computes an execution-derived stable/volatile signal and calls heartbeat_set_adaptive_period_ns() with it, rescaled to this file's kernel-appropriate base (10 ms, matching the 100 Hz rate item 0.1-0.7 configured) rather than the hosted 10 ?s HEARTBEAT_TICK_NS base. Each architecture's re-arm function reads heartbeat_next_period_ns() and converts it to raw counter units instead of using a fixed constant. Single writer (mainline, via vm_tick()'s Loop #7 site), single reader (the ISR's re-arm call) — no lock needed, per ?21.1's finding that nothing here is actually concurrent on one hart.

7.160.2 Function Documentation

7.160.2.1 heartbeat_init()

```
void heartbeat_init (
            uint64_t tsc_hz,
            uint64_t tick_hz )
```

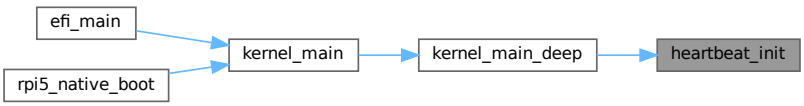
Initialise the M5 heartbeat / TIME-TRUST subsystem.

Shared across all three architectures – the fallback and calibration formula never differed between them, only what heartbeat_service() does with the resulting window did.

References time_window::count, time_window::deltas, time_trust_state::expected_delta, g_adaptive_period_ns, g_heartbeat, g_pending_valid, HEARTBEAT_BASE_PERIOD_NS, time_trust_state::last_tsc, time_window::pos, Q48_ONE, time_trust_state::ticks, TIME_WINDOW_SIZE, time_trust_state::total_samples, time_trust_state::trust, time_trust_state::variance, and time_trust_state::window.

Referenced by kernel_main_deep().

Here is the caller graph for this function:



7.160.2.2 heartbeat_next_period_ns()

```
uint64_t heartbeat_next_period_ns (
            void  )
```

Read the period the next hardware re-arm should use.

Called from interrupt context by each architecture's re-arm function (apic_timer_rearm(), riscv64_timer_rearm(), the AArch64 equivalent) in place of the fixed constant those functions used before item 0.8.

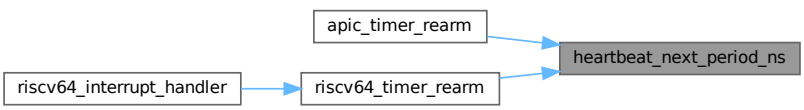
Returns

Current adaptive period in nanoseconds.

References g_adaptive_period_ns.

Referenced by apic_timer_rearm(), and riscv64_timer_rearm().

Here is the caller graph for this function:



7.160.2.3 heartbeat_service()

```
void heartbeat_service (
            void  )
```

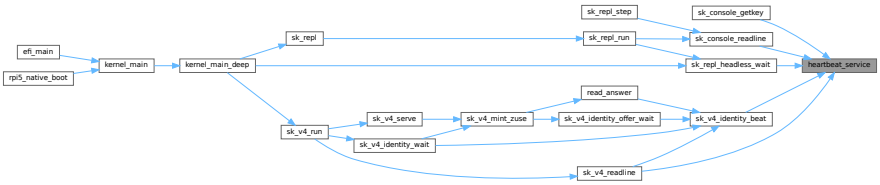
Bottom half (punch-list item 0.8).

Services one pending sample if `heartbeat_tick()` has latched one since the last call: computes the inter-tick deviation, updates the rolling window, and (architecture permitting – see `heartbeat.c`) recomputes variance and TIME-TRUST. Never runs in interrupt context. Call from the mainline, as frequently as convenient – a stale/skipped service call degrades the window's fidelity but affects nothing else, since TIME-TRUST is diagnostic only and never gates execution.

References time_trust_state::expected_delta, g_heartbeat, g_pending_counter, g_pending_valid, time_trust_state::last_tsc, Q48_ONE, time_trust_state::total_samples, time_trust_state::trust, time_trust_state::variance, and time_trust_state::window.

Referenced by sk_console_getkey(), sk_console_readline(), sk_repl_headless_wait(), sk_v4_identity_beat(), and sk_v4_readline().

Here is the caller graph for this function:



7.160.2.4 heartbeat_set_adaptive_period_ns()

```
void heartbeat_set_adaptive_period_ns (
            uint64_t ns )
```

Set the adaptive re-arm period (FABRIC-0.md ?26).

Set the adaptive re-arm period, in nanoseconds (punch-list item 0.8, FABRIC-0.md ?26).

Called from vm_runtime.c's Loop #7 site on the mainline execution path (never interrupt context) with a value already rescaled to this file's HEARTBEAT_BASE_PERIOD_NS. Clamped defensively to the same [1/4x, 4x] band Loop #7 itself enforces around its own base, so a caller forwarding an unclamped or wrongly-scaled value cannot run the ISR away.

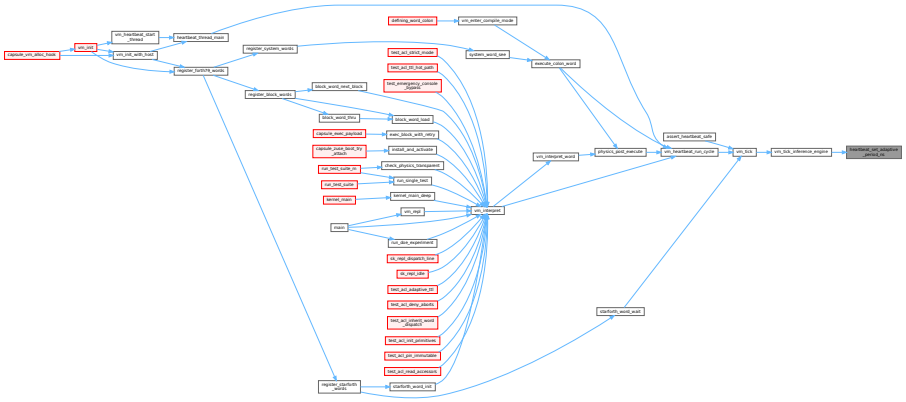
Parameters

ns	Desired period in nanoseconds for the next re-arm.

References g_adaptive_period_ns, and HEARTBEAT_BASE_PERIOD_NS.

Referenced by vm_tick_inference_engine().

Here is the caller graph for this function:



7.160.2.5 heartbeat_state()

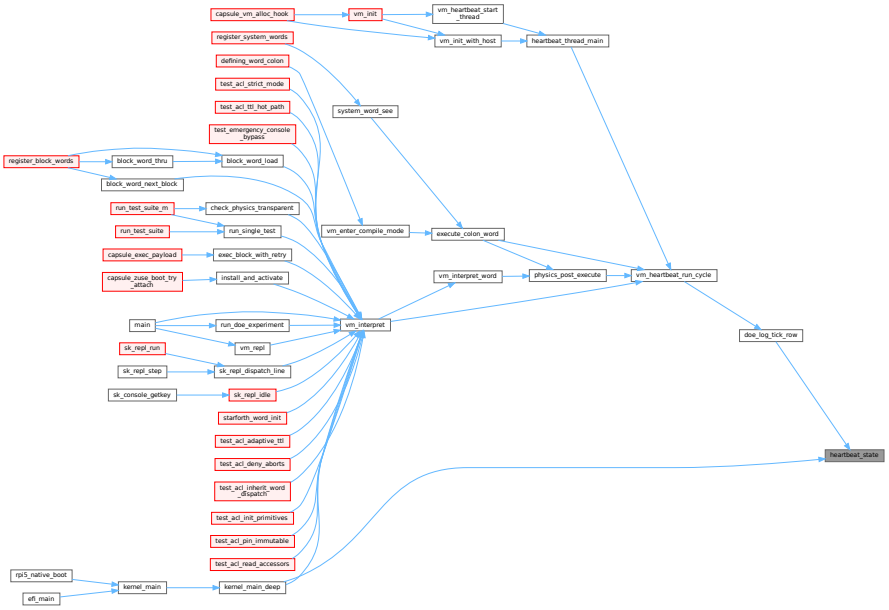
```
const TimeTrustState * heartbeat_state (
            void  )
```

Get pointer to full heartbeat state (for diagnostics).

References g_heartbeat.

Referenced by doe_log_tick_row(), and kernel_main_deep().

Here is the caller graph for this function:



7.160.2.6 heartbeat_tick()

```
void heartbeat_tick (
            void  )
```

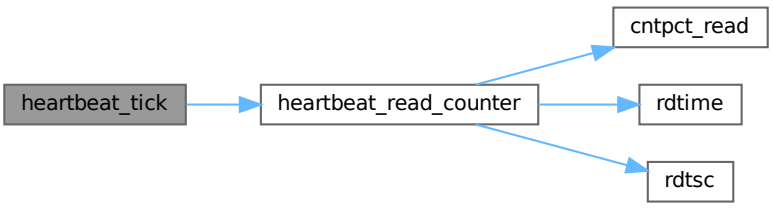
Top half.

Called directly from each architecture's ISR (punch-list item 0.8) – one call site per architecture, unchanged from before this item. Does exactly three things: reads the raw counter via `heartbeat_read_counter()`, increments TIME-TICKS, and latches the sample for `heartbeat_service()` to pick up. No window math, no variance, no loops – this must stay cheap enough for interrupt context.

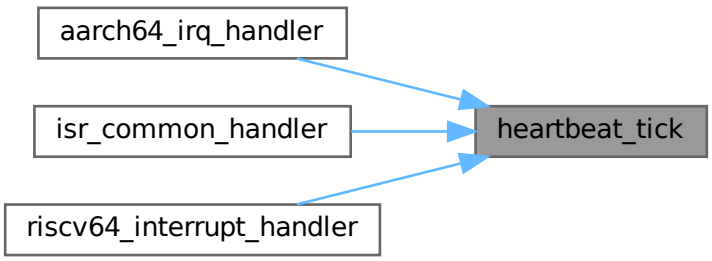
References g_heartbeat, g_pending_counter, g_pending_valid, heartbeat_read_counter(), and time_trust_state::ticks.

Referenced by aarch64_irq_handler(), isr_common_handler(), and riscv64_interrupt_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.160.2.7 heartbeat_ticks()

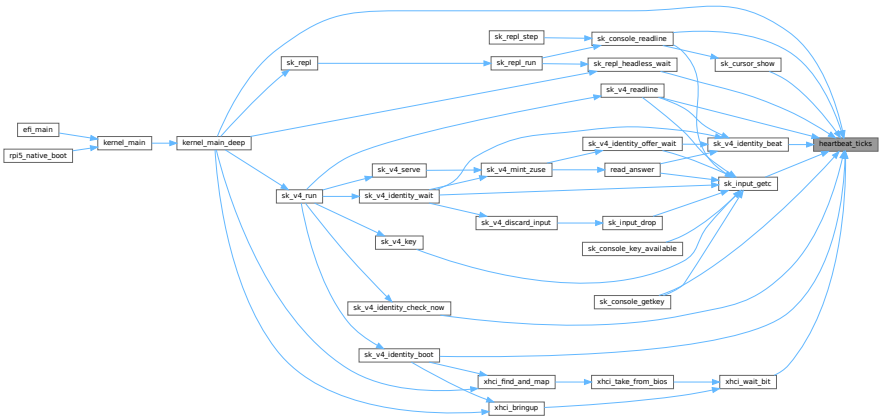
```
uint64_t heartbeat_ticks (
            void  )
```

Get current TIME-TICKS (monotonic heartbeat count).

References g_heartbeat, and time_trust_state::ticks.

Referenced by kernel_main_deep(), sk_console_getkey(), sk_console_readline(), sk_cursor_show(), sk_input_getc(), sk_repl_headless_wait(), sk_v4_identity_beat(), sk_v4_identity_boot(), sk_v4_identity_check_now(), sk_v4_readline(), and xhci_wait_bit().

Here is the caller graph for this function:



7.160.2.8 heartbeat_trust()

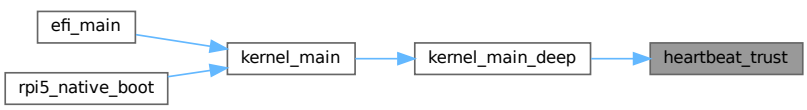
```
time_trust_t heartbeat_trust (
            void  )
```

Get current TIME-TRUST (Q48.16 confidence metric).

References g_heartbeat, and time_trust_state::trust.

Referenced by kernel_main_deep().

Here is the caller graph for this function:



7.160.3 Variable Documentation

7.160.3.1 g_adaptive_period_ns

```
volatile uint64_t g_adaptive_period_ns = 10000000ULL  [static]
```

Referenced by heartbeat_init(), heartbeat_next_period_ns(), and heartbeat_set_adaptive_period_ns().

7.160.3.2 g_heartbeat

```
TimeTrustState g_heartbeat  [static]
```

Referenced by heartbeat_init(), heartbeat_service(), heartbeat_state(), heartbeat_tick(), heartbeat_ticks(), and heartbeat_trust().

7.160.3.3 g_pending_counter

```
volatile uint64_t g_pending_counter  [static]
```

Referenced by heartbeat_service(), and heartbeat_tick().

7.160.3.4 g_pending_valid

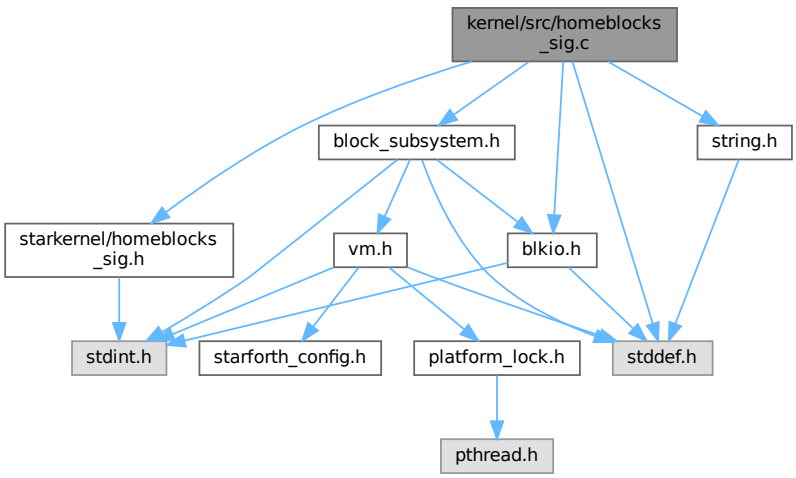
```
volatile int g_pending_valid  [static]
```

Referenced by heartbeat_init(), heartbeat_service(), and heartbeat_tick().

7.161 kernel/src/homeblocks_sig.c File Reference

```
#include "starkernel/homeblocks_sig.h"
#include <stddef.h>
#include <string.h>
#include "blkio.h"
#include "block_subsystem.h"
```

Include dependency graph for homeblocks_sig.c:



Functions

- uint64_t homeblocks_sig_compute_crc (const homeblocks_sig_t *sig)

  homeblocks_sig.c - Drive signature check (FABRIC-2.md, Milestone 4).
- homeblocks_sig_result_t homeblocks_sig_check (struct blkio_dev *dev, uint32_t sig_start_fblock, homeblocks_sig_t *out_sig)

7.161.1 Function Documentation

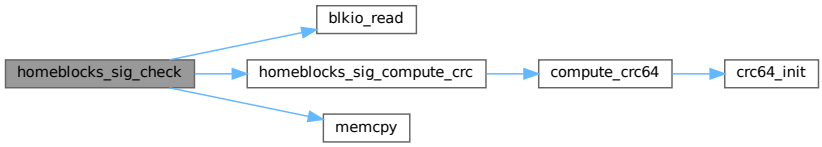
7.161.1.1 homeblocks_sig_check()

```
homeblocks_sig_result_t homeblocks_sig_check (
            struct blkio_dev * dev,
            uint32_t sig_start_fblock,
            homeblocks_sig_t * out_sig )
```

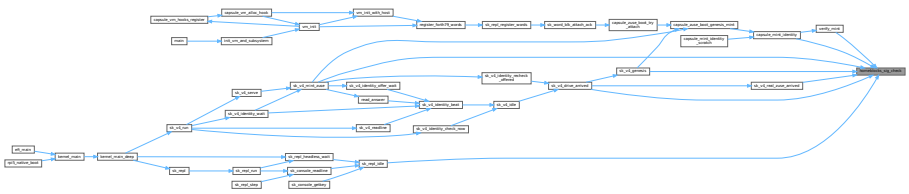
References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_read(), homeblocks_sig_t::hdr_crc, HOMEBLOCKS_SIG_BAD_CRC, HOMEBLOCKS_SIG_BAD_VERSION, HOMEBLOCKS_SIG_BLANK, homeblocks_sig_compute_crc(), HOMEBLOCKS_SIG_GET_M HOMEBLOCKS_SIG_GET_VERSION, HOMEBLOCKS_SIG_MAGIC, HOMEBLOCKS_SIG_OK, HOMEBLOCKS_SIG_READ_ERR HOMEBLOCKS_SIG_VERSION_0, homeblocks_sig_t::magic, and memcpy().

Referenced by capsule_mint_identity(), sk_repl_idle(), sk_v4_drive_arrived(), sk_v4_genesis(), sk_v4_mint_zuse(), sk_v4_real_zuse_arrived(), and verify_mint().

Here is the call graph for this function:



Here is the caller graph for this function:



7.161.1.2 homeblocks_sig_compute_crc()

```
uint64_t homeblocks_sig_compute_crc (
            const homeblocks_sig_t * sig )
```

homeblocks_sig.c - Drive signature check (FABRIC-2.md, Milestone 4).

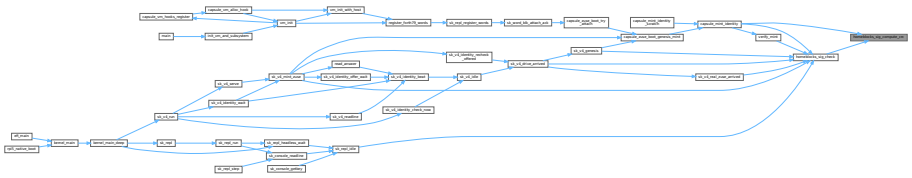
See starkernel/homeblocks_sig.h for the format and interface design.

References compute_crc64().

Referenced by capsule_mint_identity(), and homeblocks_sig_check().

Here is the call graph for this function:

Here is the caller graph for this function:

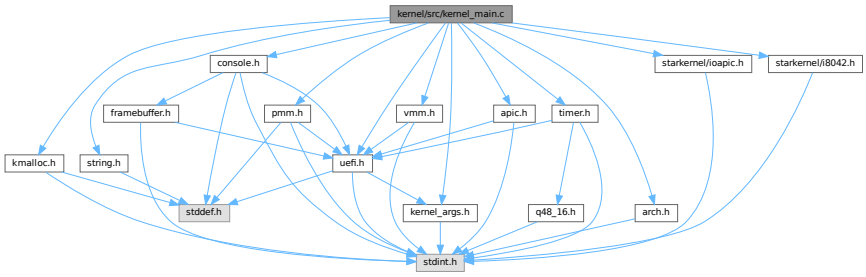


7.162 kernel/src/kernel_main.c File Reference

```
#include <string.h>
#include "uefi.h"
#include "console.h"
#include "arch.h"
#include "pmm.h"
#include "vmm.h"
#include "apic.h"
#include "timer.h"
#include "starkernel/ioapic.h"
#include "starkernel/i8042.h"
#include "kmalloc.h"
```

```
#include "starkernel/kernel_args.h"
```

Include dependency graph for kernel_main.c:



Functions

- static void kernel_main_deep (BootInfo *boot_info)

  Deep kernel initialisation — M5 heartbeat, M7 VM bootstrap, and REPL.
- static int is_ram_type (uint32_t type)

  Return non-zero if the EFI memory type represents usable or reclaimable RAM.
- static void itoa_simple (uint64_t value, char *buf, int base)

  Convert a `uint64_t` to a NUL-terminated string in the given base.
- static void print_uint (const char *label, uint64_t value)

  Print an optional label followed by a `uint64_t` in decimal to the console.
- static void print_boot_info (BootInfo *boot_info)

  Print a boot-information summary from the UEFI memory map to the console.
- static void print_pmm_stats (void)

  Print Physical Memory Manager statistics to the kernel console.
- static void print_heap_stats (void)

  Print kernel heap (kmalloc) statistics to the kernel console.
- static void print_banner (void)

  Print the StarKernel ASCII-art banner and build metadata to the console.
- void kernel_main (BootInfo *boot_info)

  Main kernel entry point after UEFI handoff — executes milestones M0–M6.

Variables

- EFI_RUNTIME_SERVICES * g_sk_runtime_services = NULL

  kernel_main.c - StarKernel main entry point (LithosAnanke branch)

7.162.1 Function Documentation

7.162.1.1 is_ram_type()

```
static int is_ram_type (
            uint32_t type )  [static]
```

Return non-zero if the EFI memory type represents usable or reclaimable RAM.

Covers all UEFI memory types that either are immediately usable by the PMM or can be reclaimed once Boot Services have exited:

- EfiConventionalMemory — general purpose RAM.
- EfiLoaderCode / EfiLoaderData — UEFI loader pages (reclaimed post-EBS).
- EfiBootServicesCode / EfiBootServicesData — boot-service pages (reclaimed post-EBS).
- EfiRuntimeServicesCode / EfiRuntimeServicesData — pages the firmware still uses for runtime calls (kept mapped, counted as physical RAM).
- EfiACPIReclaimMemory — ACPI tables; may be freed after OS has parsed them.
- EfiACPIMemoryNVS — non-volatile ACPI storage; kept reserved but is RAM.

Returns 0 for all device-memory, MMIO, persistent-memory, and special types. Used by print_boot_info() to compute the total physical RAM visible in the EFI memory map.

Parameters

type	EFI_MEMORY_TYPE value from an EFI_MEMORY_DESCRIPTOR.

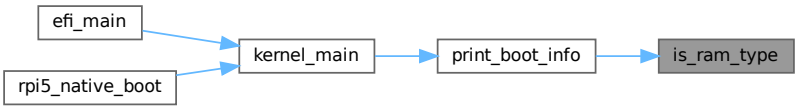
Returns

Non-zero if the type is RAM; 0 otherwise.

References EfiACPIMemoryNVS, EfiACPIReclaimMemory, EfiBootServicesCode, EfiBootServicesData, EfiConventionalMemory, EfiLoaderCode, EfiLoaderData, EfiRuntimeServicesCode, and EfiRuntimeServicesData.

Referenced by print_boot_info().

Here is the caller graph for this function:



7.162.1.2 itoa_simple()

```
static void itoa_simple (
            uint64_t value,
            char * buf,
            int base )  [static]
```

Convert a uint64_t to a NUL-terminated string in the given base.

Produces a freestanding (no libc) integer-to-string conversion for the kernel console paths. Handles bases 2–16; digits above 9 are lowercase alphabetic ('a'-'f' for hex). Special case: value == 0 writes the string "0" and returns immediately.

The algorithm builds the digit string in reverse order into a 64-byte local temp[] buffer, then reverses it into buf. buf must be at least 65 bytes to hold a 64-bit binary string plus NUL; in practice all callers pass 64-byte buffers and use base 10 or 16, where the maximum length is 20 or 16 digits respectively.

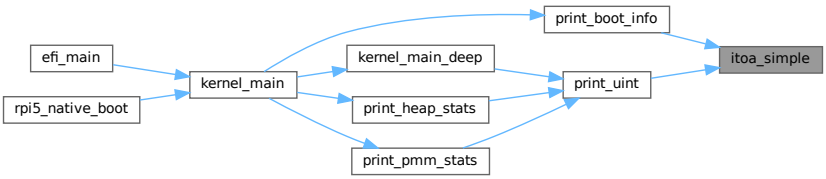
Parameters

value	Non-negative integer to convert.
buf	Caller-allocated output buffer (minimum 65 bytes for binary).
base	Numeric base (2–16).

References base.

Referenced by print_boot_info(), and print_uint().

Here is the caller graph for this function:



7.162.1.3 kernel_main()

```
void kernel_main (
            BootInfo * boot_info )
```

Main kernel entry point after UEFI handoff — executes milestones M0–M6.

On amd64 and riscv64, `kernel_entry.S` switches the stack from UEFI's default to a 2 MiB zero-initialised BSS stack and tail-calls this function as `kernel_main_impl`. On aarch64 the assembly trampoline is not yet implemented and the UEFI loader calls `kernel_main` directly.

Milestone sequence:

- **M0 — Architecture early init** (`arch_early_init()`): On amd64, installs a minimal GDT with a proper 64-bit code segment at selector 0x08 and reloads CS via `lretq`. Without this, UEFI's 64-bit segment at 0x38 is in scope and the ISR's `INT` gate (which expects CS 0x08) would fault silently.
- **M1 — Console** (`console_init()`): brings up UART 16550 at 115200 8N1 and the framebuffer VT100 terminal. Then prints banner and memory map.
- **M2 — PMM** (`pmm_init()`): initialises the physical memory manager's 4 KiB page bitmap from the EFI memory map.
- **M3 — VMM** (`vmm_init()`): builds 4-level x86-64 page tables, maps all conventional RAM at the kernel virtual base, and loads CR3.
- **M4 — IDT + APIC** (`arch_interrupts_init()` + `apic_init()`): programs the 64-entry IDT, masks the legacy 8259A PIC, and initialises the Local APIC in xAPIC MMIO mode at 0xFEE00000.
- **M5 — Timer** (`timer_init()`): calibrates the TSC and HPET.
- **M6 — Heap** (`kmalloc_init()`): initialises the kernel slab allocator with `heap_size` from the boot args or `KARGS_DEFAULT_HEAP_SIZE` (2 GiB).

Stashes `boot_info->runtime_services` in `g_sk_runtime_services` for later use by kernel FORTH words (e.g. `REBOOT`). Then tail-calls `kernel_main_deep()` for M7 and the REPL.

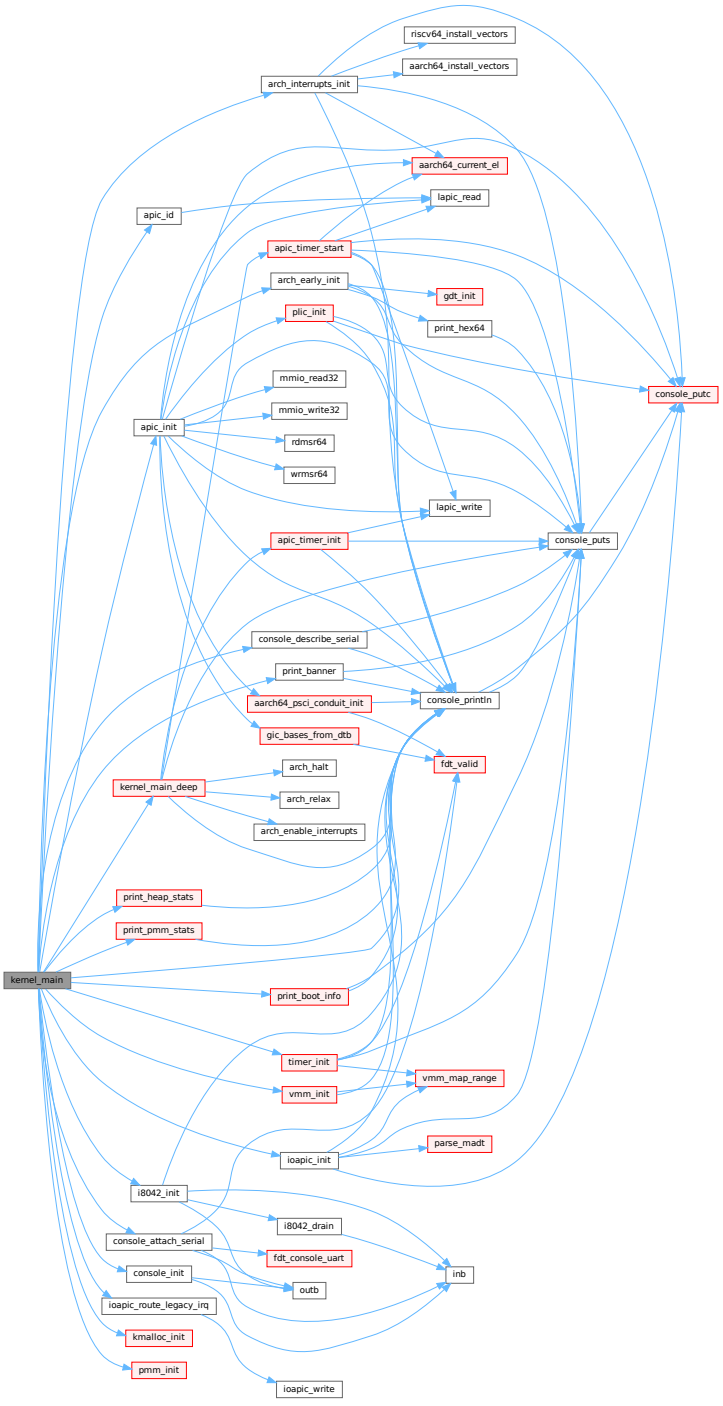
Parameters

boot_info	`BootInfo` populated by `uefi_loader.c` before `ExitBootServices()`; provides the memory map, ACPI pointer, framebuffer descriptor, runtime services pointer, and parsed kernel command-line arguments.

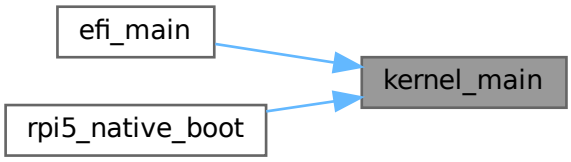
References BootInfo::acpi_table, apic_id(), apic_init(), arch_early_init(), arch_interrupts_init(), BootInfo::args, console_attach_serial(), console_describe_serial(), console_init(), console_println(), BootInfo::dtb, g_sk_runtime_services, KernelArgs::heap_size, i8042_init(), I8042_KEYBOARD_VECTOR, ioapic_init(), ioapic_route_legacy_irq(), KARGS_DEFAULT_HEAP_SIZE, kernel_main_deep(), kmalloc_init(), pmm_init(), print_banner(), print_boot_info(), print_heap_stats(), print_pmm_stats(), BootInfo::runtime_services, timer_init(), and vmm_init().

Referenced by efi_main(), and rpi5_native_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.162.1.4 kernel_main_deep()

```
static void kernel_main_deep (
            BootInfo * boot_info )  [static]
```

Deep kernel initialisation — M5 heartbeat, M7 VM bootstrap, and REPL.

Called as the final act of `kernel_main_impl()` / `kernel_main()` after all hardware milestones M0–M6 are complete. Runs on the 2 MiB BSS stack on amd64 (set up by `kernel_entry.S` before `kernel_main_impl()` was called) or the UEFI-provided stack on aarch64 and riscv64.

M5 — Heartbeat subsystem: Calls `apic_timer_init(tsc_hz, 100)` to configure the APIC timer for 100 Hz periodic delivery to vector 32, then `heartbeat_init(tsc_hz, 100)` to initialise the rolling-window heartbeat state.

M7 — VM bootstrap (when `STARFORTH_ENABLE_VM` is defined):

1. `sk_vm_bootstrap_parity()` — allocates the Mama VM and validates the capsule directory parity.
2. Allocates 1 MiB `blk_ram_buf` (LBN 0–991) and 1 MiB `krd_buf` (LBN 2048–3071 = capsule ramdrive) from `kmalloc`, then calls `capsule_blk_init()` to wire them into the Mama VM's block subsystem.
3. Copies the read-only `capsule_arena` to heap and calls `capsule_exec_init()` to load and execute `init.4th`.
4. Pins `CAPSULE-BIRTH` and `BIRTH` with `ACL_MODE_STRICT` via `vm_find_word()` so that ACL policy cannot downgrade them.

After M7, the APIC timer is started via `apic_timer_start()` and `arch_enable_interrupts()` enables IRQs.

REPL (when `STARFORTH_ENABLE_VM` is defined):

- If `boot_info->args.run_doe` is set, injects `"12345 3 EXEC-DOE BYE"` before the interactive REPL.
- If `SK_STARTUP_FORTH` is defined at build time, executes it as a compile-time startup script (lowest priority — overridden by `--doe`).
- Activates the framebuffer VT100 terminal (if the framebuffer descriptor is valid) so the REPL output appears on screen as well as the serial port.
- Clears the `StarForthRebootTries` NVRAM variable to signal a clean boot.
- Calls `sk_repl()` — the interactive FORTH REPL loop. Returns when the user executes `BYE` or `vm->halted` is set.

Terminates with an infinite `arch_halt()` idle loop regardless of the `STARFORTH_ENABLE_VM` build configuration.

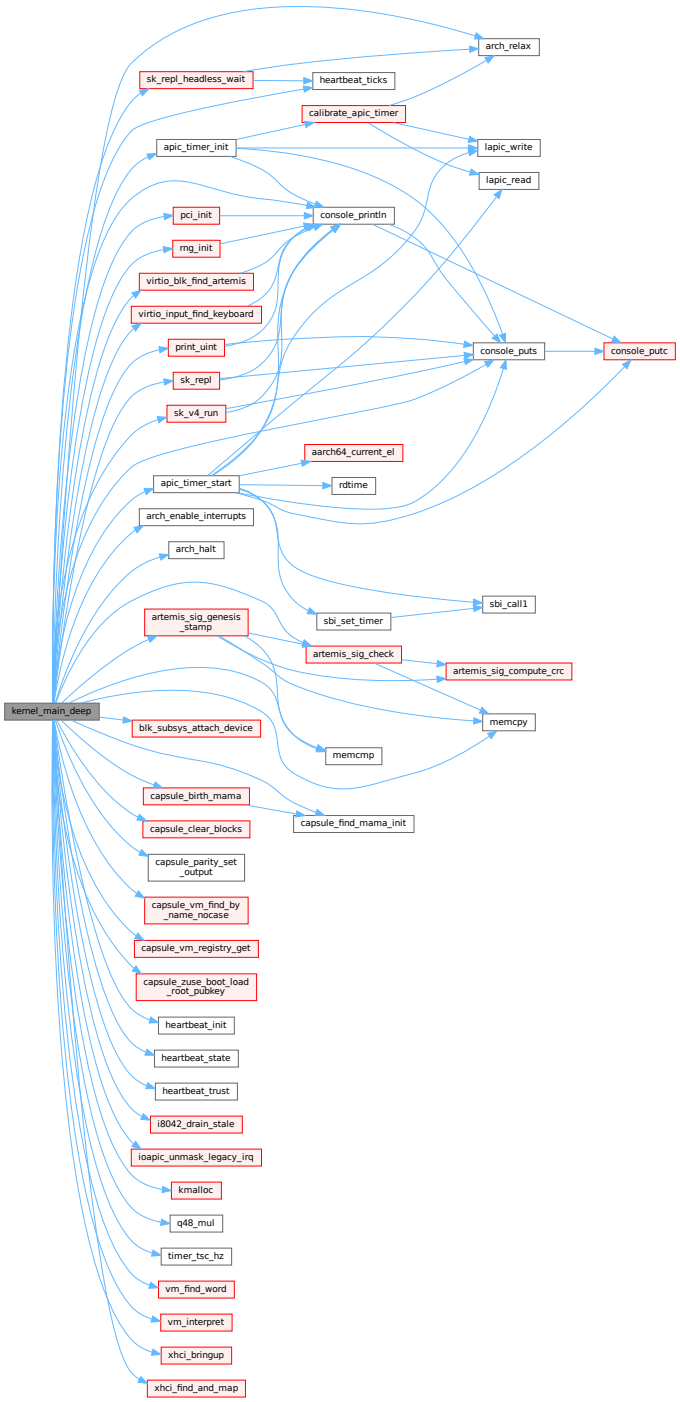
Parameters

boot_info	The `BootInfo` passed from `kernel_main_impl()`.

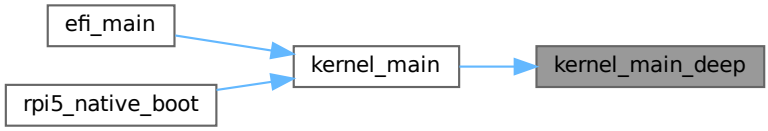
References DictEntry::acl_mode, ACL_MODE_STRICT, DictEntry::acl_pinned, BootInfo::acpi_table, apic_timer_init(), apic_timer_start(), arch_enable_interrupts(), arch_halt(), arch_relax(), CapsuleDirHeader::arena_base, CapsuleDirHeader::arena_siz BootInfo::args, ARTEMIS_SIG_BLANK, artemis_sig_check(), artemis_sig_genesis_stamp(), blk_subsys_attach_device(), BootInfo::boot_disk, capsule_birth_mama(), capsule_clear_blocks(), capsule_find_mama_init(), capsule_parity_set_output(), CAPSULE_RUN_OK, capsule_vm_find_by_name_nocase(), capsule_vm_registry_get(), capsule_zuse_boot_load_root_pubkey(), CapsuleDesc, console_println(), console_puts(), CapsuleDirHeader::desc_capacity, CapsuleDirHeader::desc_count, CapsuleDirHeader::dir_hash, EFI_VARIABLE_BOOTSERVICE_ACCESS, EFI_VARIABLE_NON_VOLATILE, EFI_VARIABLE_RUNTIME_ACCESS, entry, VM::error, g_sk_runtime_services, VM::halted, heartbeat_init(), heartbeat_state(), heartbeat_ticks(), heartbeat_trust(), i8042_drain_stale(), ioapic_unmask_legacy_irq(), BootDisk::is_stick, KARGS_SYSTEM_DEFAULT, KARGS_SYSTEM_V3, KARGS_SYSTEM_V4, kmalloc(), CapsuleDirHeader::magic, memcmp(), memcpy(), CapsuleDirHeader::name_count, pci_init(), print_uint(), q48_mul(), Q48_ONE, CapsuleDirHeader::reserved, rng_init(), KernelArgs::run_doe, _EFI_RUNTIME_SERVICES::SetVariable, SF_VAR_REBOOT_TRIES, sk_repl(), sk_repl_headless_wait(), sk_v4_run(), STARFORTH_VENDOR_GUID, VMRegistryEntry::state, KernelArgs::system, timer_tsc_hz(), time_trust_state::variance, virtio_blk_find_artemis(), virtio_input_find_keyboard(), vm_find_word(), VMRegistryEntry::vm_id, vm_interpret(), VMRegistryEntry::vm_ptr, VM_STATE_LIVE, xhci_bringup(), xhci_find_and_map(), and XHCI_MAX_CONTROLLERS.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.162.1.5 print_banner()

```
static void print_banner (
            void  )  [static]
```

Print the StarKernel ASCII-art banner and build metadata to the console.

Emits:

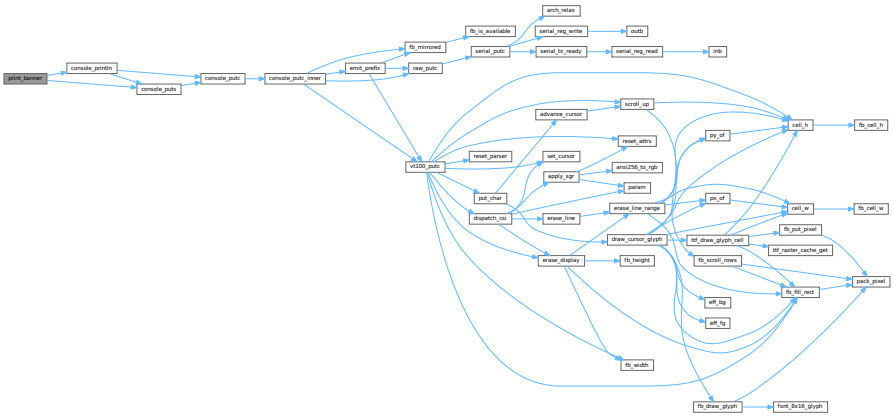
- The "StarKernel" ASCII-art logotype (six-line block font).
- `LITHOS_VERSION_STR` — the `LithosAnanke` version string from `version.h`.
- Target ISA: "amd64", "aarch64", "riscv64", or "unknown", selected by compile-time `ARCH_*` / `__riscv` preprocessor guards.
- Build date and time from `__DATE__` / `__TIME__` (compiler intrinsics).
- "UEFI BootServices: EXITED" — confirmation that the kernel is running after `ExitBootServices()` and owns all hardware.

Called first in `kernel_main_impl()` / `kernel_main()` after `console_init()` so the banner is the first visible output on the serial port, matching the `QEMU_BASELINE.log` reference.

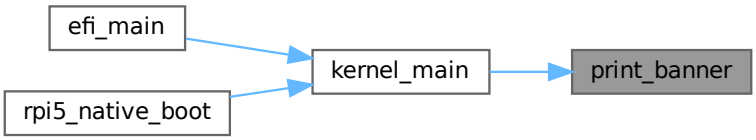
References console_println(), console_puts(), and LITHOS_VERSION_STR.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.162.1.6 print_boot_info()

```
static void print_boot_info (
            BootInfo * boot_info )  [static]
```

Print a boot-information summary from the UEFI memory map to the console.

Iterates over every EFI_MEMORY_DESCRIPTOR in boot_info->memory_map and accumulates:

- total_memory — sum of page sizes for all RAM-type regions (via is_ram_type()).
- usable_memory — sum of page sizes for EfiConventionalMemory only.

Emits a three-field report box to the kernel serial console:

- "Memory map entries: N"
- "Total memory: N MB" (rounds down to whole MiB)
- "Usable memory: N MB"

Called from kernel_main_impl() / kernel_main() immediately after M1 console initialisation, so the memory map must still be intact (it always is — the map was captured by uefi_loader.c before ExitBoot↩Services()).

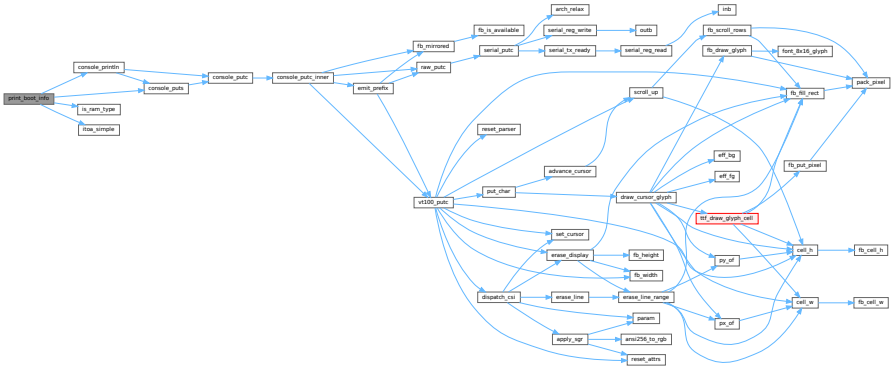
Parameters

boot_info	BootInfo structure populated by uefi_loader.c; provides memory_map, memory_map_size, and memory_map_descriptor_size.

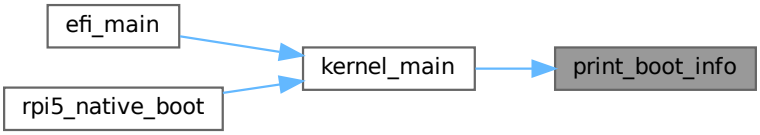
References console_println(), console_puts(), EfiConventionalMemory, is_ram_type(), itoa_simple(), BootInfo::memory_map, BootInfo::memory_map_descriptor_size, BootInfo::memory_map_size, EFI_MEMORY_DESCRIPTOR::NumberOfPages, and EFI_MEMORY_DESCRIPTOR::Type.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.162.1.7 print_heap_stats()

```
static void print_heap_stats (
            void  )  [static]
```

Print kernel heap (kmalloc) statistics to the kernel console.

Calls `kmalloc_get_stats()` to obtain a `kmalloc_stats_t` snapshot and emits four lines via `print_uint()`:

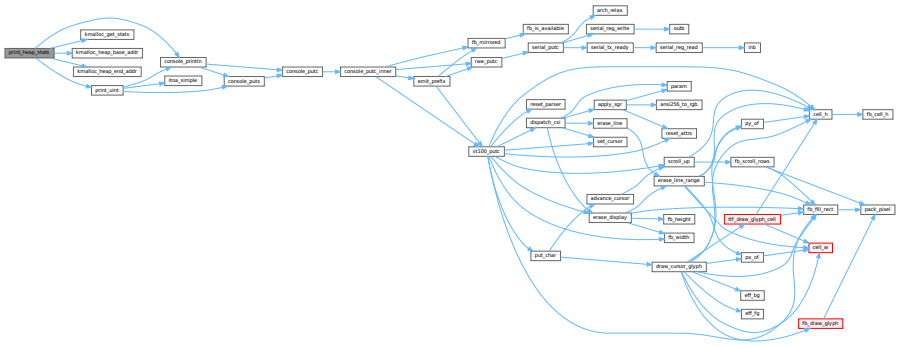
- Total bytes allocated to the heap arena.
- Free bytes currently available.
- Used bytes currently allocated by callers.
- Peak bytes — the high-water mark since `kmalloc_init()`.

Called from `kernel_main_impl()` / `kernel_main()` immediately after `kmalloc_init()` (M6) to confirm that the heap was sized correctly from the `--heap=` boot argument or its 2 GiB default.

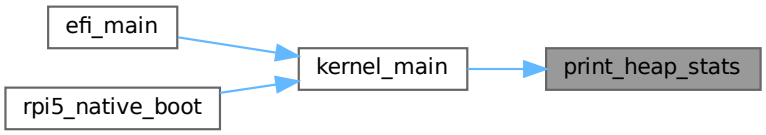
References console_println(), kmalloc_stats_t::free_bytes, kmalloc_get_stats(), kmalloc_heap_base_addr(), kmalloc_heap_end_addr(), kmalloc_stats_t::peak_bytes, print_uint(), kmalloc_stats_t::total_bytes, and kmalloc_stats_t::used_bytes.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.162.1.8 print_pmm_stats()

```
static void print_pmm_stats (
            void  )  [static]
```

Print Physical Memory Manager statistics to the kernel console.

Calls `pmm_get_stats()` to obtain a `pmm_stats_t` snapshot and then emits six lines via `print_uint()`:

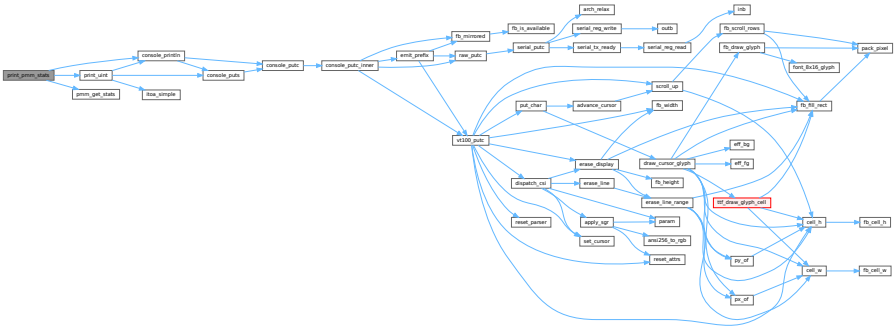
- Total pages, free pages, used pages (in 4 KiB page units).
- Total MB, free MB, used MB (bytes ÷ 1 MiB, truncated).

Called from `kernel_main_impl()` / `kernel_main()` immediately after `pmm_init()` completes (M2), providing a sanity check that the PMM saw the expected quantity of physical RAM.

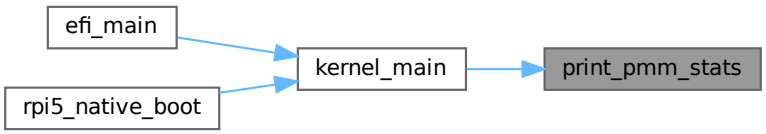
References console_putln(), pmm_stats_t::free_bytes, pmm_stats_t::free_pages, pmm_get_stats(), print_uint(), pmm_stats_t::total_bytes, pmm_stats_t::total_pages, pmm_stats_t::used_bytes, and pmm_stats_t::used_pages.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.162.1.9 print_uint()

```
static void print_uint (
            const char * label,
            uint64_t value )  [static]
```

Print an optional label followed by a `uint64_t` in decimal to the console.

Converts `value` to a decimal string via `itoa_simple()` and emits it with a trailing newline via `console_println()`. If `label` is non-NULL, it is emitted first via `console_puts()` (no newline between label and value). Used by `print_pmm_stats()` and `print_heap_stats()` to avoid repeating the convert-and-print pattern for each statistic line.

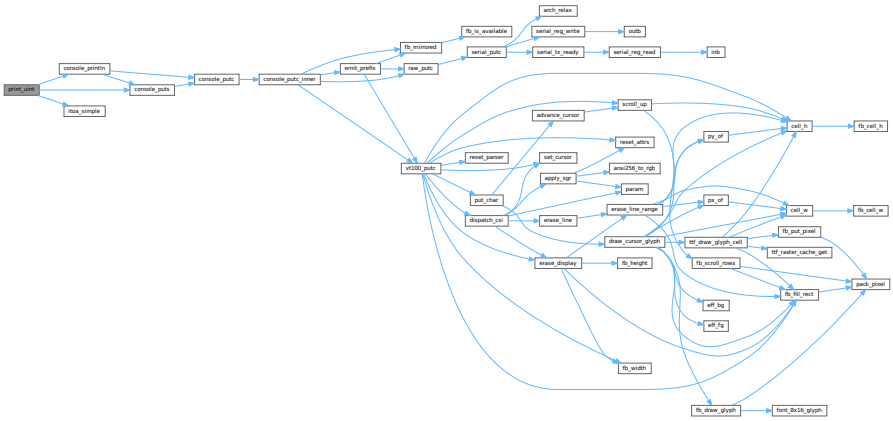
Parameters

label	Optional NUL-terminated prefix string; NULL to omit.
value	64-bit unsigned integer to display in decimal.

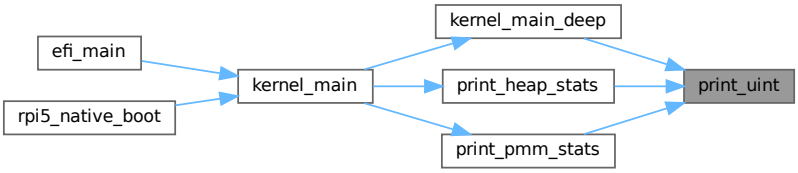
References console_println(), console_puts(), and itoa_simple().

Referenced by kernel_main_deep(), print_heap_stats(), and print_pmm_stats().

Here is the call graph for this function:



Here is the caller graph for this function:



7.162.2 Variable Documentation

7.162.2.1 g_sk_runtime_services

```
EFI_RUNTIME_SERVICES* g_sk_runtime_services = NULL
```

kernel_main.c - StarKernel main entry point (LithosAnanke branch)

Milestone status: M0-M5: Complete (build, boot, PMM, VMM, interrupts, timer) M6: Infrastructure present (kmalloc exists, validation deferred) M7: Not started (VM integration pending)

UEFI Runtime Services pointer — set once at M6 init, valid for kernel lifetime.

Populated from `boot_info->runtime_services` just before the kernel heap is initialised (between the M5 timer init and `kernel_main_deep()`). Declared `extern` in the UEFI header so kernel FORTH words (e.g. `REBOOT`) can access it without including the full `kernel_main.c` translation unit.

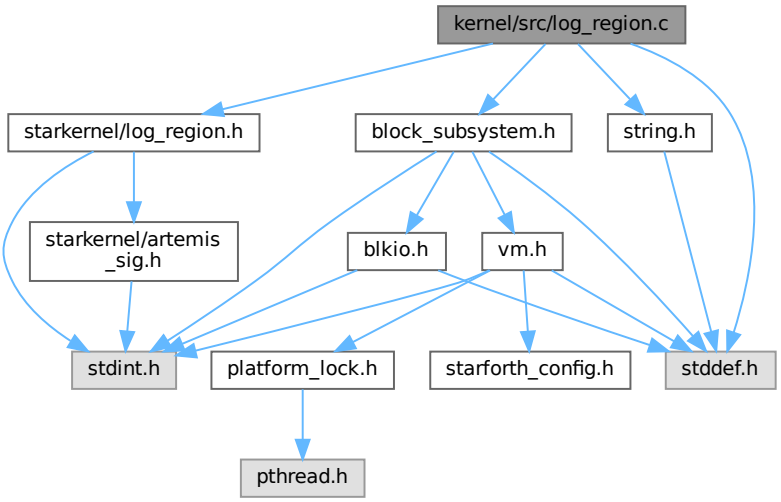
Validity note: UEFI Runtime Services remain valid in physical mode after `ExitBootServices()`. This kernel does not call `SetVirtualAddressMap()`, so the pointer is the raw physical address returned by firmware. On QEMU/OVMF this is always usable; on real hardware it is valid as long as the CPU is in physical mode (identity-mapped) — which it is for the duration of LithosAnanke, since the VMM uses a separate TTBR/CR3 but does not remap the EFI reserved regions.

Referenced by kernel_main(), kernel_main_deep(), and system_word_reboot().

7.163 kernel/src/log_region.c File Reference

```
#include "starkernel/log_region.h"
#include <stddef.h>
#include <string.h>
#include "block_subsystem.h"
```

Include dependency graph for log_region.c:



Functions

- static uint64_t log_region_ctrl_compute_crc (const log_region_ctrl_t ∗ctrl)

  log_region.c - Growable per-VM log-persistence ring on Artemis's disk.
- static int log_region_ctrl_read (log_region_ctrl_t ∗out)
- static int log_region_ctrl_write (log_region_ctrl_t ∗ctrl)
- static void log_region_ctrl_init_fresh (log_region_ctrl_t ∗ctrl)
- static uint32_t log_region_slot_devblock_from_top (uint32_t slot_index)
- static int log_region_read_slot (uint32_t slot_index, log_slot_t ∗out)
- static int log_region_write_slot (uint32_t slot_index, const log_slot_t ∗slot)
- int log_region_append (uint8_t level, uint64_t timestamp, const char ∗source, uint32_t source_len, const char ∗msg, uint32_t msg_len)
- int log_region_read_recent (uint32_t back, log_slot_t ∗out)

  log_region_read_recent - read one record back (2026-10-14, for LOG-SHOW).

7.163.1 Function Documentation

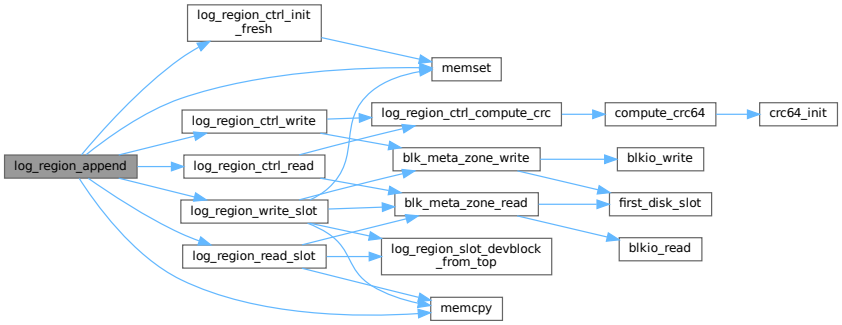
7.163.1.1 log_region_append()

```
int log_region_append (
            uint8_t level,
            uint64_t timestamp,
```

```
            const char * source,
            uint32_t source_len,
            const char * msg,
            uint32_t msg_len )
```

References log_region_ctrl_t::devblocks, log_region_ctrl_t::head_slot, log_slot_t::level, level, log_region_ctrl_init_fresh(), log_region_ctrl_read(), log_region_ctrl_write(), LOG_REGION_GROWTH_INCREMENT, LOG_REGION_MAX_DEVBLOCKS, LOG_REGION_PROTECTED_MAX_LEVEL, log_region_read_slot(), log_region_write_slot(), LOG_SLOT_MSG_MAX, LOG_SLOT_SOURCE_MAX, LOG_SLOTS_PER_DEVBLOCK, memcpy(), memset(), log_slot_t::msg, log_slot_t::msg_len, log_region_ctrl_t::record_count, log_slot_t::source, log_region_ctrl_t::tail_slot, and log_slot_t::timestamp.

Here is the call graph for this function:



7.163.1.2 log_region_ctrl_compute_crc()

```
static uint64_t log_region_ctrl_compute_crc (
            const log_region_ctrl_t * ctrl )  [static]
```

log_region.c - Growable per-VM log-persistence ring on Artemis's disk.

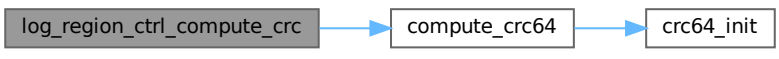
See starkernel/log_region.h for the format/interface design.

Deliberately never calls log_message() (or anything documented to call it) – this runs inside Artemis's own dictionary context during message delivery (LOG-APPEND, dispatched by a sender's MSG-SEND/MSG-TICK); a log call here would append to Artemis's own buffer and eventually flush back through this same path.

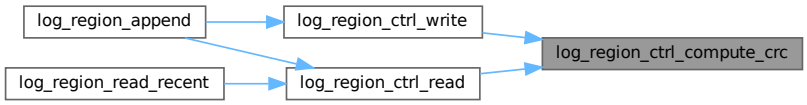
References compute_crc64().

Referenced by log_region_ctrl_read(), and log_region_ctrl_write().

Here is the call graph for this function:



Here is the caller graph for this function:



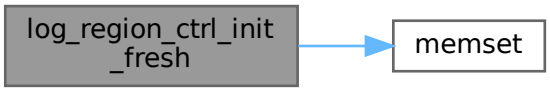
7.163.1.3 log_region_ctrl_init_fresh()

```
static void log_region_ctrl_init_fresh (
            log_region_ctrl_t * ctrl )  [static]
```

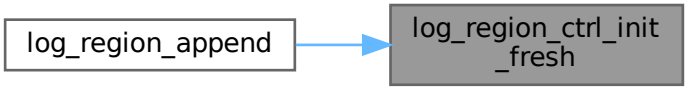
References log_region_ctrl_t::devblocks, log_region_ctrl_t::head_slot, LOG_REGION_INITIAL_DEVBLOCKS, LOG_REGION_PACK, LOG_REGION_VERSION_0, log_region_ctrl_t::magic, memset(), log_region_ctrl_t::record_count, and log_region_ctrl_t::tail_slot.

Referenced by log_region_append().

Here is the call graph for this function:



Here is the caller graph for this function:



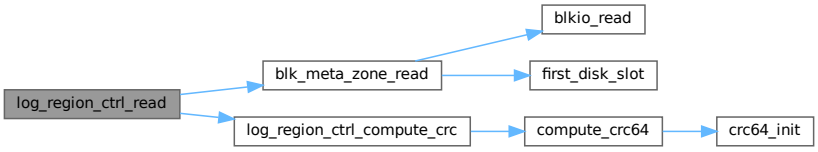
7.163.1.4 log_region_ctrl_read()

```
static int log_region_ctrl_read (
            log_region_ctrl_t * out )  [static]
```

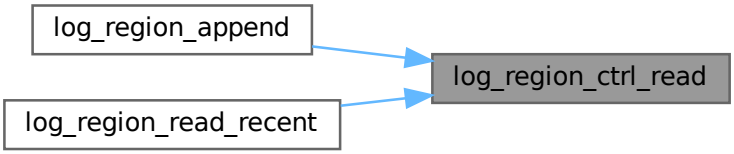
References blk_meta_zone_read(), log_region_ctrl_t::hdr_crc, log_region_ctrl_compute_crc(), LOG_REGION_DEVBLOCK_FROM_T LOG_REGION_GET_MAGIC, LOG_REGION_GET_VERSION, LOG_REGION_MAGIC, LOG_REGION_VERSION_0, and log_region_ctrl_t::magic.

Referenced by log_region_append(), and log_region_read_recent().

Here is the call graph for this function:



Here is the caller graph for this function:



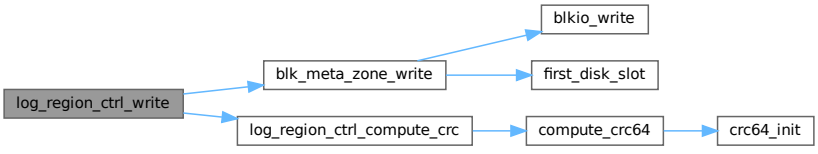
7.163.1.5 log_region_ctrl_write()

```
static int log_region_ctrl_write (
            log_region_ctrl_t * ctrl )  [static]
```

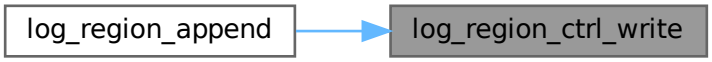
References blk_meta_zone_write(), log_region_ctrl_t::hdr_crc, log_region_ctrl_compute_crc(), and LOG_REGION_DEVBLOCK_FRO

Referenced by log_region_append().

Here is the call graph for this function:



Here is the caller graph for this function:



7.163.1.6 log_region_read_recent()

```
int log_region_read_recent (
            uint32_t back,
            log_slot_t * out )
```

log_region_read_recent - read one record back (2026-10-14, for LOG-SHOW).

Parameters

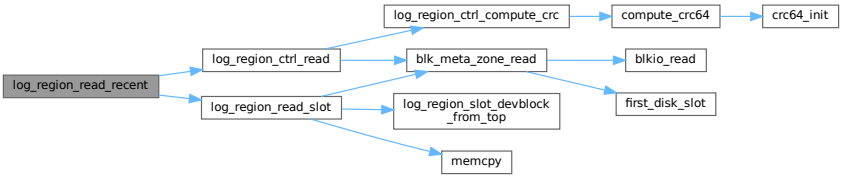
back	0 for the newest record, 1 for the one before it, ...
out	Filled with the record.

Returns

0 on success, 1 if there is no such record (the ring holds fewer than back+1, or none at all), -1 on a read failure or no ring.

References log_region_ctrl_t::devblocks, log_region_ctrl_read(), log_region_read_slot(), LOG_SLOTS_PER_DEVBLOCK, log_region_ctrl_t::record_count, and log_region_ctrl_t::tail_slot.

Here is the call graph for this function:



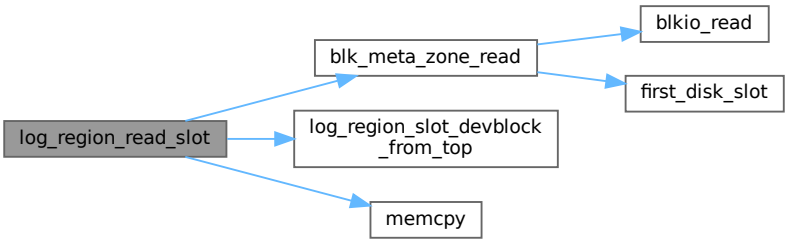
7.163.1.7 log_region_read_slot()

```
static int log_region_read_slot (
            uint32_t slot_index,
            log_slot_t * out )  [static]
```

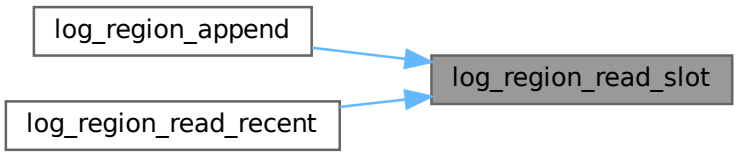
References blk_meta_zone_read(), log_region_slot_devblock_from_top(), LOG_SLOT_SIZE, LOG_SLOTS_PER_DEVBLOCK, and memcpy().

Referenced by log_region_append(), and log_region_read_recent().

Here is the call graph for this function:



Here is the caller graph for this function:



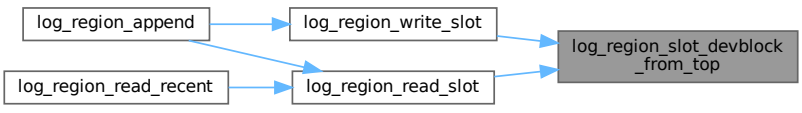
7.163.1.8 log_region_slot_devblock_from_top()

```
static uint32_t log_region_slot_devblock_from_top (
            uint32_t slot_index )  [static]
```

References LOG_REGION_DEVBLOCK_FROM_TOP_BASE, and LOG_SLOTS_PER_DEVBLOCK.

Referenced by log_region_read_slot(), and log_region_write_slot().

Here is the caller graph for this function:



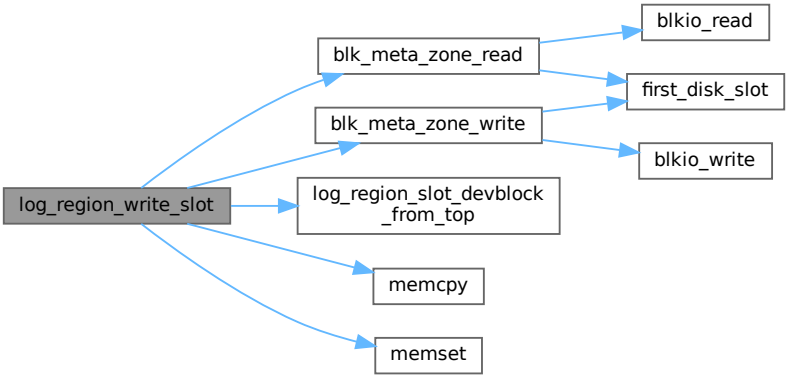
7.163.1.9 log_region_write_slot()

```
static int log_region_write_slot (
            uint32_t slot_index,
            const log_slot_t * slot )  [static]
```

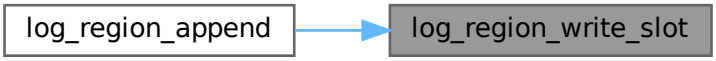
References blk_meta_zone_read(), blk_meta_zone_write(), log_region_slot_devblock_from_top(), LOG_SLOT_SIZE, LOG_SLOTS_PER_DEVBLOCK, memcpy(), and memset().

Referenced by log_region_append().

Here is the call graph for this function:



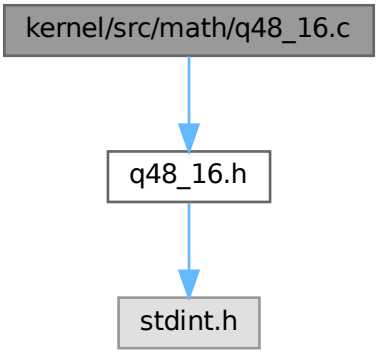
Here is the caller graph for this function:



7.164 kernel/src/math/q48_16.c File Reference

```
#include "q48_16.h"
```

Include dependency graph for q48_16.c:



Functions

- q48_16_t q48_mul (q48_16_t a, q48_16_t b)

  q48_16.c - Q48.16 Fixed-Point Arithmetic for StarKernel
- q48_16_t q48_div (q48_16_t a, q48_16_t b)

  Divide two Q48.16 fixed-point values.
- q48_16_t q48_exp_approx (q48_16_t q)

  Compute an approximation of `e^q` in Q48.16 format (Taylor series).
- q48_16_t q48_log_approx (q48_16_t x)

  Compute an approximation of `ln(x)` in Q48.16 format (integer-only).
- q48_16_t q48_sqrt_approx (q48_16_t q)

  Compute an approximation of `sqrt(q)` in Q48.16 format (Newton-Raphson).
- static int64_t q48_reduce_angle (int64_t x)
- q48_16_t q48_sin_approx (q48_16_t q)

  Compute an approximation of `sin(q)` in Q48.16 format (radians).
- q48_16_t q48_cos_approx (q48_16_t q)

  Compute an approximation of `cos(q)` in Q48.16 format (radians).

7.164.1 Function Documentation

7.164.1.1 q48_cos_approx()

```
q48_16_t q48_cos_approx (
            q48_16_t q )
```

Compute an approximation of `cos(q)` in Q48.16 format (radians).

Range-reduces into [-pi, pi] via `q48_reduce_angle()`, then sums the Taylor series `1` - x^2/2! + x^4/4! - ... for terms n = 2,4,6,8,10, with an early exit when a term falls below 10. Cosine is even (`cos`(-x) = cos(x)), so no sign correction is needed – `x^2` already absorbs the sign of `x`.

Parameters

q	Angle in Q48.16 format (radians, any magnitude)

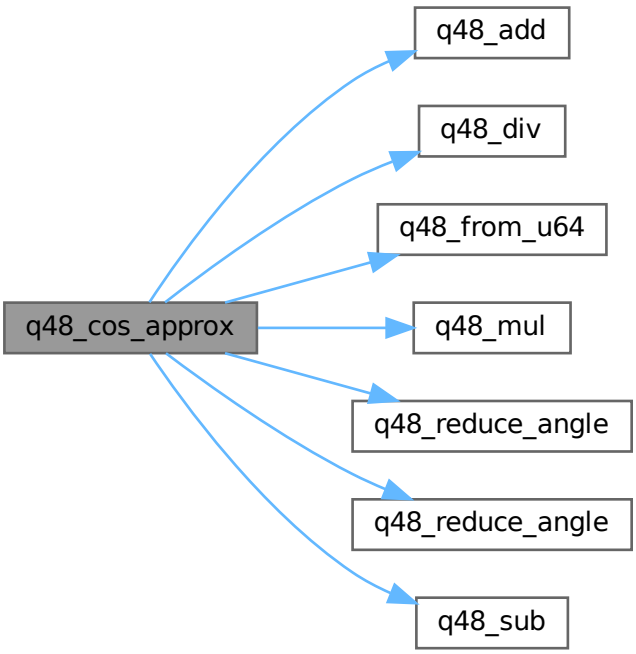
Returns

`cos(q)` in Q48.16 format

References q48_add(), q48_div(), q48_from_u64(), q48_mul(), Q48_ONE, q48_reduce_angle(), q48_reduce_angle(), and q48_sub().

Referenced by q48_word_cos().

Here is the call graph for this function:



Here is the caller graph for this function:

7.164.1.2 q48_div()

```
q48_16_t q48_div (
            q48_16_t a,
            q48_16_t b )
```

Divide two Q48.16 fixed-point values.

Computes (a << 16) / b. Returns 0 if b == 0 (safe, but callers should guard). Returns UINT64_MAX as a saturation sentinel when a exceeds 2^48, which would overflow the 16-bit left-shift.

Parameters

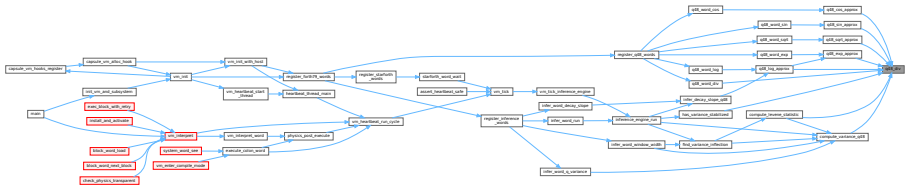
a	Dividend in Q48.16 format
b	Divisor in Q48.16 format; 0 returns 0

Returns

Quotient in Q48.16 format, or UINT64_MAX on saturation

Referenced by compute_levene_statistic(), compute_variance_q48(), has_variance_stabilized(), q48_cos_approx(), q48_exp_approx(), q48_log_approx(), q48_sin_approx(), q48_sqrt_approx(), and q48_word_div().

Here is the caller graph for this function:



7.164.1.3 q48_exp_approx()

```
q48_16_t q48_exp_approx (
            q48_16_t q )
```

Compute an approximation of e^q in Q48.16 format (Taylor series).

Supporting function for q48_log_approx() Newton iterations. Uses the Taylor series 1 + x + x^2/2! + ... up to 10 terms, with an early-exit when a term falls below 50 (sub-LSB in Q48.16). Handles negative q by computing e^|q| then returning 1/result. Returns 1.0 (65536) for q == 0, UINT64_MAX for q >= 16.0, and 0 for q <= -16.0.

Parameters

q	Exponent in Q48.16 format (signed interpretation)

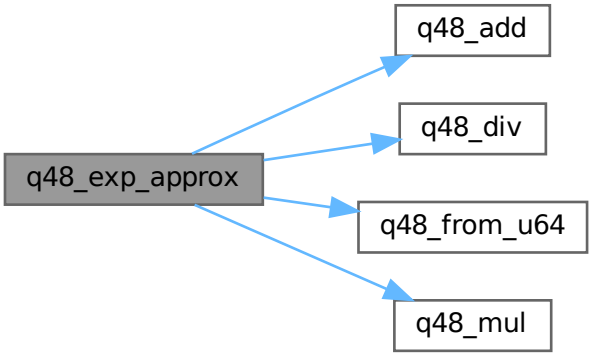
Returns

e^q in Q48.16 format

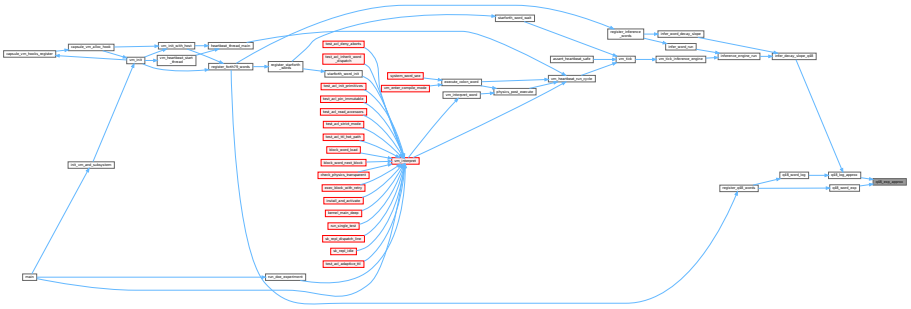
References q48_add(), q48_div(), q48_from_u64(), q48_mul(), and Q48_ONE.

Referenced by q48_log_approx(), and q48_word_exp().

Here is the call graph for this function:



Here is the caller graph for this function:



7.164.1.4 q48_log_approx()

```
q48_16_t q48_log_approx (
            q48_16_t x )
```

Compute an approximation of ln(x) in Q48.16 format (integer-only).

Used by infer_decay_slope_q48() (Loop #6) for log-linear OLS regression. Algorithm: range-reduces x to the interval [1, 2) by tracking the exponent k (so x = 2^k * m), applies 6 Newton-Raphson iterations on e^y = m starting from the first-order approximation y_0 = m - 1, then assembles the result as ln(x) = k*ln(2) + ln(m). Returns 0 for x == 0 (ln undefined) and for x == 65536 (ln(1.0) == 0.0 exactly). LN2_Q48 = 45426 (0.693147... × 65536).

Parameters

x	Input value in Q48.16 format (uint64_t; value of 65536 = 1.0)

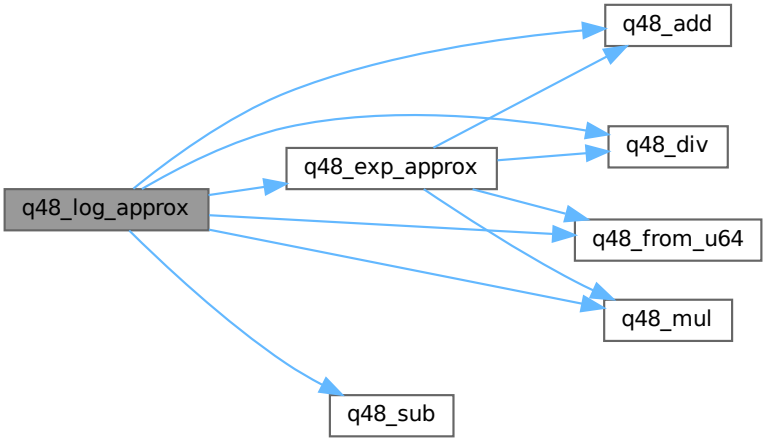
Returns

`ln(x)` in Q48.16 format, or 0 for x==0 or x==65536

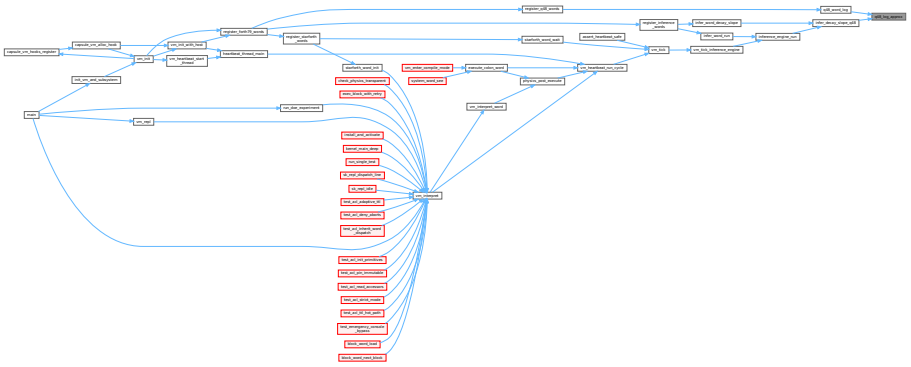
References q48_add(), q48_div(), q48_exp_approx(), q48_from_u64(), q48_mul(), Q48_ONE, and q48_sub().

Referenced by infer_decay_slope_q48(), and q48_word_log().

Here is the call graph for this function:



Here is the caller graph for this function:



7.164.1.5 q48_mul()

```
q48_16_t q48_mul (
            q48_16_t a,
            q48_16_t b )
```

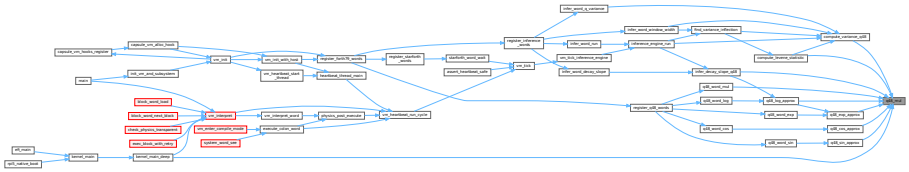
q48_16.c - Q48.16 Fixed-Point Arithmetic for StarKernel

Multiply two Q48.16 fixed-point values.

Freestanding implementation: no libc, no floating-point.

Referenced by compute_levene_statistic(), compute_variance_q48(), infer_decay_slope_q48(), kernel_main_deep(), q48_cos_approx(), q48_exp_approx(), q48_log_approx(), q48_sin_approx(), and q48_word_mul().

Here is the caller graph for this function:



7.164.1.6 q48_reduce_angle()

```
static int64_t q48_reduce_angle (
            int64_t x )  [static]
```

Referenced by q48_cos_approx(), and q48_sin_approx().

Here is the caller graph for this function:

7.164.1.7 q48_sin_approx()

```
q48_16_t q48_sin_approx (
            q48_16_t q )
```

Compute an approximation of `sin(q)` in Q48.16 format (radians).

Range-reduces into [-pi, pi] via `q48_reduce_angle()`, then sums the Taylor series `x` - x^3/3! + x^5/5! - ... for terms n = 3,5,7,9,11, with an early exit when a term falls below 10 (sub-LSB in Q48.16). Sine is odd (`sin`(-x) = -sin(x)), so the sign is factored out before the series and reapplied to the result.

Parameters

q	Angle in Q48.16 format (radians, any magnitude)

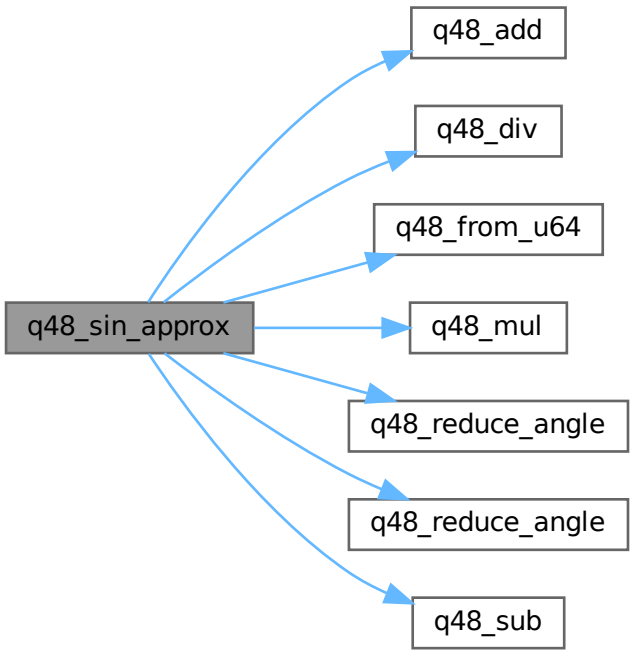
Returns

`sin(q)` in Q48.16 format

References q48_add(), q48_div(), q48_from_u64(), q48_mul(), q48_reduce_angle(), q48_reduce_angle(), and q48_sub().

Referenced by q48_word_sin().

Here is the call graph for this function:



Here is the caller graph for this function:

7.164.1.8 q48_sqrt_approx()

```
q48_16_t q48_sqrt_approx (
            q48_16_t q )
```

Compute an approximation of `sqrt(q)` in Q48.16 format (Newton-Raphson).

Iterates `x_{n+1}` = (x_n + q/x_n) / 2 starting from (q >> 1) + 16384, for up to 8 iterations. Terminates early when the delta between successive estimates falls below 10 (approximately 0.00015 in Q48.16). Used for R^2 computation in the inference engine. Returns 0 for `q` == 0, and 1.0 (65536) for `q` == 65536.

Parameters

q	Input value in Q48.16 format

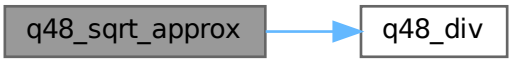
Returns

`sqrt(q)` in Q48.16 format

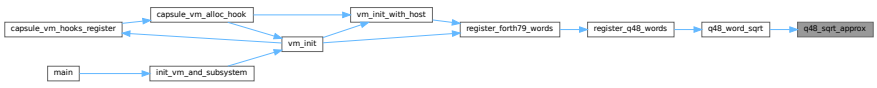
References q48_div(), and Q48_ONE.

Referenced by q48_word_sqrt().

Here is the call graph for this function:



Here is the caller graph for this function:

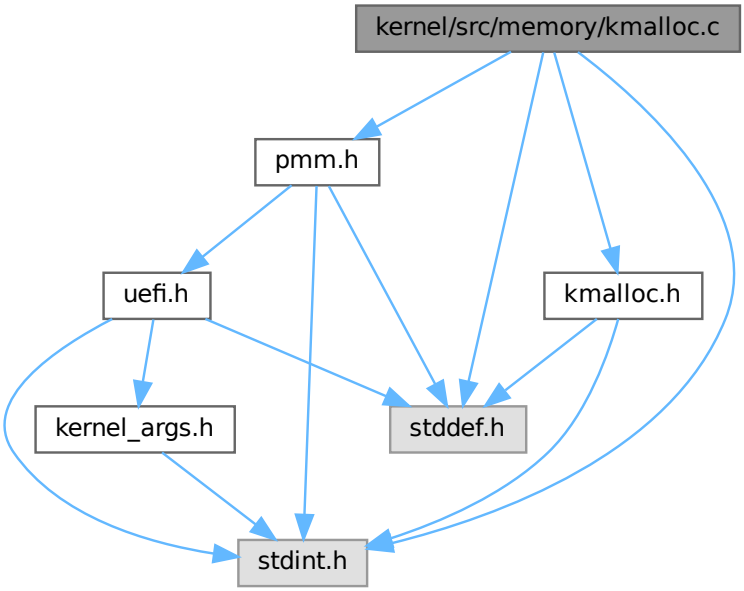


7.165 kernel/src/memory/kmalloc.c File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "kmalloc.h"
```

```
#include "pmm.h"
```

Include dependency graph for kmalloc.c:



Data Structures

- struct heap_block
- struct kmalloc_prefix_t

Macros

- #define KMALLOC_DEFAULT_HEAP_SIZE (2ULL ∗ 1024ULL ∗ 1024ULL ∗ 1024ULL) /∗ 2 GiB — floor for 256+ baby VMs ∗/

  kmalloc.c - Simple kernel heap allocator
- #define KMALLOC_MIN_ALIGN 16u
- #define KMALLOC_MAGIC 0x4B4D414Cu /∗ 'KMAL' ∗/
- #define KMALLOC_MIN_PAGES ((64u ∗ 1024u ∗ 1024u) / PMM_PAGE_SIZE)

Typedefs

- typedef struct heap_block heap_block_t

Functions

- uintptr_t kmalloc_heap_base_addr (void)
- uintptr_t kmalloc_heap_end_addr (void)
- static size_t align_up_size (size_t value, size_t align)
- static size_t normalize_alignment (size_t align)
- static void update_free_bytes (void)
- static void account_allocation (size_t requested)
- static void account_free (size_t requested)
- static int pointer_within_heap (const void *ptr)
- static void coalesce_neighbors (heap_block_t *block)
- static size_t min_splittable_payload (void)
- static int can_split (size_t remainder)
- static void * allocate_from_block (heap_block_t *block, size_t requested, size_t align)
- int kmalloc_init (uint64_t heap_size_bytes)
- int kmalloc_is_initialized (void)
- void * kmalloc (size_t size)
- void * kmalloc_aligned (size_t size, size_t align)
- void kfree (void *ptr)
- kmalloc_stats_t kmalloc_get_stats (void)
- void kmalloc_debug_census (size_t *out_largest_free, size_t *out_free_count, size_t *out_used_count, size_t *out_total_blocks)
- void kmalloc_debug_census_bytes (size_t *out_total_free_bytes, size_t *out_total_used_bytes)

Variables

- static uint8_t * heap_base = NULL
- static uint64_t heap_size = 0
- static heap_block_t * heap_head = NULL
- static kmalloc_stats_t heap_stats = {0}
- static int heap_initialized = 0

7.165.1 Macro Definition Documentation

7.165.1.1 KMALLOC_DEFAULT_HEAP_SIZE

```
#define KMALLOC_DEFAULT_HEAP_SIZE (2ULL * 1024ULL * 1024ULL * 1024ULL) /* 2 GiB -- floor for
256+ baby VMs */
```

kmalloc.c - Simple kernel heap allocator

Reserves a fixed heap from PMM and serves aligned allocations with a doubly-linked free list plus coalescing. The allocator avoids libc helpers and keeps metadata inside the heap itself.

7.165.1.2 KMALLOC_MAGIC

```
#define KMALLOC_MAGIC 0x4B4D414Cu /* 'KMAL' */
```

7.165.1.3 KMALLOC_MIN_ALIGN

```
#define KMALLOC_MIN_ALIGN 16u
```

7.165.1.4 KMALLOC_MIN_PAGES

```
#define KMALLOC_MIN_PAGES ((64u * 1024u * 1024u) / PMM_PAGE_SIZE)
```

7.165.2 Typedef Documentation

7.165.2.1 heap_block_t

```
typedef struct heap_block heap_block_t
```

7.165.3 Function Documentation

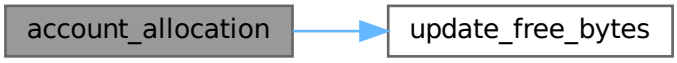
7.165.3.1 account_allocation()

```
static void account_allocation (
            size_t requested )  [static]
```

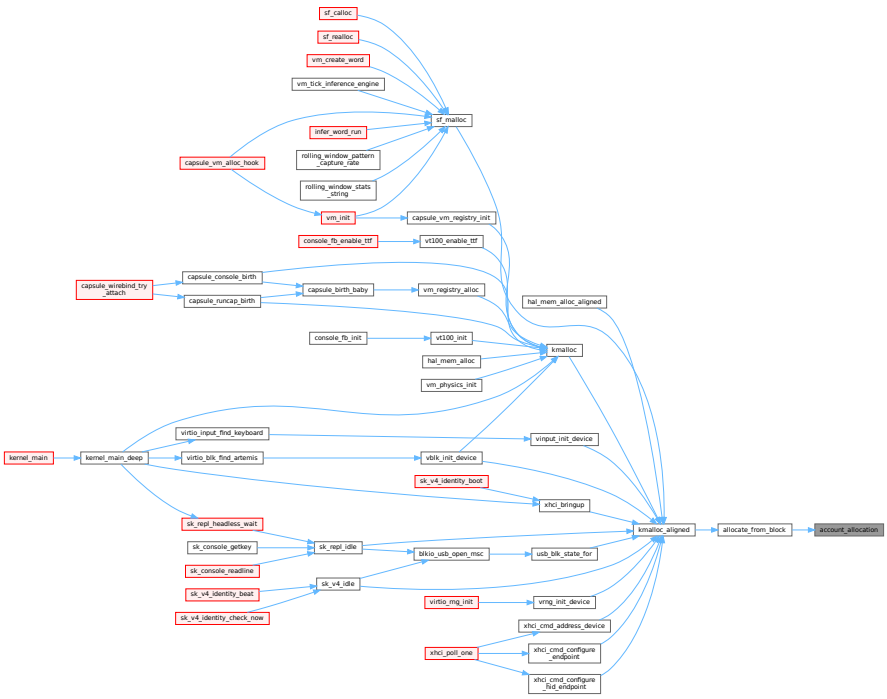
References heap_stats, kmalloc_stats_t::peak_bytes, update_free_bytes(), and kmalloc_stats_t::used_bytes.

Referenced by allocate_from_block().

Here is the call graph for this function:



Here is the caller graph for this function:



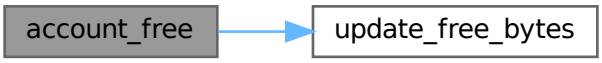
7.165.3.2 account_free()

```
static void account_free (
            size_t requested )  [static]
```

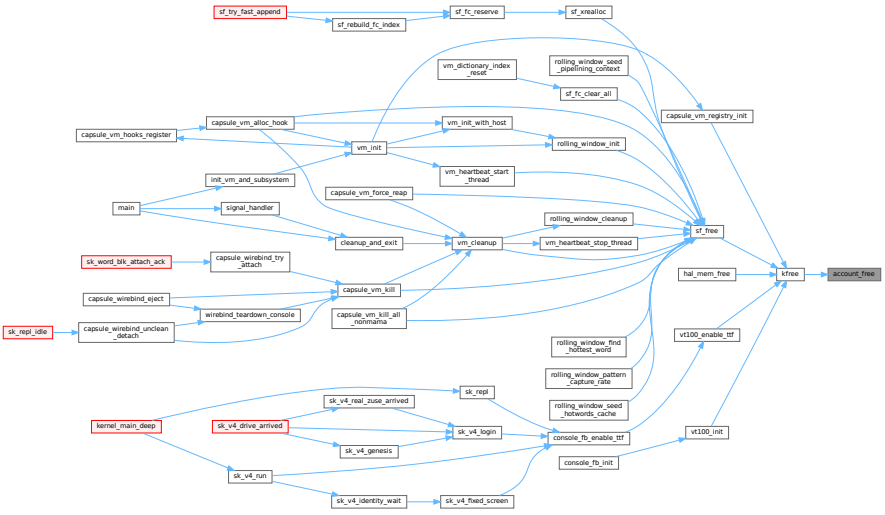
References heap_stats, update_free_bytes(), and kmalloc_stats_t::used_bytes.

Referenced by kfree().

Here is the call graph for this function:



Here is the caller graph for this function:

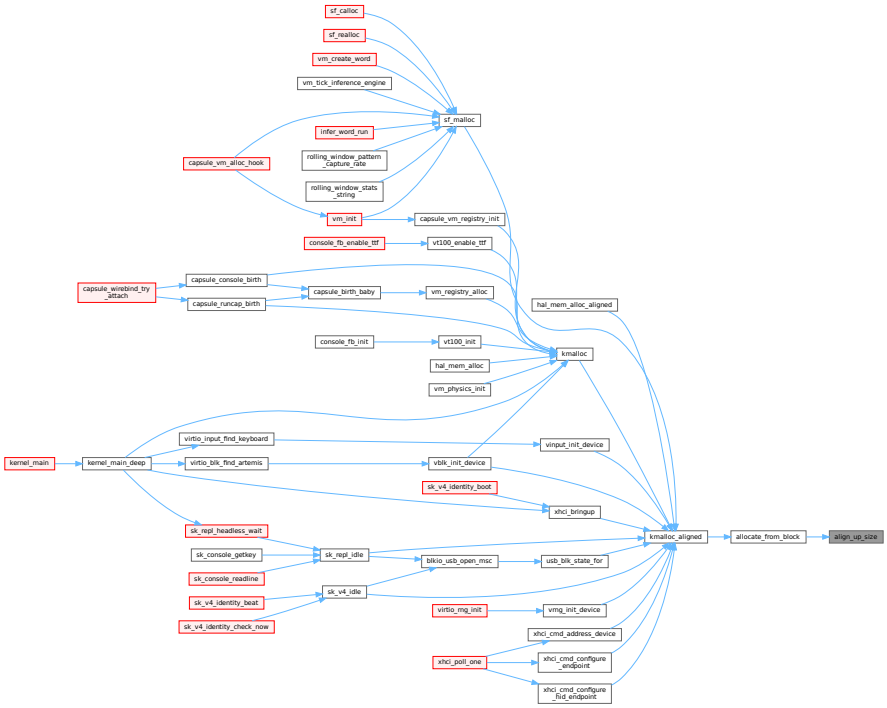


7.165.3.3 align_up_size()

```
static size_t align_up_size (
            size_t value,
            size_t align )  [static]
```

Referenced by allocate_from_block().

Here is the caller graph for this function:



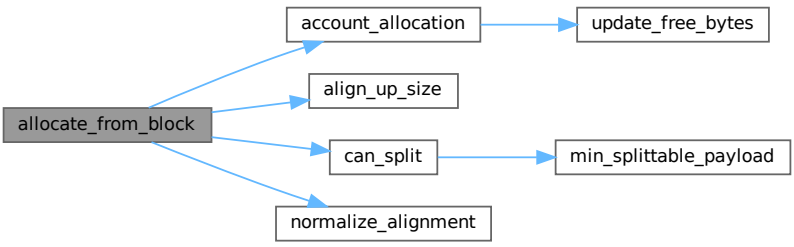
7.165.3.4 allocate_from_block()

```
static void * allocate_from_block (
            heap_block_t * block,
            size_t requested,
            size_t align )  [static]
```

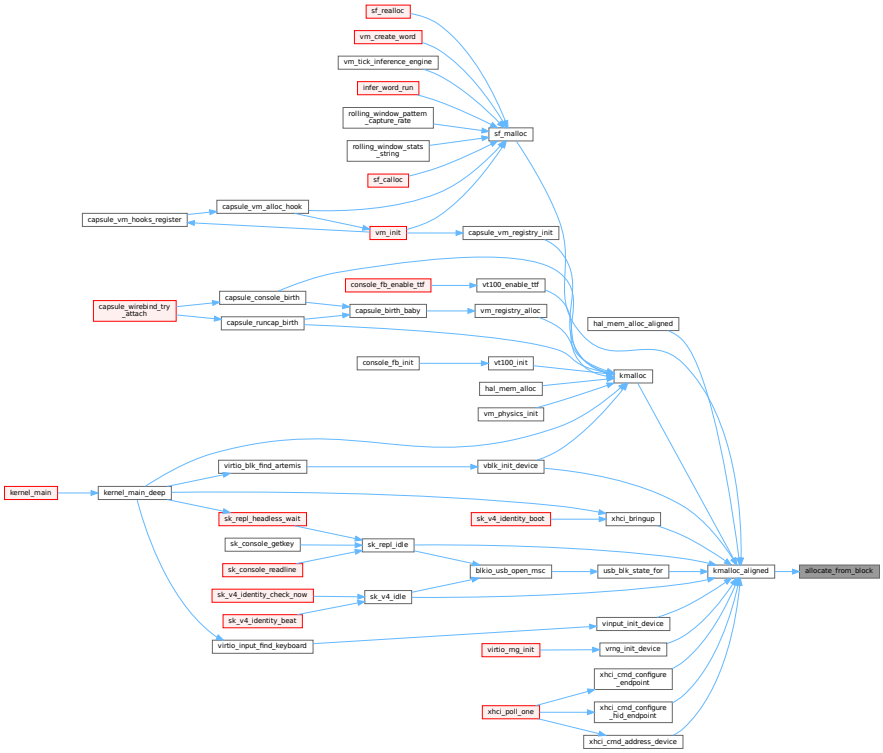
References account_allocation(), align_up_size(), kmalloc_prefix_t::block, can_split(), heap_block::free, KMALLOC_MAGIC, KMALLOC_MIN_ALIGN, kmalloc_prefix_t::magic, heap_block::next, normalize_alignment(), heap_block::prev, heap_block::requested, and heap_block::size.

Referenced by kmalloc_aligned().

Here is the call graph for this function:



Here is the caller graph for this function:



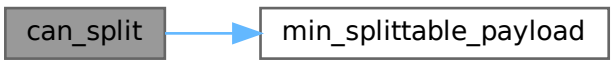
7.165.3.5 can_split()

```
static int can_split (
            size_t remainder )  [static]
```

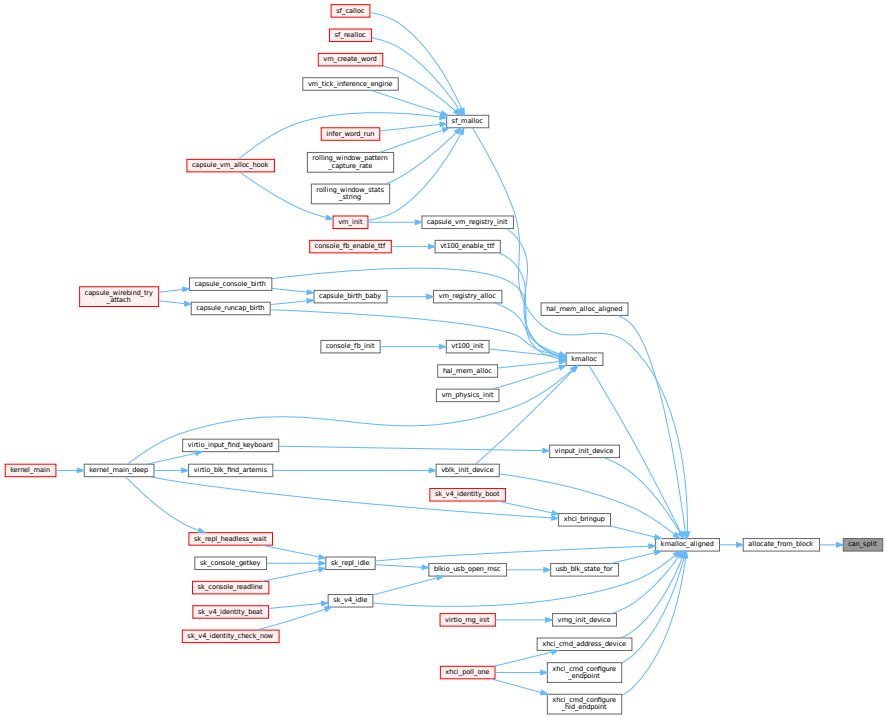
References min_splittable_payload().

Referenced by allocate_from_block().

Here is the call graph for this function:



Here is the caller graph for this function:



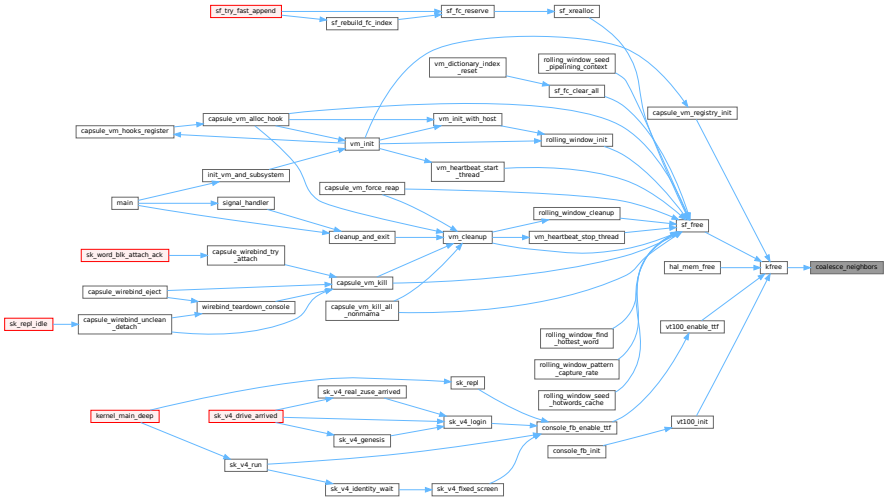
7.165.3.6 coalesce_neighbors()

```
static void coalesce_neighbors (
            heap_block_t * block )  [static]
```

References heap_block::free, heap_head, heap_block::next, heap_block::prev, and heap_block::size.

Referenced by kfree().

Here is the caller graph for this function:



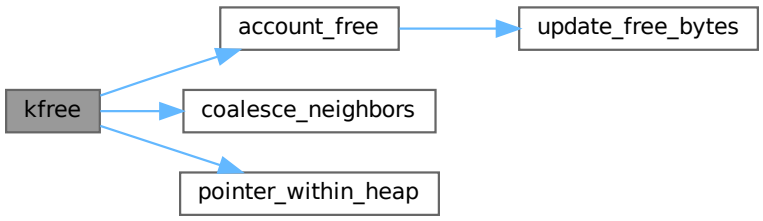
7.165.3.7 kfree()

```
void kfree (
            void * ptr )
```

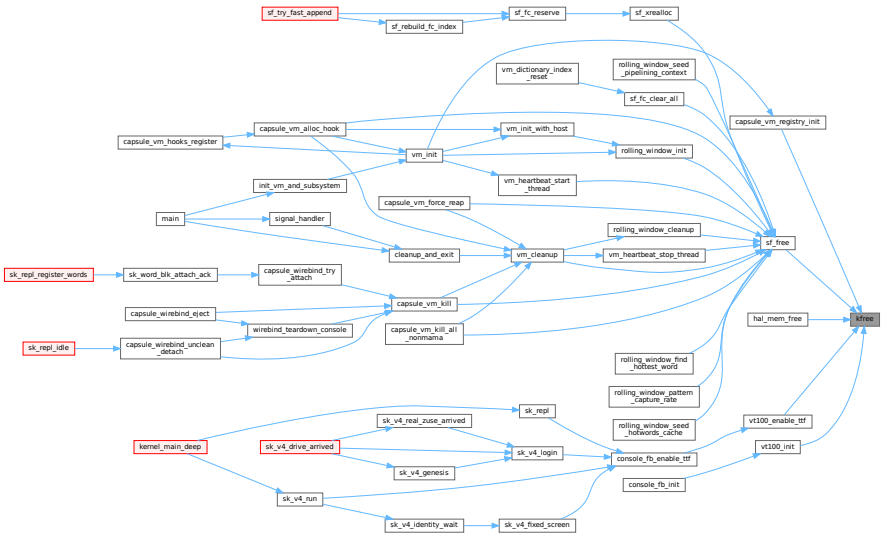
References account_free(), kmalloc_prefix_t::block, coalesce_neighbors(), heap_block::free, heap_initialized, KMALLOC_MAGIC, kmalloc_prefix_t::magic, pointer_within_heap(), and heap_block::requested.

Referenced by capsule_vm_registry_init(), hal_mem_free(), sf_free(), vt100_enable_ttf(), and vt100_init().

Here is the call graph for this function:



Here is the caller graph for this function:



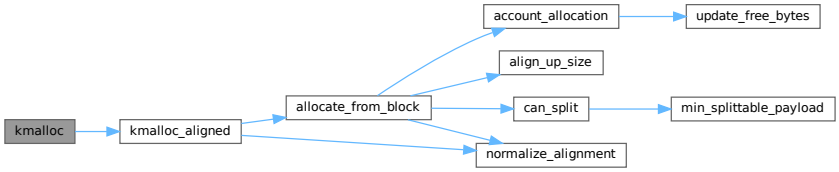
7.165.3.8 kmalloc()

```
void * kmalloc (
            size_t size )
```

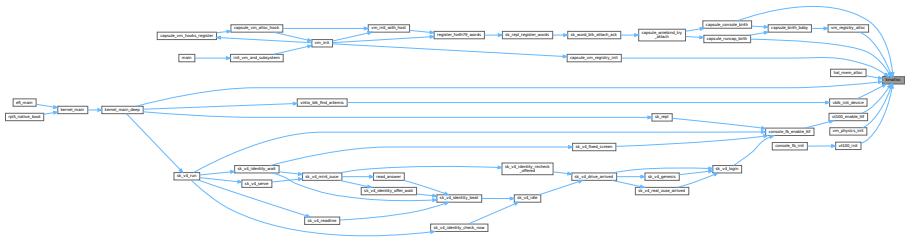
References kmalloc_aligned(), and KMALLOC_MIN_ALIGN.

Referenced by capsule_console_birth(), capsule_runcap_birth(), capsule_vm_registry_init(), hal_mem_alloc(), kernel_main_deep(), vblk_init_device(), vm_physics_init(), vm_registry_alloc(), vt100_enable_ttf(), and vt100_init().

Here is the call graph for this function:



Here is the caller graph for this function:



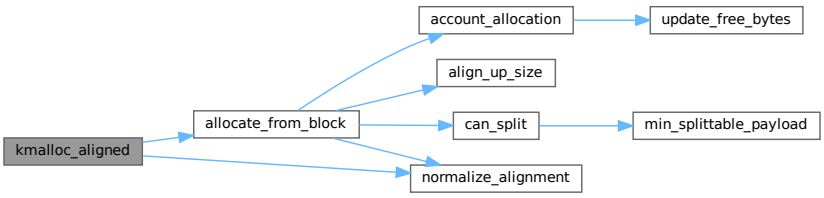
7.165.3.9 kmalloc_aligned()

```
void * kmalloc_aligned (
            size_t size,
            size_t align )
```

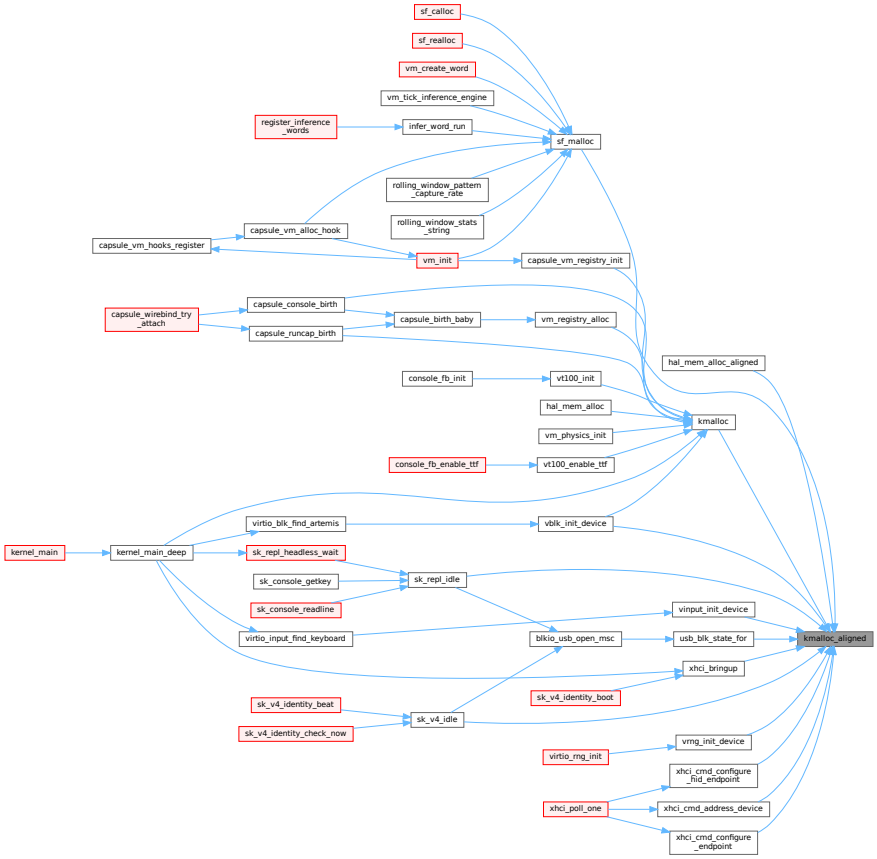
References allocate_from_block(), heap_block::free, heap_head, heap_initialized, heap_block::next, and normalize_alignment().

Referenced by hal_mem_alloc_aligned(), kmalloc(), sf_malloc(), sk_repl_idle(), sk_v4_idle(), usb_blk_state_for(), vblk_init_device(), vinput_init_device(), vrng_init_device(), xhci_bringup(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

Here is the call graph for this function:



Here is the caller graph for this function:



7.165.3.10 kmalloc_debug_census()

```
void kmalloc_debug_census (
            size_t * out_largest_free,
            size_t * out_free_count,
            size_t * out_used_count,
            size_t * out_total_blocks )
```

References heap_block::free, heap_head, heap_block::next, and heap_block::size.

7.165.3.11 kmalloc_debug_census_bytes()

```
void kmalloc_debug_census_bytes (
            size_t * out_total_free_bytes,
            size_t * out_total_used_bytes )
```

References heap_block::free, heap_head, heap_block::next, and heap_block::size.

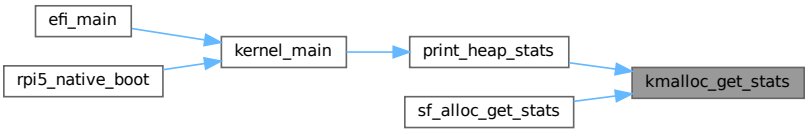
7.165.3.12 kmalloc_get_stats()

```
kmalloc_stats_t kmalloc_get_stats (
            void  )
```

References heap_stats.

Referenced by print_heap_stats(), and sf_alloc_get_stats().

Here is the caller graph for this function:



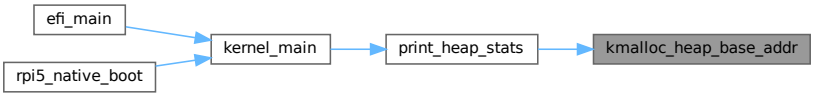
7.165.3.13 kmalloc_heap_base_addr()

```
uintptr_t kmalloc_heap_base_addr (
            void  )
```

References heap_base.

Referenced by print_heap_stats().

Here is the caller graph for this function:



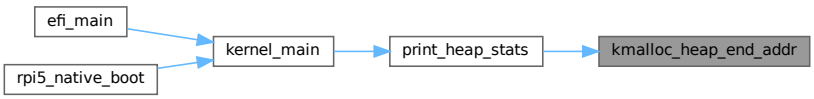
7.165.3.14 kmalloc_heap_end_addr()

```
uintptr_t kmalloc_heap_end_addr (
            void  )
```

References heap_base, and heap_size.

Referenced by print_heap_stats().

Here is the caller graph for this function:



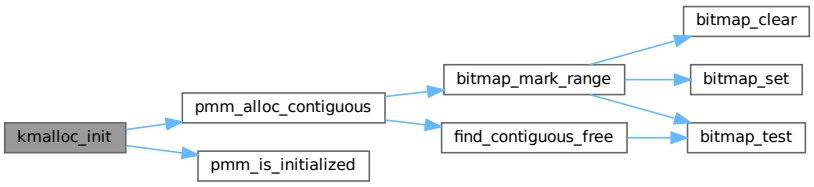
7.165.3.15 kmalloc_init()

```
int kmalloc_init (
            uint64_t heap_size_bytes )
```

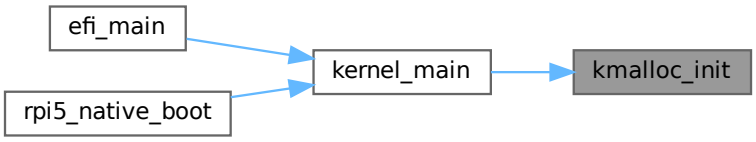
References heap_block::free, kmalloc_stats_t::free_bytes, heap_base, heap_head, heap_initialized, heap_size, heap_stats, KMALLOC_DEFAULT_HEAP_SIZE, KMALLOC_MIN_PAGES, heap_block::next, kmalloc_stats_t::peak_bytes, pmm_alloc_contiguous(), pmm_is_initialized(), PMM_PAGE_SIZE, heap_block::prev, heap_block::requested, heap_block::size, kmalloc_stats_t::total_bytes, and kmalloc_stats_t::used_bytes.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.165.3.16 kmalloc_is_initialized()

```
int kmalloc_is_initialized (
            void  )
```

References heap_initialized.

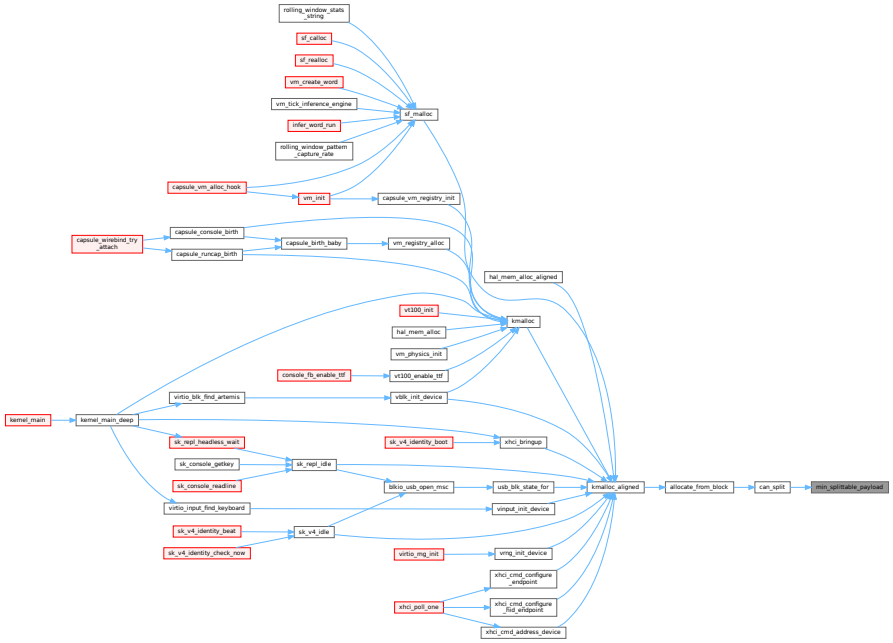
7.165.3.17 min_splittable_payload()

```
static size_t min_splittable_payload (
            void  )  [static]
```

References KMALLOC_MIN_ALIGN.

Referenced by can_split().

Here is the caller graph for this function:



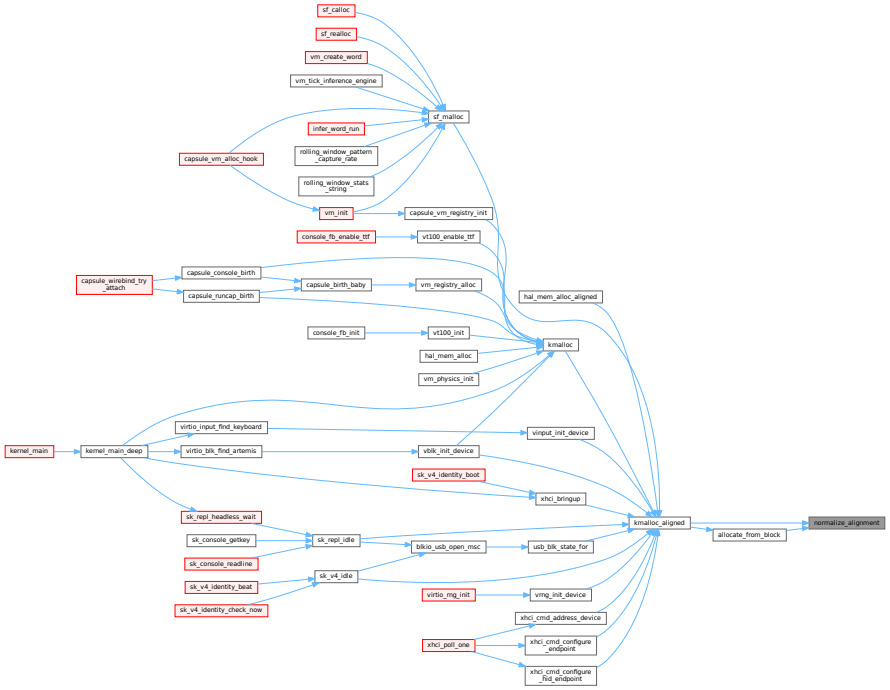
7.165.3.18 normalize_alignment()

```
static size_t normalize_alignment (
            size_t align )  [static]
```

References KMALLOC_MIN_ALIGN.

Referenced by allocate_from_block(), and kmalloc_aligned().

Here is the caller graph for this function:



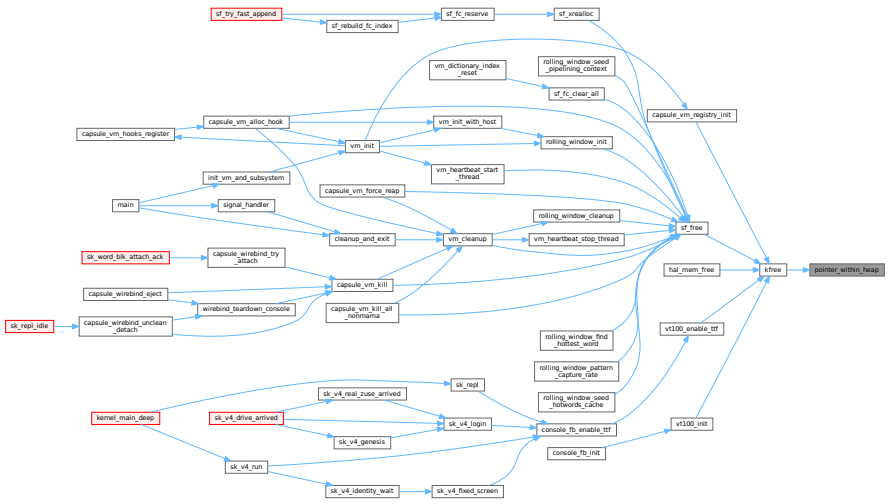
7.165.3.19 pointer_within_heap()

```
static int pointer_within_heap (
            const void * ptr )  [static]
```

References heap_base, and heap_size.

Referenced by kfree().

Here is the caller graph for this function:



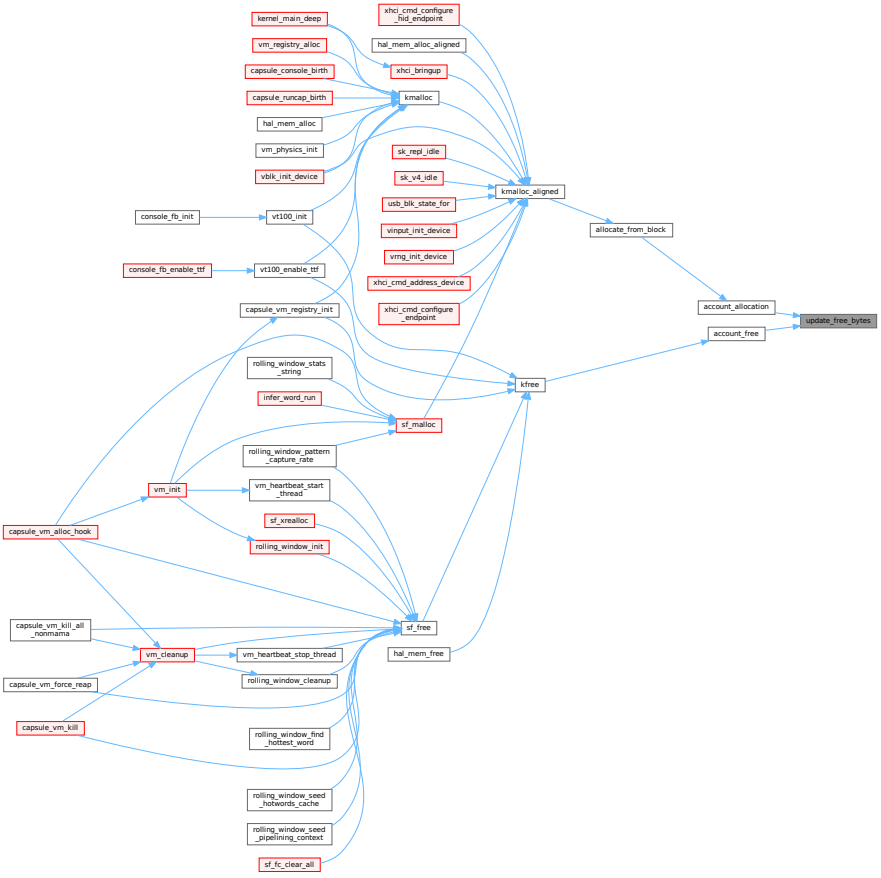
7.165.3.20 update_free_bytes()

```
static void update_free_bytes (
            void  )  [static]
```

References kmalloc_stats_t::free_bytes, heap_stats, kmalloc_stats_t::total_bytes, and kmalloc_stats_t::used_bytes.

Referenced by account_allocation(), and account_free().

Here is the caller graph for this function:



7.165.4 Variable Documentation

7.165.4.1 heap_base

```
uint8_t* heap_base = NULL  [static]
```

Referenced by kmalloc_heap_base_addr(), kmalloc_heap_end_addr(), kmalloc_init(), and pointer_within_heap().

7.165.4.2 heap_head

```
heap_block_t* heap_head = NULL  [static]
```

Referenced by coalesce_neighbors(), kmalloc_aligned(), kmalloc_debug_census(), kmalloc_debug_census_bytes(), and kmalloc_init().

7.165.4.3 heap_initialized

```
int heap_initialized = 0  [static]
```

Referenced by kfree(), kmalloc_aligned(), kmalloc_init(), and kmalloc_is_initialized().

7.165.4.4 heap_size

```
uint64_t heap_size = 0  [static]
```

Referenced by kmalloc_heap_end_addr(), kmalloc_init(), and pointer_within_heap().

7.165.4.5 heap_stats

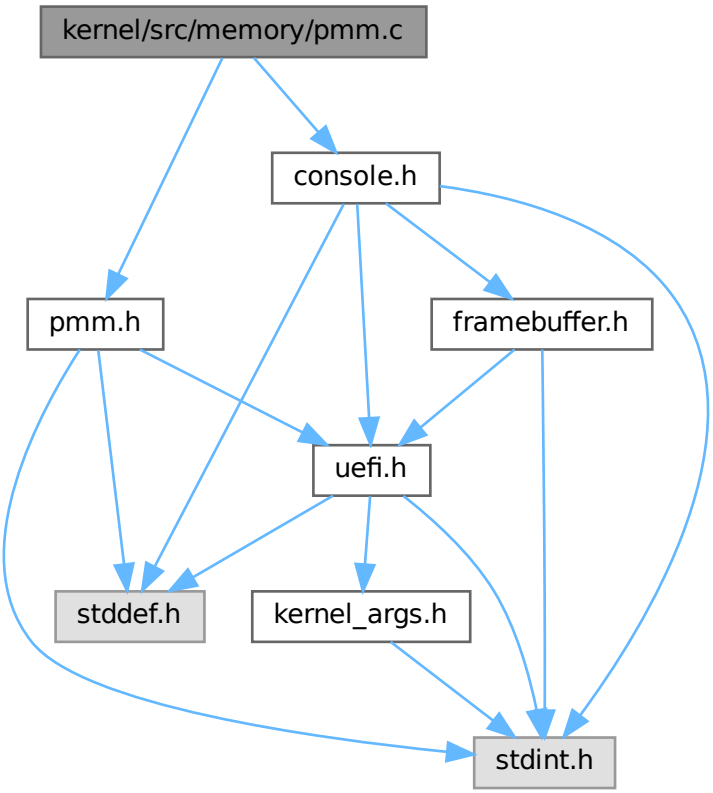
```
kmalloc_stats_t heap_stats = {0}  [static]
```

Referenced by account_allocation(), account_free(), kmalloc_get_stats(), kmalloc_init(), and update_free_bytes().

7.166 kernel/src/memory/pmm.c File Reference

```
#include "pmm.h"
#include "console.h"
```

Include dependency graph for pmm.c:



Macros

- #define PMM_MAX_BITMAP_BYTES (2u ∗ 1024u ∗ 1024u) /∗ Tracks up to ∼64 GiB ∗/

  pmm.c - Physical Memory Manager (bitmap allocator)
- #define PMM_BITMAP_BITS (PMM_MAX_BITMAP_BYTES ∗ 8ull)

Functions

- static void pmm_memset (void ∗dst, uint8_t value, size_t size)
- static uint64_t pmm_bytes_to_pages (uint64_t bytes)
- static void bitmap_set (uint64_t bit)
- static void bitmap_clear (uint64_t bit)
- static int bitmap_test (uint64_t bit)
- static void bitmap_mark_range (uint64_t start_page, uint64_t page_count, int used)
- static uint64_t find_contiguous_free (uint64_t pages_needed)
- static int is_usable_memory (uint32_t type)
- static int is_ram_type (uint32_t type)
- static void recompute_free_pages (void)
- int pmm_init (BootInfo ∗boot_info)
- int pmm_is_initialized (void)
- uint64_t pmm_alloc_page (void)
- uint64_t pmm_alloc_contiguous (uint64_t num_pages)
- void pmm_free_page (uint64_t paddr)
- void pmm_free_contiguous (uint64_t paddr, uint64_t num_pages)
- pmm_stats_t pmm_get_stats (void)

Variables

- static const uint64_t pmm_alloc_floor_page = 0x100000u / PMM_PAGE_SIZE
- static uint8_t pmm_bitmap [(2u ∗1024u ∗1024u)]
- static uint64_t pmm_total_pages = 0
- static uint64_t pmm_free_pages = 0
- static int pmm_initialized = 0
- static uint64_t pmm_tracked_pages = 0
- char __kernel_start [ ]
- char __kernel_end [ ]

7.166.1 Macro Definition Documentation

7.166.1.1 PMM_BITMAP_BITS

```
#define PMM_BITMAP_BITS (PMM_MAX_BITMAP_BYTES ∗ 8ull)
```

7.166.1.2 PMM_MAX_BITMAP_BYTES

```
#define PMM_MAX_BITMAP_BYTES (2u ∗ 1024u ∗ 1024u) /∗ Tracks up to ∼64 GiB ∗/
```

pmm.c - Physical Memory Manager (bitmap allocator)

Builds a 1-bit-per-page bitmap from the UEFI memory map and provides primitive page allocation for early kernel bring-up. The allocator favors simplicity and debuggability; performance tuning can happen once paging and interrupts are in place.

7.166.2 Function Documentation

7.166.2.1 bitmap_clear()

```
static void bitmap_clear (
            uint64_t bit )  [inline], [static]
```

References pmm_bitmap.

Referenced by bitmap_mark_range().

Here is the caller graph for this function:

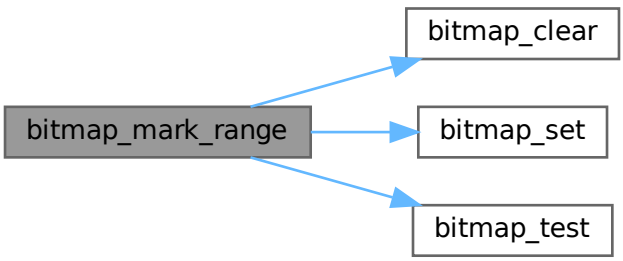
7.166.2.2 bitmap_mark_range()

```
static void bitmap_mark_range (
            uint64_t start_page,
            uint64_t page_count,
            int used )  [static]
```

References bitmap_clear(), bitmap_set(), bitmap_test(), pmm_free_pages, and pmm_tracked_pages.

Referenced by pmm_alloc_contiguous(), pmm_alloc_page(), pmm_free_contiguous(), and pmm_init().

Here is the call graph for this function:



Here is the caller graph for this function:

7.166.2.3 bitmap_set()

```
static void bitmap_set (
            uint64_t bit )  [inline], [static]
```

References pmm_bitmap.

Referenced by bitmap_mark_range().

Here is the caller graph for this function:

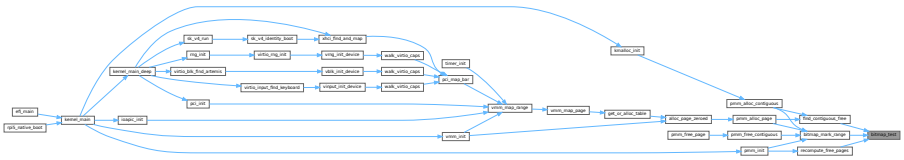
7.166.2.4 bitmap_test()

```
static int bitmap_test (
            uint64_t bit )  [inline], [static]
```

References pmm_bitmap.

Referenced by bitmap_mark_range(), find_contiguous_free(), and recompute_free_pages().

Here is the caller graph for this function:



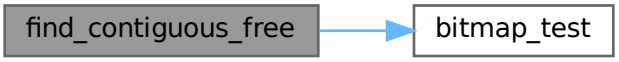
7.166.2.5 find_contiguous_free()

```
static uint64_t find_contiguous_free (
            uint64_t pages_needed )  [static]
```

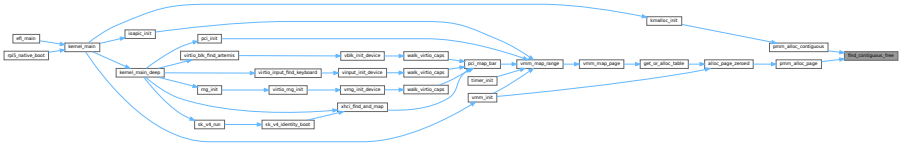
References bitmap_test(), pmm_alloc_floor_page, and pmm_tracked_pages.

Referenced by pmm_alloc_contiguous(), and pmm_alloc_page().

Here is the call graph for this function:



Here is the caller graph for this function:



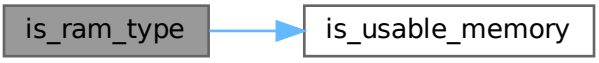
7.166.2.6 is_ram_type()

```
static int is_ram_type (
            uint32_t type ) [static]
```

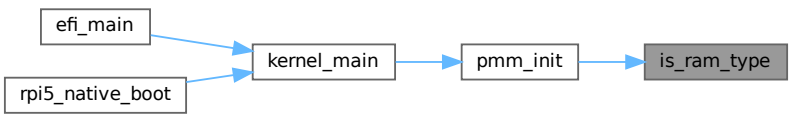
References EfiACPIMemoryNVS, EfiACPIReclaimMemory, EfiRuntimeServicesCode, EfiRuntimeServicesData, and is_usable_memory().

Referenced by pmm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



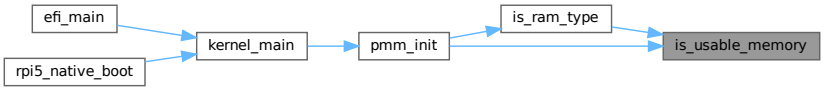
7.166.2.7 is_usable_memory()

```
static int is_usable_memory (
            uint32_t type )  [static]
```

References EfiConventionalMemory.

Referenced by is_ram_type(), and pmm_init().

Here is the caller graph for this function:



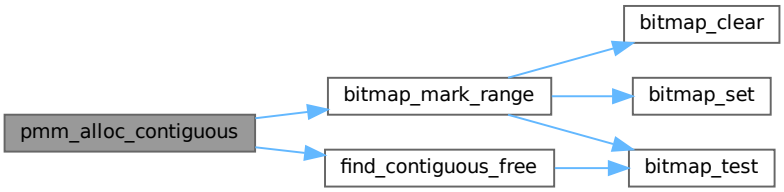
7.166.2.8 pmm_alloc_contiguous()

```
uint64_t pmm_alloc_contiguous (
            uint64_t num_pages )
```

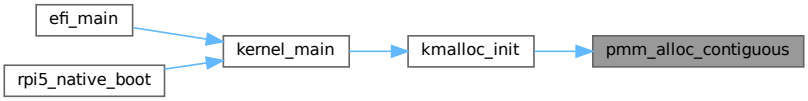
References bitmap_mark_range(), find_contiguous_free(), pmm_initialized, and PMM_PAGE_SIZE.

Referenced by kmalloc_init().

Here is the call graph for this function:



Here is the caller graph for this function:



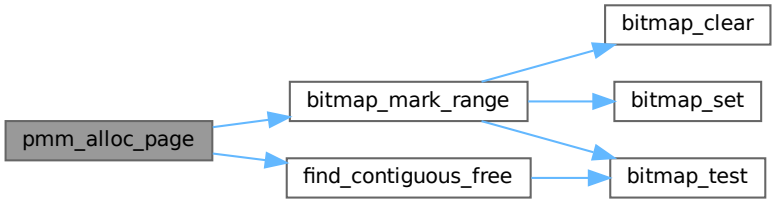
7.166.2.9 pmm_alloc_page()

```
uint64_t pmm_alloc_page (
            void  )
```

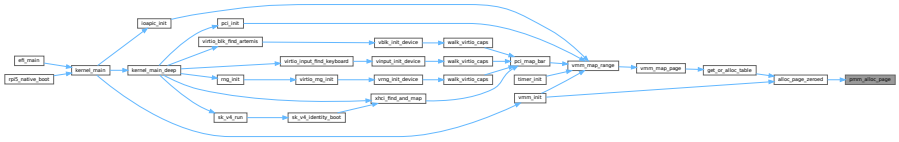
References bitmap_mark_range(), find_contiguous_free(), pmm_initialized, and PMM_PAGE_SIZE.

Referenced by alloc_page_zeroed().

Here is the call graph for this function:



Here is the caller graph for this function:



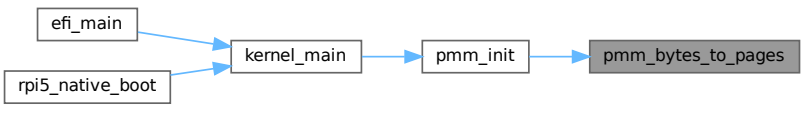
7.166.2.10 pmm_bytes_to_pages()

```
static uint64_t pmm_bytes_to_pages (
            uint64_t bytes )  [inline], [static]
```

References bytes, and PMM_PAGE_SIZE.

Referenced by pmm_init().

Here is the caller graph for this function:



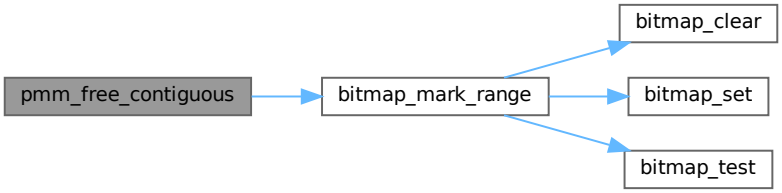
7.166.2.11 pmm_free_contiguous()

```
void pmm_free_contiguous (
            uint64_t paddr,
            uint64_t num_pages )
```

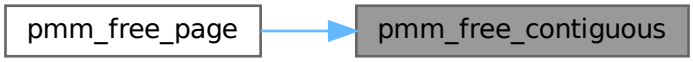
References bitmap_mark_range(), pmm_initialized, PMM_PAGE_SIZE, and pmm_tracked_pages.

Referenced by pmm_free_page().

Here is the call graph for this function:



Here is the caller graph for this function:

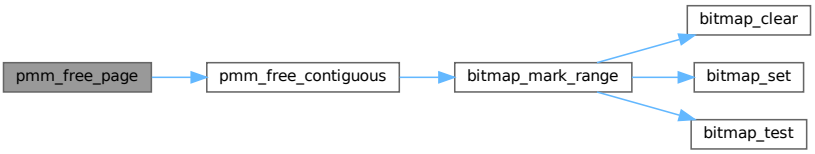


7.166.2.12 pmm_free_page()

```
void pmm_free_page (
            uint64_t paddr )
```

References pmm_free_contiguous(), pmm_initialized, and PMM_PAGE_SIZE.

Here is the call graph for this function:



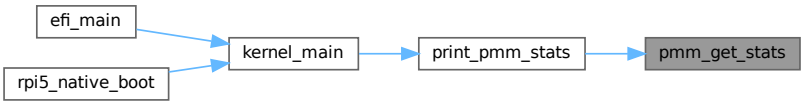
7.166.2.13 pmm_get_stats()

```
pmm_stats_t pmm_get_stats (
            void  )
```

References pmm_stats_t::free_bytes, pmm_stats_t::free_pages, pmm_free_pages, PMM_PAGE_SIZE, pmm_total_pages, pmm_stats_t::total_bytes, pmm_stats_t::total_pages, pmm_stats_t::used_bytes, and pmm_stats_t::used_pages.

Referenced by print_pmm_stats().

Here is the caller graph for this function:



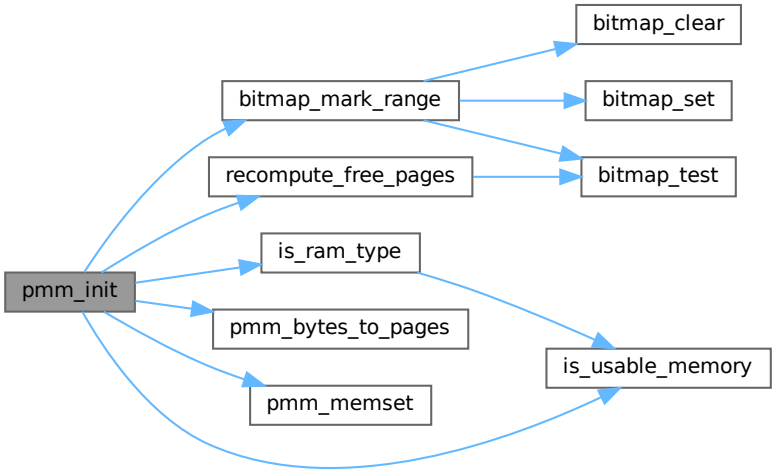
7.166.2.14 pmm_init()

```
int pmm_init (
            BootInfo * boot_info )
```

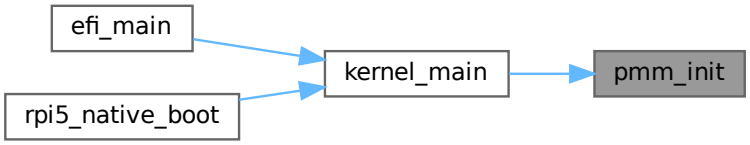
References __kernel_end, __kernel_start, bitmap_mark_range(), is_ram_type(), is_usable_memory(), BootInfo::memory_map, BootInfo::memory_map_descriptor_size, BootInfo::memory_map_size, EFI_MEMORY_DESCRIPTOR::NumberOfPages, EFI_MEMORY_DESCRIPTOR::PhysicalStart, pmm_alloc_floor_page, pmm_bitmap, PMM_BITMAP_BITS, pmm_bytes_to_pages(), pmm_free_pages, pmm_initialized, pmm_memset(), PMM_PAGE_SIZE, pmm_total_pages, pmm_tracked_pages, recompute_free_pages(), and EFI_MEMORY_DESCRIPTOR::Type.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



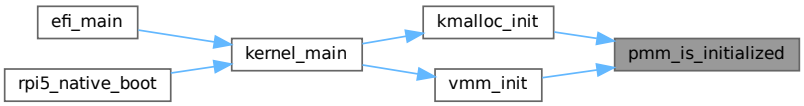
7.166.2.15 pmm_is_initialized()

```
int pmm_is_initialized (
            void  )
```

References pmm_initialized.

Referenced by kmalloc_init(), and vmm_init().

Here is the caller graph for this function:

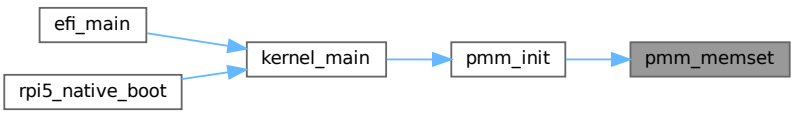


7.166.2.16 pmm_memset()

```
static void pmm_memset (
            void * dst,
            uint8_t value,
            size_t size )  [static]
```

Referenced by pmm_init().

Here is the caller graph for this function:



7.166.2.17 recompute_free_pages()

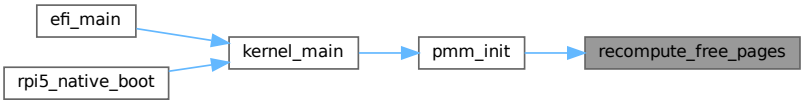
```
static void recompute_free_pages (
            void  )  [static]
```

References bitmap_test(), pmm_free_pages, and pmm_tracked_pages.

Referenced by pmm_init().

Here is the call graph for this function:

Here is the caller graph for this function:



7.166.3 Variable Documentation

7.166.3.1 __kernel_end

```
char __kernel_end[]  [extern]
```

Referenced by pmm_init(), and rpi5_mmu_uncached().

7.166.3.2 __kernel_start

```
char __kernel_start[]  [extern]
```

Referenced by pmm_init().

7.166.3.3 pmm_alloc_floor_page

```
const uint64_t pmm_alloc_floor_page = 0x100000u / PMM_PAGE_SIZE  [static]
```

Referenced by find_contiguous_free(), and pmm_init().

7.166.3.4 pmm_bitmap

```
uint8_t pmm_bitmap[(2u *1024u *1024u)]  [static]
```

Referenced by bitmap_clear(), bitmap_set(), bitmap_test(), and pmm_init().

7.166.3.5 pmm_free_pages

```
uint64_t pmm_free_pages = 0  [static]
```

Referenced by bitmap_mark_range(), pmm_get_stats(), pmm_init(), and recompute_free_pages().

7.166.3.6 pmm_initialized

```
int pmm_initialized = 0  [static]
```

Referenced by pmm_alloc_contiguous(), pmm_alloc_page(), pmm_free_contiguous(), pmm_free_page(), pmm_init(), and pmm_is_initialized().

7.166.3.7 pmm_total_pages

```
uint64_t pmm_total_pages = 0  [static]
```

Referenced by pmm_get_stats(), and pmm_init().

7.166.3.8 pmm_tracked_pages

```
uint64_t pmm_tracked_pages = 0  [static]
```

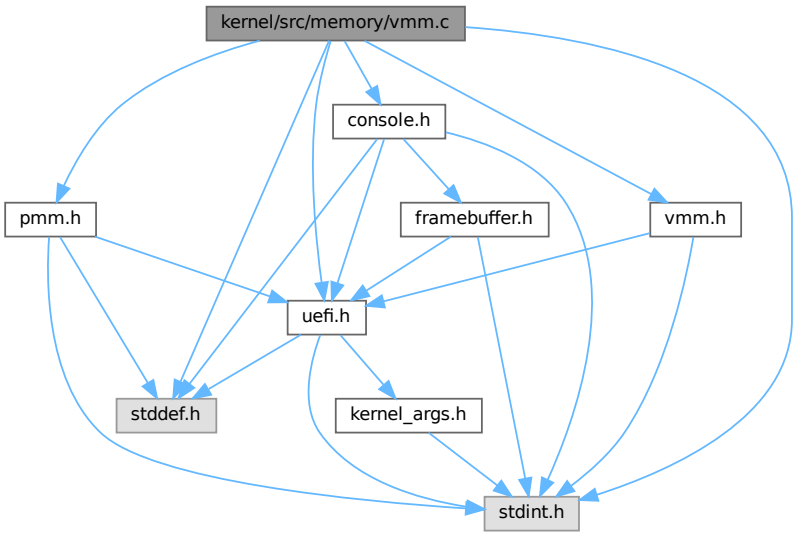
Referenced by bitmap_mark_range(), find_contiguous_free(), pmm_free_contiguous(), pmm_init(), and recompute_free_pages().

7.167 kernel/src/memory/vmm.c File Reference

```
#include <stddef.h>
#include <stdint.h>
#include "vmm.h"
#include "pmm.h"
#include "console.h"
```

```
#include "uefi.h"
```

Include dependency graph for vmm.c:



Macros

- #define PTE_PRESENT (1ull << 0)

  vmm.c - 4-level paging Virtual Memory Manager (x86_64)
- #define PTE_WRITABLE (1ull << 1)
- #define PTE_USER (1ull << 2)
- #define PTE_PWT (1ull << 3)
- #define PTE_PCD (1ull << 4)
- #define PTE_ACCESSED (1ull << 5)
- #define PTE_DIRTY (1ull << 6)
- #define PTE_PS (1ull << 7)
- #define PTE_GLOBAL (1ull << 8)
- #define PTE_NX (1ull << 63)
- #define ADDR_MASK 0x000FFFFFFFFFF000ull

Functions

- static int is_ram_type (uint32_t type)
- static uint64_t * paddr_to_virt (uint64_t paddr)
- static uint64_t alloc_page_zeroed (void)
- static uint16_t pml4_index (uint64_t vaddr)
- static uint16_t pdpt_index (uint64_t vaddr)
- static uint16_t pd_index (uint64_t vaddr)
- static uint16_t pt_index (uint64_t vaddr)
- static uint64_t * get_or_alloc_table (uint64_t *parent, uint16_t idx)
- static uint64_t * get_table (uint64_t *parent, uint16_t idx)
- static uint64_t make_pte (uint64_t paddr, uint64_t flags)
- static void invlpg (uint64_t vaddr)

- static void load_cr3 (uint64_t pml4_phys)
- static uint64_t * resolve_pt (uint64_t vaddr, uint16_t *idx_out)
- int vmm_map_page (uint64_t vaddr, uint64_t paddr, uint64_t flags)
- int vmm_unmap_page (uint64_t vaddr)
- uint64_t vmm_get_paddr (uint64_t vaddr)
- int vmm_map_range (uint64_t vaddr, uint64_t paddr, uint64_t size, uint64_t flags)
- int vmm_query_page (uint64_t vaddr, vmm_page_info_t *info)
- int vmm_init (BootInfo *boot_info)

Variables

- static uint64_t vmm_root_pml4_phys = 0
- static const uint64_t lapic_phys_base = 0xFEE00000ull
- static const uint64_t hpet_phys_base = 0xFED00000ull

7.167.1 Macro Definition Documentation

7.167.1.1 ADDR_MASK

```
#define ADDR_MASK 0x000FFFFFFFFFF000ull
```

7.167.1.2 PTE_ACCESSED

```
#define PTE_ACCESSED (1ull << 5)
```

7.167.1.3 PTE_DIRTY

```
#define PTE_DIRTY (1ull << 6)
```

7.167.1.4 PTE_GLOBAL

```
#define PTE_GLOBAL (1ull << 8)
```

7.167.1.5 PTE_NX

```
#define PTE_NX (1ull << 63)
```

7.167.1.6 PTE_PCD

```
#define PTE_PCD (1ull << 4)
```

7.167.1.7 PTE_PRESENT

```
#define PTE_PRESENT (1ull << 0)
```

vmm.c - 4-level paging Virtual Memory Manager (x86_64)

Minimal bring-up: identity maps the first 16 MiB, supports single-page map/unmap/query, and switches to the new page table root.

7.167.1.8 PTE_PS

```
#define PTE_PS (1ull << 7)
```

7.167.1.9 PTE_PWT

```
#define PTE_PWT (1ull << 3)
```

7.167.1.10 PTE_USER

```
#define PTE_USER (1ull << 2)
```

7.167.1.11 PTE_WRITABLE

```
#define PTE_WRITABLE (1ull << 1)
```

7.167.2 Function Documentation

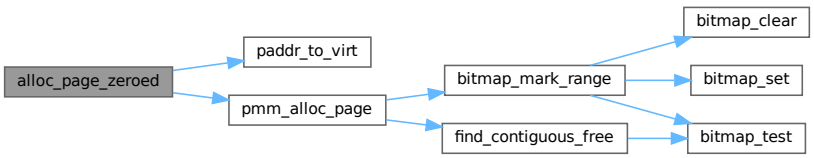
7.167.2.1 alloc_page_zeroed()

```
static uint64_t alloc_page_zeroed (
            void  )  [static]
```

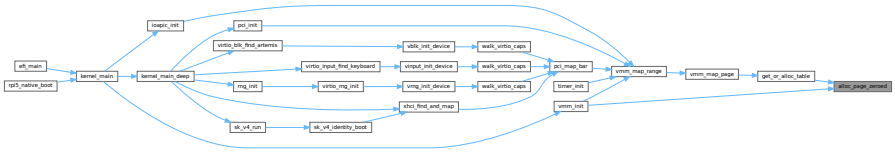
References paddr_to_virt(), pmm_alloc_page(), and VMM_PAGE_SIZE.

Referenced by get_or_alloc_table(), and vmm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



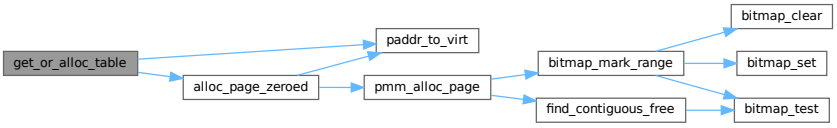
7.167.2.2 get_or_alloc_table()

```
static uint64_t * get_or_alloc_table (
            uint64_t * parent,
            uint16_t idx )  [static]
```

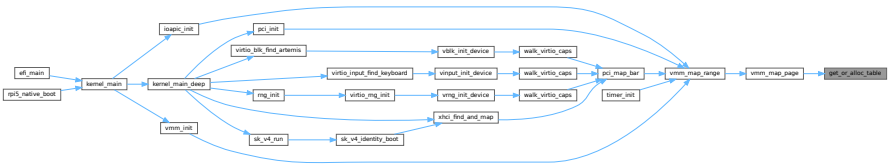
References ADDR_MASK, alloc_page_zeroed(), entry, paddr_to_virt(), PTE_PRESENT, and PTE_WRITABLE.

Referenced by vmm_map_page().

Here is the call graph for this function:



Here is the caller graph for this function:



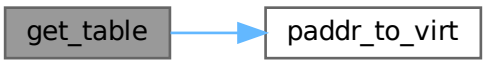
7.167.2.3 get_table()

```
static uint64_t * get_table (
            uint64_t * parent,
            uint16_t idx )  [static]
```

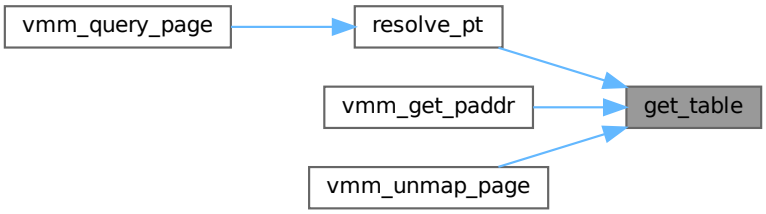
References ADDR_MASK, entry, paddr_to_virt(), and PTE_PRESENT.

Referenced by resolve_pt(), vmm_get_paddr(), and vmm_unmap_page().

Here is the call graph for this function:



Here is the caller graph for this function:

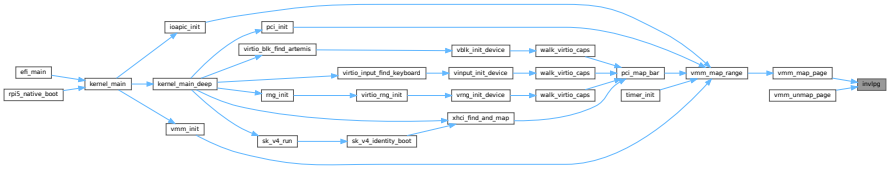


7.167.2.4 invlpg()

```
static void invlpg (
            uint64_t vaddr )  [inline], [static]
```

Referenced by vmm_map_page(), and vmm_unmap_page().

Here is the caller graph for this function:



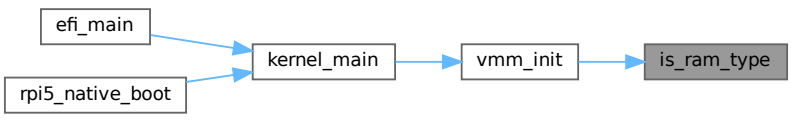
7.167.2.5 is_ram_type()

```
static int is_ram_type (
            uint32_t type )  [static]
```

References EfiACPIMemoryNVS, EfiACPIReclaimMemory, EfiBootServicesCode, EfiBootServicesData, EfiConventionalMemory, EfiLoaderCode, EfiLoaderData, EfiRuntimeServicesCode, and EfiRuntimeServicesData.

Referenced by vmm_init().

Here is the caller graph for this function:

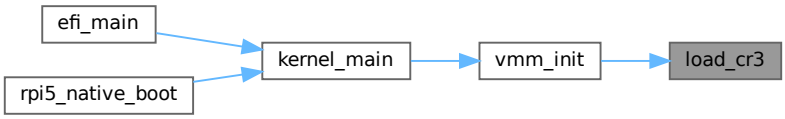


7.167.2.6 load_cr3()

```
static void load_cr3 (
            uint64_t pml4_phys )  [inline], [static]
```

Referenced by vmm_init().

Here is the caller graph for this function:



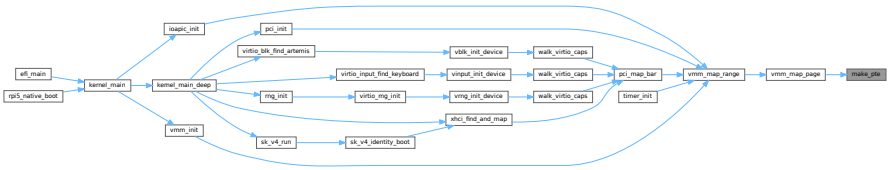
7.167.2.7 make_pte()

```
static uint64_t make_pte (
            uint64_t paddr,
            uint64_t flags )  [static]
```

References ADDR_MASK, flags, PTE_NX, PTE_PCD, PTE_PRESENT, PTE_USER, PTE_WRITABLE, VMM_FLAG_CACHE_DISABLE, VMM_FLAG_NX, VMM_FLAG_PRESENT, VMM_FLAG_USER, and VMM_FLAG_WRITABLE.

Referenced by vmm_map_page().

Here is the caller graph for this function:

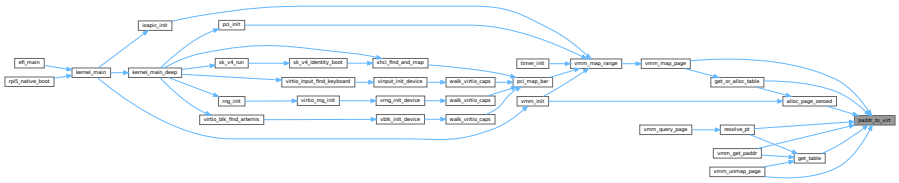


7.167.2.8 paddr_to_virt()

```
static uint64_t * paddr_to_virt (
            uint64_t paddr )  [inline], [static]
```

Referenced by alloc_page_zeroed(), get_or_alloc_table(), get_table(), resolve_pt(), vmm_get_paddr(), vmm_map_page(), and vmm_unmap_page().

Here is the caller graph for this function:

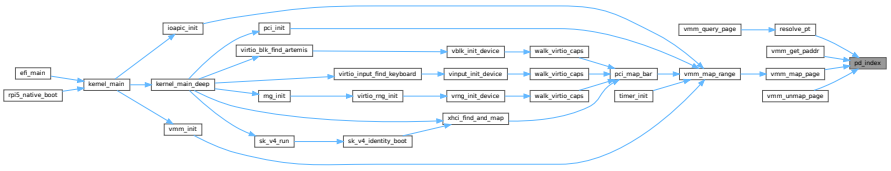


7.167.2.9 pd_index()

```
static uint16_t pd_index (
            uint64_t vaddr )  [inline], [static]
```

Referenced by resolve_pt(), vmm_get_paddr(), vmm_map_page(), and vmm_unmap_page().

Here is the caller graph for this function:

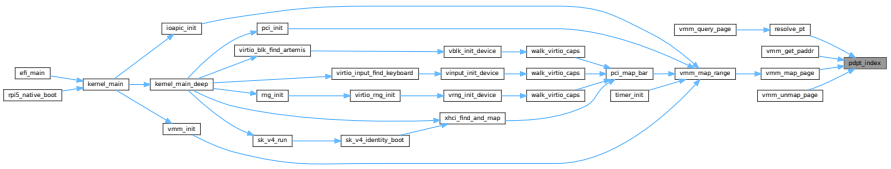


7.167.2.10 pdpt_index()

```
static uint16_t pdpt_index (
            uint64_t vaddr )  [inline], [static]
```

Referenced by resolve_pt(), vmm_get_paddr(), vmm_map_page(), and vmm_unmap_page().

Here is the caller graph for this function:

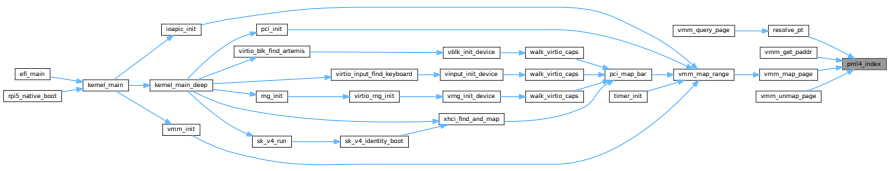


7.167.2.11 pml4_index()

```
static uint16_t pml4_index (
            uint64_t vaddr )  [inline], [static]
```

Referenced by resolve_pt(), vmm_get_paddr(), vmm_map_page(), and vmm_unmap_page().

Here is the caller graph for this function:

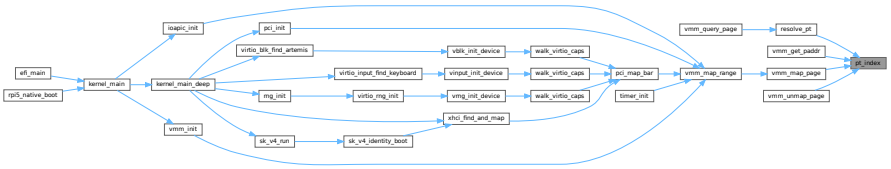


7.167.2.12 pt_index()

```
static uint16_t pt_index (
            uint64_t vaddr )  [inline], [static]
```

Referenced by resolve_pt(), vmm_get_paddr(), vmm_map_page(), and vmm_unmap_page().

Here is the caller graph for this function:



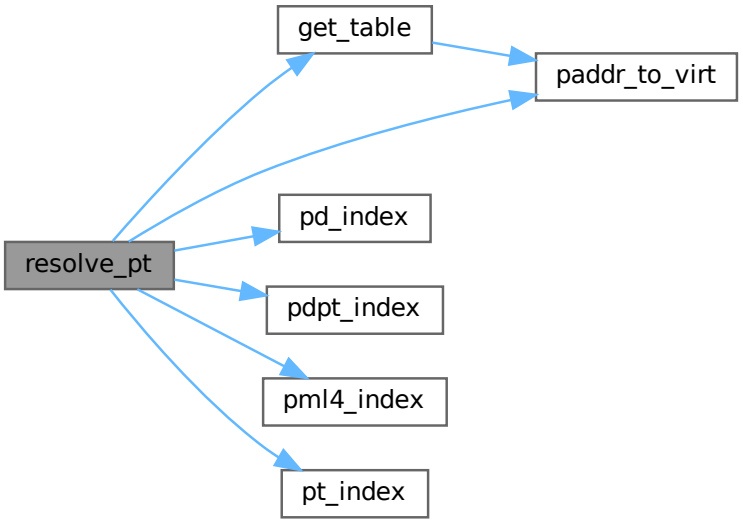
7.167.2.13 resolve_pt()

```
static uint64_t * resolve_pt (
            uint64_t vaddr,
            uint16_t * idx_out )  [static]
```

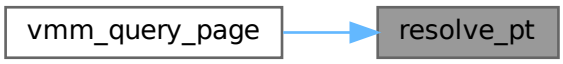
References get_table(), paddr_to_virt(), pd_index(), pdpt_index(), pml4_index(), pt_index(), and vmm_root_pml4_phys.

Referenced by vmm_query_page().

Here is the call graph for this function:



Here is the caller graph for this function:

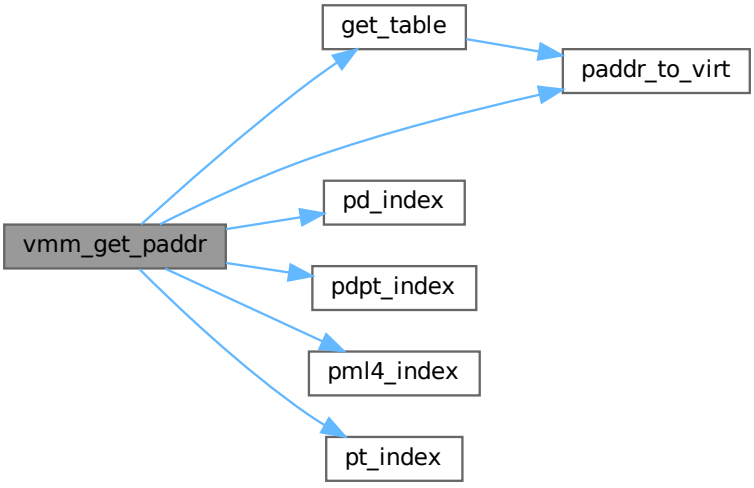


7.167.2.14 vmm_get_paddr()

```
uint64_t vmm_get_paddr (
            uint64_t vaddr )
```

References ADDR_MASK, get_table(), paddr_to_virt(), pd_index(), pdpt_index(), pml4_index(), pt_index(), PTE_PRESENT, and vmm_root_pml4_phys.

Here is the call graph for this function:



7.167.2.15 vmm_init()

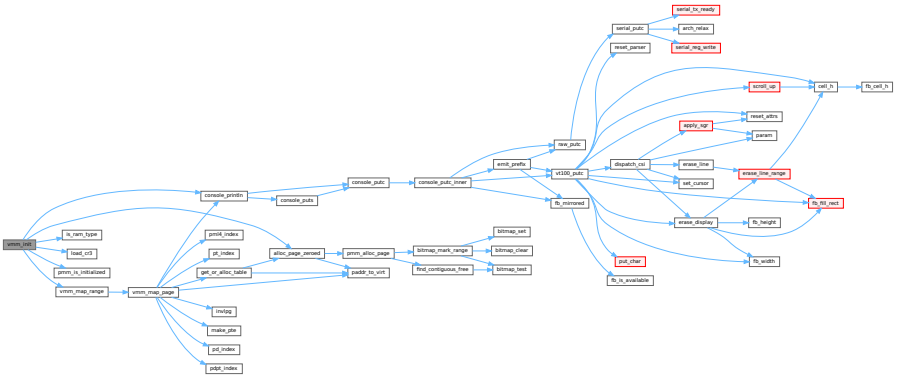
```
int vmm_init (
            BootInfo * boot_info )
```

References alloc_page_zeroed(), FramebufferInfo::base, console_println(), BootInfo::framebuffer, hpet_phys_base, is_ram_type(), lapic_phys_base, load_cr3(), BootInfo::memory_map, BootInfo::memory_map_descriptor_size, BootInfo::memory_map_size, EFI_MEMORY_DESCRIPTOR::NumberOfPages, EFI_MEMORY_DESCRIPTOR::PhysicalStart,

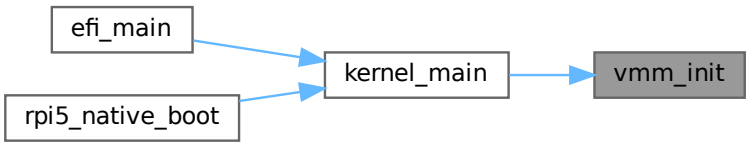
pmm_is_initialized(), FramebufferInfo::size, EFI_MEMORY_DESCRIPTOR::Type, VMM_FLAG_CACHE_DISABLE, VMM_FLAG_WRITABLE, vmm_map_range(), VMM_PAGE_SIZE, and vmm_root_pml4_phys.

Referenced by kernel_main().

Here is the call graph for this function:



Here is the caller graph for this function:



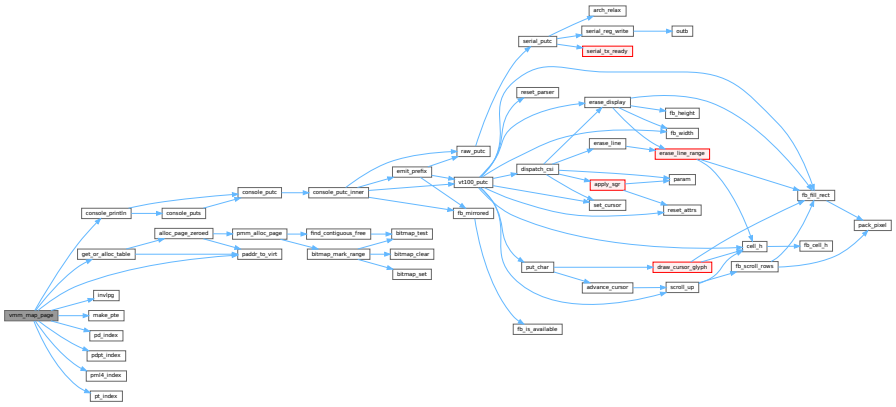
7.167.2.16 vmm_map_page()

```
int vmm_map_page (
            uint64_t vaddr,
            uint64_t paddr,
            uint64_t flags )
```

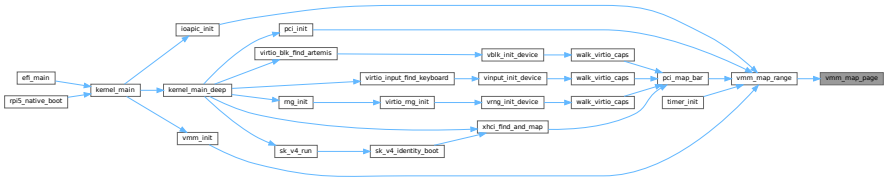
References ADDR_MASK, console_println(), flags, get_or_alloc_table(), invlpg(), make_pte(), paddr_to_virt(), pd_index(), pdpt_index(), pml4_index(), pt_index(), PTE_PRESENT, PTE_WRITABLE, VMM_FLAG_PRESENT, and vmm_root_pml4_phys.

Referenced by vmm_map_range().

Here is the call graph for this function:



Here is the caller graph for this function:



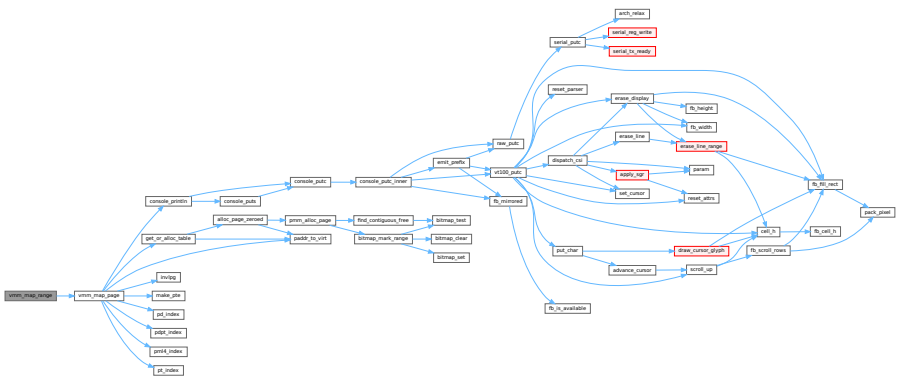
7.167.2.17 vmm_map_range()

```
int vmm_map_range (
            uint64_t vaddr,
            uint64_t paddr,
            uint64_t size,
            uint64_t flags )
```

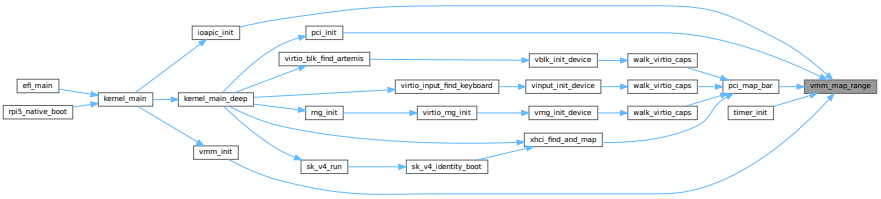
References flags, vmm_map_page(), and VMM_PAGE_SIZE.

Referenced by ioapic_init(), pci_init(), pci_map_bar(), timer_init(), and vmm_init().

Here is the call graph for this function:



Here is the caller graph for this function:

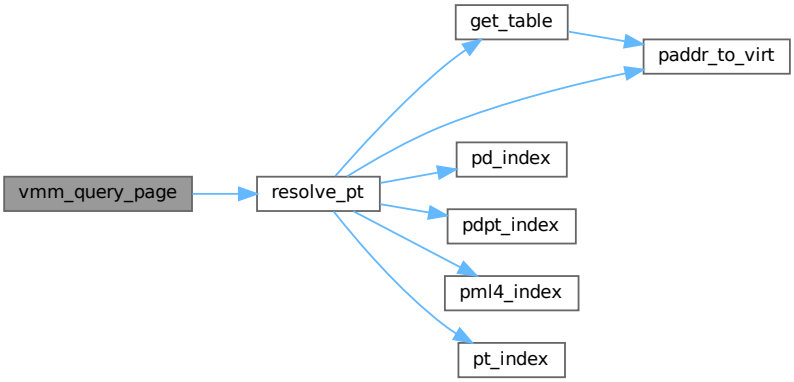


7.167.2.18 vmm_query_page()

```
int vmm_query_page (
            uint64_t vaddr,
            vmm_page_info_t * info )
```

References entry, vmm_page_info_t::executable, vmm_page_info_t::present, PTE_NX, PTE_PRESENT, PTE_WRITABLE, resolve_pt(), and vmm_page_info_t::writable.

Here is the call graph for this function:

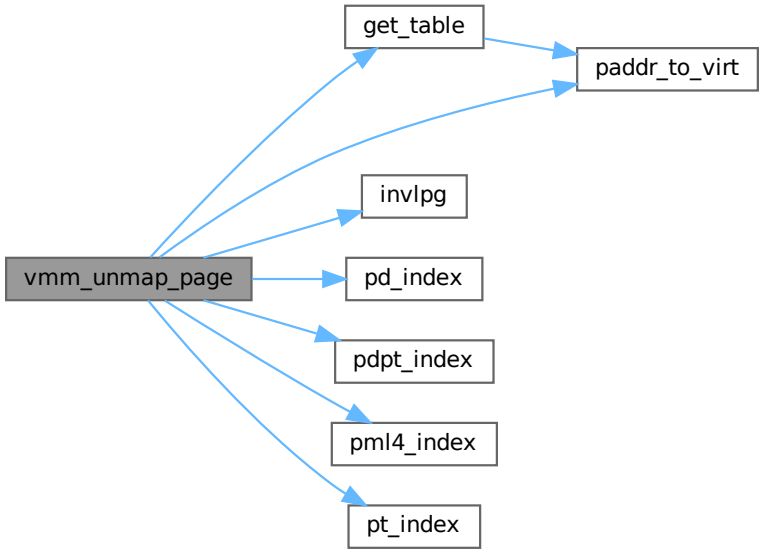


7.167.2.19 vmm_unmap_page()

```
int vmm_unmap_page (
            uint64_t vaddr )
```

References get_table(), invlpg(), paddr_to_virt(), pd_index(), pdpt_index(), pml4_index(), pt_index(), PTE_PRESENT, and vmm_root_pml4_phys.

Here is the call graph for this function:



7.167.3 Variable Documentation

7.167.3.1 hpet_phys_base

```
const uint64_t hpet_phys_base = 0xFED00000ull  [static]
```

Referenced by vmm_init().

7.167.3.2 lapic_phys_base

```
const uint64_t lapic_phys_base = 0xFEE00000ull  [static]
```

Referenced by vmm_init().

7.167.3.3 vmm_root_pml4_phys

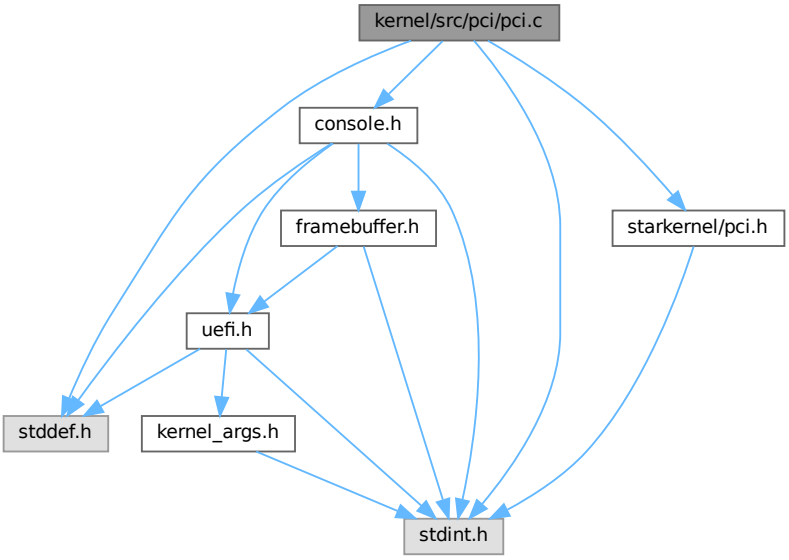
```
uint64_t vmm_root_pml4_phys = 0  [static]
```

Referenced by resolve_pt(), vmm_get_paddr(), vmm_init(), vmm_map_page(), and vmm_unmap_page().

7.168 kernel/src/pci/pci.c File Reference

```
#include <stddef.h>
#include <stdint.h>
#include "starkernel/pci.h"
#include "console.h"
```

Include dependency graph for pci.c:



Data Structures

- struct __attribute__

  ioapic.c - I/O APIC driver (amd64)

Functions

- static volatile uint32_t * ecam_cfg_ptr (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off)
- static uint32_t ecam_read32 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off)
- static uint16_t ecam_read16 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off)
- static uint8_t ecam_read8 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off)
- static void ecam_write32 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off, uint32_t v)
- static void ecam_write16 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off, uint16_t v)
- static void ecam_write8 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off, uint8_t v)
- static uint32_t cfg_read32 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off)
- static uint16_t cfg_read16 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off)
- static uint8_t cfg_read8 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off)
- static void cfg_write32 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off, uint32_t v)
- static void cfg_write16 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off, uint16_t v)
- static void cfg_write8 (uint8_t bus, uint8_t slot, uint8_t fn, uint16_t off, uint8_t v)
- static int sig4_eq (const uint8_t *p, const char *s)
- static int sig8_eq (const uint8_t *p, const char *s)

- static int parse_mcfg (void *rsdp)
- int pci_init (void *rsdp)
- int pci_find_first (uint16_t vendor_id, uint16_t device_id, PciDevice *out)
- int pci_find_class (uint8_t class_code, uint8_t subclass, uint8_t prog_if, unsigned index, PciDevice *out)
- uint32_t pci_read32 (const PciDevice *d, uint16_t off)
- uint16_t pci_read16 (const PciDevice *d, uint16_t off)
- uint8_t pci_read8 (const PciDevice *d, uint16_t off)
- void pci_write32 (const PciDevice *d, uint16_t off, uint32_t v)
- void pci_write16 (const PciDevice *d, uint16_t off, uint16_t v)
- void pci_write8 (const PciDevice *d, uint16_t off, uint8_t v)
- void pci_enable (const PciDevice *d)
- uint64_t pci_bar (const PciDevice *d, int bar_idx)
- uint64_t pci_bar_size (const PciDevice *d, int bar_idx)
- int pci_map_bar (uint64_t phys_addr, uint64_t size)

Variables

- static uint64_t g_ecam_base = 0
- static uint8_t g_bus_first = 0
- static uint8_t g_bus_last = 255

7.168.1 Function Documentation

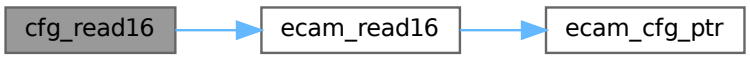
7.168.1.1 cfg_read16()

```
static uint16_t cfg_read16 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off )  [static]
```

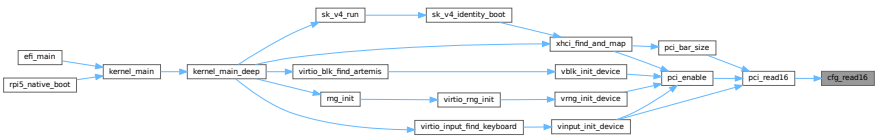
References ecam_read16().

Referenced by pci_read16().

Here is the call graph for this function:



Here is the caller graph for this function:



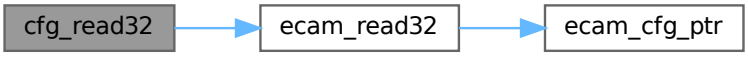
7.168.1.2 cfg_read32()

```
static uint32_t cfg_read32 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off )  [static]
```

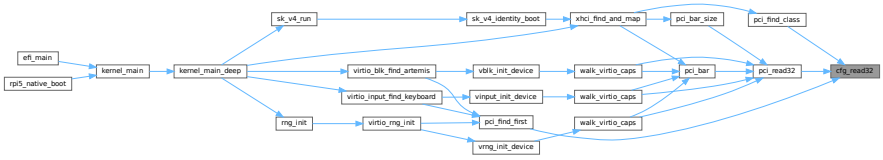
References ecam_read32().

Referenced by pci_find_class(), pci_find_first(), and pci_read32().

Here is the call graph for this function:



Here is the caller graph for this function:



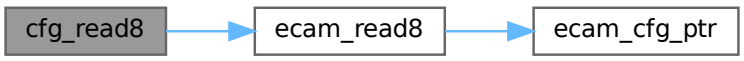
7.168.1.3 cfg_read8()

```
static uint8_t cfg_read8 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off )  [static]
```

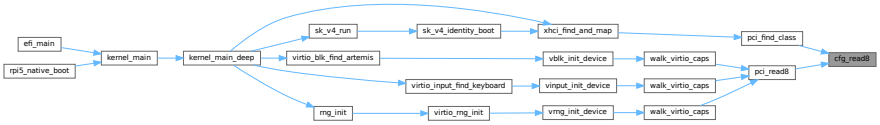
References ecam_read8().

Referenced by pci_find_class(), and pci_read8().

Here is the call graph for this function:



Here is the caller graph for this function:



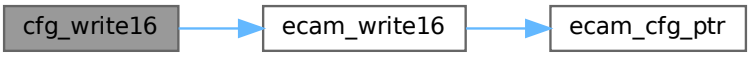
7.168.1.4 cfg_write16()

```
static void cfg_write16 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off,
            uint16_t v )  [static]
```

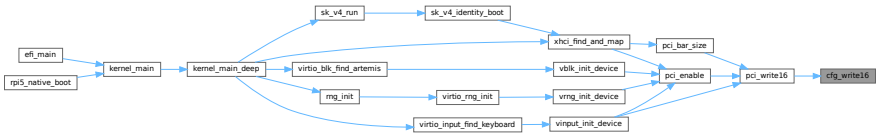
References ecam_write16().

Referenced by pci_write16().

Here is the call graph for this function:



Here is the caller graph for this function:



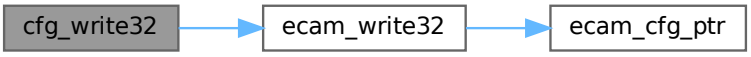
7.168.1.5 cfg_write32()

```
static void cfg_write32 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off,
            uint32_t v )  [static]
```

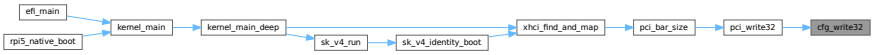
References ecam_write32().

Referenced by pci_write32().

Here is the call graph for this function:



Here is the caller graph for this function:



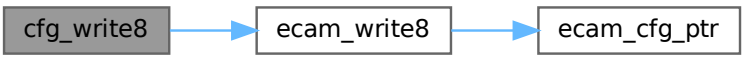
7.168.1.6 cfg_write8()

```
static void cfg_write8 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off,
            uint8_t v )  [static]
```

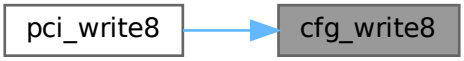
References ecam_write8().

Referenced by pci_write8().

Here is the call graph for this function:



Here is the caller graph for this function:



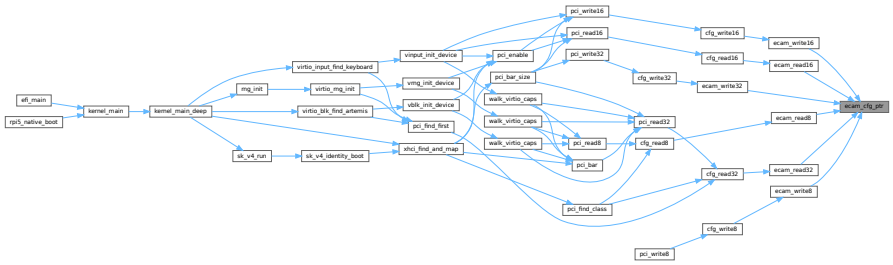
7.168.1.7 ecam_cfg_ptr()

```
static volatile uint32_t * ecam_cfg_ptr (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off )  [static]
```

References g_ecam_base.

Referenced by ecam_read16(), ecam_read32(), ecam_read8(), ecam_write16(), ecam_write32(), and ecam_write8().

Here is the caller graph for this function:



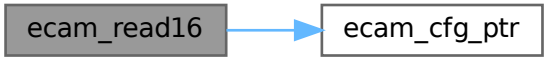
7.168.1.8 ecam_read16()

```
static uint16_t ecam_read16 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off )  [static]
```

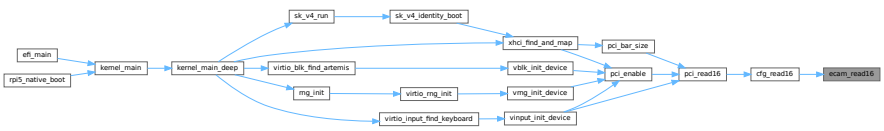
References ecam_cfg_ptr().

Referenced by cfg_read16().

Here is the call graph for this function:



Here is the caller graph for this function:



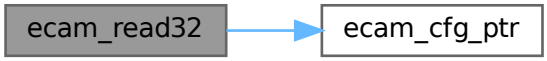
7.168.1.9 ecam_read32()

```
static uint32_t ecam_read32 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off )  [static]
```

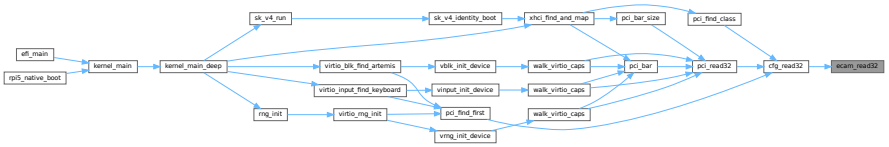
References ecam_cfg_ptr().

Referenced by cfg_read32().

Here is the call graph for this function:



Here is the caller graph for this function:



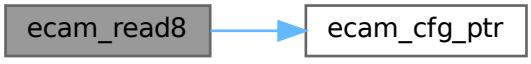
7.168.1.10 ecam_read8()

```
static uint8_t ecam_read8 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off )  [static]
```

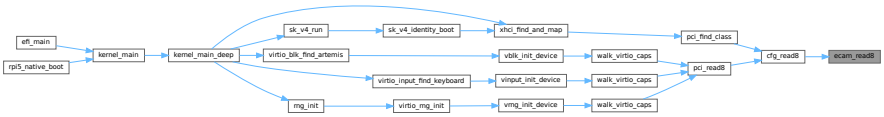
References ecam_cfg_ptr().

Referenced by cfg_read8().

Here is the call graph for this function:



Here is the caller graph for this function:



7.168.1.11 ecam_write16()

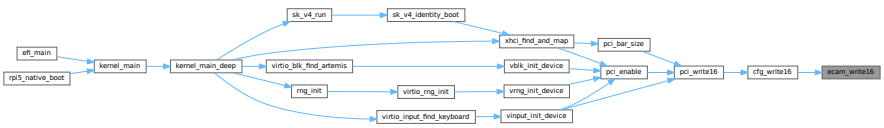
```
static void ecam_write16 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off,
            uint16_t v )  [static]
```

References ecam_cfg_ptr().

Referenced by cfg_write16().

Here is the call graph for this function:

Here is the caller graph for this function:



7.168.1.12 ecam_write32()

```
static void ecam_write32 (
            uint8_t bus,
            uint8_t slot,
            uint8_t fn,
            uint16_t off,
            uint32_t v )  [static]
```

References ecam_cfg_ptr().

Referenced by cfg_write32().

Here is the call graph for this function:

Here is the caller graph for this function:

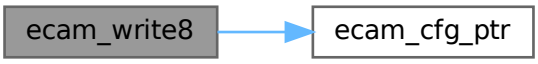
7.168.1.13 ecam_write8()

```
static void ecam_write8 (
             uint8_t bus,
             uint8_t slot,
             uint8_t fn,
             uint16_t off,
             uint8_t v )  [static]
```

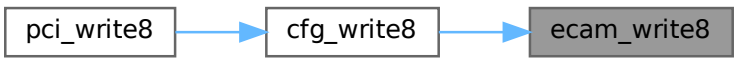
References ecam_cfg_ptr().

Referenced by cfg_write8().

Here is the call graph for this function:



Here is the caller graph for this function:



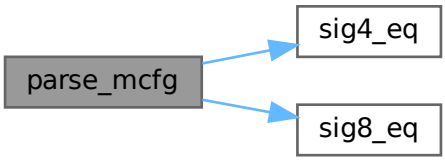
7.168.1.14 parse_mcfg()

```
static int parse_mcfg (
            void * rsdp )  [static]
```

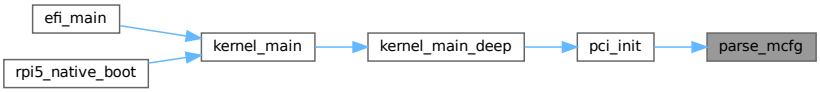
References g_bus_first, g_bus_last, g_ecam_base, sig4_eq(), and sig8_eq().

Referenced by pci_init().

Here is the call graph for this function:



Here is the caller graph for this function:



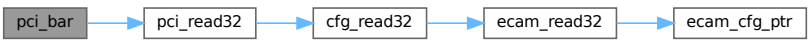
7.168.1.15 pci_bar()

```
uint64_t pci_bar (
            const PciDevice * d,
            int bar_idx )
```

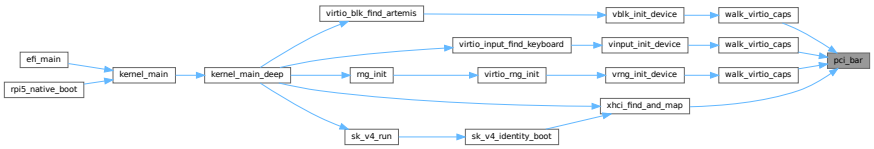
References PCI_BAR_MEM_64, PCI_BAR_TYPE_MASK, PCI_CFG_BAR0, and pci_read32().

Referenced by walk_virtio_caps(), walk_virtio_caps(), walk_virtio_caps(), and xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:



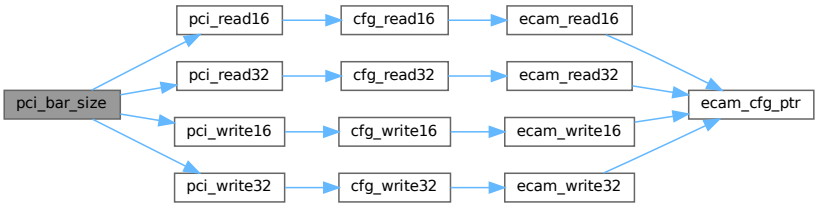
7.168.1.16 pci_bar_size()

```
uint64_t pci_bar_size (
            const PciDevice * d,
            int bar_idx )
```

References PCI_BAR_MEM_64, PCI_BAR_TYPE_MASK, PCI_CFG_BAR0, PCI_CFG_COMMAND, PCI_CMD_MEM_SPACE, pci_read16(), pci_read32(), pci_write16(), and pci_write32().

Referenced by xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:

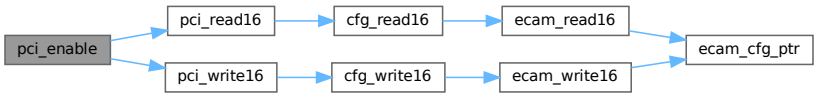
7.168.1.17 pci_enable()

```
void pci_enable (
            const PciDevice * d )
```

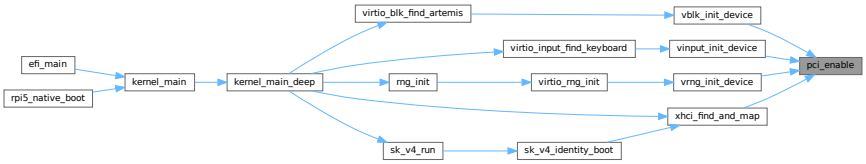
References PCI_CFG_COMMAND, PCI_CMD_BUS_MASTER, PCI_CMD_IO_SPACE, PCI_CMD_MEM_SPACE, pci_read16(), and pci_write16().

Referenced by vblk_init_device(), vinput_init_device(), vrng_init_device(), and xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:



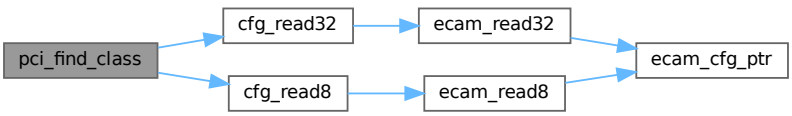
7.168.1.18 pci_find_class()

```
int pci_find_class (
            uint8_t class_code,
            uint8_t subclass,
            uint8_t prog_if,
            unsigned index,
            PciDevice * out )
```

References PciDevice::bus, cfg_read32(), cfg_read8(), PciDevice::device, PciDevice::device_id, PciDevice::function, g_bus_first, g_bus_last, g_ecam_base, PCI_CFG_CLASS_REV, PCI_CFG_HEADER_TYPE, PCI_CFG_VENDOR_ID, and PciDevice::vendor_id.

Referenced by xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:

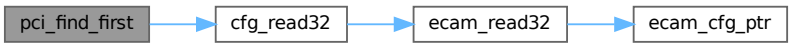
7.168.1.19 pci_find_first()

```
int pci_find_first (
            uint16_t vendor_id,
            uint16_t device_id,
            PciDevice * out )
```

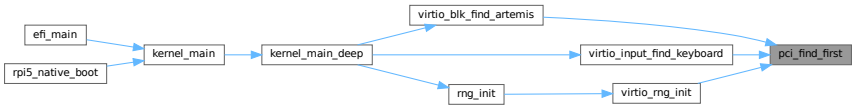
References PciDevice::bus, cfg_read32(), PciDevice::device, PciDevice::device_id, PciDevice::function, PCI_CFG_VENDOR_ID, and PciDevice::vendor_id.

Referenced by virtio_blk_find_artemis(), virtio_input_find_keyboard(), and virtio_rng_init().

Here is the call graph for this function:



Here is the caller graph for this function:

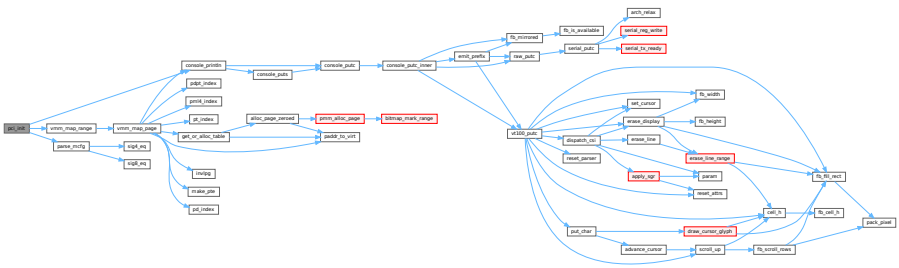


7.168.1.20 pci_init()

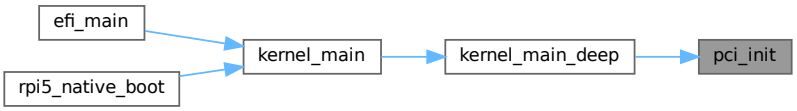
```
int pci_init (
            void * rsdp )
```

References console_println(), g_ecam_base, parse_mcfg(), VMM_FLAG_CACHE_DISABLE, VMM_FLAG_WRITABLE, and vmm_map_range().

Here is the call graph for this function:



Here is the caller graph for this function:



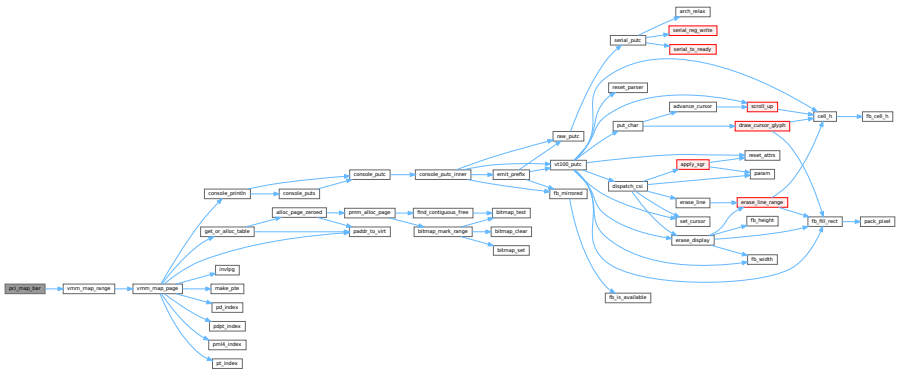
7.168.1.21 pci_map_bar()

```
int pci_map_bar (
            uint64_t phys_addr,
            uint64_t size )
```

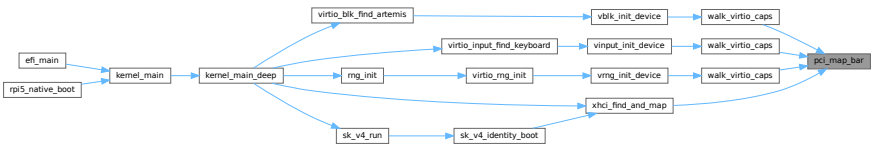
References VMM_FLAG_CACHE_DISABLE, VMM_FLAG_WRITABLE, and vmm_map_range().

Referenced by walk_virtio_caps(), walk_virtio_caps(), walk_virtio_caps(), and xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:



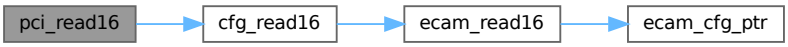
7.168.1.22 pci_read16()

```
uint16_t pci_read16 (
            const PciDevice * d,
            uint16_t off )
```

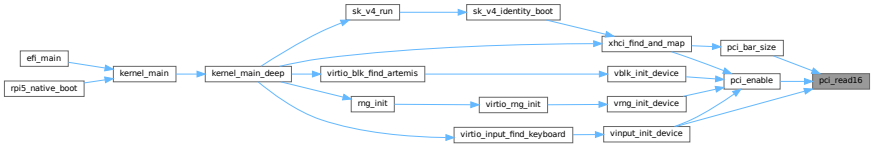
References PciDevice::bus, cfg_read16(), PciDevice::device, and PciDevice::function.

Referenced by pci_bar_size(), pci_enable(), and vinput_init_device().

Here is the call graph for this function:



Here is the caller graph for this function:



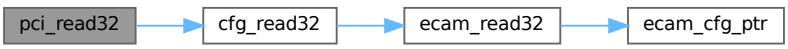
7.168.1.23 pci_read32()

```
uint32_t pci_read32 (
            const PciDevice * d,
            uint16_t off )
```

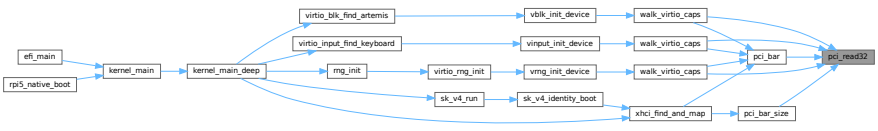
References PciDevice::bus, cfg_read32(), PciDevice::device, and PciDevice::function.

Referenced by pci_bar(), pci_bar_size(), walk_virtio_caps(), walk_virtio_caps(), and walk_virtio_caps().

Here is the call graph for this function:



Here is the caller graph for this function:



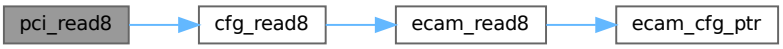
7.168.1.24 pci_read8()

```
uint8_t pci_read8 (
            const PciDevice * d,
            uint16_t off )
```

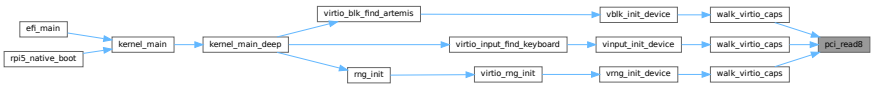
References PciDevice::bus, cfg_read8(), PciDevice::device, and PciDevice::function.

Referenced by walk_virtio_caps(), walk_virtio_caps(), and walk_virtio_caps().

Here is the call graph for this function:



Here is the caller graph for this function:



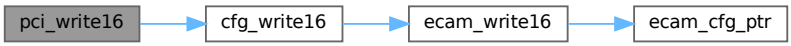
7.168.1.25 pci_write16()

```
void pci_write16 (
            const PciDevice * d,
            uint16_t off,
            uint16_t v )
```

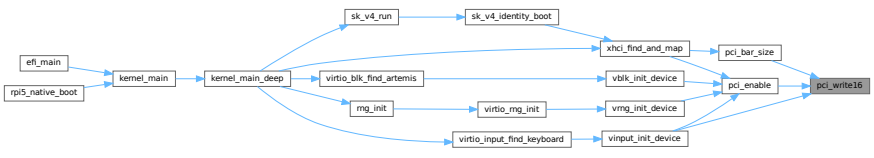
References PciDevice::bus, cfg_write16(), PciDevice::device, and PciDevice::function.

Referenced by pci_bar_size(), pci_enable(), and vinput_init_device().

Here is the call graph for this function:



Here is the caller graph for this function:



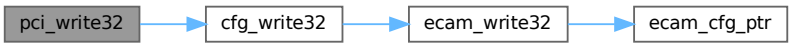
7.168.1.26 pci_write32()

```
void pci_write32 (
            const PciDevice * d,
            uint16_t off,
            uint32_t v )
```

References PciDevice::bus, cfg_write32(), PciDevice::device, and PciDevice::function.

Referenced by pci_bar_size().

Here is the call graph for this function:



Here is the caller graph for this function:

7.168.1.27 pci_write8()

```
void pci_write8 (
            const PciDevice * d,
            uint16_t off,
            uint8_t v )
```

References PciDevice::bus, cfg_write8(), PciDevice::device, and PciDevice::function.

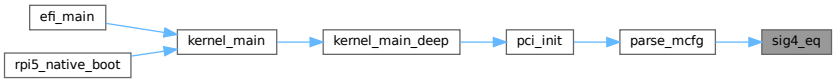
Here is the call graph for this function:

7.168.1.28 sig4_eq()

```
static int sig4_eq (
            const uint8_t * p,
            const char * s )  [static]
```

Referenced by parse_mcfg().

Here is the caller graph for this function:

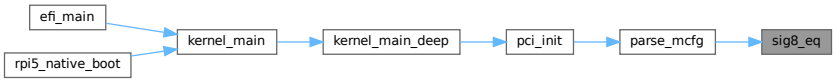


7.168.1.29 sig8_eq()

```
static int sig8_eq (
            const uint8_t * p,
            const char * s )  [static]
```

Referenced by parse_mcfg().

Here is the caller graph for this function:



7.168.2 Variable Documentation

7.168.2.1 g_bus_first

```
uint8_t g_bus_first = 0  [static]
```

Referenced by parse_mcfg(), and pci_find_class().

7.168.2.2 g_bus_last

```
uint8_t g_bus_last = 255  [static]
```

Referenced by parse_mcfg(), and pci_find_class().

7.168.2.3 g_ecam_base

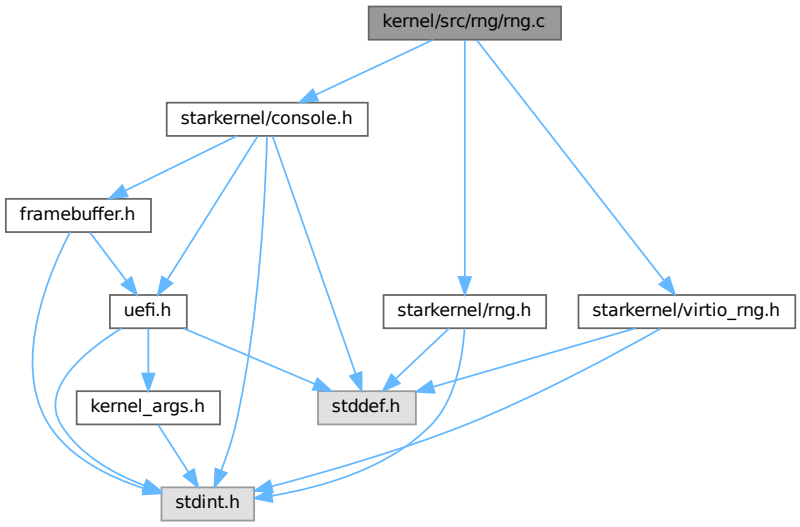
```
uint64_t g_ecam_base = 0  [static]
```

Referenced by ecam_cfg_ptr(), parse_mcfg(), pci_find_class(), and pci_init().

7.169 kernel/src/rng/rng.c File Reference

```
#include "starkernel/rng.h"
#include "starkernel/virtio_rng.h"
#include "starkernel/console.h"
```

Include dependency graph for rng.c:



Enumerations

- enum { RNG_BACKEND_NONE = 0 , RNG_BACKEND_VIRTIO }

Functions

- int rng_init (void)
- int rng_ready (void)
- int rng_get_bytes (uint8_t *buf, size_t n)

Variables

- static int g_rng_backend = RNG_BACKEND_NONE

7.169.1 Enumeration Type Documentation

7.169.1.1 anonymous enum

```
anonymous enum
```

Enumerator

RNG_BACKEND_NONE	
RNG_BACKEND_VIRTIO	

7.169.2 Function Documentation

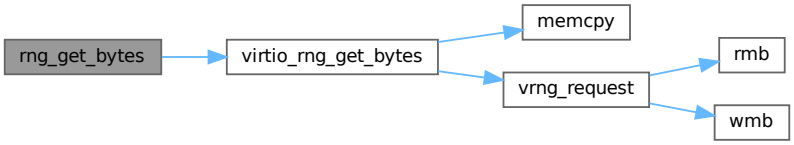
7.169.2.1 rng_get_bytes()

```
int rng_get_bytes (
            uint8_t * buf,
            size_t n )
```

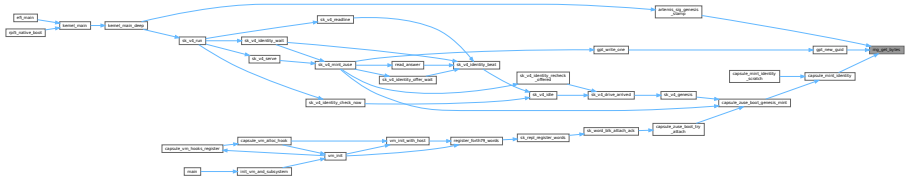
References g_rng_backend, RNG_BACKEND_VIRTIO, RNG_ERR_NO_BACKEND, and virtio_rng_get_bytes().

Referenced by artemis_sig_genesis_stamp(), capsule_mint_identity(), and gpt_new_guid().

Here is the call graph for this function:



Here is the caller graph for this function:



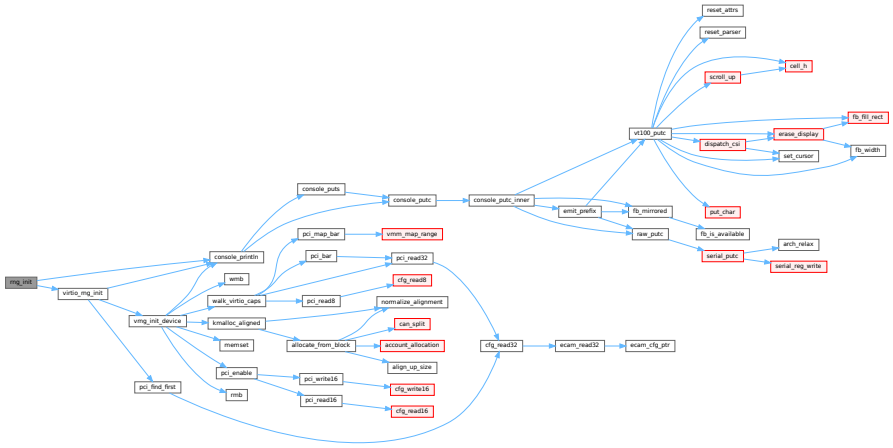
7.169.2.2 rng_init()

```
int rng_init (
            void  )
```

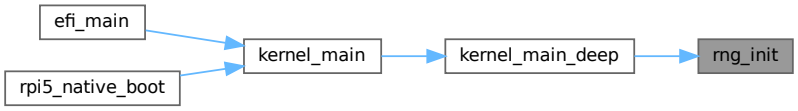
References console_println(), g_rng_backend, RNG_BACKEND_NONE, RNG_BACKEND_VIRTIO, and virtio_rng_init().

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.169.2.3 rng_ready()

```
int rng_ready (
            void  )
```

References g_rng_backend, and RNG_BACKEND_NONE.

Referenced by capsule_mint_identity().

Here is the caller graph for this function:

7.169.3 Variable Documentation

7.169.3.1 g_rng_backend

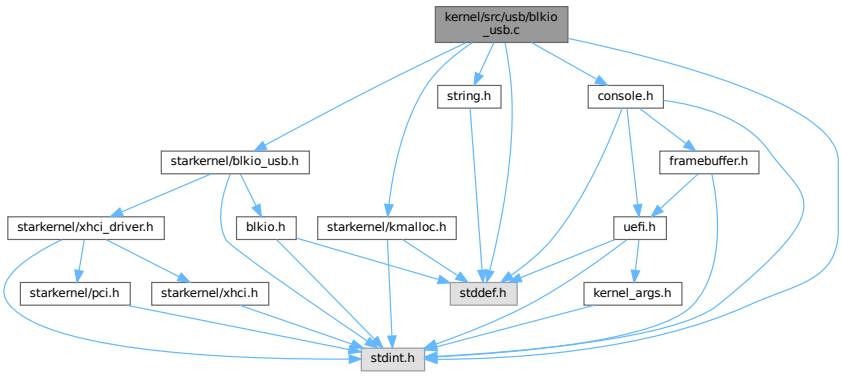
```
int g_rng_backend = RNG_BACKEND_NONE  [static]
```

Referenced by rng_get_bytes(), rng_init(), and rng_ready().

7.170 kernel/src/usb/blkio_usb.c File Reference

```
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include "starkernel/blkio_usb.h"
#include "starkernel/kmalloc.h"
#include "console.h"
```

Include dependency graph for blkio_usb.c:



Data Structures

- struct BlkioUsbState

Functions

- static BlkioUsbState * usb_blk_state_for (xhci_dev_t *xdev, uint32_t slot_id)
- static int usb_blk_open (blkio_dev_t *dev, const blkio_params_t *p)
- static int usb_blk_close (blkio_dev_t *dev)
- static int usb_blk_read (blkio_dev_t *dev, uint32_t fblock, void *dst)
- static int usb_blk_write (blkio_dev_t *dev, uint32_t fblock, const void *src)
- static int usb_blk_flush (blkio_dev_t *dev)
- static int usb_blk_info (blkio_dev_t *dev, blkio_info_t *out)
- int blkio_usb_open_msc (blkio_dev_t *dev_out, xhci_dev_t *xdev, uint32_t slot_id)

Variables

- struct {
  xhci_dev_t * xdev
  BlkioUsbState * slots
  uint32_t count
  } g_usb_blk [8u]

- static const blkio_vtable_t g_usb_blk_vtable

7.170.1 Function Documentation

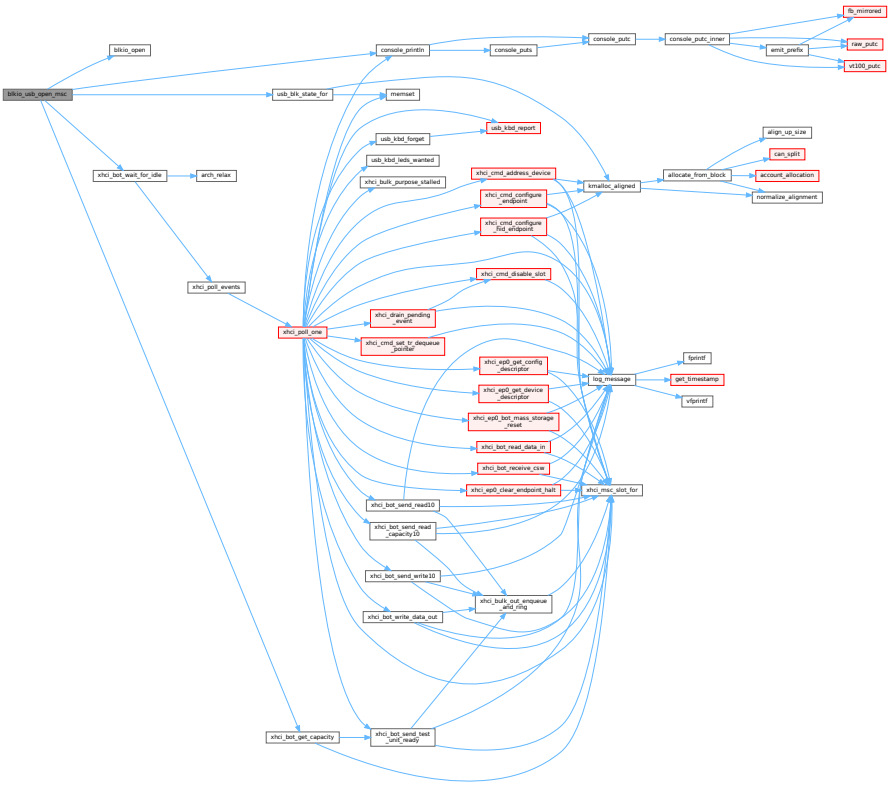
7.170.1.1 blkio_usb_open_msc()

```
int blkio_usb_open_msc (
            blkio_dev_t * dev_out,
            xhci_dev_t * xdev,
            uint32_t slot_id )
```

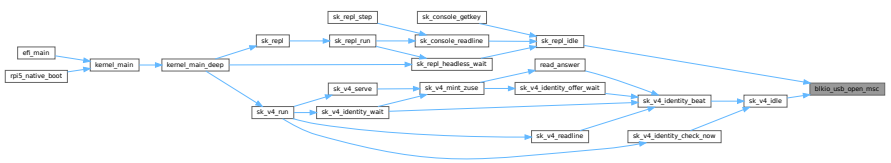
References BLKIO_FORTH_BLOCK_SIZE, blkio_open(), block_size, xhci_dev_t::bot_cap_block_size, xhci_dev_t::bot_cap_last_lba, console_println(), blkio_params_t::forth_block_size, g_usb_blk_vtable, blkio_params_t::opaque, BlkioUsbState::scsi_block_size, BlkioUsbState::scsi_blocks_per_fblock, BlkioUsbState::slot_id, blkio_params_t::total_blocks, BlkioUsbState::total_forth_blocks, usb_blk_state_for(), BlkioUsbState::xdev, xdev, xhci_bot_get_capacity(), and xhci_bot_wait_for_idle().

Referenced by sk_repl_idle(), and sk_v4_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.170.1.2 usb_blk_close()

```
static int usb_blk_close (
            blkio_dev_t * dev )  [static]
```

References BLKIO_OK.

7.170.1.3 usb_blk_flush()

```
static int usb_blk_flush (
            blkio_dev_t * dev )  [static]
```

References BLKIO_OK.

7.170.1.4 usb_blk_info()

```
static int usb_blk_info (
            blkio_dev_t * dev,
            blkio_info_t * out )  [static]
```

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_info_t::forth_block_size, blkio_info_t::phys_sector_size, blkio_info_t::phys_size_bytes, blkio_info_t::read_only, BlkioUsbState::scsi_block_size, blkio_dev::state, blkio_info_t::total_blocks, and BlkioUsbState::total_forth_blocks.

7.170.1.5 usb_blk_open()

```
static int usb_blk_open (
            blkio_dev_t * dev,
            const blkio_params_t * p )  [static]
```

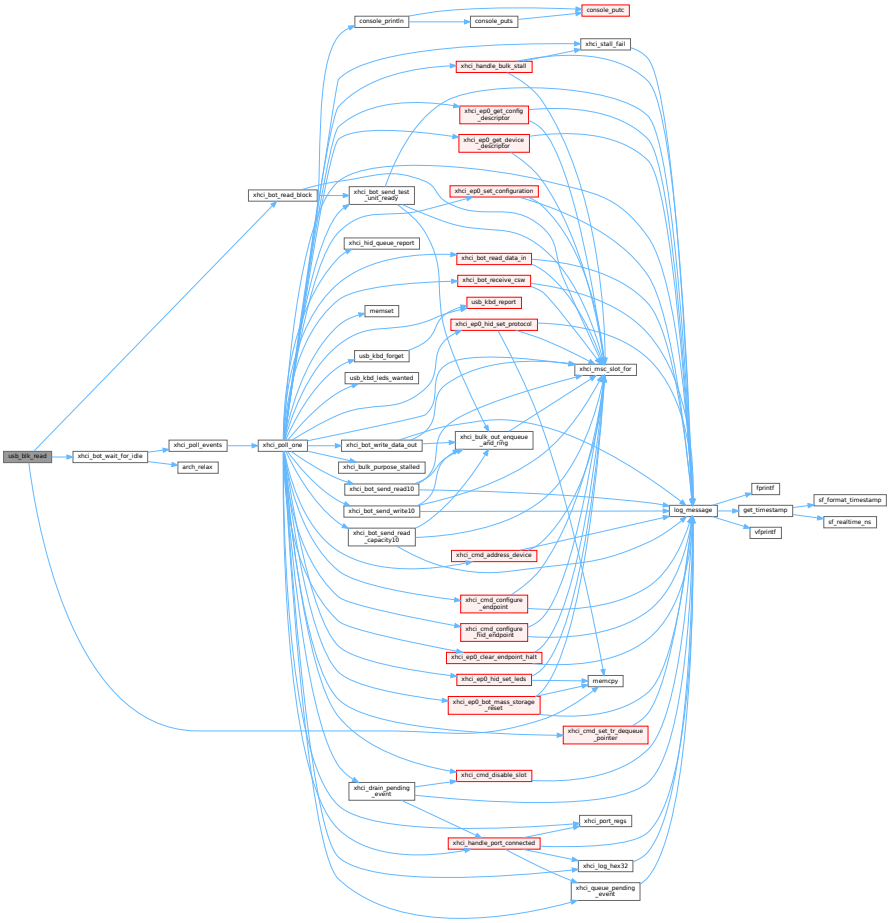
References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_dev::forth_block_size, blkio_params_t::opaque, blkio_dev::state, blkio_dev::total_blocks, and BlkioUsbState::total_forth_blocks.

7.170.1.6 usb_blk_read()

```
static int usb_blk_read (
            blkio_dev_t * dev,
            uint32_t fblock,
            void * dst )  [static]
```

References BLKIO_EINVAL, BLKIO_EIO, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, xhci_dev_t::bot_data_buf, memcpy(), BlkioUsbState::scsi_block_size, BlkioUsbState::scsi_blocks_per_fblock, BlkioUsbState::slot_id, blkio_dev::state, BlkioUsbState::total_forth_blocks, BlkioUsbState::xdev, xhci_bot_read_block(), and xhci_bot_wait_for_idle().

Here is the call graph for this function:



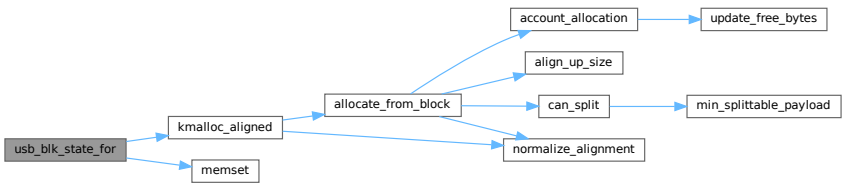
7.170.1.7 usb_blk_state_for()

```
static BlkioUsbState * usb_blk_state_for (
            xhci_dev_t * xdev,
            uint32_t slot_id )  [static]
```

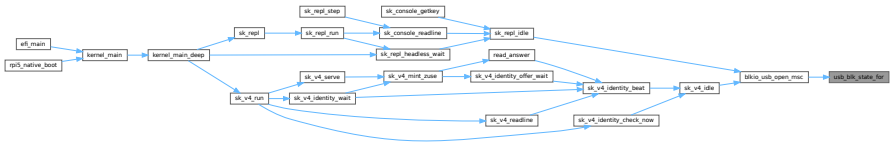
References bytes, g_usb_blk, kmalloc_aligned(), xhci_dev_t::max_slots, memset(), slots, xdev, and XHCI_MAX_CONTROLLERS.

Referenced by blkio_usb_open_msc().

Here is the call graph for this function:



Here is the caller graph for this function:

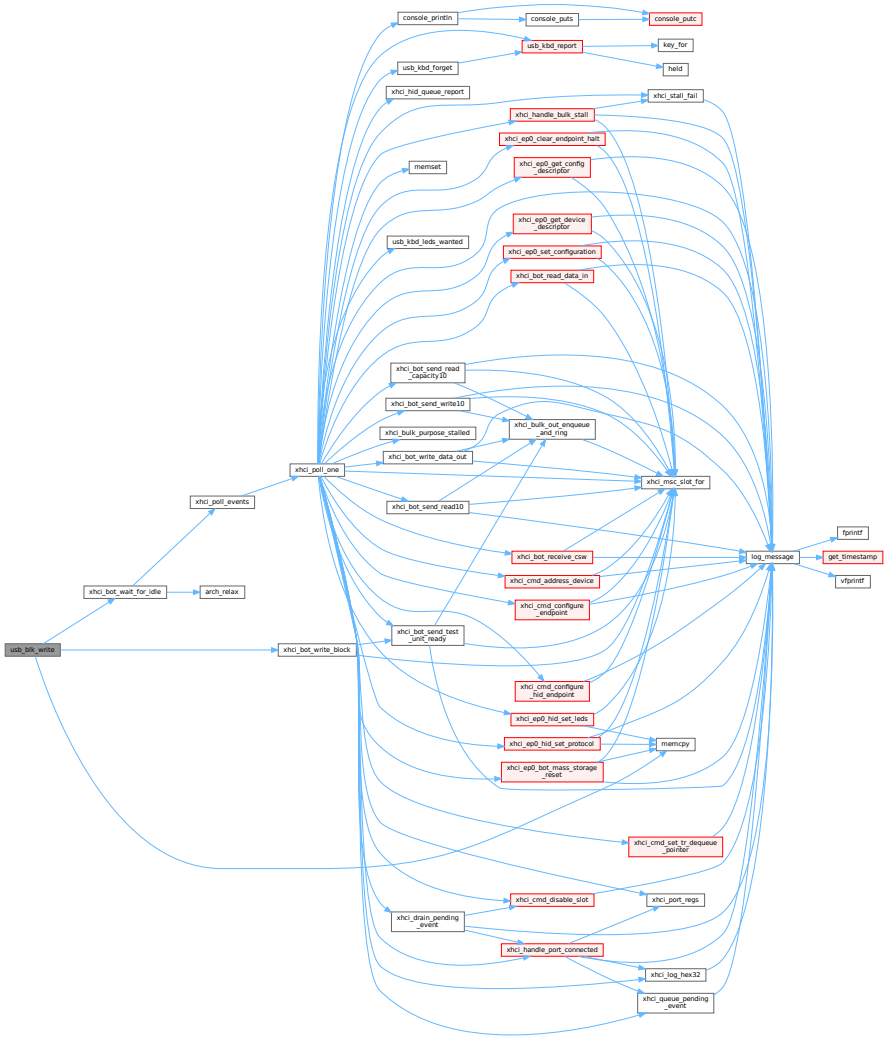


7.170.1.8 usb_blk_write()

```
static int usb_blk_write (
            blkio_dev_t * dev,
            uint32_t fblock,
            const void * src )  [static]
```

References BLKIO_EINVAL, BLKIO_EIO, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, xhci_dev_t::bot_data_buf, memcpy(), BlkioUsbState::scsi_block_size, BlkioUsbState::scsi_blocks_per_fblock, BlkioUsbState::slot_id, blkio_dev::state, BlkioUsbState::total_forth_blocks, BlkioUsbState::xdev, xhci_bot_wait_for_idle(), and xhci_bot_write_block().

Here is the call graph for this function:



7.170.2 Variable Documentation

7.170.2.1 count

```
uint32_t count
```

Referenced by blk_firsttouch_claim(), cd_classify_ids(), collect_ram_regions(), collect_reserved_ranges(), dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), flatten_quad(), forth_ALL_HEATS(), forth_PIPELINING_RESET_ALL(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), gpt_find(), gpt_grow_last(), gpt_read(), hotwords_bucket_reorder(), io_word_spaces(), io_word_type(), reloc_load_from_disk(), rolling_window_count_transition(), sf_calloc(), sk_dict_canonical_hash(), sk_dict_word_count(), string_word_count(), system_word_vlist(), system_word_words(), transition_metrics_get_probability_q48(), and update_stats().

7.170.2.2 [struct]

```
struct { ...  } g_usb_blk[ 8u ]  [static]
```

Referenced by usb_blk_state_for().

7.170.2.3 g_usb_blk_vtable

```
const blkio_vtable_t g_usb_blk_vtable  [static]
```

Initial value:

```
= {
    .open  = usb_blk_open,
    .close = usb_blk_close,
    .read  = usb_blk_read,
    .write = usb_blk_write,
    .flush = usb_blk_flush,
    .info  = usb_blk_info
}
```

Referenced by blkio_usb_open_msc().

7.170.2.4 slots

```
BlkioUsbState* slots
```

Referenced by sk_word_blk_attach_ack(), ttf_raster_cache_init(), usb_blk_state_for(), and vt100_enable_ttf().

7.170.2.5 xdev

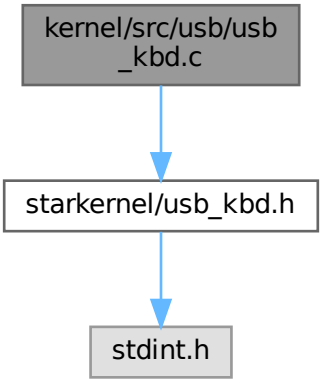
```
xhci_dev_t* xdev
```

Referenced by blkio_usb_open_msc(), sk_repl_idle(), sk_v4_idle(), sk_word_blk_attach_ack(), and usb_blk_state_for().

7.171 kernel/src/usb/usb_kbd.c File Reference

```
#include "starkernel/usb_kbd.h"
```

Include dependency graph for usb_kbd.c:



Macros

- #define KBD_EVENTS 64u

Functions

- static void push (uint16_t keycode, int pressed)
- static uint16_t key_for (uint8_t usage)
- static int held (const uint8_t *keys, uint32_t n, uint8_t usage)
- void usb_kbd_report (uint8_t prev[8], const uint8_t *report, uint32_t len)

  usb_kbd.h - a USB boot-protocol keyboard's keys, as key events (2026-10-14, boards/BRINGUP.md A3; usb_kbd.c).
- void usb_kbd_forget (uint8_t prev[8])
- void usb_kbd_set_leds (uint8_t leds)
- uint8_t usb_kbd_leds_wanted (void)
- int usb_kbd_pop_event (uint16_t *keycode, int *pressed)

Variables

- static const uint8_t usage_to_key [0x66]

  usb_kbd.c - a USB boot-protocol keyboard's reports, as key events (2026-10-14, boards/BRINGUP.md A3).
- static const uint8_t modifier_key [8] = { 29, 42, 56, 125, 97, 54, 100, 126 }
- struct {
  uint16_t keycode
  uint8_t pressed
  } g_events [64u]

- static uint32_t g_head
- static uint32_t g_count
- static uint8_t g_leds_want = 0x01u

7.171.1 Macro Definition Documentation

7.171.1.1 KBD_EVENTS

```
#define KBD_EVENTS 64u
```

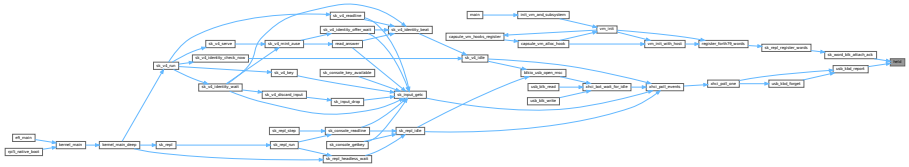
7.171.2 Function Documentation

7.171.2.1 held()

```
static int held (
            const uint8_t * keys,
            uint32_t n,
            uint8_t usage )  [static]
```

Referenced by sk_word_blk_attach_ack(), and usb_kbd_report().

Here is the caller graph for this function:



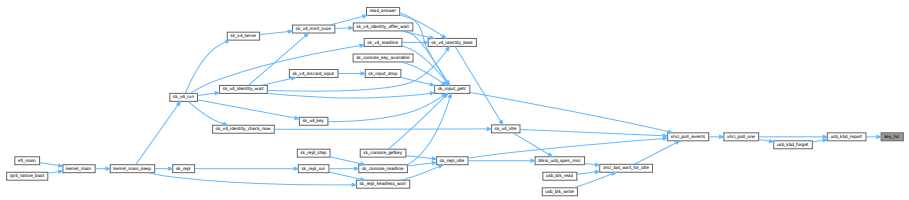
7.171.2.2 key_for()

```
static uint16_t key_for (
            uint8_t usage )  [static]
```

References usage_to_key.

Referenced by usb_kbd_report().

Here is the caller graph for this function:



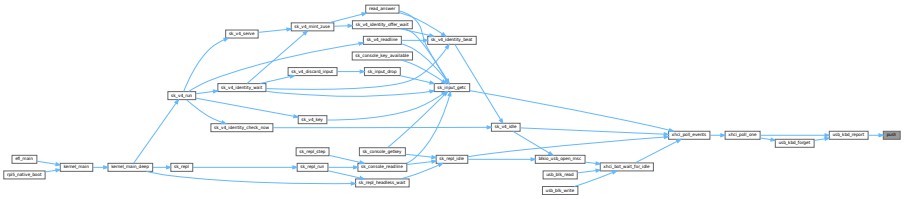
7.171.2.3 push()

```
static void push (
            uint16_t keycode,
            int pressed )  [static]
```

References g_count, g_events, g_head, KBD_EVENTS, keycode, and pressed.

Referenced by usb_kbd_report().

Here is the caller graph for this function:



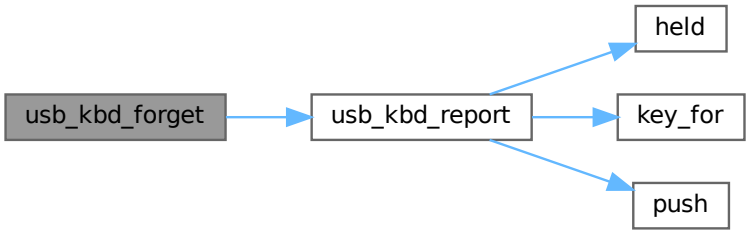
7.171.2.4 usb_kbd_forget()

```
void usb_kbd_forget (
            uint8_t prev[8] )
```

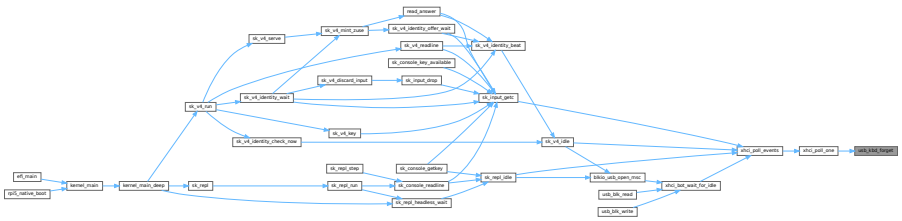
References usb_kbd_report().

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



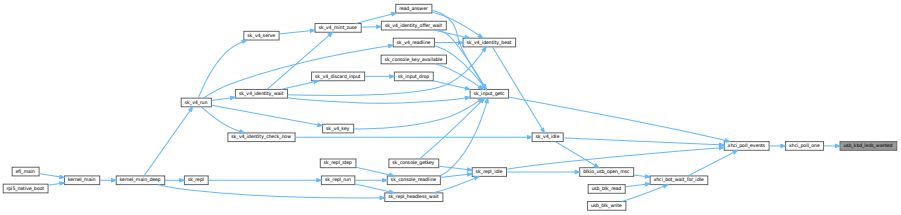
7.171.2.5 usb_kbd_leds_wanted()

```
uint8_t usb_kbd_leds_wanted (
            void  )
```

References g_leds_want.

Referenced by xhci_poll_one().

Here is the caller graph for this function:



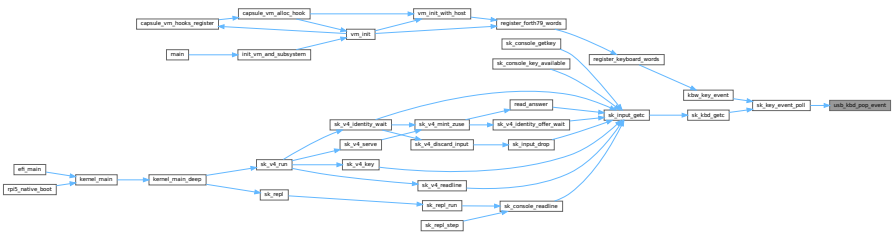
7.171.2.6 usb_kbd_pop_event()

```
int usb_kbd_pop_event (
            uint16_t * keycode,
            int * pressed )
```

References g_count, g_events, g_head, KBD_EVENTS, keycode, and pressed.

Referenced by sk_key_event_poll().

Here is the caller graph for this function:



7.171.2.7 usb_kbd_report()

```
void usb_kbd_report (
            uint8_t prev[8],
            const uint8_t * report,
            uint32_t len )
```

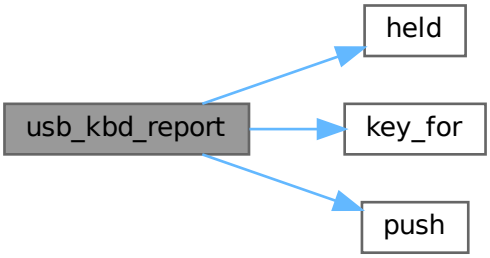
usb_kbd.h - a USB boot-protocol keyboard's keys, as key events (2026-10-14, boards/BRINGUP.md A3; usb_kbd.c).

The xHCI driver hands each 8-byte boot report here; the keys pressed and released since the report before come out as Linux input keycodes, the namespace every keyboard in this kernel already speaks (sk_key_event_poll(), keyboard_words.c).

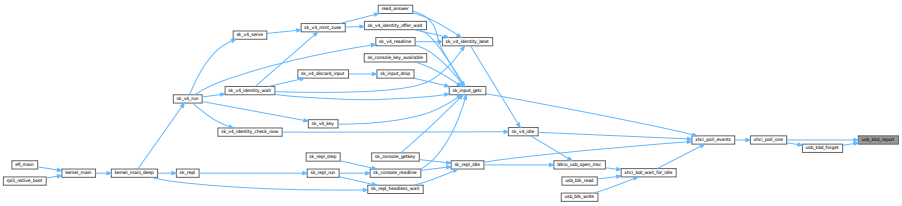
References held(), key_for(), modifier_key, and push().

Referenced by usb_kbd_forget(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



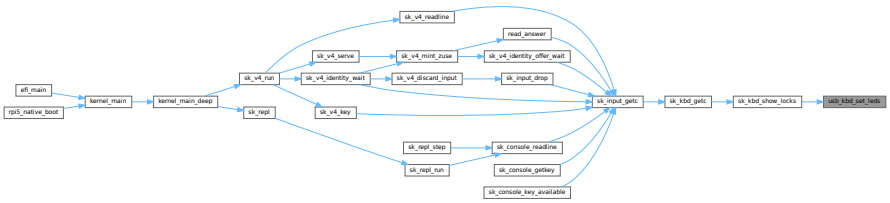
7.171.2.8 usb_kbd_set_leds()

```
void usb_kbd_set_leds (
            uint8_t leds )
```

References g_leds_want.

Referenced by sk_kbd_show_locks().

Here is the caller graph for this function:



7.171.3 Variable Documentation

7.171.3.1 g_count

```
uint32_t g_count  [static]
```

Referenced by push(), and usb_kbd_pop_event().

7.171.3.2 [struct]

```
struct { ...  } g_events[ 64u ]  [static]
```

Referenced by push(), and usb_kbd_pop_event().

7.171.3.3 g_head

```
uint32_t g_head  [static]
```

Referenced by push(), and usb_kbd_pop_event().

7.171.3.4 g_leds_want

```
uint8_t g_leds_want = 0x01u  [static]
```

Referenced by usb_kbd_leds_wanted(), and usb_kbd_set_leds().

7.171.3.5 keycode

```
uint16_t keycode
```

Referenced by kbw_key_event(), push(), sk_kbd_getc(), sk_kbd_keypad(), sk_key_event_poll(), and usb_kbd_pop_event().

7.171.3.6 modifier_key

```
const uint8_t modifier_key[8] = { 29, 42, 56, 125, 97, 54, 100, 126 }  [static]
```

Referenced by usb_kbd_report().

7.171.3.7 pressed

```
uint8_t pressed
```

Referenced by kbw_key_event(), push(), sk_kbd_getc(), sk_key_event_poll(), and usb_kbd_pop_event().

7.171.3.8 usage_to_key

```
const uint8_t usage_to_key[0x66]  [static]
```

Initial value:

```
= {
    [0x04] = 30, [0x05] = 48, [0x06] = 46, [0x07] = 32, [0x08] = 18,
    [0x09] = 33, [0x0A] = 34, [0x0B] = 35, [0x0C] = 23, [0x0D] = 36,
    [0x0E] = 37, [0x0F] = 38, [0x10] = 50, [0x11] = 49, [0x12] = 24,
    [0x13] = 25, [0x14] = 16, [0x15] = 19, [0x16] = 31, [0x17] = 20,
    [0x18] = 22, [0x19] = 47, [0x1A] = 17, [0x1B] = 45, [0x1C] = 21,
    [0x1D] = 44,
    [0x1E] = 2,  [0x1F] = 3,  [0x20] = 4,  [0x21] = 5,  [0x22] = 6,
    [0x23] = 7,  [0x24] = 8,  [0x25] = 9,  [0x26] = 10, [0x27] = 11,
    [0x28] = 28, [0x29] = 1,  [0x2A] = 14, [0x2B] = 15, [0x2C] = 57,
    [0x2D] = 12, [0x2E] = 13, [0x2F] = 26, [0x30] = 27, [0x31] = 43,
    [0x32] = 43, [0x33] = 39, [0x34] = 40, [0x35] = 41, [0x36] = 51,
    [0x37] = 52, [0x38] = 53, [0x39] = 58,
    [0x3A] = 59, [0x3B] = 60, [0x3C] = 61, [0x3D] = 62, [0x3E] = 63,
    [0x3F] = 64, [0x40] = 65, [0x41] = 66, [0x42] = 67, [0x43] = 68,
    [0x44] = 87, [0x45] = 88,
    [0x46] = 99,  [0x47] = 70,  [0x48] = 119,
    [0x49] = 110, [0x4A] = 102, [0x4B] = 104,
    [0x4C] = 111, [0x4D] = 107, [0x4E] = 109,
    [0x4F] = 106, [0x50] = 105, [0x51] = 108, [0x52] = 103,
    [0x53] = 69,
    [0x54] = 98,  [0x55] = 55,  [0x56] = 74,  [0x57] = 78,  [0x58] = 96,
    [0x59] = 79,  [0x5A] = 80,  [0x5B] = 81,  [0x5C] = 75,  [0x5D] = 76,
    [0x5E] = 77,  [0x5F] = 71,  [0x60] = 72,  [0x61] = 73,  [0x62] = 82,
    [0x63] = 83,  [0x64] = 86,  [0x65] = 127,
}
```

usb_kbd.c - a USB boot-protocol keyboard's reports, as key events (2026-10-14, boards/BRINGUP.md A3).

See usb_kbd.h.

A boot report says which keys are held now, not what changed: what was held in the report before and is not now has been released, and the other way round has been pressed. The eight modifier bits are keys too (left/right Ctrl, Shift, Alt, GUI). A report whose keys are all 0x01 is the keyboard's "too many keys" (ErrorRollOver) and changes nothing.

Usage IDs (HID Usage Tables 1.12, page 7) become Linux keycodes (input-event-codes.h) – the same numbers the PS/2 and virtio keyboards already give sk_key_event_poll(). Only the keys a console uses are mapped; others are dropped. The keyboard does not repeat a held key: held is held, and nothing here repeats it either.

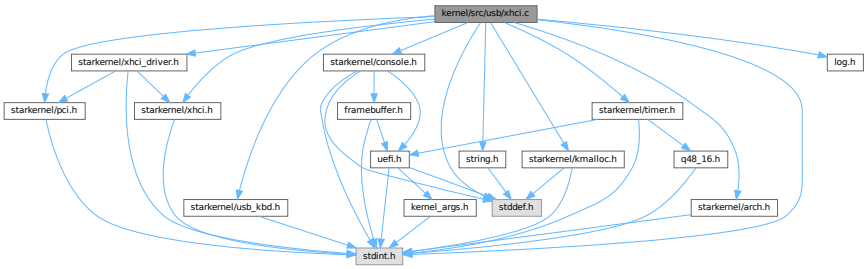
Referenced by key_for().

7.172 kernel/src/usb/xhci.c File Reference

```
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include "starkernel/pci.h"
#include "starkernel/xhci.h"
#include "starkernel/xhci_driver.h"
#include "starkernel/usb_kbd.h"
#include "starkernel/console.h"
#include "starkernel/kmalloc.h"
#include "starkernel/timer.h"
#include "starkernel/arch.h"
```

```
#include "log.h"
```

Include dependency graph for xhci.c:



Macros

- #define XHCI_BAR0_MIN_MAP_SIZE 0x10000ull
- #define XHCI_PCI_CLASS 0x0Cu
- #define XHCI_PCI_SUBCLASS 0x03u
- #define XHCI_PCI_PROG_IF 0x30u
- #define XHCI_XCAP_LEGACY 1u
- #define XHCI_LEGSUP_BIOS_OWNED (1u << 16)
- #define XHCI_LEGSUP_OS_OWNED (1u << 24)
- #define USB_HID_REQ_SET_REPORT 0x09u
- #define USB_HID_REPORT_OUTPUT 0x0200u /* wValue: report type Output (2), report ID 0 */

Functions

- static int xhci_wait_bit (volatile uint32_t *reg, uint32_t mask, int want_set, uint64_t max_ticks)
- static void xhci_log_hex32 (const char *prefix, uint32_t v)
- static void xhci_take_from_bios (xhci_dev_t *dev)
- int xhci_find_and_map (xhci_dev_t *dev, unsigned index)
- int xhci_cmd_enable_slot (xhci_dev_t *dev)
- int xhci_cmd_disable_slot (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_cmd_address_device (xhci_dev_t *dev, uint32_t slot_id, uint32_t port_id, uint32_t speed)
- int xhci_cmd_configure_endpoint (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_cmd_reset_endpoint (xhci_dev_t *dev, uint32_t slot_id, uint32_t ep_id)
- int xhci_cmd_set_tr_dequeue_pointer (xhci_dev_t *dev, uint32_t slot_id, uint32_t ep_id, uint64_t new_↩ dequeue, uint32_t dcs)
- int xhci_bot_send_read10 (xhci_dev_t *dev, uint32_t slot_id, uint32_t lba, uint16_t num_blocks, uint32_↩ t block_size)
- int xhci_bot_send_write10 (xhci_dev_t *dev, uint32_t slot_id, uint32_t lba, uint16_t num_blocks, uint32_↩ t block_size)
- int xhci_bot_send_test_unit_ready (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_send_read_capacity10 (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_get_capacity (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_read_data_in (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_write_data_out (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_bot_receive_csw (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_ep0_get_device_descriptor (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_ep0_get_config_descriptor (xhci_dev_t *dev, uint32_t slot_id, uint16_t length)
- int xhci_ep0_set_configuration (xhci_dev_t *dev, uint32_t slot_id, uint8_t config_value)
- static void xhci_scan_ports_for_already_connected (xhci_dev_t *dev)

- int xhci_bringup (xhci_dev_t *dev)
- uint32_t xhci_dev_count (void)
- xhci_dev_t * xhci_dev_at (uint32_t i)
- xhci_msc_slot_t * xhci_msc_slot_for (xhci_dev_t *dev, uint32_t slot_id)
- static void xhci_submit_command (xhci_dev_t *dev, uint64_t parameter, uint32_t status, uint32_t trb_type, uint32_t extra_control_bits)
- static uint32_t xhci_default_ep0_max_packet (uint32_t speed)
- static size_t xhci_ctx_bytes (const xhci_dev_t *dev)
- static void xhci_init_bulk_ring (xhci_trb_t *ring)
- int xhci_cmd_configure_hid_endpoint (xhci_dev_t *dev, uint32_t slot_id)
- static void xhci_bulk_out_enqueue_and_ring (xhci_dev_t *dev, uint32_t slot_id, uint64_t parameter, uint32_t status, uint32_t control_bits)
- int xhci_bot_read_block (xhci_dev_t *dev, uint32_t slot_id, uint32_t lba, uint16_t num_blocks, uint32_t block_size)
- int xhci_bot_write_block (xhci_dev_t *dev, uint32_t slot_id, uint32_t lba, uint16_t num_blocks, uint32_t block_size)
- int xhci_bot_wait_for_idle (xhci_dev_t *dev, uint32_t max_iters)
- static void xhci_bulk_in_enqueue_and_ring (xhci_dev_t *dev, uint32_t slot_id, uint64_t parameter, uint32_t status, uint32_t control_bits)
- static void xhci_ep0_enqueue_trb (xhci_dev_t *dev, uint32_t slot_id, uint64_t parameter, uint32_t status, uint32_t control_bits)
- static void xhci_ep0_control_read (xhci_dev_t *dev, uint32_t slot_id, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, uint8_t *buf, uint16_t len)
- static void xhci_ep0_control_write_nodata (xhci_dev_t *dev, uint32_t slot_id, uint8_t bRequest, uint16_t wValue, uint16_t wIndex)
- int xhci_ep0_clear_endpoint_halt (xhci_dev_t *dev, uint32_t slot_id, uint8_t ep_addr)
- int xhci_ep0_bot_mass_storage_reset (xhci_dev_t *dev, uint32_t slot_id)
- int xhci_ep0_hid_set_protocol (xhci_dev_t *dev, uint32_t slot_id)
- static int xhci_ep0_hid_set_leds (xhci_dev_t *dev, uint32_t slot_id, uint8_t leds)
- static void xhci_hid_queue_report (xhci_dev_t *dev, uint32_t slot_id, xhci_msc_slot_t *ms)
- static xhci_port_regs_t * xhci_port_regs (xhci_dev_t *dev, uint32_t port_id)
- static void xhci_queue_pending_event (xhci_dev_t *dev, uint32_t port_id, int is_connect, uint32_t portsc, uint32_t slot_id)
- static void xhci_handle_port_connected (xhci_dev_t *dev, uint32_t port_id, uint32_t portsc)
- static void xhci_drain_pending_event (xhci_dev_t *dev)
- static void xhci_stall_fail (xhci_dev_t *dev)
- static int xhci_bulk_purpose_stalled (xhci_dev_t *dev, uint32_t purpose)
- static void xhci_handle_bulk_stall (xhci_dev_t *dev, uint32_t slot_id, uint32_t purpose)
- int xhci_keyboard_running (void)
- static void xhci_poll_one (xhci_dev_t *dev)
- void xhci_poll_events (void)

Variables

- static xhci_dev_t * g_xhci_devs [8u]
- static uint32_t g_xhci_dev_count
- static uint32_t g_xhci_keyboards

7.172.1 Macro Definition Documentation

7.172.1.1 USB_HID_REPORT_OUTPUT

```
#define USB_HID_REPORT_OUTPUT 0x0200u /* wValue:  report type Output (2), report ID 0 */
```

7.172.1.2 USB_HID_REQ_SET_REPORT

```
#define USB_HID_REQ_SET_REPORT 0x09u
```

7.172.1.3 XHCI_BAR0_MIN_MAP_SIZE

```
#define XHCI_BAR0_MIN_MAP_SIZE 0x10000ull
```

7.172.1.4 XHCI_LEGSUP_BIOS_OWNED

```
#define XHCI_LEGSUP_BIOS_OWNED (1u << 16)
```

7.172.1.5 XHCI_LEGSUP_OS_OWNED

```
#define XHCI_LEGSUP_OS_OWNED (1u << 24)
```

7.172.1.6 XHCI_PCI_CLASS

```
#define XHCI_PCI_CLASS 0x0Cu
```

7.172.1.7 XHCI_PCI_PROG_IF

```
#define XHCI_PCI_PROG_IF 0x30u
```

7.172.1.8 XHCI_PCI_SUBCLASS

```
#define XHCI_PCI_SUBCLASS 0x03u
```

7.172.1.9 XHCI_XCAP_LEGACY

```
#define XHCI_XCAP_LEGACY 1u
```

7.172.2 Function Documentation

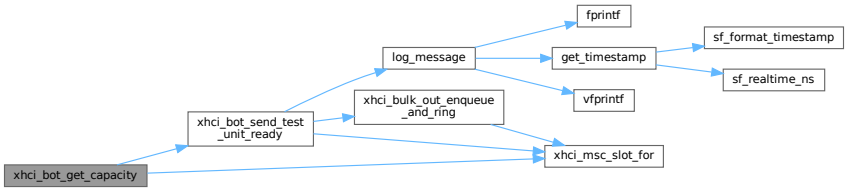
7.172.2.1 xhci_bot_get_capacity()

```
int xhci_bot_get_capacity (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

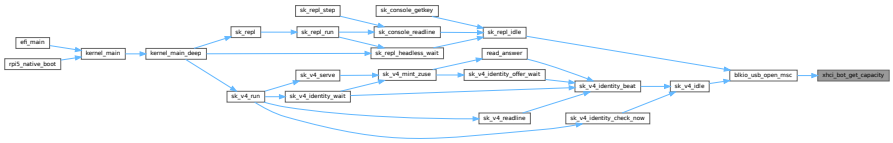
References xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, xhci_bot_send_test_unit_ready(), and xhci_msc_slot_for().

Referenced by blkio_usb_open_msc().

Here is the call graph for this function:



Here is the caller graph for this function:



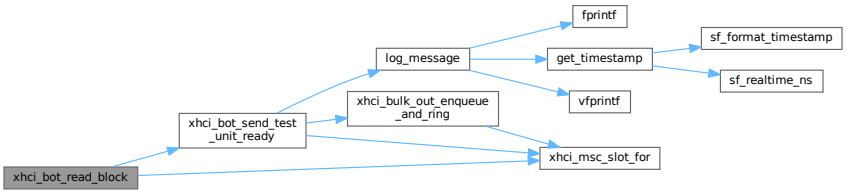
7.172.2.2 xhci_bot_read_block()

```
int xhci_bot_read_block (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t lba,
            uint16_t num_blocks,
            uint32_t block_size )
```

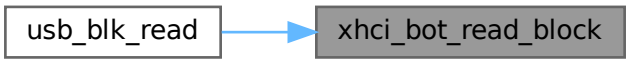
References block_size, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_read10_block_size, xhci_dev_t::bot_read10_lba, xhci_dev_t::bot_read10_num_blocks, xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, xhci_bot_send_test_unit_ready(), and xhci_msc_slot_for().

Referenced by usb_blk_read().

Here is the call graph for this function:



Here is the caller graph for this function:



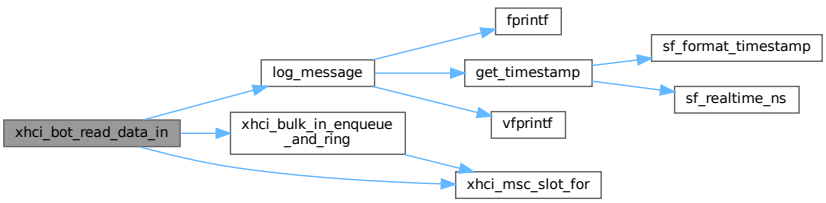
7.172.2.3 xhci_bot_read_data_in()

```
int xhci_bot_read_data_in (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

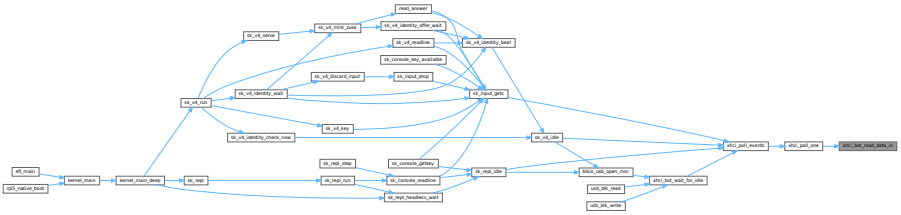
References xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_msc_slot_t::bulk_in_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, xhci_bulk_in_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_BOT_DATA_IN

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



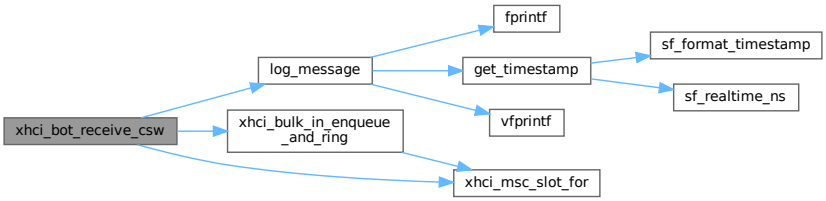
7.172.2.4 xhci_bot_receive_csw()

```
int xhci_bot_receive_csw (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

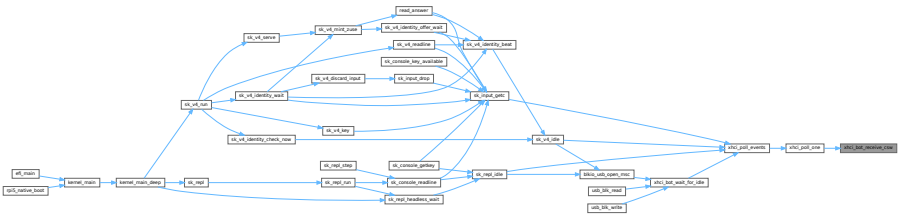
References xhci_dev_t::bot_csw, xhci_msc_slot_t::bulk_in_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_BOT_CSW_LENGTH, xhci_bulk_in_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CSW_RECEIVED.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



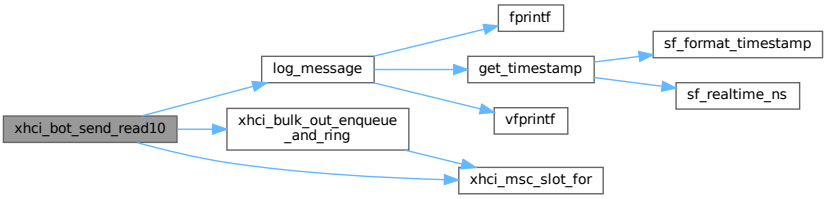
7.172.2.5 xhci_bot_send_read10()

```
int xhci_bot_send_read10 (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t lba,
            uint16_t num_blocks,
            uint32_t block_size )
```

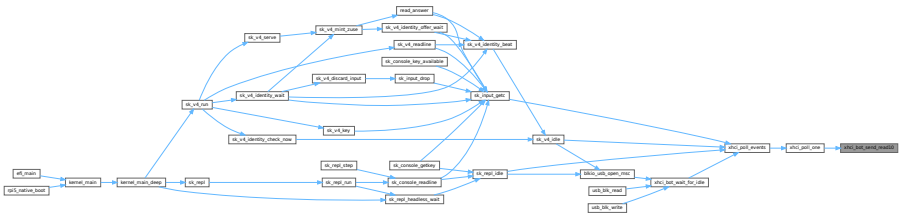
References usb_bot_cbw_t::bCBWCBLength, usb_bot_cbw_t::bCBWLUN, block_size, usb_bot_cbw_t::bmCBWFlags, xhci_dev_t::bot_cbw, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, usb_bot_cbw_t::CBWCB, usb_bot_cbw_t::dCBWDataTransferLe usb_bot_cbw_t::dCBWSignature, usb_bot_cbw_t::dCBWTag, LOG_DEBUG, log_message(), SCSI_CDB_LEN_READ10, SCSI_CMD_READ10, xhci_msc_slot_t::transfer_purpose, USB_BOT_CBW_FLAG_DATA_IN, USB_BOT_CBW_LENGTH, USB_BOT_CBW_LUN_DEFAULT, USB_BOT_CBW_SIGNATURE, xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CBW_SENT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



7.172.2.6 xhci_bot_send_read_capacity10()

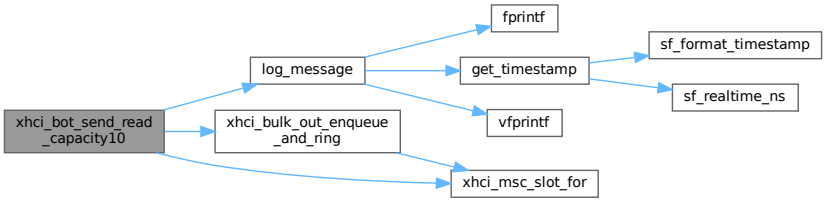
```
int xhci_bot_send_read_capacity10 (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

References usb_bot_cbw_t::bCBWCBLength, usb_bot_cbw_t::bCBWLUN, usb_bot_cbw_t::bmCBWFlags, xhci_dev_t::bot_cbw, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr,

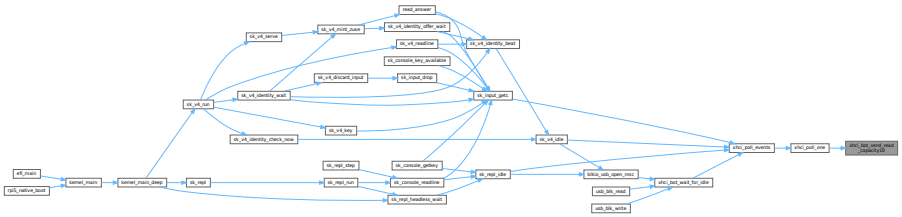
xhci_msc_slot_t::bulk_out_ring, usb_bot_cbw_t::CBWCB, usb_bot_cbw_t::dCBWDataTransferLength, usb_bot_cbw_t::dCBWSignatu
usb_bot_cbw_t::dCBWTag, LOG_DEBUG, log_message(), SCSI_CDB_LEN_READ_CAPACITY10, SCSI_CMD_READ_CAPACITY1
SCSI_READ_CAPACITY10_DATA_LEN, xhci_msc_slot_t::transfer_purpose, USB_BOT_CBW_FLAG_DATA_IN,
USB_BOT_CBW_LENGTH, USB_BOT_CBW_LUN_DEFAULT, USB_BOT_CBW_SIGNATURE, xhci_bulk_out_enqueue_and_ring(),
xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL,
and XHCI_XFER_CBW_SENT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



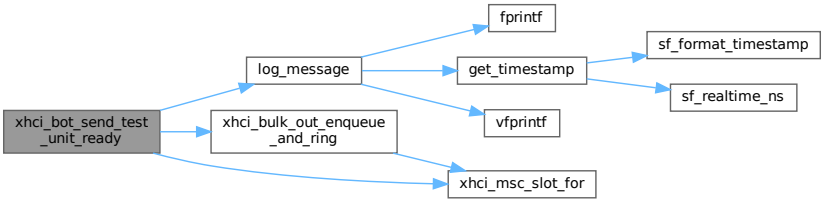
7.172.2.7 xhci_bot_send_test_unit_ready()

```
int xhci_bot_send_test_unit_ready (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

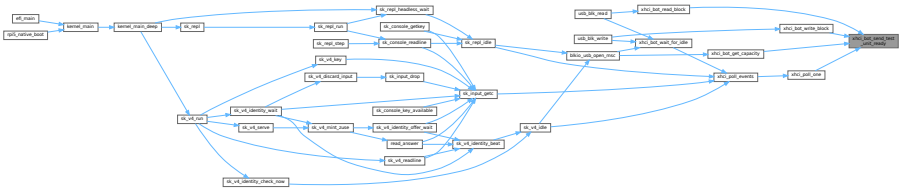
References usb_bot_cbw_t::bCBWCBLength, usb_bot_cbw_t::bCBWLUN, usb_bot_cbw_t::bmCBWFlags,
xhci_dev_t::bot_cbw, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_tag,
xhci_dev_t::bot_next_tag, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr,
xhci_msc_slot_t::bulk_out_ring, usb_bot_cbw_t::CBWCB, usb_bot_cbw_t::dCBWDataTransferLength, usb_bot_cbw_t::dCBWSignatu
usb_bot_cbw_t::dCBWTag, LOG_DEBUG, log_message(), SCSI_CDB_LEN_TEST_UNIT_READY, SCSI_CMD_TEST_UNIT_READ
xhci_msc_slot_t::transfer_purpose, USB_BOT_CBW_LENGTH, USB_BOT_CBW_LUN_DEFAULT, USB_BOT_CBW_SIGNATURE,
xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT,
XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CBW_SENT.

Referenced by xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_write_block(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



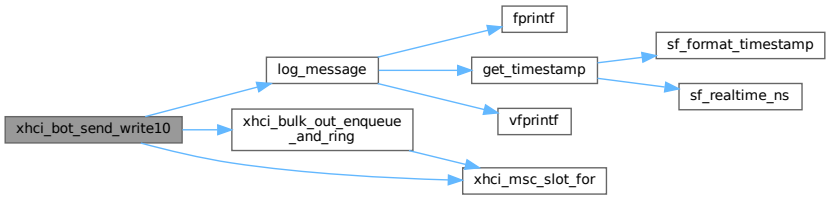
7.172.2.8 xhci_bot_send_write10()

```
int xhci_bot_send_write10 (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t lba,
            uint16_t num_blocks,
            uint32_t block_size )
```

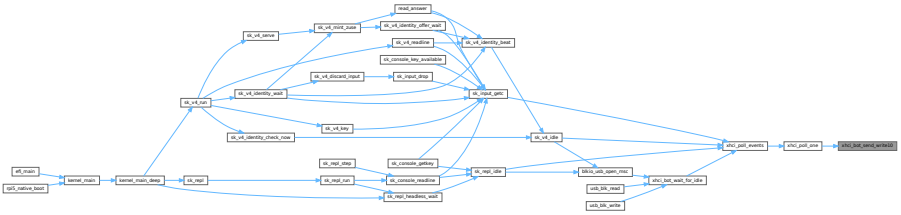
References usb_bot_cbw_t::bCBWCBLength, usb_bot_cbw_t::bCBWLUN, block_size, usb_bot_cbw_t::bmCBWFlags, xhci_dev_t::bot_cbw, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, usb_bot_cbw_t::CBWCB, usb_bot_cbw_t::dCBWDataTransferLe usb_bot_cbw_t::dCBWSignature, usb_bot_cbw_t::dCBWTag, LOG_DEBUG, log_message(), SCSI_CDB_LEN_WRITE10, SCSI_CMD_WRITE10, xhci_msc_slot_t::transfer_purpose, USB_BOT_CBW_LENGTH, USB_BOT_CBW_LUN_DEFAULT, USB_BOT_CBW_SIGNATURE, xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_CBW_SENT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



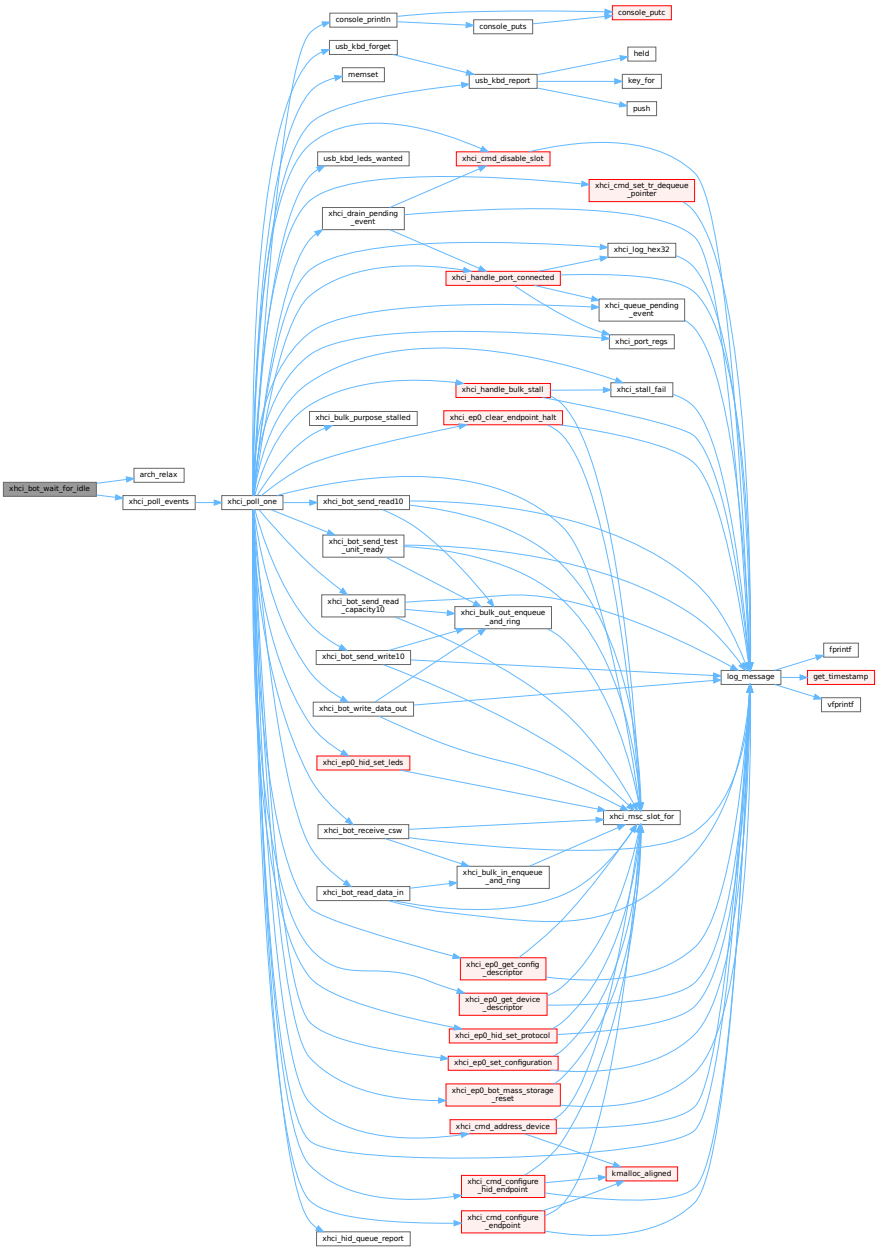
7.172.2.9 xhci_bot_wait_for_idle()

```
int xhci_bot_wait_for_idle (
            xhci_dev_t * dev,
            uint32_t max_iters )
```

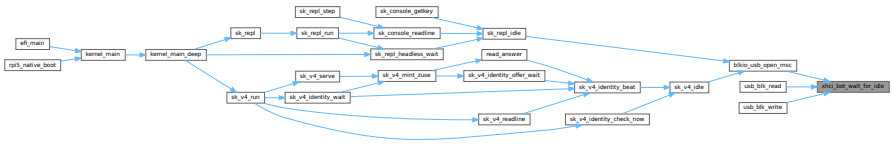
References arch_relax(), xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_last_status, and xhci_poll_events().

Referenced by blkio_usb_open_msc(), usb_blk_read(), and usb_blk_write().

Here is the call graph for this function:



Here is the caller graph for this function:



7.172.2.10 xhci_bot_write_block()

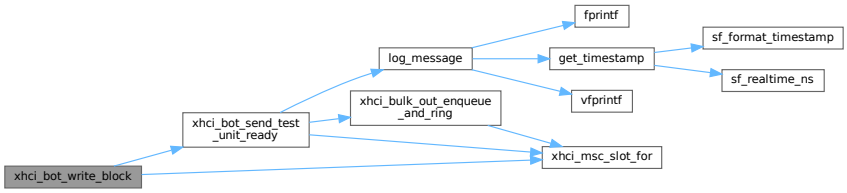
```
int xhci_bot_write_block (
```

```
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t lba,
            uint16_t num_blocks,
            uint32_t block_size )
```

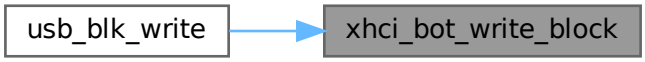
References block_size, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_dev_t::bot_write10_block_size, xhci_dev_t::bot_write10_lba, xhci_dev_t::bot_write10_num_blocks, xhci_msc_slot_t::bulk_in_ep, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, xhci_bot_send_test_unit_ready(), and xhci_msc_slot_for().

Referenced by usb_blk_write().

Here is the call graph for this function:



Here is the caller graph for this function:



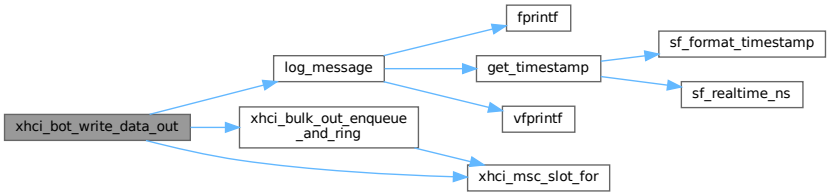
7.172.2.11 xhci_bot_write_data_out()

```
int xhci_bot_write_data_out (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

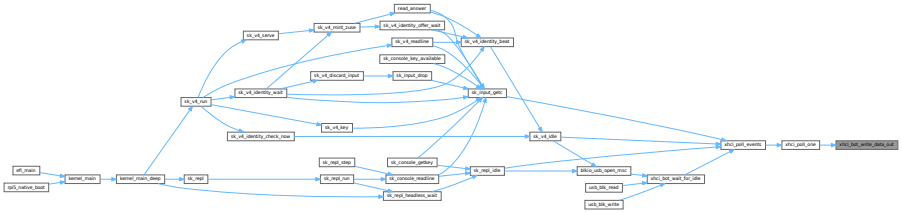
References xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_msc_slot_t::bulk_out_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, xhci_bulk_out_enqueue_and_ring(), xhci_msc_slot_for(), XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_NORMAL, and XHCI_XFER_BOT_DATA_O

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



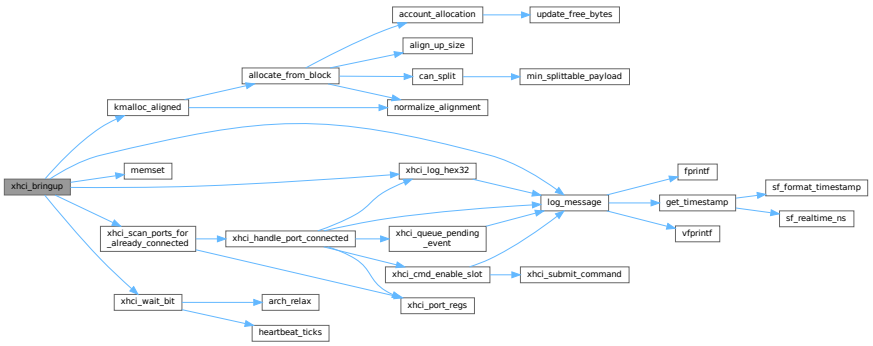
7.172.2.12 xhci_bringup()

```
int xhci_bringup (
            xhci_dev_t * dev )
```

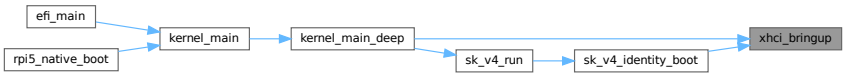
References xhci_dev_t::bot_cap_block_size, xhci_dev_t::bot_cap_last_lba, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_expected_dat xhci_dev_t::bot_last_status, xhci_dev_t::bot_last_tag, xhci_dev_t::bot_next_tag, xhci_dev_t::bot_read10_block_size, xhci_dev_t::bot_read10_lba, xhci_dev_t::bot_read10_num_blocks, xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_dev_t::cap, xhci_dev_t::cmd_ring, xhci_dev_t::cmd_ring_cycle, xhci_dev_t::cmd_ring_enq, xhci_op_regs_t::config, xhci_dev_t::connect_state, xhci_trb_t::control, xhci_op_regs_t::crcr, xhci_dev_t::dcbaa, xhci_op_regs_t::dcbaap, xhci_intr_regs_t::erdp, xhci_intr_regs_t::erstba, xhci_intr_regs_t::erstsz, xhci_dev_t::evt_ring, xhci_dev_t::evt_ring_cycle, xhci_dev_t::evt_ring_deq, xhci_dev_t::evt_ring_seg_table, g_xhci_dev_count, g_xhci_devs, xhci_cap_regs_t::hcc_params1, xhci_dev_t::intr0, kmalloc_aligned(), LOG_DEBUG, LOG_ERROR, log_message(), xhci_dev_t::max_scratchpad_bufs, xhci_dev_t::max_slots, memset(), xhci_dev_t::msc_slots, xhci_dev_t::op, xhci_op_regs_t::page_size, xhci_trb_t::parameter, xhci_dev_t::pending_connect_port_id, xhci_dev_t::pending_connect_slot_id, xhci_dev_t::pending_connect_speed, xhci_dev_t::pending_disable_slot_id, xhci_dev_t::pending_events, xhci_dev_t::port_slot_id, xhci_dev_t::runtime, xhci_dev_t::scratchpad_arr, xhci_trb_t::status, xhci_op_regs_t::usb_cmd, xhci_op_regs_t::usb_sts, xhci_dev_t::valid, XHCI_CONFIG_MAX_SLOTS_EN, XHCI_CRCR_PTR_MASK, XHCI_CRCR_RCS, XHCI_ERDP_PTR_MASK, XHCI_HCCPARAMS1_CSZ, xhci_log_hex32(), XHCI_MAX_CONTROLLERS, XHCI_MAX_TRACKED_PORTS, XHCI_RING_BYTES, XHCI_RING_TRB_COUNT, xhci_scan_ports_for_already_connected(), XHCI_TRB_CONTROL_CYCLE, XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_LINK, XHCI_USBCMD_HCRST, XHCI_USBCMD_RUN, XHCI_USBSTS_CNR, XHCI_USBSTS_HCH, and xhci_wait_bit().

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.172.2.13 xhci_bulk_in_enqueue_and_ring()

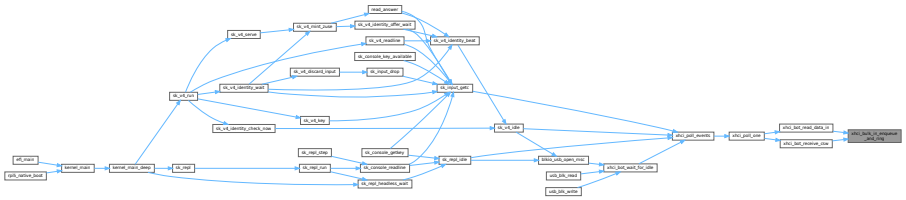
```
static void xhci_bulk_in_enqueue_and_ring (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint64_t parameter,
            uint32_t status,
            uint32_t control_bits )  [static]
```

References xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_in_ring_cycle, xhci_msc_slot_t::bulk_in_ring_enq, xhci_trb_t::control, xhci_dev_t::doorbell, xhci_trb_t::parameter, xhci_trb_t::status, XHCI_DB_TARGET, XHCI_EP_ADDR_TO_DCI, xhci_msc_slot_for(), XHCI_RING_TRB_COUNT, XHCI_TRB_CONTROL_CYCLE, XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, and XHCI_TRB_TYPE_LINK.

Referenced by xhci_bot_read_data_in(), and xhci_bot_receive_csw().

Here is the call graph for this function:

Here is the caller graph for this function:



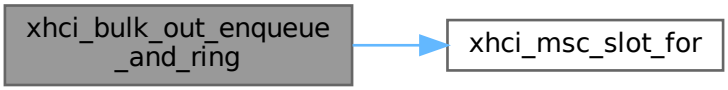
7.172.2.14 xhci_bulk_out_enqueue_and_ring()

```
static void xhci_bulk_out_enqueue_and_ring (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint64_t parameter,
            uint32_t status,
            uint32_t control_bits )  [static]
```

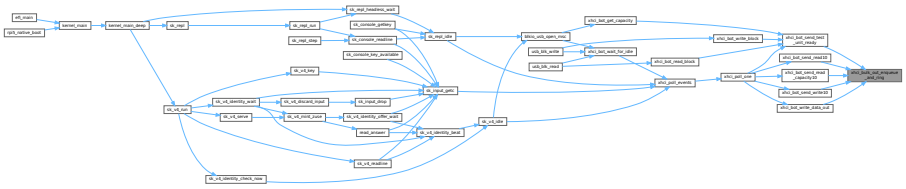
References xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_ring, xhci_msc_slot_t::bulk_out_ring_cycle, xhci_msc_slot_t::bulk_out_ring_enq, xhci_trb_t::control, xhci_dev_t::doorbell, xhci_trb_t::parameter, xhci_trb_t::status, XHCI_DB_TARGET, XHCI_EP_ADDR_TO_DCI, xhci_msc_slot_for(), XHCI_RING_TRB_COUNT, XHCI_TRB_CONTROL_CYCLE, XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, and XHCI_TRB_TYPE_LINK.

Referenced by xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), and xhci_bot_write_data_out().

Here is the call graph for this function:



Here is the caller graph for this function:



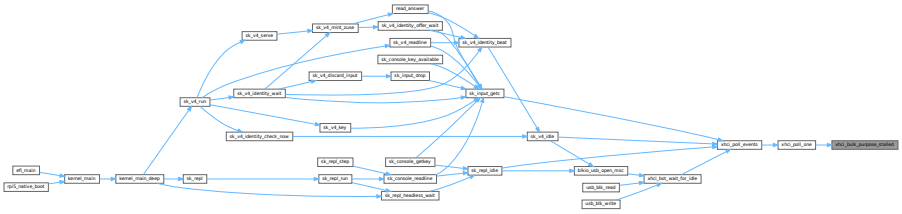
7.172.2.15 xhci_bulk_purpose_stalled()

```
static int xhci_bulk_purpose_stalled (
            xhci_dev_t * dev,
            uint32_t purpose )  [static]
```

References XHCI_XFER_BOT_DATA_IN, XHCI_XFER_BOT_DATA_OUT, XHCI_XFER_CBW_SENT, and XHCI_XFER_CSW_RECEIVED.

Referenced by xhci_poll_one().

Here is the caller graph for this function:



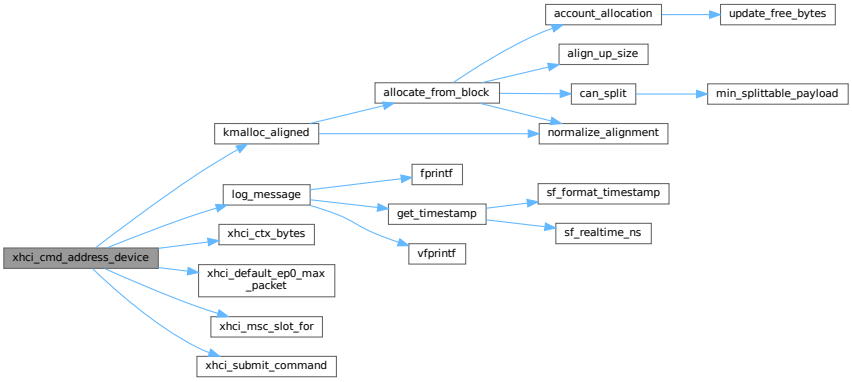
7.172.2.16 xhci_cmd_address_device()

```
int xhci_cmd_address_device (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t port_id,
            uint32_t speed )
```

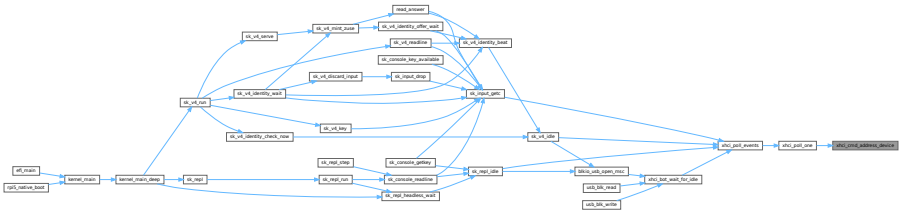
References xhci_input_ctrl_ctx32_t::add_flags, xhci_dev_t::cmd_ring, xhci_trb_t::control, xhci_dev_t::dcbaa, xhci_msc_slot_t::device_ctx, xhci_slot_ctx32_t::dword0, xhci_slot_ctx32_t::dword1, xhci_ep_ctx32_t::dword1, xhci_slot_ctx32_t::dword2, xhci_ep_ctx32_t::dword4, xhci_msc_slot_t::ep0_ring, xhci_msc_slot_t::ep0_ring_cycle, xhci_msc_slot_t::ep0_ring_enq, xhci_msc_slot_t::hid_ep_addr, xhci_msc_slot_t::hid_kbd, xhci_msc_slot_t::hid_running, xhci_msc_slot_t::in_use, xhci_msc_slot_t::input_ctx, kmalloc_aligned(), LOG_DEBUG, log_message(), xhci_trb_t::parameter, xhci_trb_t::status, xhci_ep_ctx32_t::tr_dequeue_ptr, xhci_ctx_bytes(), xhci_default_ep0_max_packet(), XHCI_EP_CTX_CERR_SHIF XHCI_EP_CTX_MAX_PACKET_SHIFT, XHCI_EP_CTX_TYPE_CONTROL_BIDI, XHCI_EP_CTX_TYPE_SHIFT, XHCI_INPUT_CTRL_ADD_EP0, XHCI_INPUT_CTRL_ADD_SLOT, xhci_msc_slot_for(), XHCI_RING_BYTES, XHCI_RING_TRB_COUNT, XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT, XHCI_SLOT_CTX_INTR_TARGET_SHIFT, XHCI_SLOT_CTX_ROOT_PORT_SHIFT, XHCI_SLOT_CTX_SPEED_SHIFT, xhci_submit_command(), XHCI_TRB_CONTROL_CY XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_ADDRESS_DEVICE_CMD, and XHCI_TRB_TYPE_LINK.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



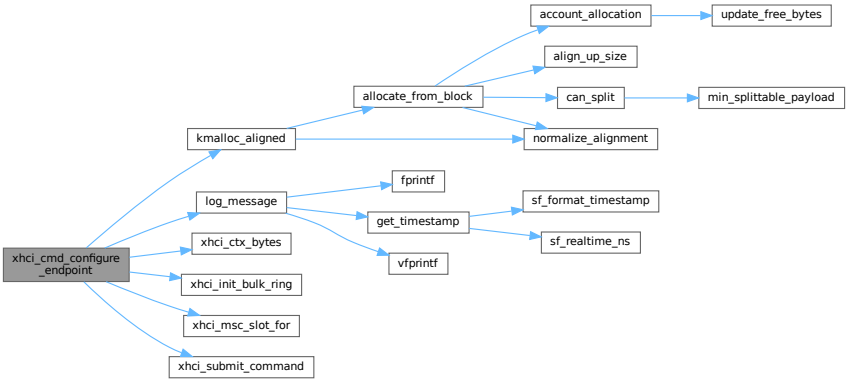
7.172.2.17 xhci_cmd_configure_endpoint()

```
int xhci_cmd_configure_endpoint (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

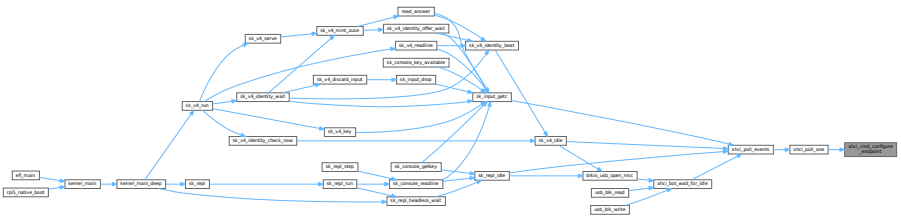
References xhci_input_ctrl_ctx32_t::add_flags, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_max_packet, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_in_ring_cycle, xhci_msc_slot_t::bulk_in_ring_enq, xhci_msc_slot_t::bulk_out_ep xhci_msc_slot_t::bulk_out_max_packet, xhci_msc_slot_t::bulk_out_ring, xhci_msc_slot_t::bulk_out_ring_cycle, xhci_msc_slot_t::bulk_out_ring_enq, xhci_dev_t::cmd_ring, xhci_dev_t::dcbaa, xhci_msc_slot_t::device_ctx, xhci_slot_ctx32_t::dword0, xhci_slot_ctx32_t::dword1, xhci_ep_ctx32_t::dword1, xhci_slot_ctx32_t::dword2, xhci_ep_ctx32_t::dword4, xhci_msc_slot_t::input_ctx, kmalloc_aligned(), LOG_DEBUG, log_message(), xhci_ep_ctx32_t::tr_dequeue_ptr, xhci_ctx_bytes(), XHCI_EP_ADDR_TO_DCI, XHCI_EP_CTX_CERR_SHIFT, XHCI_EP_CTX_MAX_PACKET_SHIFT, XHCI_EP_CTX_TYPE_BULK_IN, XHCI_EP_CTX_TYPE_BULK_OUT, XHCI_EP_CTX_TYPE_SHIFT, xhci_init_bulk_ring(), XHCI_INPUT_CTRL_ADD_SLOT, xhci_msc_slot_for(), XHCI_RING_BYTES, XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT, xhci_submit_command(), and XHCI_TRB_TYPE_CONFIGU

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



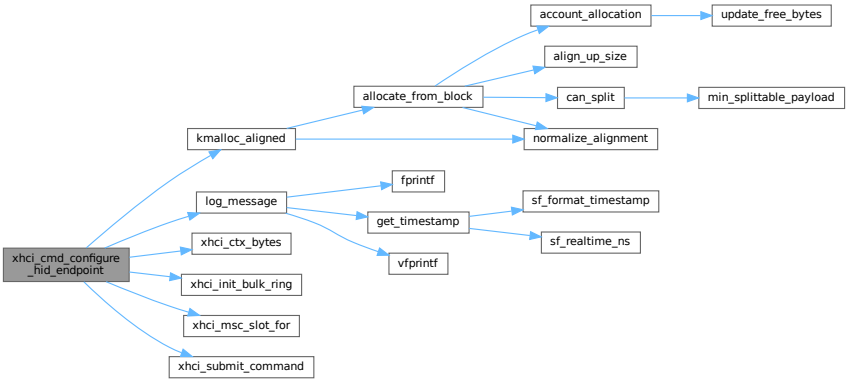
7.172.2.18 xhci_cmd_configure_hid_endpoint()

```
int xhci_cmd_configure_hid_endpoint (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

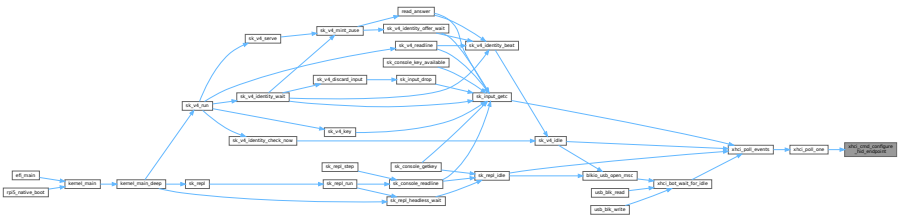
References xhci_input_ctrl_ctx32_t::add_flags, xhci_dev_t::cmd_ring, xhci_dev_t::dcbaa, xhci_msc_slot_t::device_ctx, xhci_slot_ctx32_t::dword0, xhci_ep_ctx32_t::dword0, xhci_slot_ctx32_t::dword1, xhci_ep_ctx32_t::dword1, xhci_slot_ctx32_t::dword2, xhci_ep_ctx32_t::dword4, xhci_msc_slot_t::hid_buf, xhci_msc_slot_t::hid_ep_addr, xhci_msc_slot_t::hid_interval, xhci_msc_slot_t::hid_max_packet, xhci_msc_slot_t::hid_ring, xhci_msc_slot_t::hid_ring_cycle, xhci_msc_slot_t::hid_ring_enq, xhci_msc_slot_t::input_ctx, kmalloc_aligned(), LOG_DEBUG, log_message(), xhci_ep_ctx32_t::tr_dequeue_ptr, xhci_ctx_bytes(), XHCI_EP_ADDR_TO_DCI, XHCI_EP_CTX_CERR_SHIFT, XHCI_EP_CTX_INTERVAL_SHIFT, XHCI_EP_CTX_MAX_PACKET_SHIFT, XHCI_EP_CTX_TYPE_INTERRUPT_IN, XHCI_EP_CTX_TYPE_SHIFT, xhci_init_bulk_ring(), XHCI_INPUT_CTRL_ADD_SLOT, xhci_msc_slot_for(), XHCI_RING_BYTES, XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT, XHCI_SLOT_CTX_SPEED_SHIFT, xhci_submit_command(), and XHCI_TRB_TYPE_CONFIGURE_ENDPOINT_CMD.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



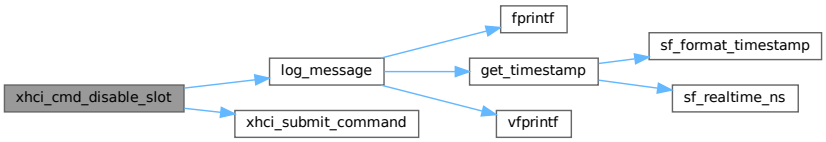
7.172.2.19 xhci_cmd_disable_slot()

```
int xhci_cmd_disable_slot (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

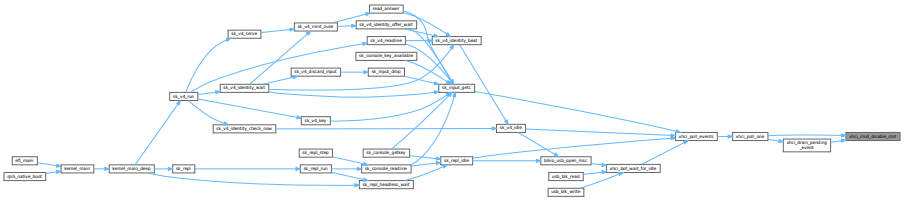
References xhci_dev_t::cmd_ring, LOG_DEBUG, log_message(), xhci_submit_command(), and XHCI_TRB_TYPE_DISABLE_SLOT.

Referenced by xhci_drain_pending_event(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



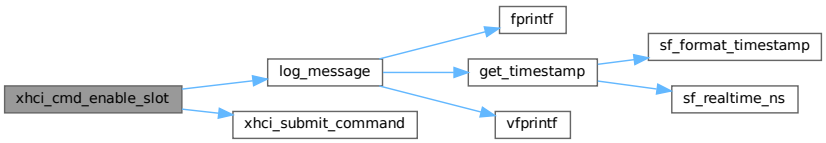
7.172.2.20 xhci_cmd_enable_slot()

```
int xhci_cmd_enable_slot (
            xhci_dev_t * dev )
```

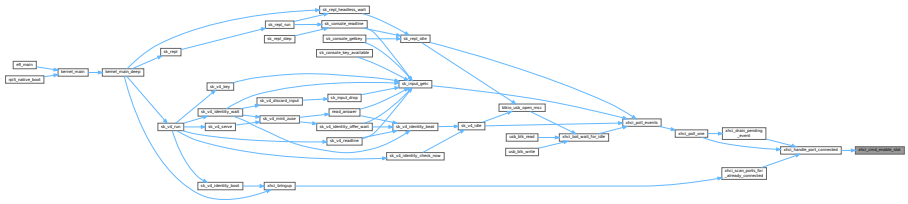
References xhci_dev_t::cmd_ring, LOG_DEBUG, log_message(), xhci_submit_command(), and XHCI_TRB_TYPE_ENABLE_SLOT_

Referenced by xhci_handle_port_connected().

Here is the call graph for this function:



Here is the caller graph for this function:



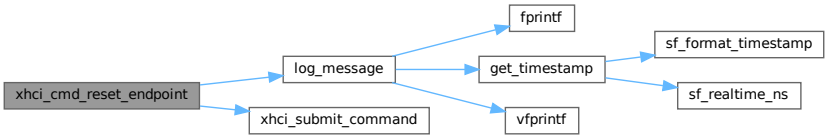
7.172.2.21 xhci_cmd_reset_endpoint()

```
int xhci_cmd_reset_endpoint (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t ep_id )
```

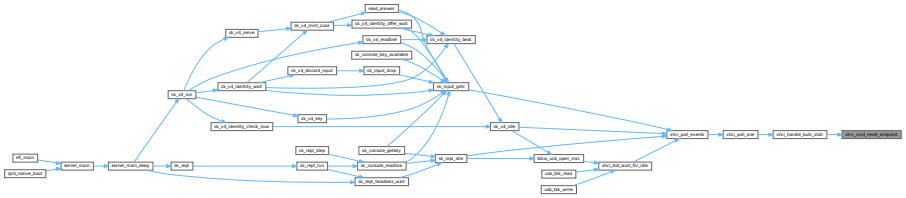
References xhci_dev_t::cmd_ring, LOG_DEBUG, log_message(), XHCI_EP_ADDR_TO_DCI, xhci_submit_command(), and XHCI_TRB_TYPE_RESET_ENDPOINT_CMD.

Referenced by xhci_handle_bulk_stall().

Here is the call graph for this function:



Here is the caller graph for this function:



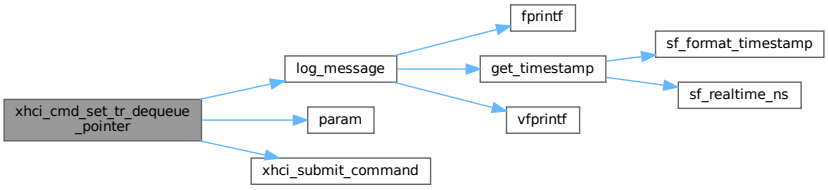
7.172.2.22 xhci_cmd_set_tr_dequeue_pointer()

```
int xhci_cmd_set_tr_dequeue_pointer (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t ep_id,
            uint64_t new_dequeue,
            uint32_t dcs )
```

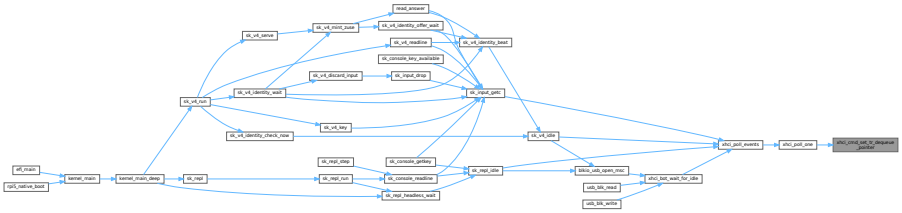
References xhci_dev_t::cmd_ring, LOG_DEBUG, log_message(), param(), xhci_submit_command(), and XHCI_TRB_TYPE_SET_TR_DEQUEUE_POINTER_CMD.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



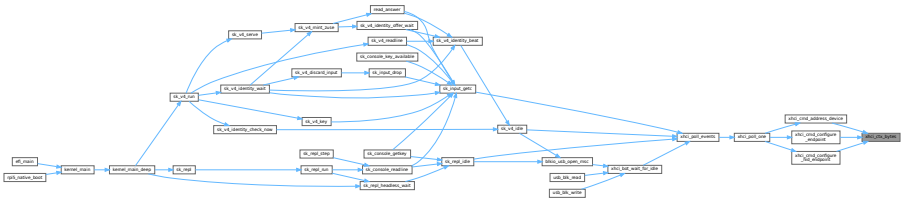
7.172.2.23 xhci_ctx_bytes()

```
static size_t xhci_ctx_bytes (
            const xhci_dev_t * dev )  [static]
```

References xhci_dev_t::cap, xhci_cap_regs_t::hcc_params1, and XHCI_HCCPARAMS1_CSZ.

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

Here is the caller graph for this function:

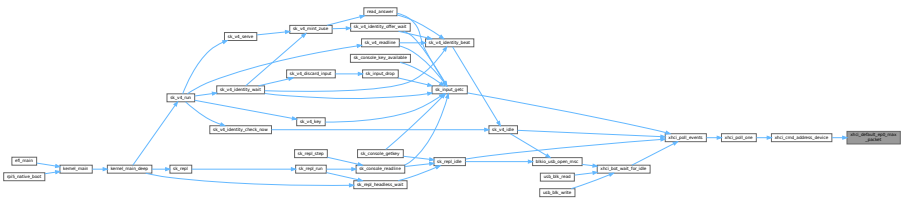


7.172.2.24 xhci_default_ep0_max_packet()

```
static uint32_t xhci_default_ep0_max_packet (
            uint32_t speed )  [static]
```

Referenced by xhci_cmd_address_device().

Here is the caller graph for this function:



7.172.2.25 xhci_dev_at()

```
xhci_dev_t * xhci_dev_at (
            uint32_t i )
```

References g_xhci_dev_count, and g_xhci_devs.

Referenced by sk_repl_idle(), sk_v4_idle(), and sk_word_blk_attach_ack().

Here is the caller graph for this function:

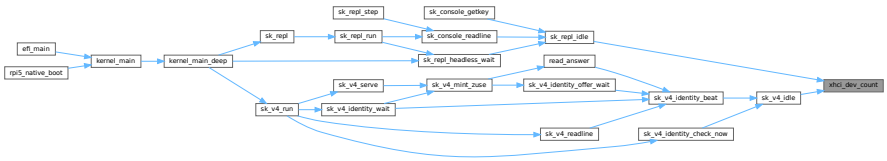
7.172.2.26 xhci_dev_count()

```
uint32_t xhci_dev_count (
            void  )
```

References g_xhci_dev_count.

Referenced by sk_repl_idle(), and sk_v4_idle().

Here is the caller graph for this function:



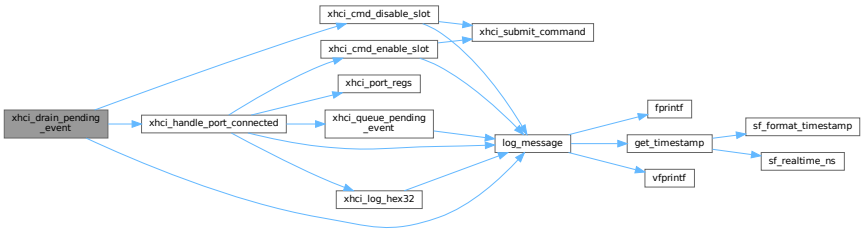
7.172.2.27 xhci_drain_pending_event()

```
static void xhci_drain_pending_event (
            xhci_dev_t * dev )  [static]
```

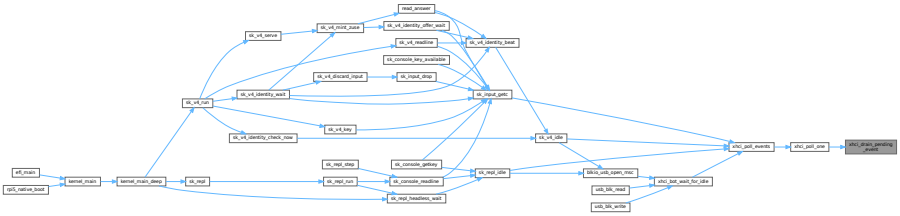
References xhci_dev_t::connect_state, xhci_dev_t::is_connect, LOG_DEBUG, log_message(), xhci_dev_t::pending_disable_slot_id, xhci_dev_t::pending_events, xhci_dev_t::port_id, xhci_dev_t::portsc, xhci_dev_t::slot_id, xhci_dev_t::valid, xhci_cmd_disable_slot(), xhci_handle_port_connected(), and XHCI_MAX_TRACKED_PORTS.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



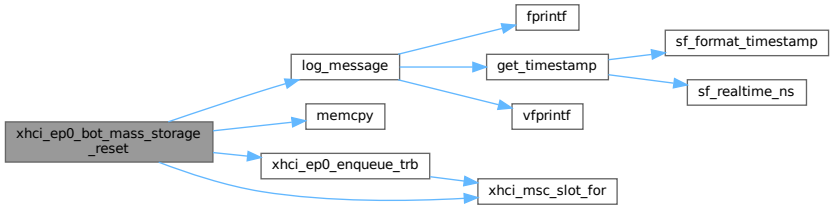
7.172.2.28 xhci_ep0_bot_mass_storage_reset()

```
int xhci_ep0_bot_mass_storage_reset (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

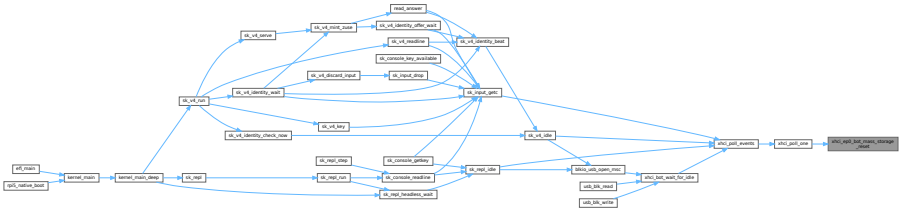
References usb_setup_packet_t::bmRequestType, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), memcpy(), xhci_msc_slot_t::transfer_purpose, USB_BOT_MASS_STORAGE_RESET, USB_RECIP_INTERFACE, USB_REQ_TYPE_CLASS, XHCI_DB_TARGET, xhci_ep0_enqueue_trb(), xhci_msc_slot_for(), XHCI_SETUP_TRT_NO_DATA, XHCI_TRB_CONTROL_DIR_IN, XHCI_TRB_CONTROL_IDT, XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TRT_SHIFT, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_SETUP_STAGE, XHCI_TRB_TYPE_STATUS_STAGE, and XHCI_XFER_BOT_RESET.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



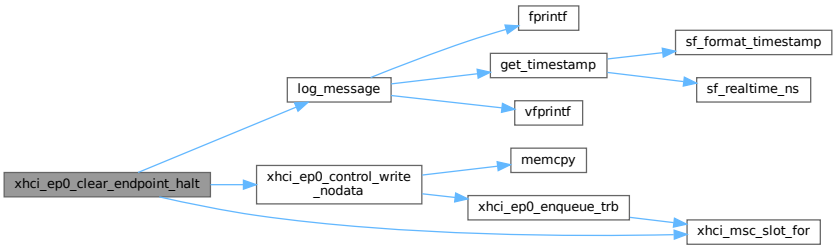
7.172.2.29 xhci_ep0_clear_endpoint_halt()

```
int xhci_ep0_clear_endpoint_halt (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint8_t ep_addr )
```

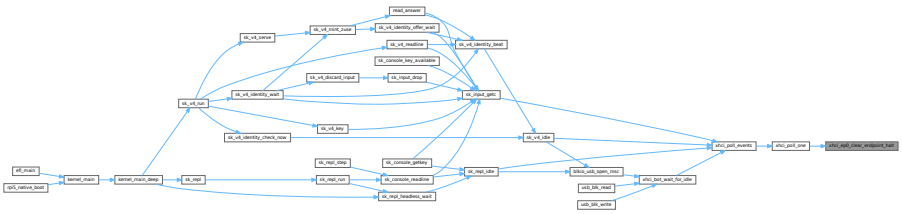
References xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_FEATURE_ENDPOINT_HALT, USB_REQ_CLEAR_FEATURE, XHCI_DB_TARGET, xhci_ep0_control_write_nodata(), xhci_msc_slot_for(), and XHCI_XFER_CLEAR_HALT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



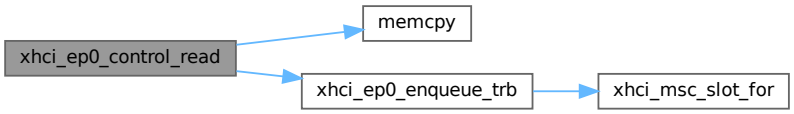
7.172.2.30 xhci_ep0_control_read()

```
static void xhci_ep0_control_read (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint8_t bRequest,
            uint16_t wValue,
            uint16_t wIndex,
            uint8_t * buf,
            uint16_t len )  [static]
```

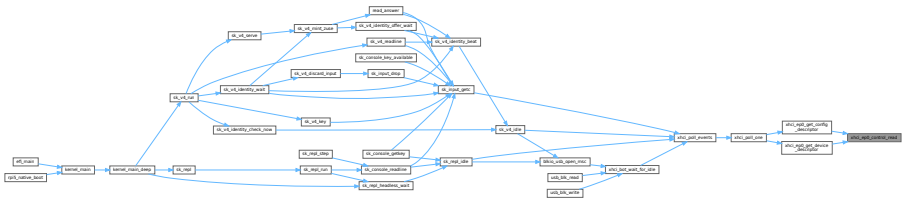
References usb_setup_packet_t::bmRequestType, memcpy(), USB_DIR_DEVICE_TO_HOST, xhci_ep0_enqueue_trb(), XHCI_SETUP_TRT_IN_DATA, XHCI_TRB_CONTROL_DIR_IN, XHCI_TRB_CONTROL_IDT, XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TRT_SHIFT, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_DATA_STAGE, XHCI_TRB_TYPE_SETUP_STAGE, and XHCI_TRB_TYPE_STATUS_STAGE.

Referenced by xhci_ep0_get_config_descriptor(), and xhci_ep0_get_device_descriptor().

Here is the call graph for this function:



Here is the caller graph for this function:



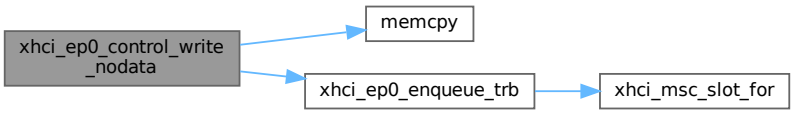
7.172.2.31 xhci_ep0_control_write_nodata()

```
static void xhci_ep0_control_write_nodata (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint8_t bRequest,
            uint16_t wValue,
            uint16_t wIndex )  [static]
```

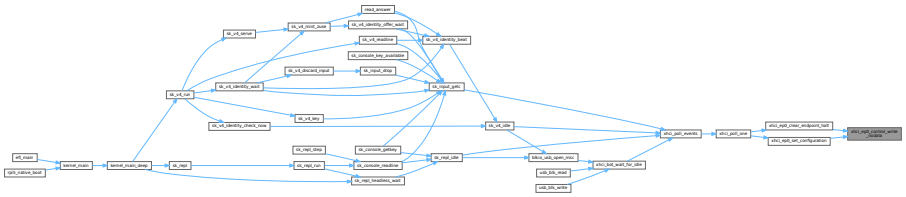
References usb_setup_packet_t::bmRequestType, memcpy(), USB_DIR_HOST_TO_DEVICE, xhci_ep0_enqueue_trb(), XHCI_SETUP_TRT_NO_DATA, XHCI_TRB_CONTROL_DIR_IN, XHCI_TRB_CONTROL_IDT, XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TRT_SHIFT, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_SETUP_STAGE, and XHCI_TRB_TYPE_STATUS_STAGE.

Referenced by xhci_ep0_clear_endpoint_halt(), and xhci_ep0_set_configuration().

Here is the call graph for this function:



Here is the caller graph for this function:



7.172.2.32 xhci_ep0_enqueue_trb()

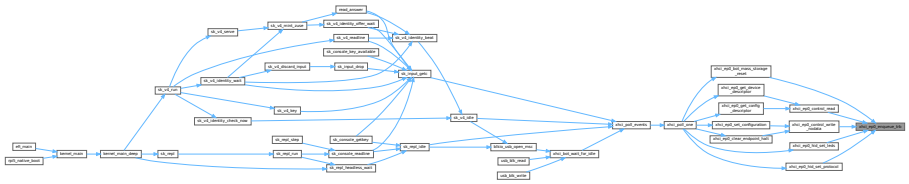
```
static void xhci_ep0_enqueue_trb (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint64_t parameter,
            uint32_t status,
            uint32_t control_bits )  [static]
```

References xhci_trb_t::control, xhci_msc_slot_t::ep0_ring, xhci_msc_slot_t::ep0_ring_cycle, xhci_msc_slot_t::ep0_ring_enq, xhci_trb_t::parameter, xhci_trb_t::status, xhci_msc_slot_for(), XHCI_RING_TRB_COUNT, XHCI_TRB_CONTROL_CYCLE, XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, and XHCI_TRB_TYPE_LINK.

Referenced by xhci_ep0_bot_mass_storage_reset(), xhci_ep0_control_read(), xhci_ep0_control_write_nodata(), xhci_ep0_hid_set_leds(), and xhci_ep0_hid_set_protocol().

Here is the call graph for this function:

Here is the caller graph for this function:



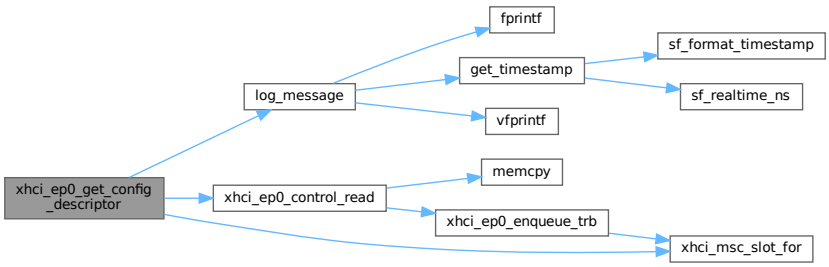
7.172.2.33 xhci_ep0_get_config_descriptor()

```
int xhci_ep0_get_config_descriptor (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint16_t length )
```

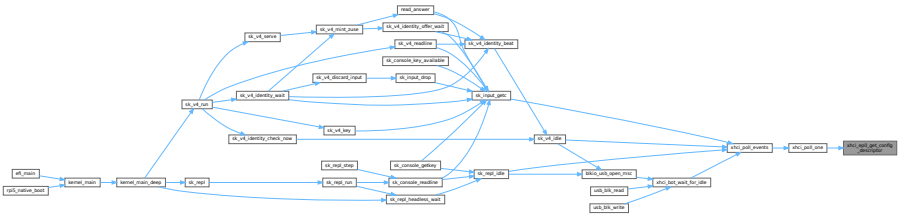
References xhci_msc_slot_t::config_descriptor, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_DESC_TYPE_CONFIG, USB_REQ_GET_DESCRIPTOR, XHCI_DB_TARGET, xhci_ep0_control_read(), xhci_msc_slot_for(), XHCI_XFER_CONFIG_DESC_FULL, and XHCI_XFER_CONFIG_DESC_SHORT.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



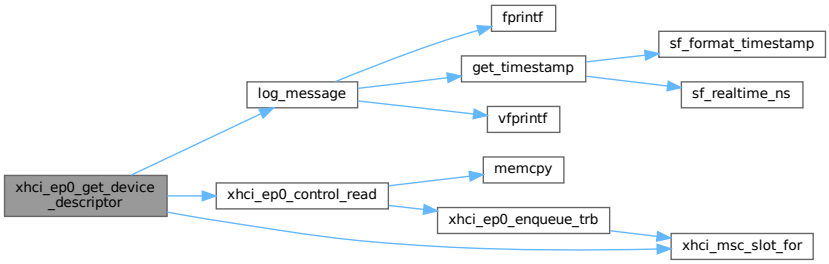
7.172.2.34 xhci_ep0_get_device_descriptor()

```
int xhci_ep0_get_device_descriptor (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

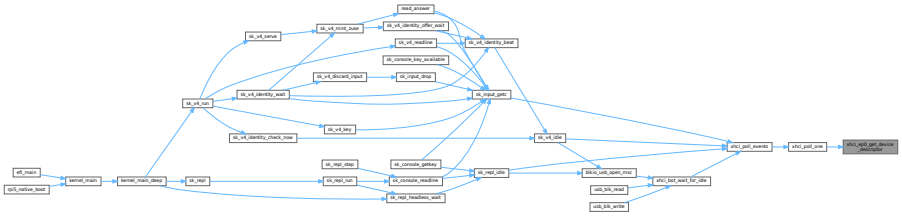
References xhci_msc_slot_t::device_descriptor, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_DESC_TYPE_DEVICE, USB_REQ_GET_DESCRIPTOR, XHCI_DB_TARGET, xhci_ep0_control_read(), xhci_msc_slot_for(), and XHCI_XFER_DEVICE_DESC.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



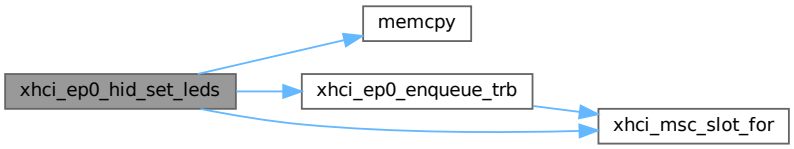
7.172.2.35 xhci_ep0_hid_set_leds()

```
static int xhci_ep0_hid_set_leds (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint8_t leds )  [static]
```

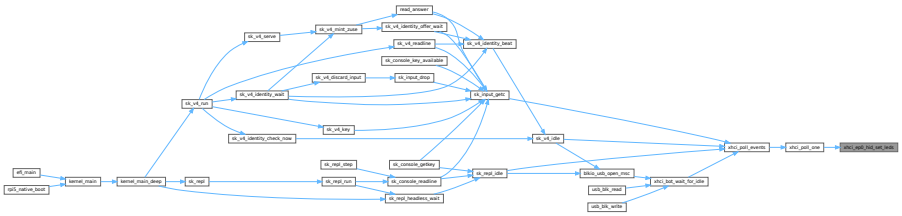
References usb_setup_packet_t::bmRequestType, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, xhci_msc_slot_t::hid_buf, xhci_msc_slot_t::hid_iface, xhci_msc_slot_t::hid_leds_sent, memcpy(), xhci_msc_slot_t::transfer_purpose, USB_HID_REPORT_OUTPUT, USB_HID_REQ_SET_REPORT, USB_RECIP_INTERFACE, USB_REQ_TYPE_CLASS, XHCI_DB_TARGET, xhci_ep0_enqueue_trb(), xhci_msc_slot_for(), XHCI_SETUP_TRT_OUT_DATA, XHCI_TRB_CONTROL_DIR_IN, XHCI_TRB_CONTROL_IDT, XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TRT_SHIFT, XHCI_TRB_CONTROL_TYPE_SH, XHCI_TRB_TYPE_DATA_STAGE, XHCI_TRB_TYPE_SETUP_STAGE, XHCI_TRB_TYPE_STATUS_STAGE, and XHCI_XFER_HID_SET_LEDS.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



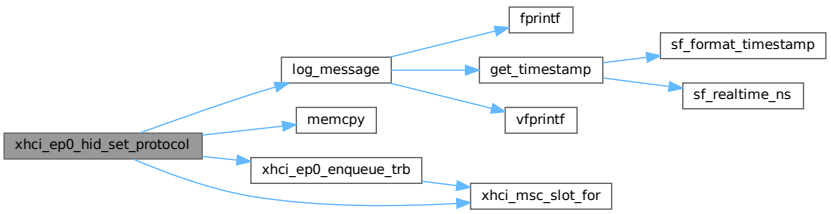
7.172.2.36 xhci_ep0_hid_set_protocol()

```
int xhci_ep0_hid_set_protocol (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

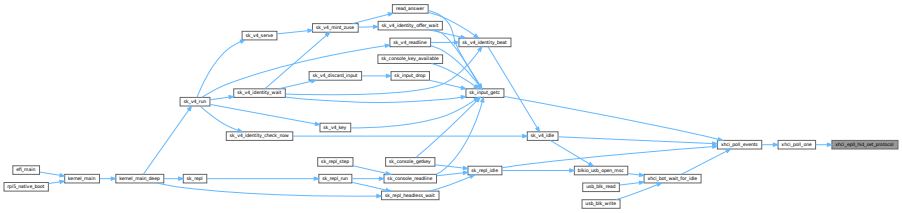
References usb_setup_packet_t::bmRequestType, xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, xhci_msc_slot_t::hid_iface, LOG_DEBUG, log_message(), memcpy(), xhci_msc_slot_t::transfer_purpose, USB_HID_PROTOCOL_BOOT, USB_HID_REQ_SET_PROTOCOL, USB_RECIP_INTERFACE, USB_REQ_TYPE_CLASS, XHCI_DB_TARGET, xhci_ep0_enqueue_trb(), xhci_msc_slot_for(), XHCI_SETUP_TRT_NO_DATA, XHCI_TRB_CONTROL_DIR_IN, XHCI_TRB_CONTROL_IDT, XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TRT_SHIFT, XHCI_TRB_CONTROL_TYPE_SH XHCI_TRB_TYPE_SETUP_STAGE, XHCI_TRB_TYPE_STATUS_STAGE, and XHCI_XFER_HID_SET_PROTOCOL.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



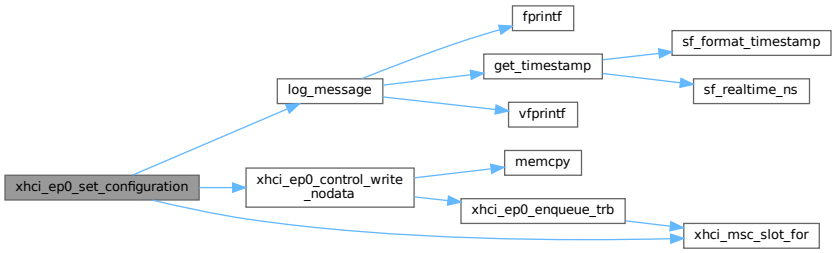
7.172.2.37 xhci_ep0_set_configuration()

```
int xhci_ep0_set_configuration (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint8_t config_value )
```

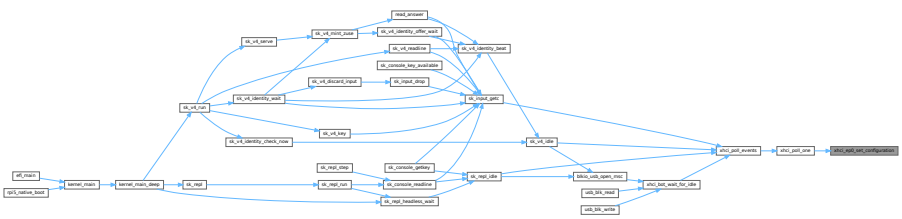
References xhci_dev_t::doorbell, xhci_msc_slot_t::ep0_ring, LOG_DEBUG, log_message(), xhci_msc_slot_t::transfer_purpose, USB_REQ_SET_CONFIGURATION, XHCI_DB_TARGET, xhci_ep0_control_write_nodata(), xhci_msc_slot_for(), and XHCI_XFER_SET_CONFIG.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



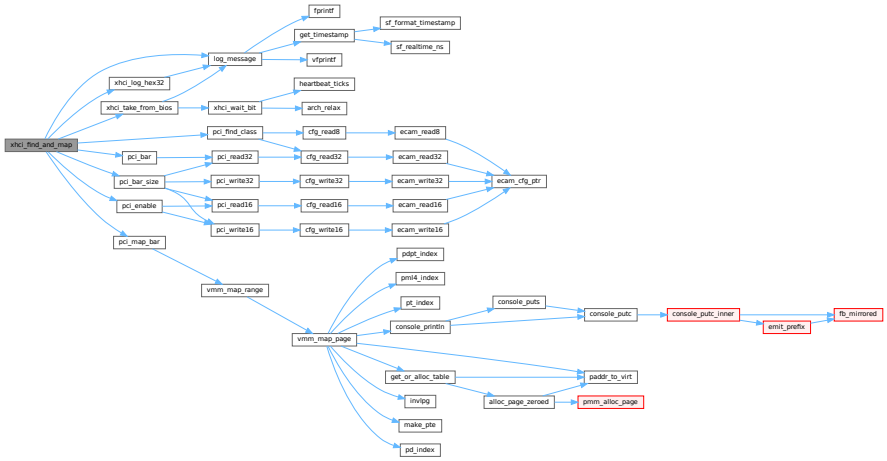
7.172.2.38 xhci_find_and_map()

```
int xhci_find_and_map (
            xhci_dev_t * dev,
            unsigned index )
```

References xhci_dev_t::bar0_phys, xhci_dev_t::cap, xhci_cap_regs_t::cap_length, xhci_cap_regs_t::db_off, PciDevice::device_id, xhci_dev_t::doorbell, xhci_cap_regs_t::hcs_params1, xhci_cap_regs_t::hcs_params2, LOG_DEBUG, LOG_ERROR, log_message(), xhci_dev_t::max_intrs, xhci_dev_t::max_ports, xhci_dev_t::max_scratchpad_bufs, xhci_dev_t::max_slots, xhci_dev_t::op, xhci_dev_t::pci, pci_bar(), pci_bar_size(), pci_enable(), pci_find_class(), pci_map_bar(), xhci_cap_regs_t::rts_off, xhci_dev_t::runtime, PciDevice::vendor_id, XHCI_BAR0_MIN_MAP_SIZE, XHCI_HCSPARAMS1_MAX_INTRS, XHCI_HCSPARAMS1_MAX_PORTS, XHCI_HCSPARAMS1_MAX_SLOTS, XHCI_HCSPARAMS2_MAX_SCRATCHPAD_BUFS, xhci_log_hex32(), XHCI_PCI_CLASS, XHCI_PCI_PROG_IF, XHCI_PCI_SUBCLASS, and xhci_take_from_bios().

Referenced by kernel_main_deep(), and sk_v4_identity_boot().

Here is the call graph for this function:



Here is the caller graph for this function:

7.172.2.39 xhci_handle_bulk_stall()

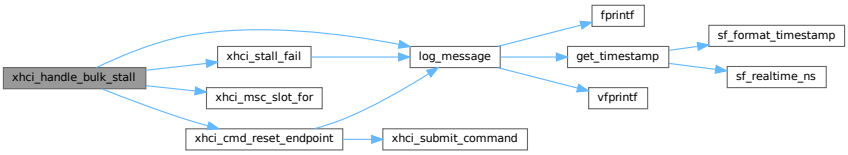
```
static void xhci_handle_bulk_stall (
            xhci_dev_t * dev,
            uint32_t slot_id,
            uint32_t purpose )  [static]
```

References xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_stall_recoveries, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_out_ep_addr, xhci_dev_t::connect_state, LOG_ERROR, log_message(), LOG_WARN,

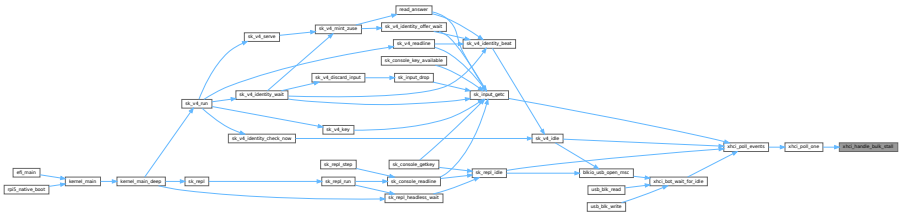
xhci_msc_slot_t::next_action, xhci_dev_t::stall_dci, xhci_dev_t::stall_ep_addr, xhci_dev_t::stall_retry_action, XHCI_BOT_STALL_MAX_RECOVERIES, xhci_cmd_reset_endpoint(), XHCI_EP_ADDR_TO_DCI, xhci_msc_slot_for(), XHCI_NEXT_ACTION_BOT_CSW_RECEIVE, XHCI_NEXT_ACTION_BOT_DATA_IN, XHCI_NEXT_ACTION_BOT_DATA_OUT, XHCI_NEXT_ACTION_BOT_RESET, XHCI_NEXT_ACTION_BOT_SEND_READ10, XHCI_NEXT_ACTION_BOT_SEND_READ_CA XHCI_NEXT_ACTION_BOT_SEND_TUR, XHCI_NEXT_ACTION_BOT_SEND_WRITE10, xhci_stall_fail(), XHCI_XFER_BOT_DATA_IN, XHCI_XFER_BOT_DATA_OUT, XHCI_XFER_CBW_SENT, and XHCI_XFER_CSW_RECEIVED.

Referenced by xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



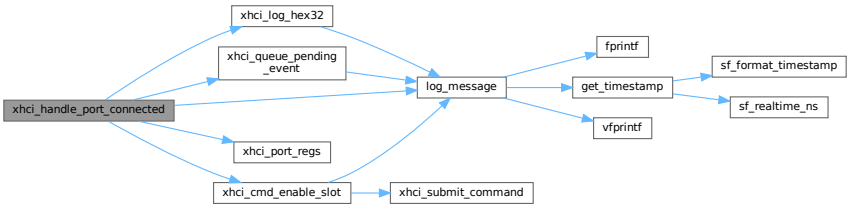
7.172.2.40 xhci_handle_port_connected()

```
static void xhci_handle_port_connected (
            xhci_dev_t * dev,
            uint32_t port_id,
            uint32_t portsc )  [static]
```

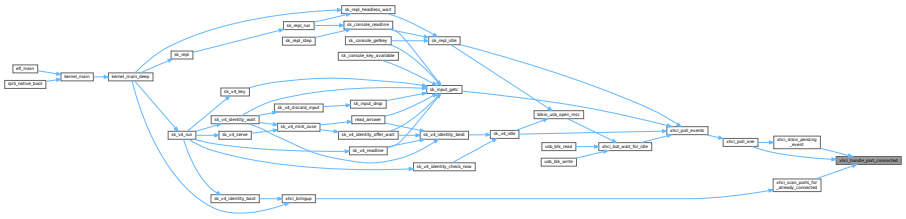
References xhci_dev_t::connect_state, LOG_DEBUG, log_message(), LOG_WARN, xhci_dev_t::pending_connect_port_id, xhci_dev_t::pending_connect_speed, xhci_port_regs_t::portsc, xhci_cmd_enable_slot(), xhci_log_hex32(), XHCI_MAX_TRACKED_PORTS, xhci_port_regs(), XHCI_PORTSC_PED, XHCI_PORTSC_PP, XHCI_PORTSC_PR, XHCI_PORTSC_SPEED, and xhci_queue_pending_event().

Referenced by xhci_drain_pending_event(), xhci_poll_one(), and xhci_scan_ports_for_already_connected().

Here is the call graph for this function:



Here is the caller graph for this function:



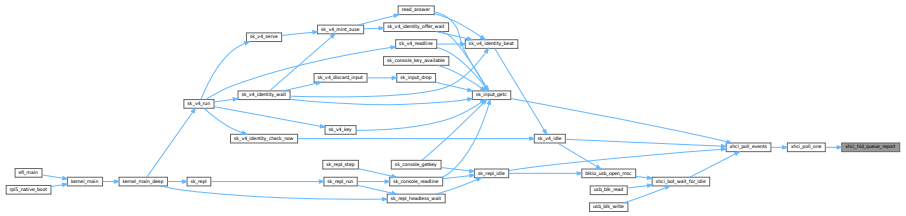
7.172.2.41 xhci_hid_queue_report()

```
static void xhci_hid_queue_report (
            xhci_dev_t * dev,
            uint32_t slot_id,
            xhci_msc_slot_t * ms )  [static]
```

References xhci_trb_t::control, xhci_dev_t::doorbell, xhci_msc_slot_t::hid_buf, xhci_msc_slot_t::hid_ep_addr, xhci_msc_slot_t::hid_max_packet, xhci_msc_slot_t::hid_ring, xhci_msc_slot_t::hid_ring_cycle, xhci_msc_slot_t::hid_ring_enq, xhci_trb_t::parameter, xhci_trb_t::status, XHCI_DB_TARGET, XHCI_EP_ADDR_TO_DCI, XHCI_RING_TRB_COUNT, XHCI_TRB_CONTROL_CYCLE, XHCI_TRB_CONTROL_IOC, XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, XHCI_TRB_TYPE_LINK, and XHCI_TRB_TYPE_NORMAL.

Referenced by xhci_poll_one().

Here is the caller graph for this function:



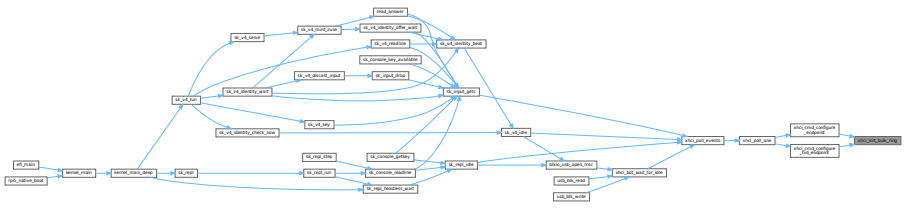
7.172.2.42 xhci_init_bulk_ring()

```
static void xhci_init_bulk_ring (
            xhci_trb_t * ring )  [static]
```

References ring, XHCI_RING_TRB_COUNT, XHCI_TRB_CONTROL_CYCLE, XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, and XHCI_TRB_TYPE_LINK.

Referenced by xhci_cmd_configure_endpoint(), and xhci_cmd_configure_hid_endpoint().

Here is the caller graph for this function:



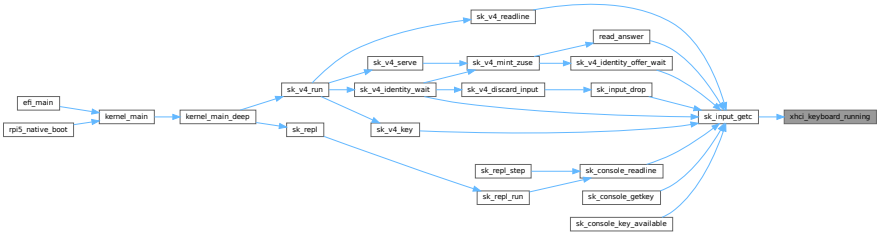
7.172.2.43 xhci_keyboard_running()

```
int xhci_keyboard_running (
            void  )
```

References g_xhci_keyboards.

Referenced by sk_input_getc().

Here is the caller graph for this function:



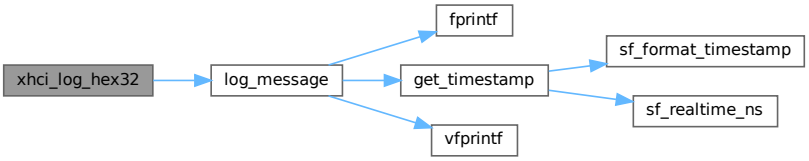
7.172.2.44 xhci_log_hex32()

```
static void xhci_log_hex32 (
            const char * prefix,
            uint32_t v )  [static]
```

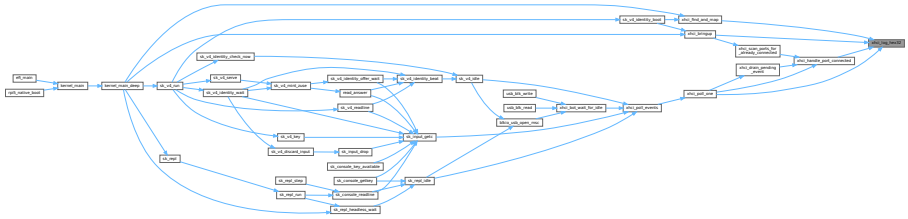
References LOG_DEBUG, and log_message().

Referenced by xhci_bringup(), xhci_find_and_map(), xhci_handle_port_connected(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



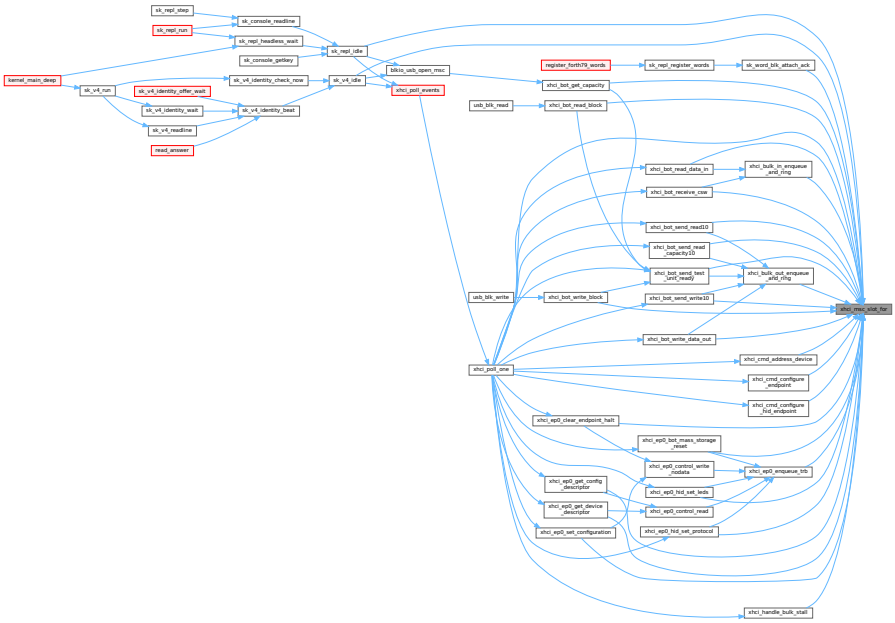
7.172.2.45 xhci_msc_slot_for()

```
xhci_msc_slot_t * xhci_msc_slot_for (
            xhci_dev_t * dev,
            uint32_t slot_id )
```

References xhci_dev_t::max_slots, and xhci_dev_t::msc_slots.

Referenced by sk_repl_idle(), sk_v4_idle(), sk_word_blk_attach_ack(), xhci_bot_get_capacity(), xhci_bot_read_block(), xhci_bot_read_data_in(), xhci_bot_receive_csw(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_block(), xhci_bot_write_data_out(), xhci_bulk_in_enqueue_and_ring(), xhci_bulk_out_enqueue_and_ring(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), xhci_cmd_configure_hid_endpoint(), xhci_ep0_bot_mass_storage_reset(), xhci_ep0_clear_endpoint_halt(), xhci_ep0_enqueue_trb(), xhci_ep0_get_config_descriptor(), xhci_ep0_get_device_descriptor(), xhci_ep0_hid_set_leds(), xhci_ep0_hid_set_protocol(), xhci_ep0_set_configuration(), xhci_handle_bulk_stall(), and xhci_poll_one().

Here is the caller graph for this function:



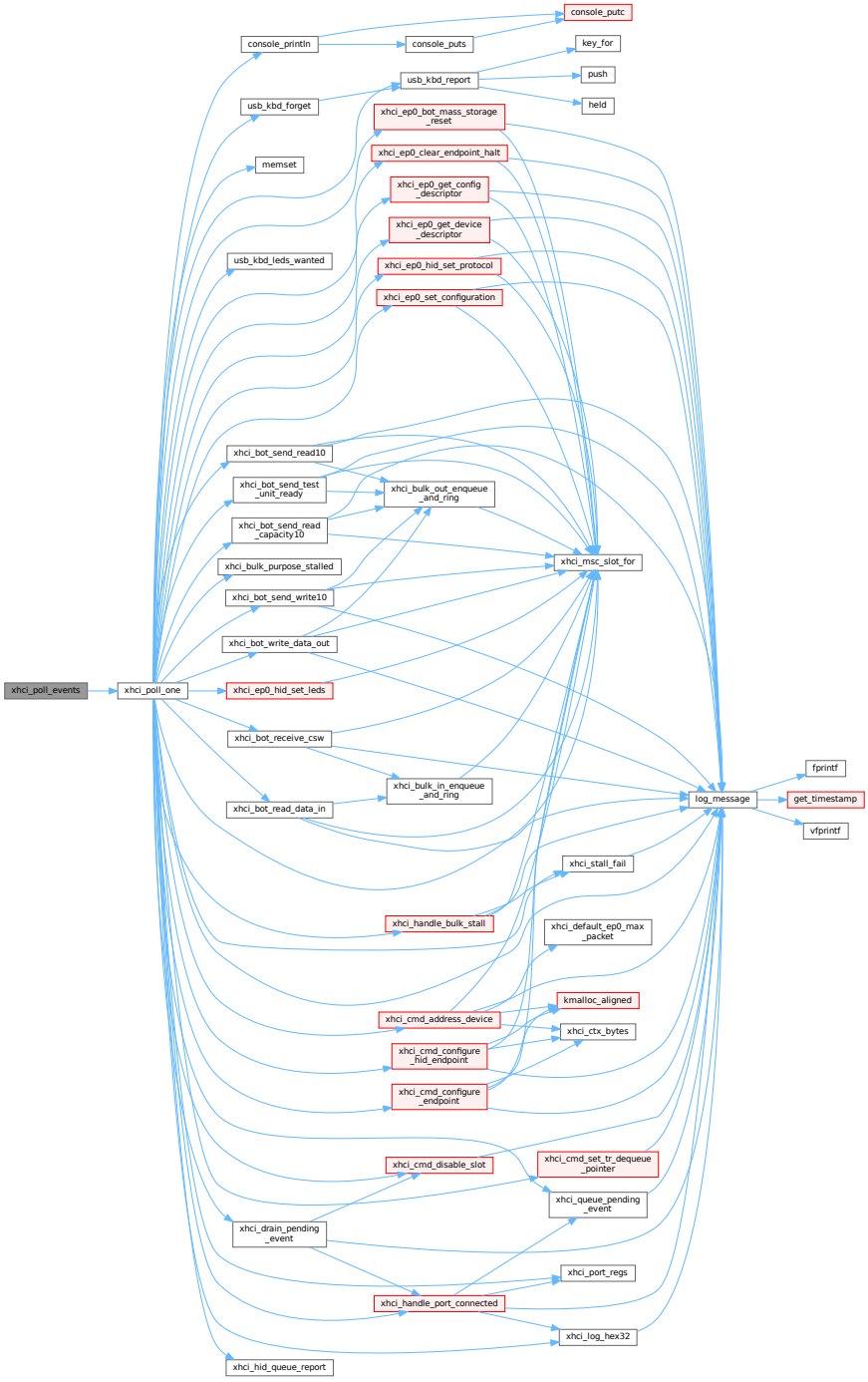
7.172.2.46 xhci_poll_events()

```
void xhci_poll_events (
            void  )
```

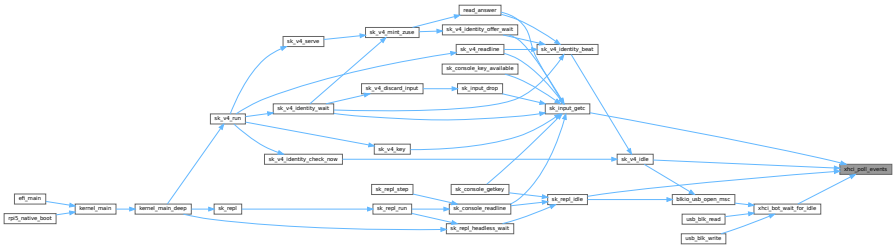
References g_xhci_dev_count, g_xhci_devs, and xhci_poll_one().

Referenced by sk_input_getc(), sk_repl_idle(), sk_v4_idle(), and xhci_bot_wait_for_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.172.2.47 xhci_poll_one()

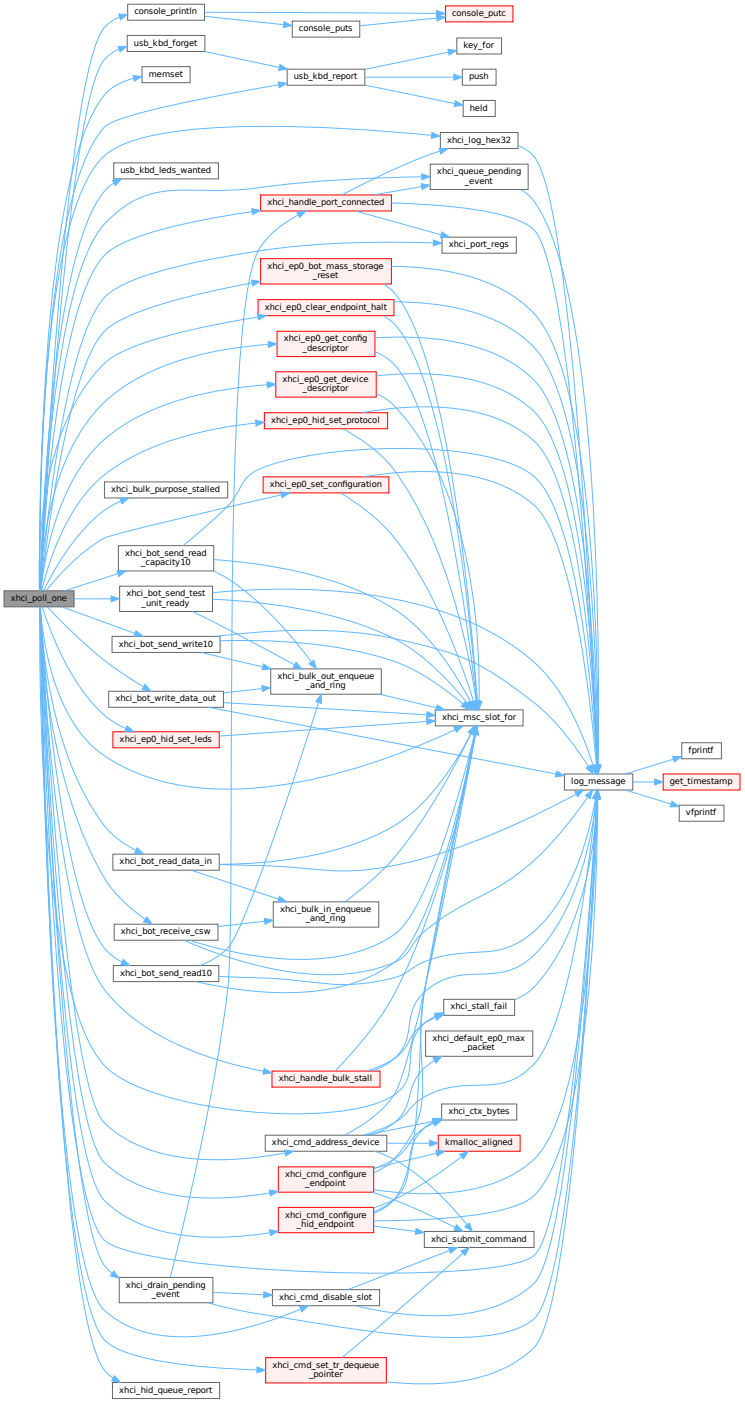
```
static void xhci_poll_one (
            xhci_dev_t * dev )  [static]
```

References usb_bot_csw_t::bCSWStatus, xhci_dev_t::bot_cap_block_size, xhci_dev_t::bot_cap_last_lba, xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_csw, xhci_dev_t::bot_data_buf, xhci_dev_t::bot_expected_data_len, xhci_dev_t::bot_last_status, xhci_dev_t::bot_last_tag, xhci_msc_slot_t::bot_msc_attach_pending, xhci_msc_slot_t::bot_msc_attache xhci_msc_slot_t::bot_msc_detach_pending, xhci_dev_t::bot_read10_block_size, xhci_dev_t::bot_read10_lba, xhci_dev_t::bot_read10_num_blocks, xhci_dev_t::bot_reset_clear_remaining, xhci_dev_t::bot_tur_chain_target, xhci_dev_t::bot_tur_retries, xhci_dev_t::bot_write10_block_size, xhci_dev_t::bot_write10_lba, xhci_dev_t::bot_write10_num_blocks, xhci_msc_slot_t::bulk_in_ep_addr, xhci_msc_slot_t::bulk_in_max_packet, xhci_msc_slot_t::bulk_in_ring, xhci_msc_slot_t::bulk_in_ri xhci_msc_slot_t::bulk_in_ring_enq, xhci_msc_slot_t::bulk_out_ep_addr, xhci_msc_slot_t::bulk_out_max_packet, xhci_msc_slot_t::bulk_out_ring, xhci_msc_slot_t::bulk_out_ring_cycle, xhci_msc_slot_t::bulk_out_ring_enq, xhci_msc_slot_t::config_descriptor, xhci_msc_slot_t::config_total_length, xhci_dev_t::connect_state, console_println(), xhci_trb_t::control, xhci_dev_t::dcbaa, usb_bot_csw_t::dCSWDataResidue, usb_bot_csw_t::dCSWSignature, usb_bot_csw_t::dCSWTag, xhci_msc_slot_t::device_descriptor, xhci_intr_regs_t::erdp, xhci_dev_t::evt_ring, xhci_dev_t::evt_ring_cycle, xhci_dev_t::evt_ring_deq, g_xhci_keyboards, xhci_msc_slot_t::hid_buf, xhci_msc_slot_t::hid_ep_addr, xhci_msc_slot_t::hid_iface, xhci_msc_slot_t::hid_interval, xhci_msc_slot_t::hid_kbd, xhci_msc_slot_t::hid_leds_sent, xhci_msc_slot_t::hid_max_packet, xhci_msc_slot_t::hid_prev, xhci_msc_slot_t::hid_running, xhci_msc_slot_t::in_use, xhci_dev_t::intr0, LOG_DEBUG, LOG_ERROR, LOG_INFO, log_message(), LOG_WARN, xhci_dev_t::max_slots, memset(), xhci_msc_slot_t::next_action, xhci_msc_slot_t::next_action_config_value, xhci_msc_slot_t::next_action_length, xhci_trb_t::parameter, xhci_dev_t::pending_connect_port_id, xhci_dev_t::pending_connect_slot_id, xhci_dev_t::pending_connect_sp xhci_dev_t::pending_disable_slot_id, xhci_dev_t::port_slot_id, xhci_port_regs_t::portsc, xhci_dev_t::stall_ep_addr, xhci_dev_t::stall_retry_action, xhci_trb_t::status, xhci_msc_slot_t::transfer_purpose, USB_BOT_CSW_SIGNATURE, USB_BOT_CSW_STATUS_FAILED, USB_BOT_CSW_STATUS_PASS, USB_CLASS_HID, USB_CLASS_MASS_STORAGE, USB_CONFIG_OFF_CONFIG_VALUE, USB_CONFIG_OFF_TOTAL_LENGTH, USB_DESC_OFF_LENGTH, USB_DESC_OFF_TYPE, USB_DESC_TYPE_ENDPOINT, USB_DESC_TYPE_INTERFACE, USB_EP_ADDR_DIR_MASK, USB_EP_ATTR_TYPE_MASK, USB_EP_OFF_ADDRESS, USB_EP_OFF_ATTRIBUTES, USB_EP_OFF_INTERVAL, USB_EP_OFF_MAX_PACKET_SIZE, USB_EP_TYPE_BULK, USB_EP_TYPE_INTERRUPT, USB_IFACE_OFF_CLASS, USB_IFACE_OFF_NUMBER, USB_IFACE_OFF_PROTOCOL, USB_IFACE_OFF_SUBCLASS, usb_kbd_forget(), usb_kbd_leds_wanted(), usb_kbd_report(), USB_PROTOCOL_BOT, USB_PROTOCOL_HID_KEYBOARD, USB_SUBCLASS_HID_BOOT, USB_SUBCLASS_SCSI, xhci_bot_read_data_in(), xhci_bot_receive_csw(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), XHCI_BOT_TUR_MAX_RETRIES, xhci_bot_write_data_out(), xhci_bulk_purpose_stalled(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), xhci_cmd_configure_hid_endpoint(), xhci_cmd_disable_slot(), xhci_cmd_set_tr_dequeue_pointer(), XHCI_COMPLETION_CODE_SHORT_PACKET, XHCI_COMPLETION_CODE_STALL_ERROR, XHCI_COMPLETION_CODE_SUC xhci_drain_pending_event(), xhci_ep0_bot_mass_storage_reset(), xhci_ep0_clear_endpoint_halt(), xhci_ep0_get_config_descriptor(xhci_ep0_get_device_descriptor(), xhci_ep0_hid_set_leds(), xhci_ep0_hid_set_protocol(), xhci_ep0_set_configuration(), XHCI_EP_ADDR_TO_DCI, XHCI_ERDP_EHB, XHCI_ERDP_PTR_MASK, XHCI_EVT_COMPLETION_CODE, XHCI_EVT_EP_ID, XHCI_EVT_RING_MAX_DRAIN, XHCI_EVT_SLOT_ID, xhci_handle_bulk_stall(), xhci_handle_port_connected() xhci_hid_queue_report(), xhci_log_hex32(), XHCI_MAX_TRACKED_PORTS, xhci_msc_slot_for(), XHCI_NEXT_ACTION_BOT_CSW XHCI_NEXT_ACTION_BOT_DATA_IN, XHCI_NEXT_ACTION_BOT_DATA_OUT, XHCI_NEXT_ACTION_BOT_RESET,

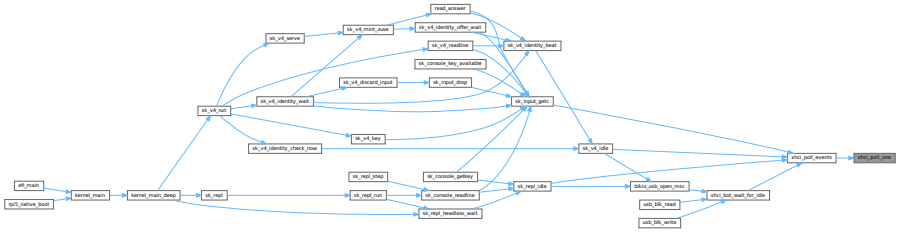
XHCI_NEXT_ACTION_BOT_SEND_READ10, XHCI_NEXT_ACTION_BOT_SEND_READ_CAPACITY10, XHCI_NEXT_ACTION_BO
XHCI_NEXT_ACTION_BOT_SEND_WRITE10, XHCI_NEXT_ACTION_CLEAR_HALT, XHCI_NEXT_ACTION_CONFIGURE_ENDPO
XHCI_NEXT_ACTION_GET_CONFIG_DESC, XHCI_NEXT_ACTION_GET_DEVICE_DESC, XHCI_NEXT_ACTION_HID_POLL,
XHCI_NEXT_ACTION_HID_SET_PROTOCOL, XHCI_NEXT_ACTION_NONE, XHCI_NEXT_ACTION_SET_CONFIG,
xhci_port_regs(), XHCI_PORTSC_CCS, XHCI_PORTSC_CSC, XHCI_PORTSC_PP, XHCI_PORTSC_PRC,
XHCI_PSC_EVT_PORT_ID, xhci_queue_pending_event(), XHCI_RING_TRB_COUNT, xhci_stall_fail(), XHCI_TRB_CONTROL_CYC
XHCI_TRB_TYPE, XHCI_TRB_TYPE_COMMAND_COMPLETION_EVT, XHCI_TRB_TYPE_PORT_STATUS_CHANGE_EVT,
XHCI_TRB_TYPE_TRANSFER_EVENT, XHCI_XFER_BOT_DATA_IN, XHCI_XFER_BOT_DATA_OUT, XHCI_XFER_BOT_RESET,
XHCI_XFER_CBW_SENT, XHCI_XFER_CLEAR_HALT, XHCI_XFER_CONFIG_DESC_FULL, XHCI_XFER_CONFIG_DESC_SHOR
XHCI_XFER_CSW_RECEIVED, XHCI_XFER_DEVICE_DESC, XHCI_XFER_HID_SET_LEDS, XHCI_XFER_HID_SET_PROTOCOL
XHCI_XFER_NONE, and XHCI_XFER_SET_CONFIG.

Referenced by xhci_poll_events().

Here is the call graph for this function:



Here is the caller graph for this function:



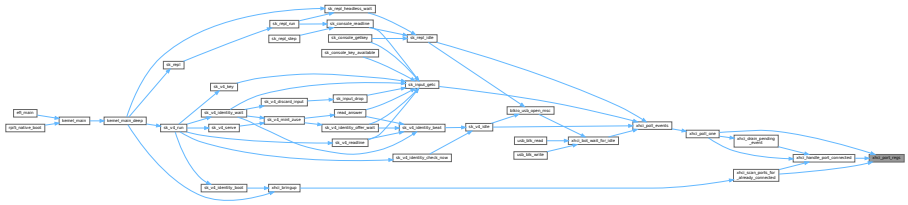
7.172.2.48 xhci_port_regs()

```
static xhci_port_regs_t * xhci_port_regs (
            xhci_dev_t * dev,
            uint32_t port_id )  [static]
```

References xhci_dev_t::max_ports, xhci_dev_t::op, and XHCI_PORT_REGS_OFFSET.

Referenced by xhci_handle_port_connected(), xhci_poll_one(), and xhci_scan_ports_for_already_connected().

Here is the caller graph for this function:



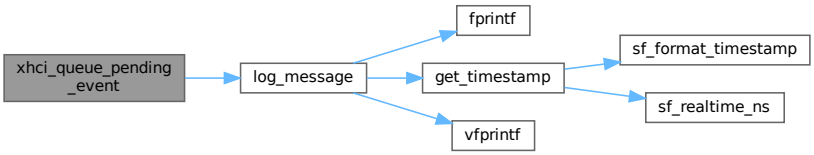
7.172.2.49 xhci_queue_pending_event()

```
static void xhci_queue_pending_event (
            xhci_dev_t * dev,
            uint32_t port_id,
            int is_connect,
            uint32_t portsc,
            uint32_t slot_id )  [static]
```

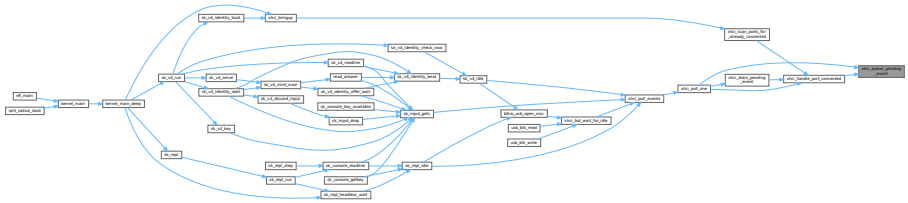
References xhci_dev_t::is_connect, LOG_DEBUG, log_message(), xhci_dev_t::pending_events, xhci_dev_t::port_id, xhci_dev_t::portsc, xhci_dev_t::slot_id, xhci_dev_t::valid, and XHCI_MAX_TRACKED_PORTS.

Referenced by xhci_handle_port_connected(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



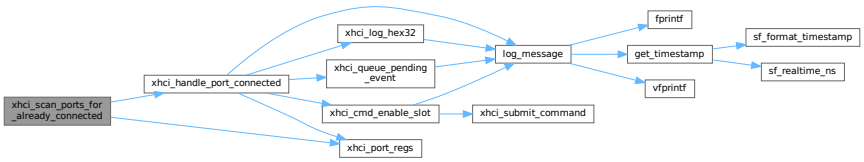
7.172.2.50 xhci_scan_ports_for_already_connected()

```
static void xhci_scan_ports_for_already_connected (
            xhci_dev_t * dev )  [static]
```

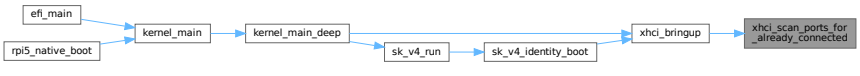
References xhci_dev_t::max_ports, xhci_port_regs_t::portsc, xhci_handle_port_connected(), XHCI_MAX_TRACKED_PORTS, xhci_port_regs(), XHCI_PORTSC_CCS, XHCI_PORTSC_CSC, XHCI_PORTSC_PP, and XHCI_PORTSC_PRC.

Referenced by xhci_bringup().

Here is the call graph for this function:



Here is the caller graph for this function:



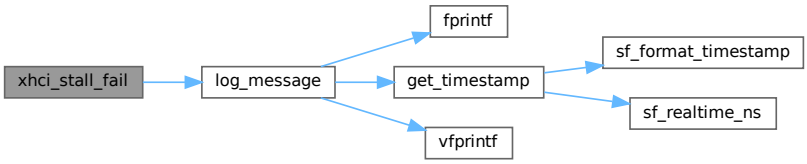
7.172.2.51 xhci_stall_fail()

```
static void xhci_stall_fail (
            xhci_dev_t * dev ) [static]
```

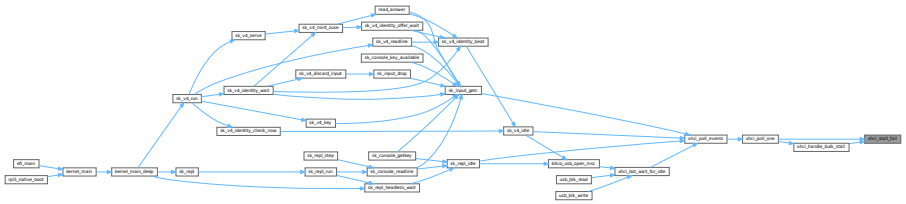
References xhci_dev_t::bot_cmd_kind, xhci_dev_t::bot_last_status, xhci_dev_t::bot_reset_clear_remaining, LOG_ERROR, log_message(), xhci_dev_t::stall_retry_action, and XHCI_NEXT_ACTION_NONE.

Referenced by xhci_handle_bulk_stall(), and xhci_poll_one().

Here is the call graph for this function:



Here is the caller graph for this function:



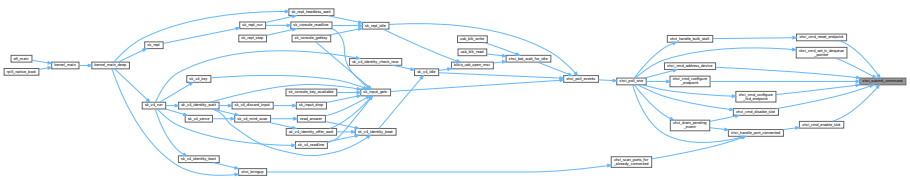
7.172.2.52 xhci_submit_command()

```
static void xhci_submit_command (
            xhci_dev_t * dev,
            uint64_t parameter,
            uint32_t status,
            uint32_t trb_type,
            uint32_t extra_control_bits ) [static]
```

References xhci_dev_t::cmd_ring, xhci_dev_t::cmd_ring_cycle, xhci_dev_t::cmd_ring_enq, xhci_trb_t::control, xhci_dev_t::doorbell, xhci_trb_t::parameter, xhci_trb_t::status, XHCI_DB_TARGET, XHCI_RING_TRB_COUNT, XHCI_TRB_CONTROL_CYCLE, XHCI_TRB_CONTROL_TC, XHCI_TRB_CONTROL_TYPE_SHIFT, and XHCI_TRB_TYPE_LINK.

Referenced by xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), xhci_cmd_configure_hid_endpoint(), xhci_cmd_disable_slot(), xhci_cmd_enable_slot(), xhci_cmd_reset_endpoint(), and xhci_cmd_set_tr_dequeue_pointer().

Here is the caller graph for this function:



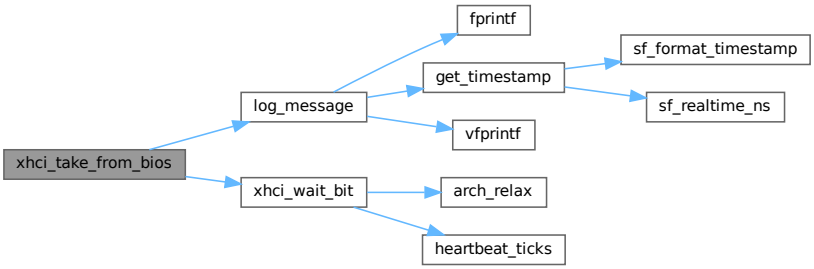
7.172.2.53 xhci_take_from_bios()

```
static void xhci_take_from_bios (
            xhci_dev_t * dev )  [static]
```

References xhci_dev_t::cap, xhci_cap_regs_t::hcc_params1, LOG_DEBUG, log_message(), LOG_WARN, XHCI_LEGSUP_BIOS_OWNED, XHCI_LEGSUP_OS_OWNED, xhci_wait_bit(), and XHCI_XCAP_LEGACY.

Referenced by xhci_find_and_map().

Here is the call graph for this function:



Here is the caller graph for this function:

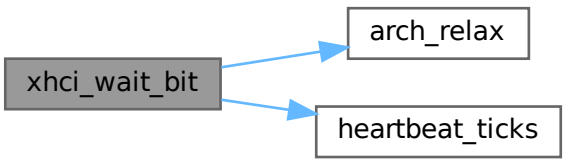
7.172.2.54 xhci_wait_bit()

```
static int xhci_wait_bit (
            volatile uint32_t * reg,
            uint32_t mask,
            int want_set,
            uint64_t max_ticks )  [static]
```

References arch_relax(), and heartbeat_ticks().

Referenced by xhci_bringup(), and xhci_take_from_bios().

Here is the call graph for this function:



Here is the caller graph for this function:

7.172.3 Variable Documentation

7.172.3.1 g_xhci_dev_count

```
uint32_t g_xhci_dev_count  [static]
```

Referenced by xhci_bringup(), xhci_dev_at(), xhci_dev_count(), and xhci_poll_events().

7.172.3.2 g_xhci_devs

```
xhci_dev_t* g_xhci_devs[8u]  [static]
```

Referenced by xhci_bringup(), xhci_dev_at(), and xhci_poll_events().

7.172.3.3 g_xhci_keyboards

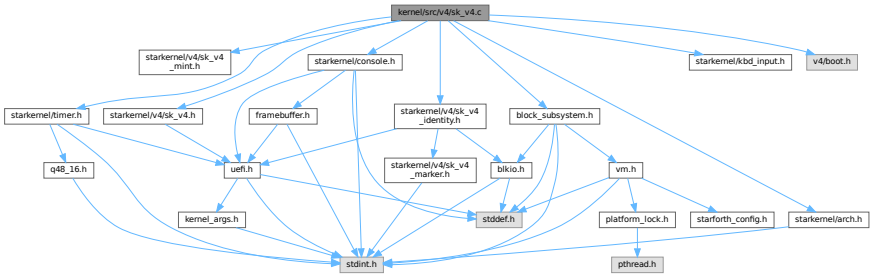
```
uint32_t g_xhci_keyboards  [static]
```

Referenced by xhci_keyboard_running(), and xhci_poll_one().

7.173 kernel/src/v4/sk_v4.c File Reference

```
#include "starkernel/v4/sk_v4_mint.h"
#include "starkernel/v4/sk_v4.h"
#include "starkernel/v4/sk_v4_identity.h"
#include "starkernel/kbd_input.h"
#include "starkernel/console.h"
#include "starkernel/arch.h"
#include "starkernel/timer.h"
#include "v4/boot.h"
#include "block_subsystem.h"
```

Include dependency graph for sk_v4.c:



Macros

- #define SK_V4_LINE 1024u /∗ what the line editor here takes: a line, as the boot takes one (v4/system/boot.c) ∗/
- #define SK_V4_PROMPT "\x1b[38;2;85;255;255mok> \x1b[39m"
- #define SK_V4_CURSOR_BLINK 50u /∗ ticks between blinks, as v3's (500 ms at 100 Hz) ∗/

Functions

- static void sk_v4_out (const char ∗text, unsigned len)
- static int sk_v4_key (void)
- static void sk_v4_serve (v4_node ∗node, unsigned request)
- static int sk_v4_readline (void)
- void sk_v4_run (void(∗chain)(void), const BootDisk ∗boot_disk)

Variables

- static v4_node sk_v4_node
- static v4_exec_state sk_v4_es
- static v4_heat sk_v4_heat
- static v4_image sk_v4_image
- static char sk_v4_text [1024u+1u]
- static uint8_t sk_v4_post_blk_ram [2080u ∗1024u]
- static const char ∗const sk_v4_words [ ] = { "BYE", "MINT-ZUSE" }

7.173.1 Macro Definition Documentation

7.173.1.1 SK_V4_CURSOR_BLINK

```
#define SK_V4_CURSOR_BLINK 50u /* ticks between blinks, as v3's (500 ms at 100 Hz) */
```

7.173.1.2 SK_V4_LINE

```
#define SK_V4_LINE 1024u /* what the line editor here takes:  a line, as the boot takes one
(v4/system/boot.c) */
```

7.173.1.3 SK_V4_PROMPT

```
#define SK_V4_PROMPT "\x1b[38;2;85;255;255mok> \x1b[39m"
```

7.173.2 Function Documentation

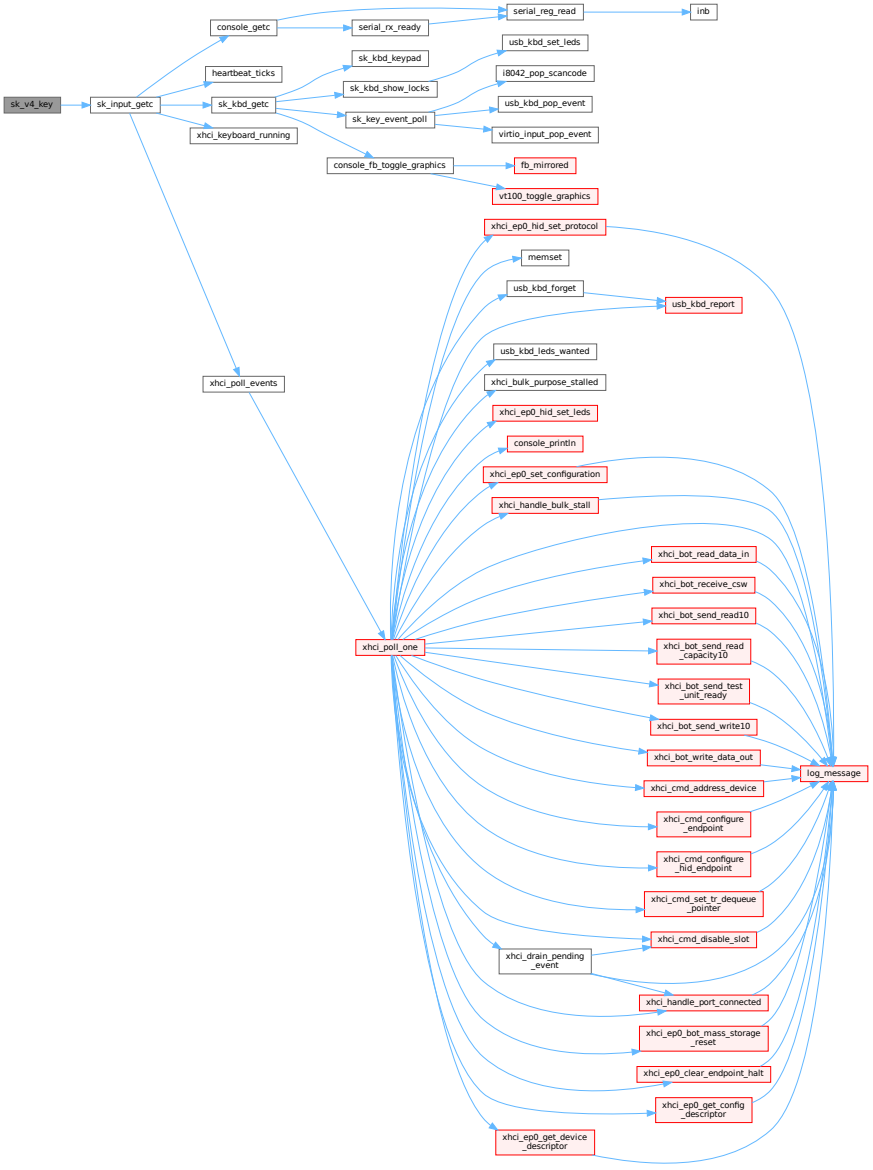
7.173.2.1 sk_v4_key()

```
static int sk_v4_key (
            void  )  [static]
```

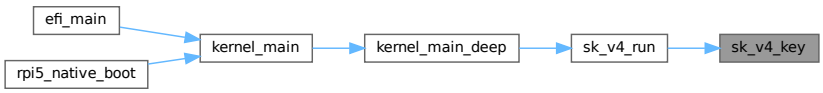
References sk_input_getc().

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



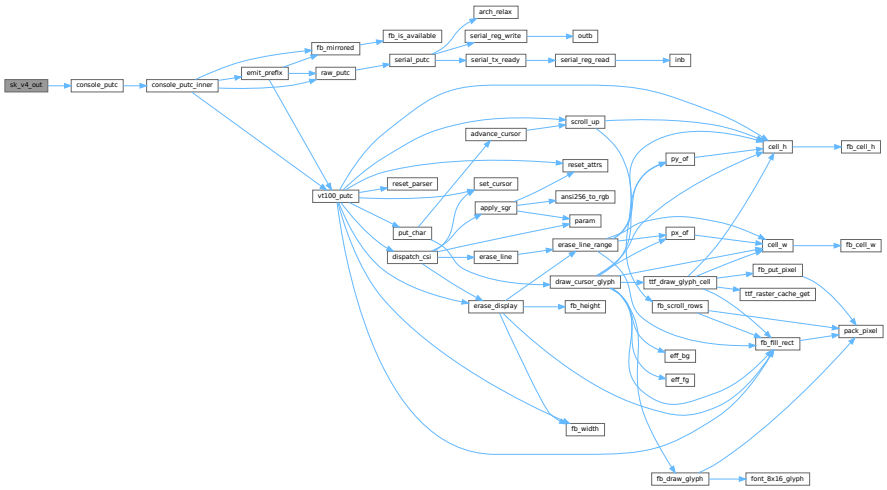
7.173.2.2 sk_v4_out()

```
static void sk_v4_out (
            const char * text,
            unsigned len )  [static]
```

References console_putc().

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:

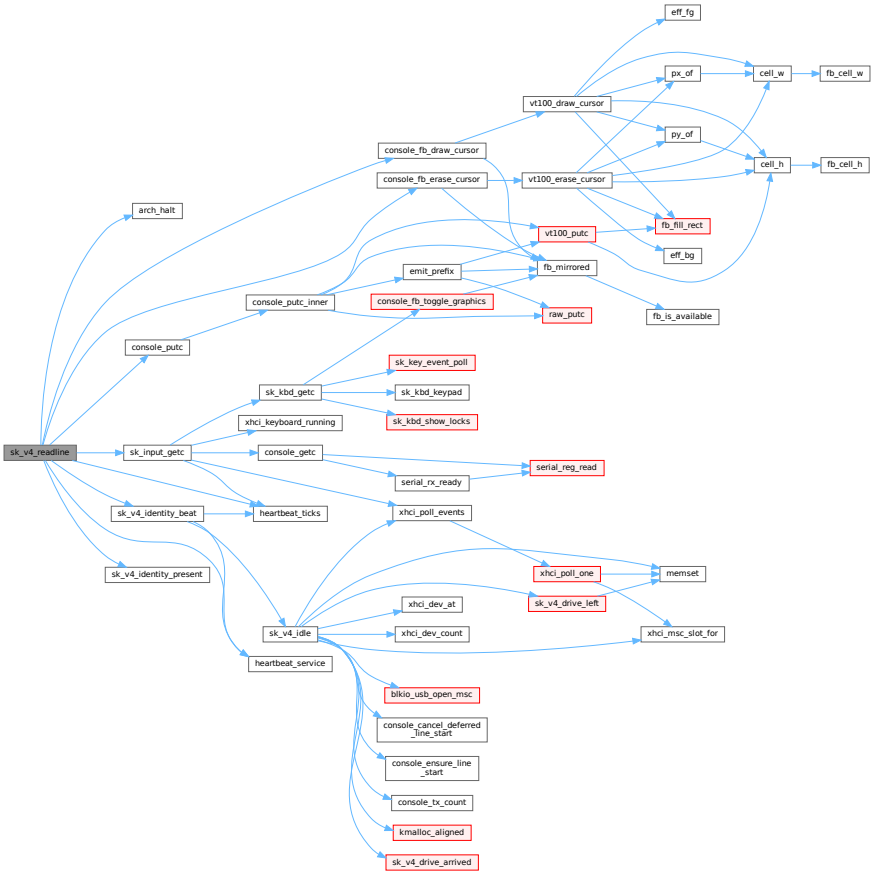
7.173.2.3 sk_v4_readline()

```
static int sk_v4_readline (
            void  )  [static]
```

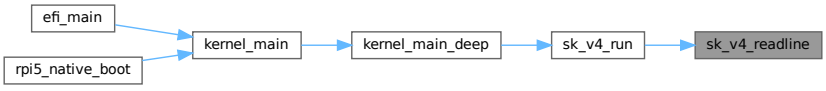
References arch_halt(), console_fb_draw_cursor(), console_fb_erase_cursor(), console_putc(), heartbeat_service(), heartbeat_ticks(), sk_input_getc(), SK_V4_CURSOR_BLINK, sk_v4_identity_beat(), sk_v4_identity_present(), SK_V4_LINE, and sk_v4_text.

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



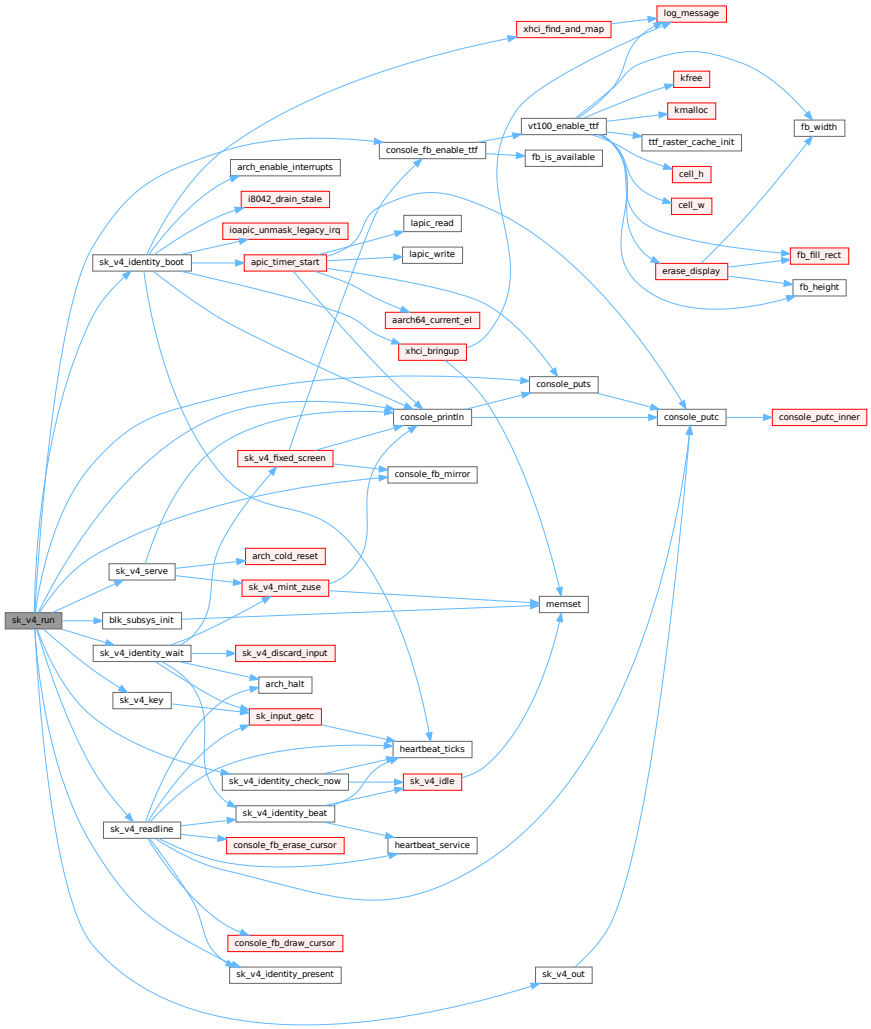
7.173.2.4 sk_v4_run()

```
void sk_v4_run (
            void(*)(void) chain,
            const BootDisk * boot_disk )
```

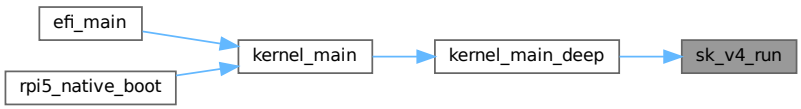
References blk_subsys_init(), console_fb_enable_ttf(), console_fb_mirror(), console_println(), console_puts(), sk_v4_es, sk_v4_heat, sk_v4_identity_boot(), sk_v4_identity_check_now(), sk_v4_identity_present(), sk_v4_identity_wait(), sk_v4_image, sk_v4_key(), sk_v4_node, sk_v4_out(), sk_v4_post_blk_ram, SK_V4_PROMPT, sk_v4_readline(), sk_v4_serve(), sk_v4_text, and sk_v4_words.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



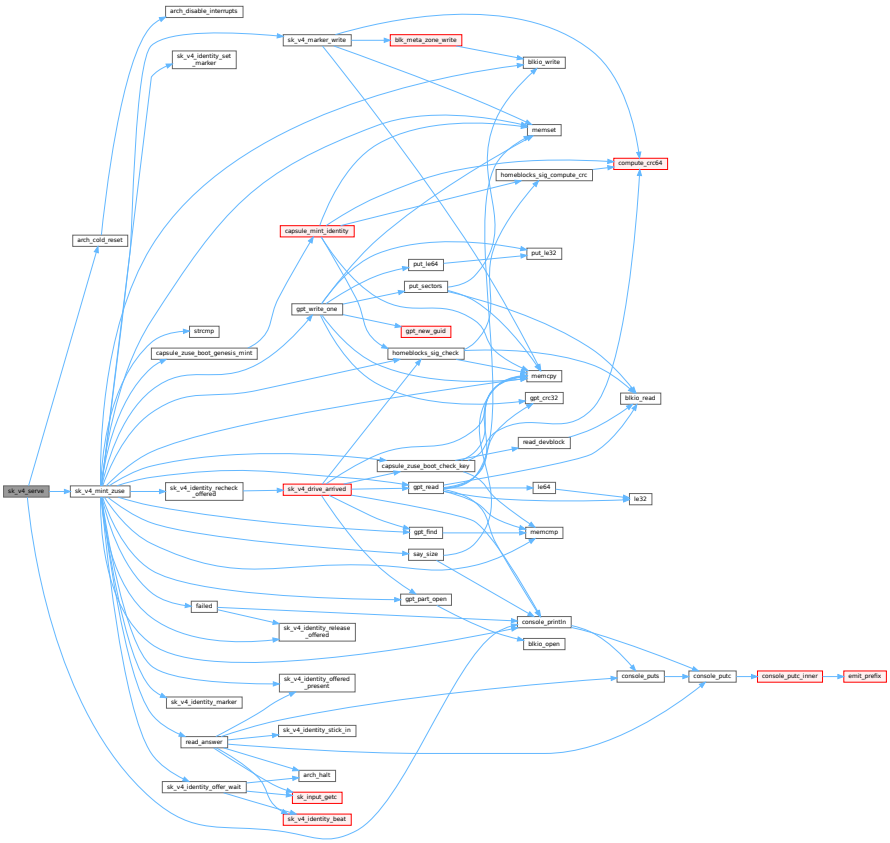
7.173.2.5 sk_v4_serve()

```
static void sk_v4_serve (
            v4_node * node,
            unsigned request )  [static]
```

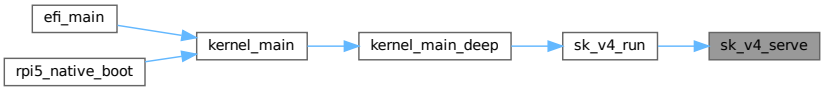
References arch_cold_reset(), console_println(), and sk_v4_mint_zuse().

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.173.3 Variable Documentation

7.173.3.1 sk_v4_es

```
v4_exec_state sk_v4_es  [static]
```

Referenced by sk_v4_run().

7.173.3.2 sk_v4_heat

```
v4_heat sk_v4_heat  [static]
```

Referenced by sk_v4_run().

7.173.3.3 sk_v4_image

```
v4_image sk_v4_image  [static]
```

Referenced by sk_v4_run().

7.173.3.4 sk_v4_node

```
v4_node sk_v4_node  [static]
```

Referenced by sk_v4_run().

7.173.3.5 sk_v4_post_blk_ram

```
uint8_t sk_v4_post_blk_ram[2080u * 1024u]  [static]
```

Referenced by sk_v4_run().

7.173.3.6 sk_v4_text

```
char sk_v4_text[1024u+1u]  [static]
```

Referenced by sk_v4_readline(), and sk_v4_run().

7.173.3.7 sk_v4_words

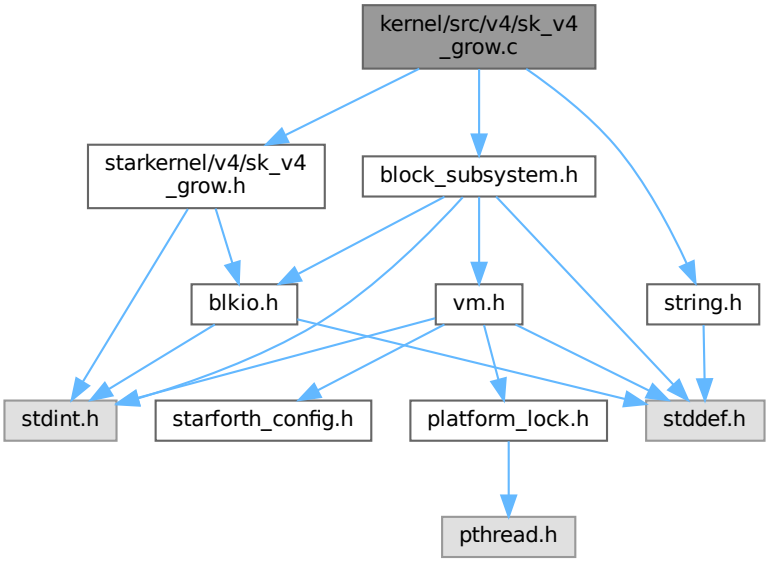
```
const char* const sk_v4_words[] = { "BYE", "MINT-ZUSE" }  [static]
```

Referenced by sk_v4_run().

7.174 kernel/src/v4/sk_v4_grow.c File Reference

```
#include "starkernel/v4/sk_v4_grow.h"
#include "block_subsystem.h"
#include <string.h>
```

Include dependency graph for sk_v4_grow.c:



Functions

- static int dev_rw4k (blkio_dev_t *dev, uint64_t devblock, uint8_t *buf, int write)
- static int read_header (blkio_dev_t *art)
- static uint64_t bam_cover_devblocks (const blk_volume_meta_t *m)
- uint64_t sk_v4_grow_cap_sectors (blkio_dev_t *art, uint64_t cap_sectors)
- int sk_v4_grow_artemis (blkio_dev_t *art)

Variables

- union {
    uint8_t bytes [4096]
    blk_volume_meta_t meta
  } g_hdr

- static uint8_t g_move [4096]

7.174.1 Function Documentation

7.174.1.1 bam_cover_devblocks()

```
static uint64_t bam_cover_devblocks (
            const blk_volume_meta_t * m )  [static]
```

References blk_volume_meta_t::bam_devblocks, blk_volume_meta_t::meta_fence_blocks, and blk_volume_meta_t::reloc_devblocks.

Referenced by sk_v4_grow_artemis(), and sk_v4_grow_cap_sectors().

Here is the caller graph for this function:

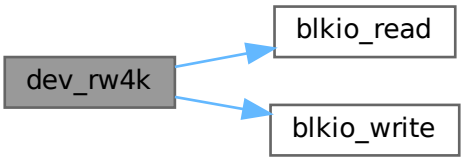
7.174.1.2 dev_rw4k()

```
static int dev_rw4k (
            blkio_dev_t * dev,
            uint64_t devblock,
            uint8_t * buf,
            int write )  [static]
```

References base, BLKIO_OK, blkio_read(), and blkio_write().

Referenced by read_header(), and sk_v4_grow_artemis().

Here is the call graph for this function:



Here is the caller graph for this function:

7.174.1.3 read_header()

```
static int read_header (
            blkio_dev_t * art )  [static]
```

References dev_rw4k(), and g_hdr.

Referenced by sk_v4_grow_artemis(), and sk_v4_grow_cap_sectors().

Here is the call graph for this function:

Here is the caller graph for this function:

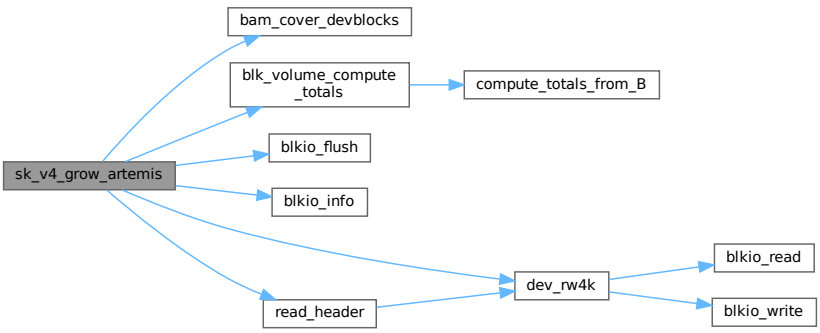
7.174.1.4 sk_v4_grow_artemis()

```
int sk_v4_grow_artemis (
            blkio_dev_t * art )
```

References bam_cover_devblocks(), blk_volume_compute_totals(), blkio_flush(), blkio_info(), BLKIO_OK, dev_rw4k(), g_hdr, g_move, read_header(), and blkio_info_t::total_blocks.

Referenced by sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

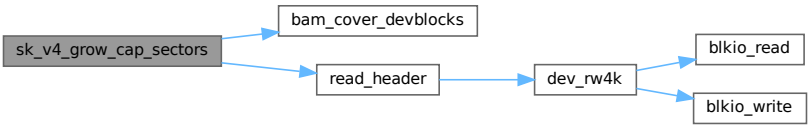
7.174.1.5 sk_v4_grow_cap_sectors()

```
uint64_t sk_v4_grow_cap_sectors (
            blkio_dev_t * art,
            uint64_t cap_sectors )
```

References bam_cover_devblocks(), g_hdr, and read_header().

Referenced by sk_v4_grow_stick().

Here is the call graph for this function:



Here is the caller graph for this function:

7.174.2 Variable Documentation

7.174.2.1 bytes

```
uint8_t bytes[4096]
```

Referenced by analytics_heap_allocate(), array_ptr(), hosted_xt_entry_owned(), pmm_bytes_to_pages(), profiler_memory_alloc(), profiler_memory_read(), profiler_memory_write(), say_size(), sk_repl_idle(), sk_v4_identity_offer_wait(), sk_v4_idle(), sk_v4_mint_zuse(), usb_blk_state_for(), vm_align(), and vm_allot().

7.174.2.2 [union]

```
union { ... } g_hdr [static]
```

Referenced by read_header(), sk_v4_grow_artemis(), and sk_v4_grow_cap_sectors().

7.174.2.3 g_move

```
uint8_t g_move[4096] [static]
```

Referenced by sk_v4_grow_artemis().

7.174.2.4 meta

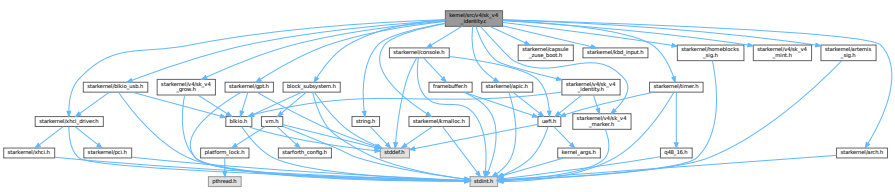
```
blk_volume_meta_t meta
```

Referenced by blk_acl_allow_get(), blk_acl_allow_set(), blk_acl_ttl_get(), blk_acl_ttl_set(), blk_firsttouch_claim(), blk_flags_get(), blk_flags_set(), blk_get_meta(), blk_get_volume_meta(), blk_migration_idle_check_slot(), blk_owner_fp_get(), blk_owner_fp_set(), blk_owner_has_claim(), blk_set_meta(), blk_set_volume_meta(), and devblock_claimed_by_lbn().

7.175 kernel/src/v4/sk_v4_identity.c File Reference

```
#include "starkernel/v4/sk_v4_identity.h"
#include "starkernel/console.h"
#include "starkernel/arch.h"
#include "starkernel/apic.h"
#include "starkernel/timer.h"
#include "starkernel/kmalloc.h"
#include "starkernel/xhci_driver.h"
#include "starkernel/blkio_usb.h"
#include "starkernel/homeblocks_sig.h"
#include "starkernel/artemis_sig.h"
#include "starkernel/capsule_zuse_boot.h"
#include "starkernel/kbd_input.h"
#include "starkernel/gpt.h"
#include "starkernel/v4/sk_v4_marker.h"
#include "starkernel/v4/sk_v4_grow.h"
#include "starkernel/v4/sk_v4_mint.h"
#include "block_subsystem.h"
#include <string.h>
```

Include dependency graph for sk_v4_identity.c:



Data Structures

- struct sk_v4_drive_t

Macros

- #define SK_V4_IDLE_BEAT 100u /∗ ticks between idle beats, v3's SK_IDLE_BEAT_INTERVAL (1 s at 100 Hz) ∗/
- #define SK_V4_GPT_PARTS 16u
- #define SK_V4_GROW_MIN_BYTES STICK_GROW_MIN_BYTES
- #define SK_V4_CAP_SECTORS (STICK_ARTEMIS_CAP_GIB ∗ 2097152ULL)

Functions

- static const char ∗ sk_v4_user_prefix (void)
- int sk_v4_identity_present (void)
- void sk_v4_identity_boot (const BootDisk ∗boot_disk)
- static int sk_v4_is_this_stick (unsigned nparts)
- static int sk_v4_grow_stick (sk_v4_drive_t ∗d, unsigned ∗nparts)
- static void sk_v4_login (blkio_dev_t ∗dev, const uint8_t seed[32], const uint8_t pubkey[32], const char ∗line)
- static int sk_v4_genesis (blkio_dev_t ∗dev, blkio_dev_t ∗art)
- static int sk_v4_real_zuse_arrived (sk_v4_drive_t ∗d, unsigned nparts)
- static int sk_v4_drive_arrived (sk_v4_drive_t ∗d)
- static void sk_v4_session_end (void)
- static void sk_v4_drive_left (sk_v4_drive_t ∗d)
- static void sk_v4_idle (void)
- void sk_v4_identity_beat (void)
- void sk_v4_identity_check_now (void)
- int sk_v4_identity_offer_wait (blkio_dev_t ∗∗raw, uint64_t ∗bytes)
- int sk_v4_identity_stick_in (void)
- int sk_v4_identity_offered_present (void)
- void sk_v4_identity_release_offered (void)
- const sk_v4_marker_t ∗ sk_v4_identity_marker (void)
- void sk_v4_identity_set_marker (const sk_v4_marker_t ∗m)
- void sk_v4_identity_recheck_offered (void)
- static void sk_v4_discard_input (void)
- static void sk_v4_fixed_screen (void)
- void sk_v4_identity_wait (void)

Variables

- static int g_zuse_session
- static int g_zuse_cert_installed
- static uint8_t g_zuse_seed [32]
- static uint8_t g_zuse_pubkey [32]
- static uint8_t g_zuse_root_pubkey [32]
- static int g_zuse_root_known
- static struct blkio_dev ∗ g_zuse_dev
- static sk_v4_marker_t g_marker
- static int g_proxy_fallback

- struct {
  sk_v4_drive_t * slots
  uint32_t count
  } g_usb [8u]

- static gpt_part_t g_gpt_parts [16u]
- static uint64_t g_last_beat
- static int g_mint_waiting
- static sk_v4_drive_t * g_offered
- static sk_v4_drive_t * g_spare
- static sk_v4_drive_t * g_pending_real
- static sk_v4_drive_t * g_real_drive
- static const BootDisk * g_boot_disk
- static int g_stick_in

7.175.1 Macro Definition Documentation

7.175.1.1 SK_V4_CAP_SECTORS

```
#define SK_V4_CAP_SECTORS (STICK_ARTEMIS_CAP_GIB * 2097152ULL)
```

7.175.1.2 SK_V4_GPT_PARTS

```
#define SK_V4_GPT_PARTS 16u
```

7.175.1.3 SK_V4_GROW_MIN_BYTES

```
#define SK_V4_GROW_MIN_BYTES STICK_GROW_MIN_BYTES
```

7.175.1.4 SK_V4_IDLE_BEAT

```
#define SK_V4_IDLE_BEAT 100u /* ticks between idle beats, v3's SK_IDLE_BEAT_INTERVAL (1 s at
100 Hz) */
```

7.175.2 Function Documentation

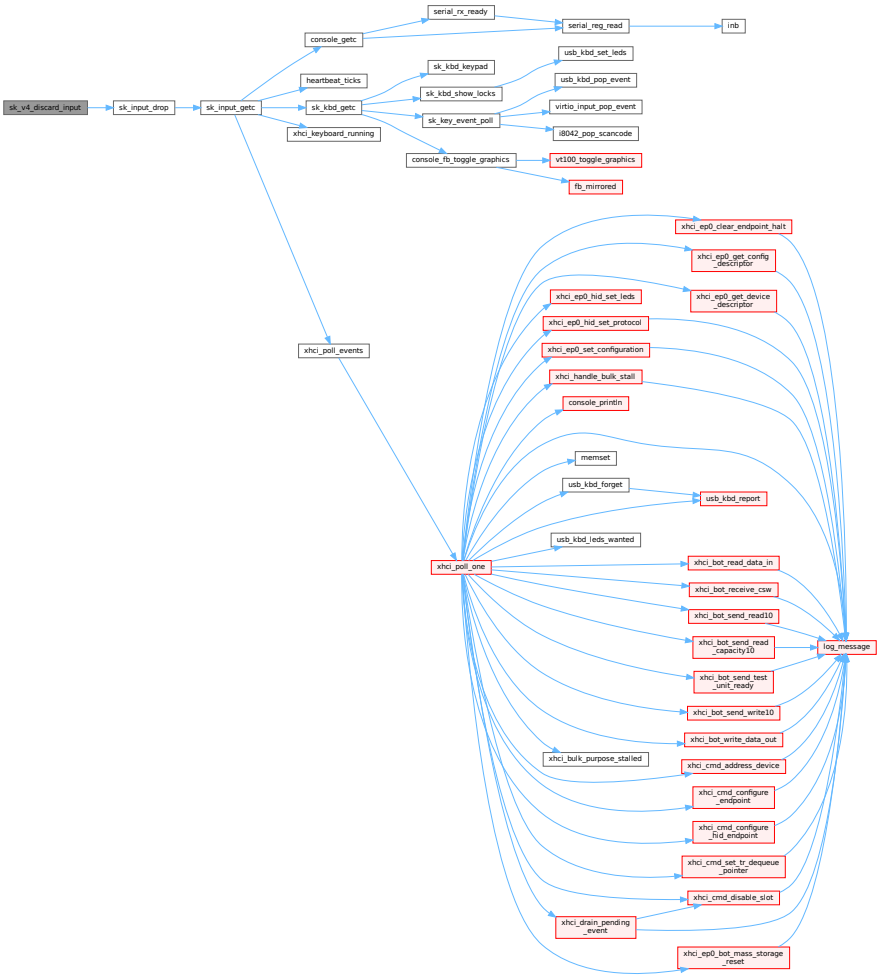
7.175.2.1 sk_v4_discard_input()

```
static void sk_v4_discard_input (
            void  )  [static]
```

References sk_input_drop().

Referenced by sk_v4_identity_wait().

Here is the call graph for this function:



Here is the caller graph for this function:

7.175.2.2 sk_v4_drive_arrived()

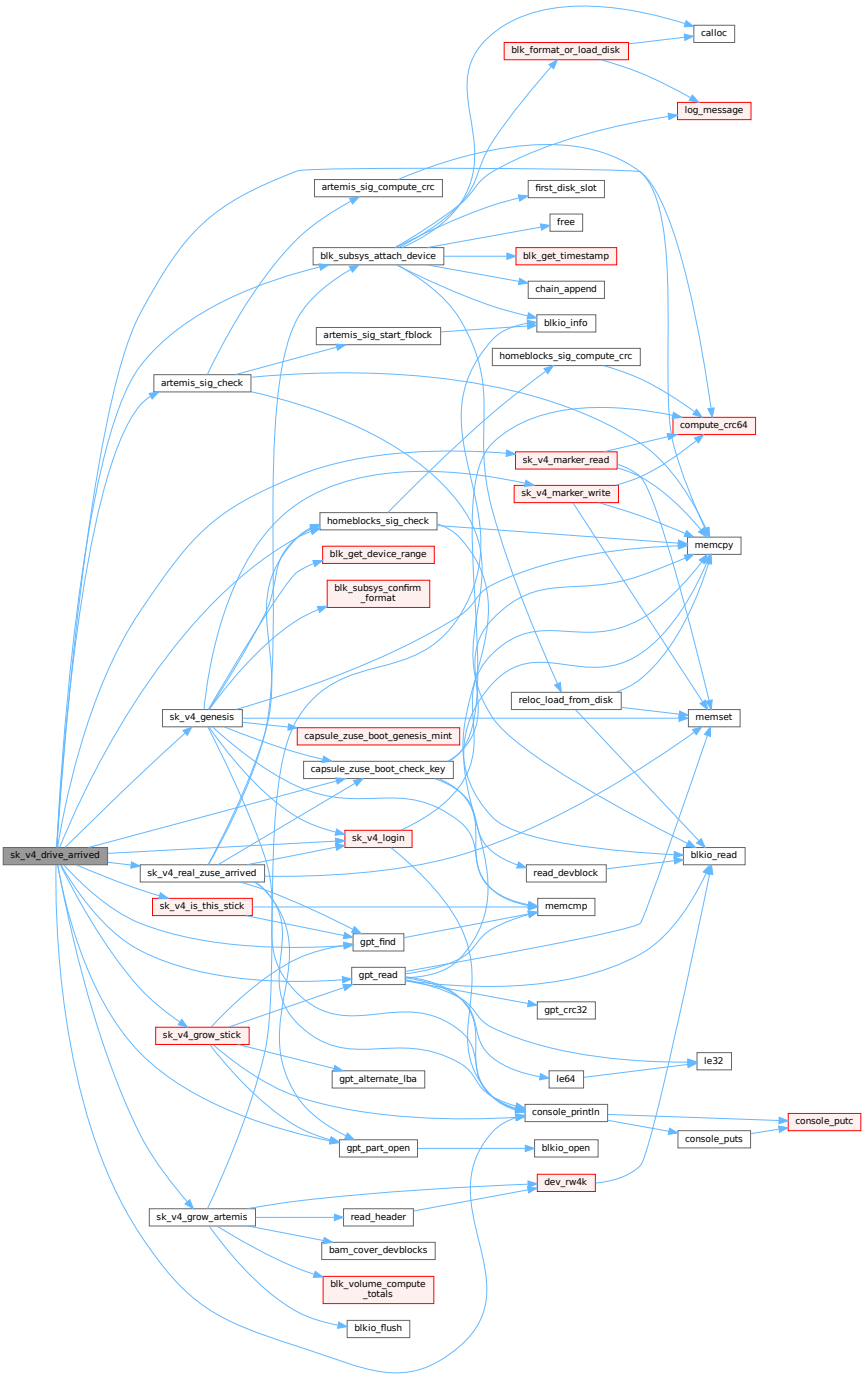
```
static int sk_v4_drive_arrived (
            sk_v4_drive_t * d )  [static]
```

References sk_v4_drive_t::art, sk_v4_drive_t::art_state, sk_v4_drive_t::art_used, artemis_sig_check(), ARTEMIS_SIG_OK, BLK_OK, blk_subsys_attach_device(), BLKIO_OK, capsule_zuse_boot_check_key(), console_println(), g, g_gpt_parts, g_marker, g_mint_waiting, g_offered, g_pending_real, g_proxy_fallback, g_spare, g_stick_in, g_zuse_root_known, g_zuse_root_pubkey, g_zuse_session, GPT_BAD, GPT_EIO, gpt_find(), GPT_OK, gpt_part_open(), gpt_read(), gpt_type_starship_artemis, gpt_type_starship_identity, sk_v4_marker_t::has_real,

HOMEBLOCKS_SIG_BLANK, homeblocks_sig_check(), HOMEBLOCKS_SIG_START_FBLOCK, memcpy(), sk_v4_drive_t::part, sk_v4_drive_t::part_state, sk_v4_marker_t::present, sk_v4_marker_t::proxy, sk_v4_drive_t::raw, sk_v4_genesis(), SK_V4_GPT_PARTS, sk_v4_grow_artemis(), sk_v4_grow_stick(), sk_v4_is_this_stick(), sk_v4_login(), sk_v4_marker_read(), sk_v4_real_zuse_arrived(), and sk_v4_drive_t::used.

Referenced by sk_v4_identity_recheck_offered(), and sk_v4_idle().

Here is the call graph for this function:



Here is the caller graph for this function:

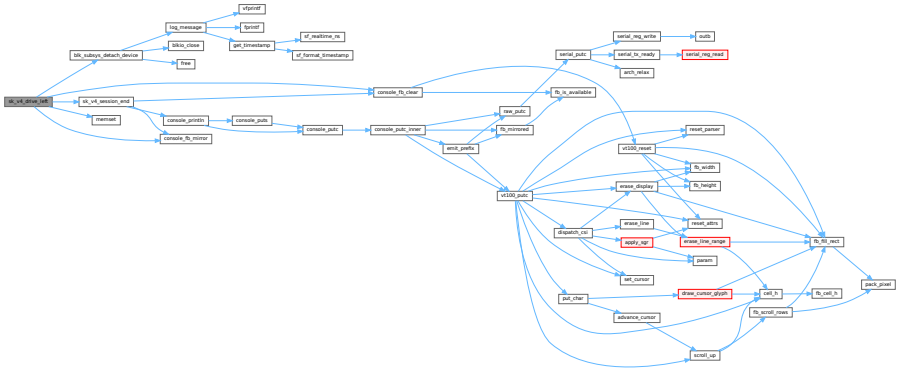
7.175.2.3 sk_v4_drive_left()

```
static void sk_v4_drive_left (
            sk_v4_drive_t * d )  [static]
```

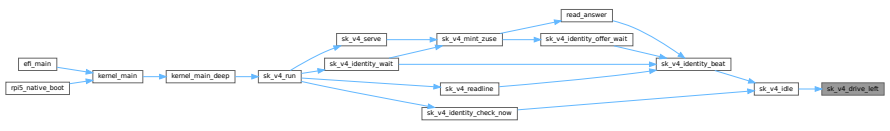
References sk_v4_drive_t::art_used, blk_subsys_detach_device(), console_fb_clear(), console_fb_mirror(), g_marker, g_offered, g_pending_real, g_proxy_fallback, g_real_drive, g_spare, g_stick_in, g_zuse_dev, sk_v4_marker_t::has_real, memset(), sk_v4_marker_t::present, sk_v4_session_end(), and sk_v4_drive_t::used.

Referenced by sk_v4_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



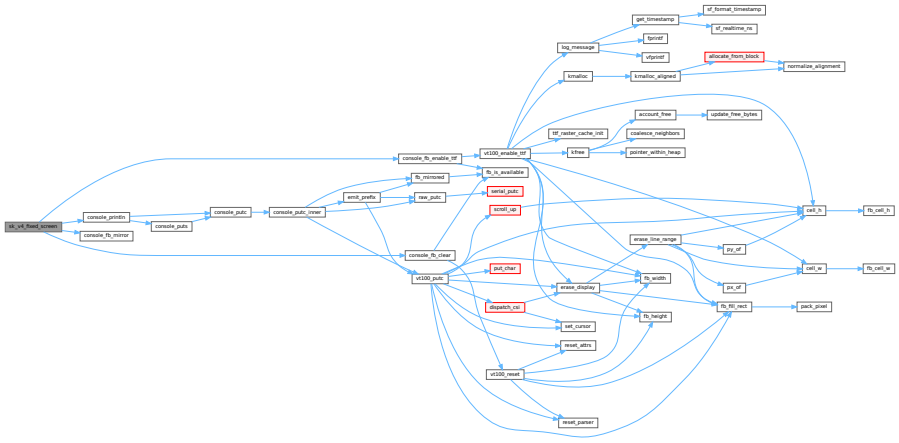
7.175.2.4 sk_v4_fixed_screen()

```
static void sk_v4_fixed_screen (
            void  )  [static]
```

References console_fb_clear(), console_fb_enable_ttf(), console_fb_mirror(), and console_println().

Referenced by sk_v4_identity_wait().

Here is the call graph for this function:



Here is the caller graph for this function:

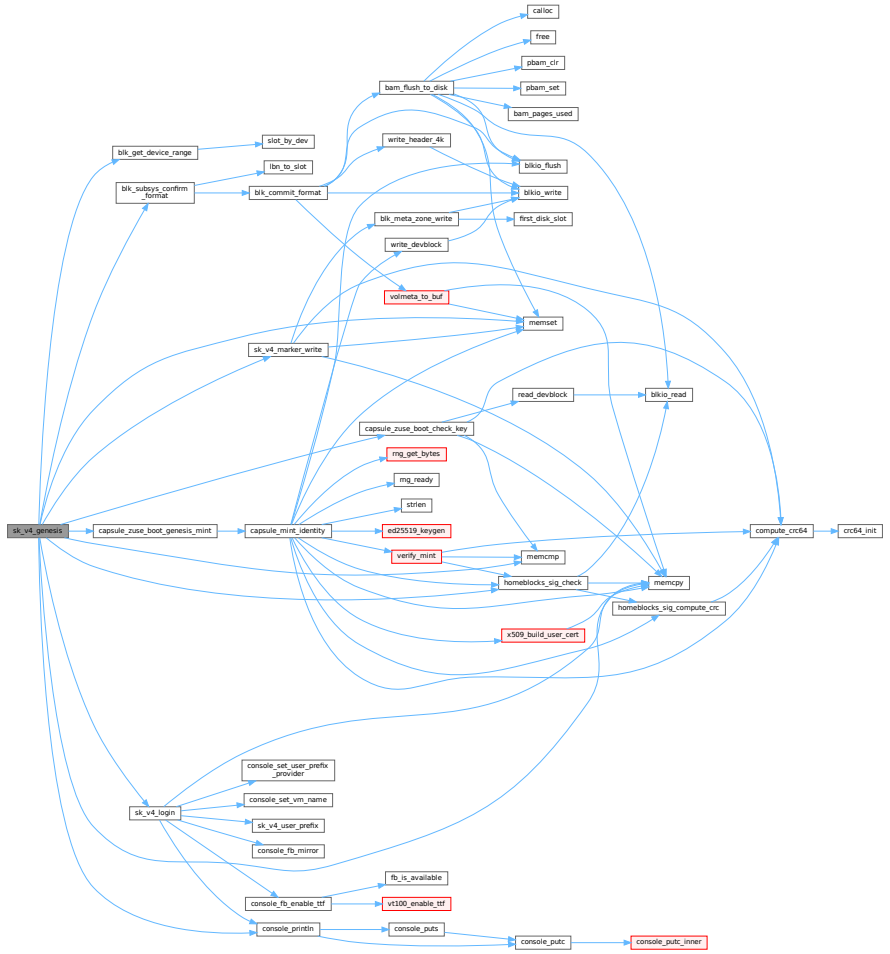
7.175.2.5 sk_v4_genesis()

```
static int sk_v4_genesis (
            blkio_dev_t * dev,
            blkio_dev_t * art )  [static]
```

References blk_get_device_range(), BLK_OK, blk_subsys_confirm_format(), capsule_zuse_boot_check_key(), capsule_zuse_boot_genesis_mint(), console_println(), g_marker, homeblocks_sig_check(), HOMEBLOCKS_SIG_START_FBLOCK, memcmp(), memcpy(), memset(), sk_v4_marker_t::present, sk_v4_marker_t::proxy, sk_v4_login(), and sk_v4_marker_write().

Referenced by sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

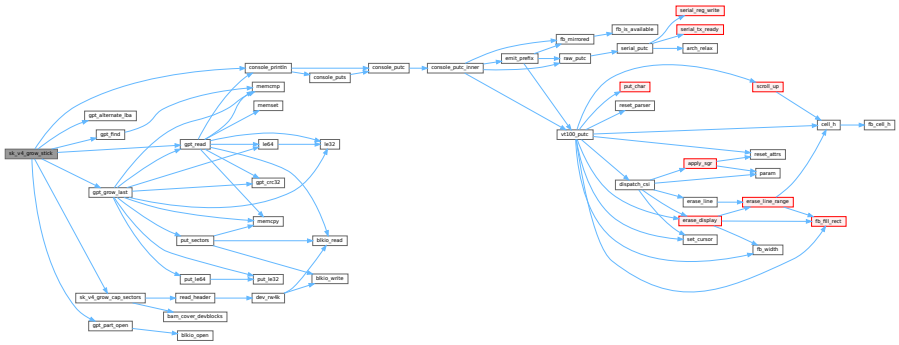
7.175.2.6 sk_v4_grow_stick()

```
static int sk_v4_grow_stick (
            sk_v4_drive_t * d,
            unsigned * nparts )  [static]
```

References sk_v4_drive_t::art, sk_v4_drive_t::art_state, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, console_println(), gpt_part_t::first_lba, g_gpt_parts, gpt_alternate_lba(), gpt_find(), gpt_grow_last(), GPT_OK, gpt_part_open(), gpt_read(), gpt_type_starship_artemis, gpt_type_starship_identity, sk_v4_drive_t::raw, SK_V4_CAP_SECTORS, SK_V4_GPT_PARTS, sk_v4_grow_cap_sectors(), SK_V4_GROW_MIN_BYTES, and blkio_dev::total_blocks.

Referenced by sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

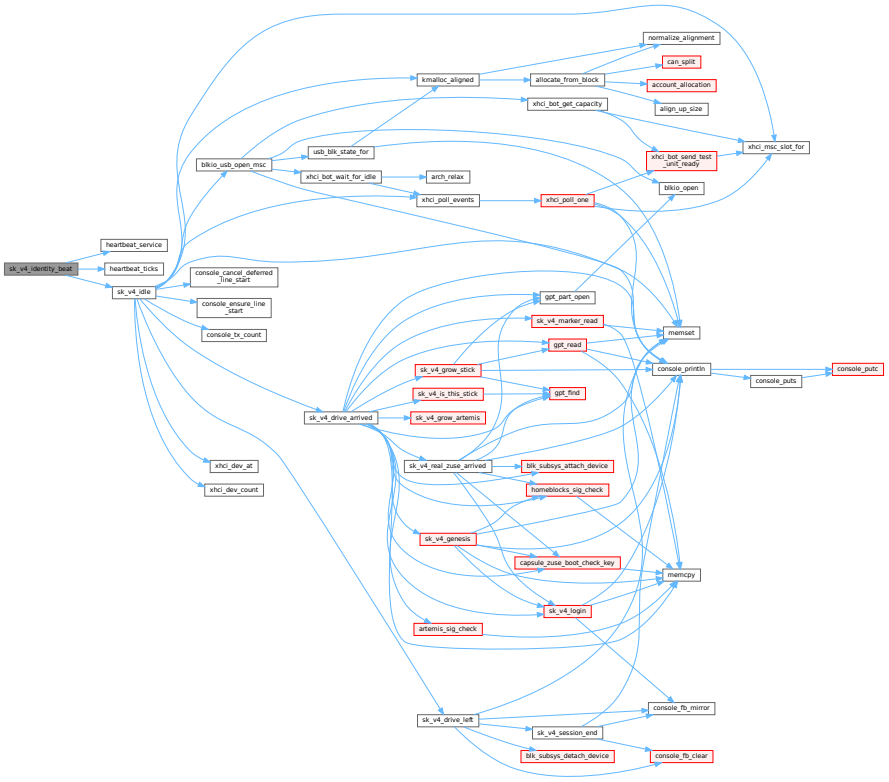
7.175.2.7 sk_v4_identity_beat()

```
void sk_v4_identity_beat (
            void  )
```

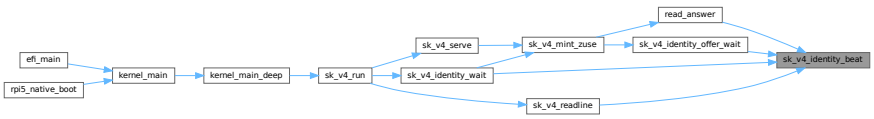
References g_last_beat, heartbeat_service(), heartbeat_ticks(), sk_v4_idle(), and SK_V4_IDLE_BEAT.

Referenced by read_answer(), sk_v4_identity_offer_wait(), sk_v4_identity_wait(), and sk_v4_readline().

Here is the call graph for this function:



Here is the caller graph for this function:



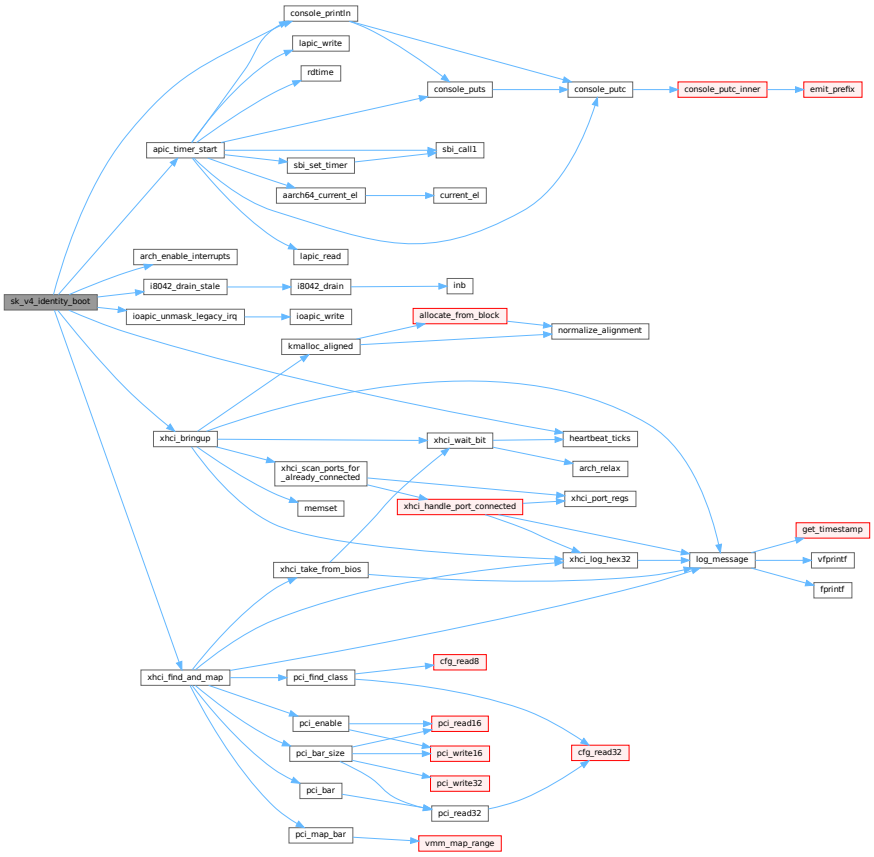
7.175.2.8 sk_v4_identity_boot()

```
void sk_v4_identity_boot (
            const BootDisk * boot_disk )
```

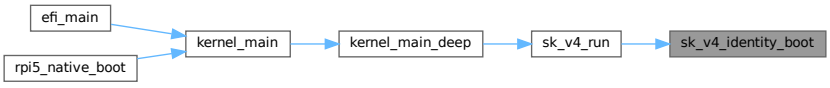
References apic_timer_start(), arch_enable_interrupts(), console_println(), g_boot_disk, g_last_beat, heartbeat_ticks(), i8042_drain_stale(), ioapic_unmask_legacy_irq(), xhci_bringup(), xhci_find_and_map(), and XHCI_MAX_CONTROLLERS.

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



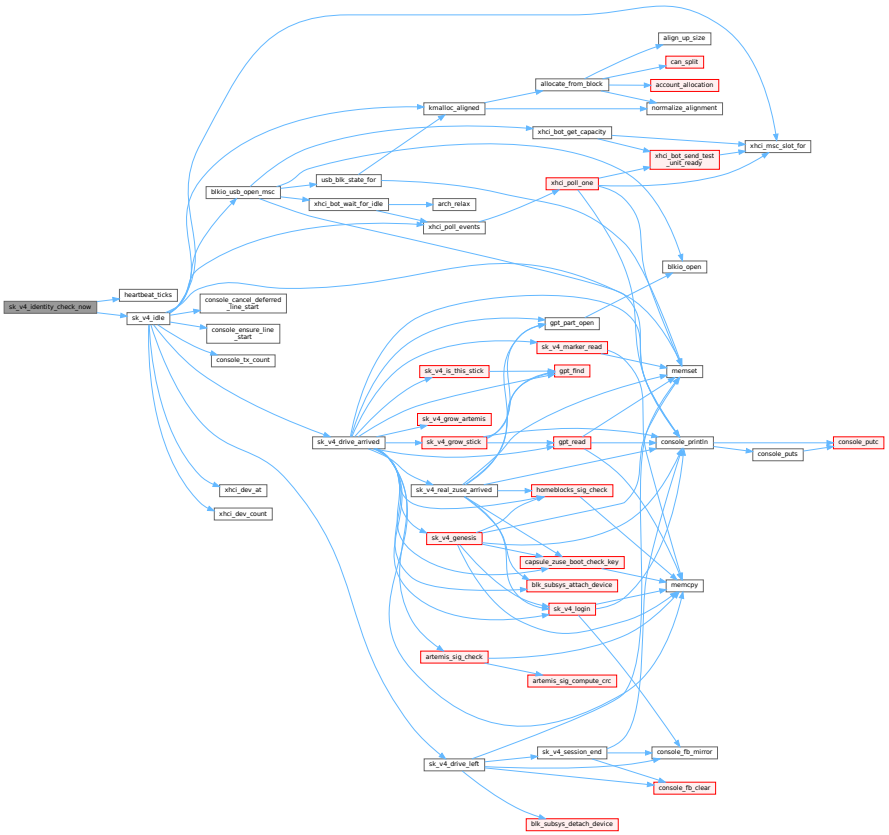
7.175.2.9 sk_v4_identity_check_now()

```
void sk_v4_identity_check_now (
            void  )
```

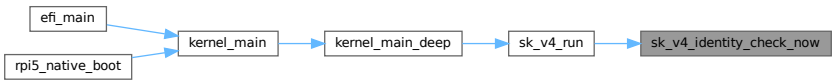
References g_last_beat, heartbeat_ticks(), and sk_v4_idle().

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.175.2.10 sk_v4_identity_marker()

```
const sk_v4_marker_t * sk_v4_identity_marker (
            void  )
```

References g_marker.

Referenced by sk_v4_mint_zuse().

Here is the caller graph for this function:

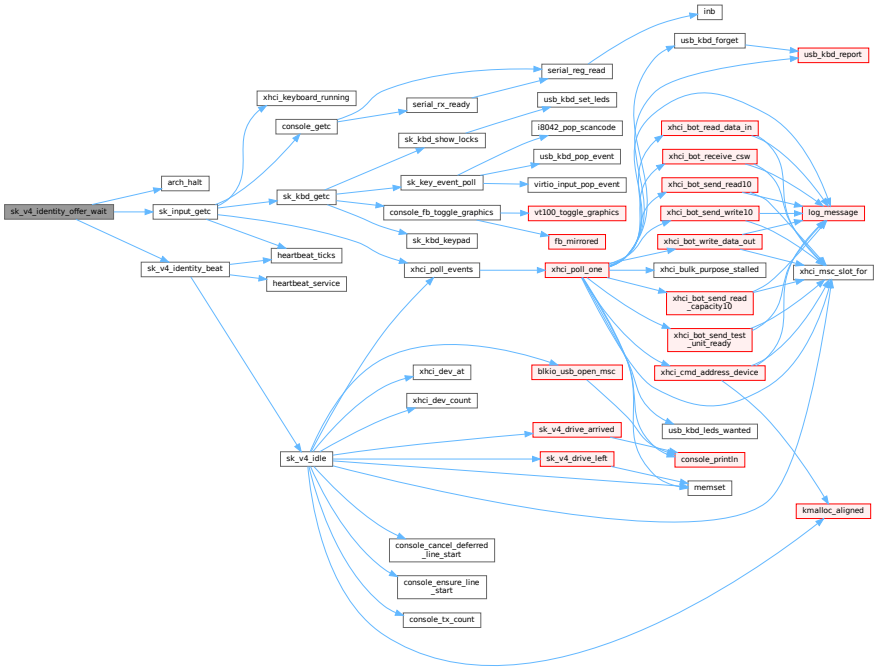
7.175.2.11 sk_v4_identity_offer_wait()

```
int sk_v4_identity_offer_wait (
            blkio_dev_t ** raw,
            uint64_t * bytes )
```

References arch_halt(), BLKIO_FORTH_BLOCK_SIZE, bytes, g_mint_waiting, g_offered, g_spare, g_stick_in, sk_v4_drive_t::raw, sk_input_getc(), sk_v4_identity_beat(), and blkio_dev::total_blocks.

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

7.175.2.12 sk_v4_identity_offered_present()

```
int sk_v4_identity_offered_present (
            void  )
```

References g_offered.

Referenced by read_answer(), and sk_v4_mint_zuse().

Here is the caller graph for this function:

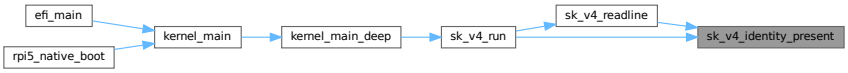
7.175.2.13 sk_v4_identity_present()

```
int sk_v4_identity_present (
            void  )
```

References g_zuse_session.

Referenced by sk_v4_readline(), and sk_v4_run().

Here is the caller graph for this function:



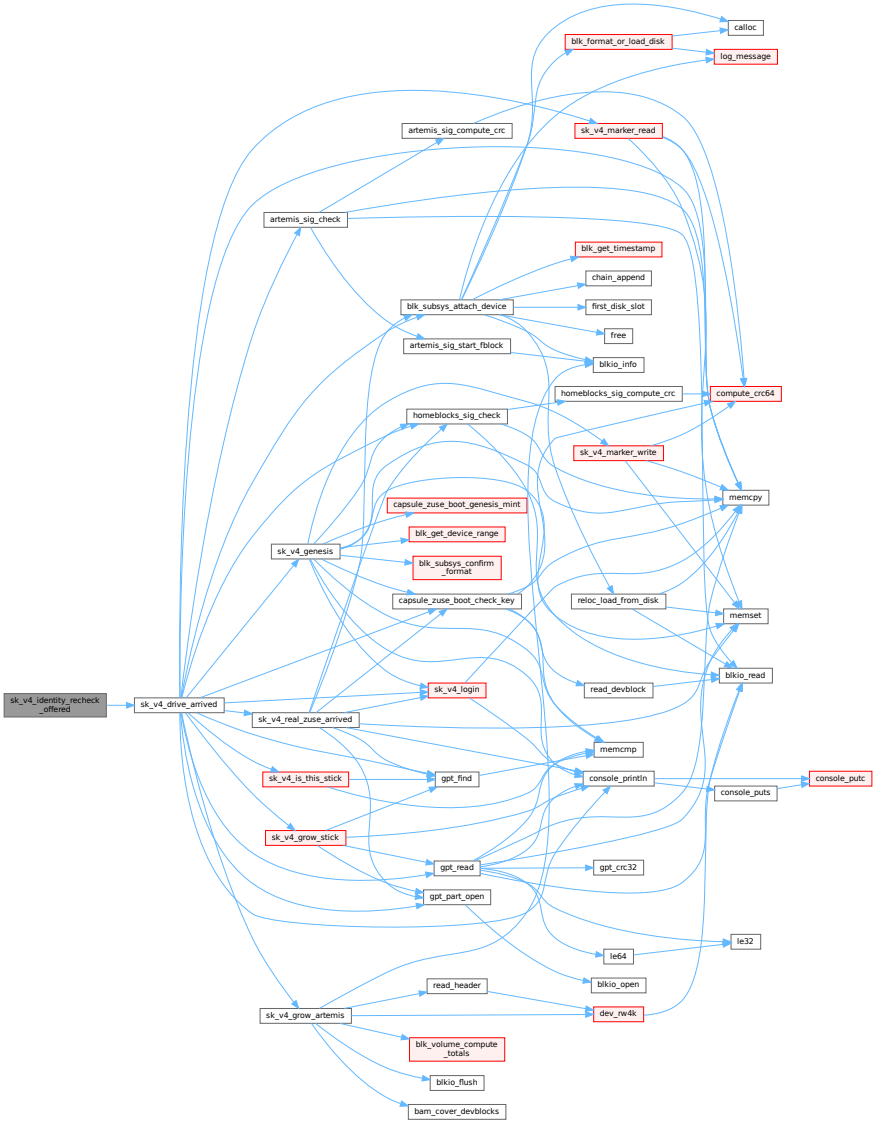
7.175.2.14 sk_v4_identity_recheck_offered()

```
void sk_v4_identity_recheck_offered (
            void  )
```

References g_offered, and sk_v4_drive_arrived().

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

7.175.2.15 sk_v4_identity_release_offered()

```
void sk_v4_identity_release_offered (
            void  )
```

References g_offered, and g_spare.

Referenced by failed(), and sk_v4_mint_zuse().

Here is the caller graph for this function:

7.175.2.16 sk_v4_identity_set_marker()

```
void sk_v4_identity_set_marker (
            const sk_v4_marker_t * m )
```

References g_marker.

Referenced by sk_v4_mint_zuse().

Here is the caller graph for this function:

7.175.2.17 sk_v4_identity_stick_in()

```
int sk_v4_identity_stick_in (
            void  )
```

References g_stick_in.

Referenced by read_answer().

Here is the caller graph for this function:

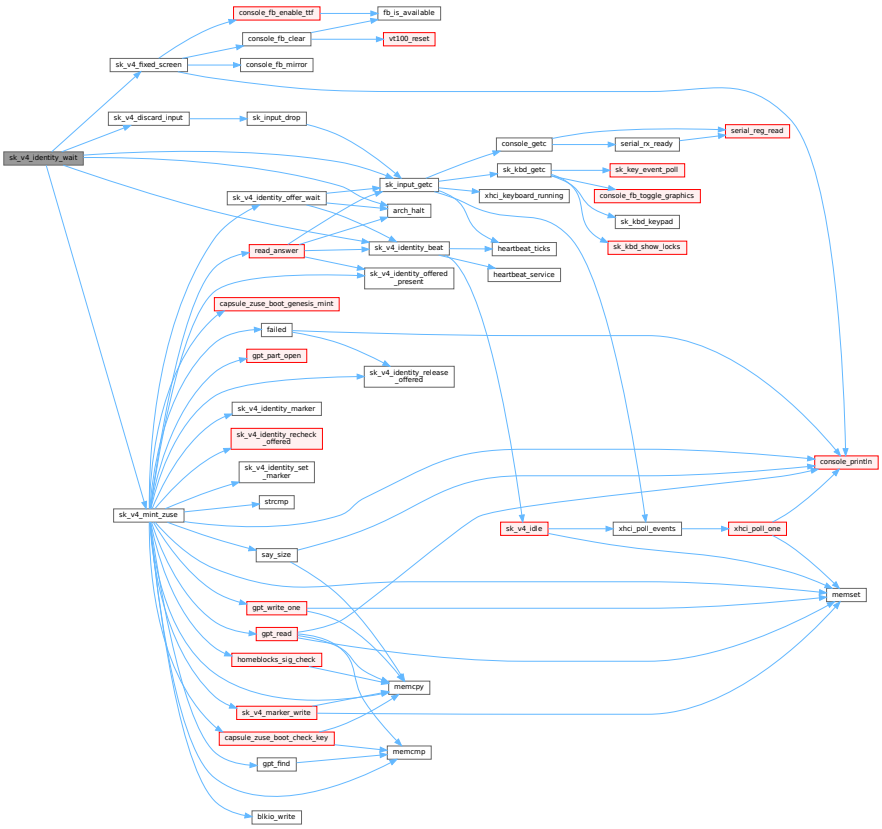
7.175.2.18 sk_v4_identity_wait()

```
void sk_v4_identity_wait (
            void  )
```

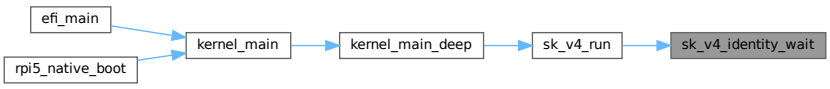
References arch_halt(), g_proxy_fallback, g_zuse_session, sk_input_getc(), sk_v4_discard_input(), sk_v4_fixed_screen(), sk_v4_identity_beat(), and sk_v4_mint_zuse().

Referenced by sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



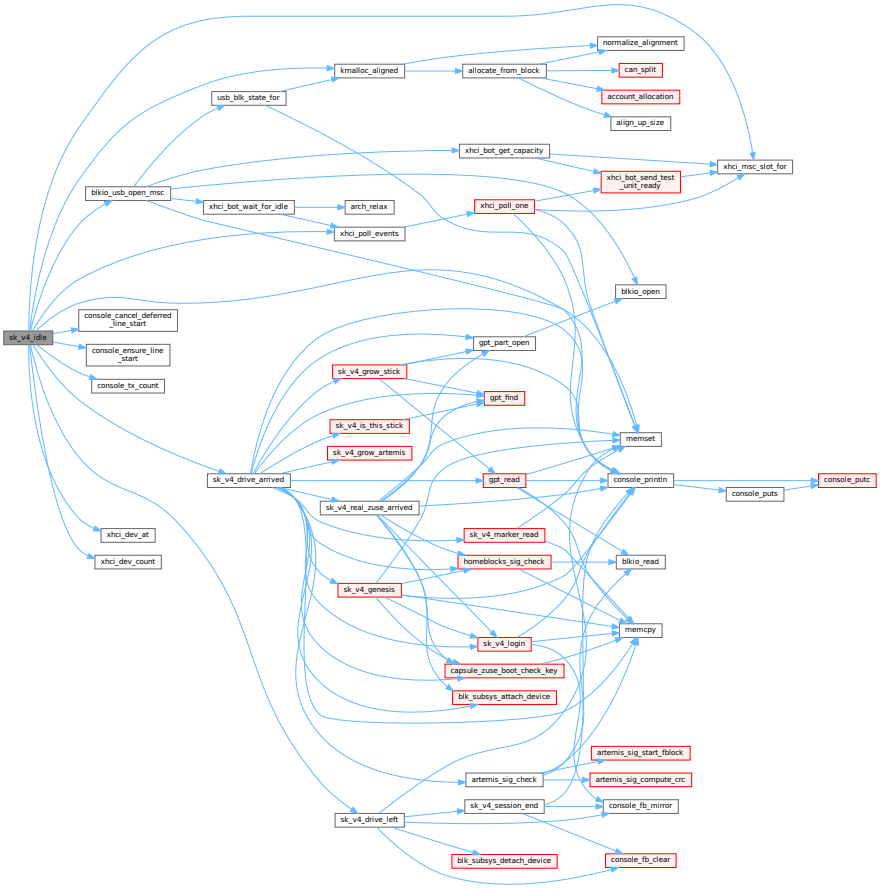
7.175.2.19 sk_v4_idle()

```
static void sk_v4_idle (
            void  )  [static]
```

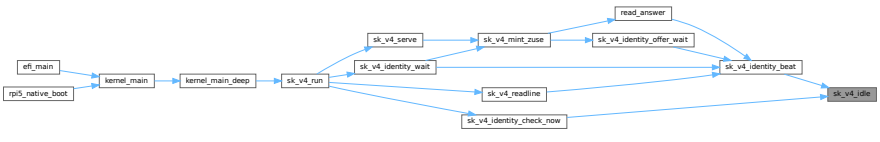
References blkio_usb_open_msc(), xhci_msc_slot_t::bot_msc_attach_pending, xhci_msc_slot_t::bot_msc_attached, xhci_msc_slot_t::bot_msc_detach_pending, bytes, console_cancel_deferred_line_start(), console_ensure_line_start(), console_tx_count(), g_usb, kmalloc_aligned(), xhci_dev_t::max_slots, memset(), sk_v4_drive_arrived(), sk_v4_drive_left(), slots, xdev, xhci_dev_at(), xhci_dev_count(), XHCI_MAX_CONTROLLERS, xhci_msc_slot_for(), and xhci_poll_events().

Referenced by sk_v4_identity_beat(), and sk_v4_identity_check_now().

Here is the call graph for this function:



Here is the caller graph for this function:



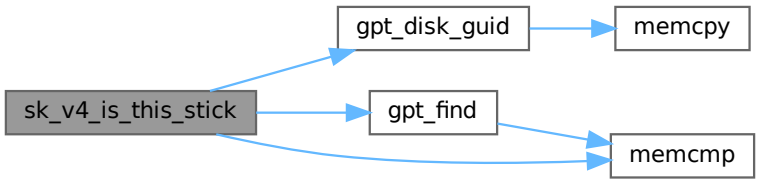
7.175.2.20 sk_v4_is_this_stick()

```
static int sk_v4_is_this_stick (
            unsigned nparts )  [static]
```

References BootDisk::disk_guid, g_boot_disk, g_gpt_parts, g_stick_in, gpt_disk_guid(), gpt_find(), gpt_type_starship_artemis, gpt_type_starship_identity, BootDisk::is_stick, and memcmp().

Referenced by sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

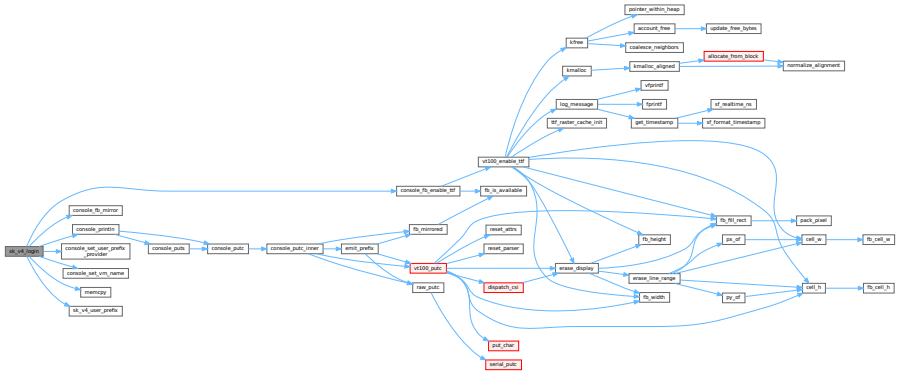
7.175.2.21 sk_v4_login()

```
static void sk_v4_login (
            blkio_dev_t * dev,
            const uint8_t seed[32],
            const uint8_t pubkey[32],
            const char * line )  [static]
```

References console_fb_enable_ttf(), console_fb_mirror(), console_println(), console_set_user_prefix_provider(), console_set_vm_name(), g_zuse_cert_installed, g_zuse_dev, g_zuse_pubkey, g_zuse_root_known, g_zuse_root_pubkey, g_zuse_seed, g_zuse_session, memcpy(), and sk_v4_user_prefix().

Referenced by sk_v4_drive_arrived(), sk_v4_genesis(), and sk_v4_real_zuse_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

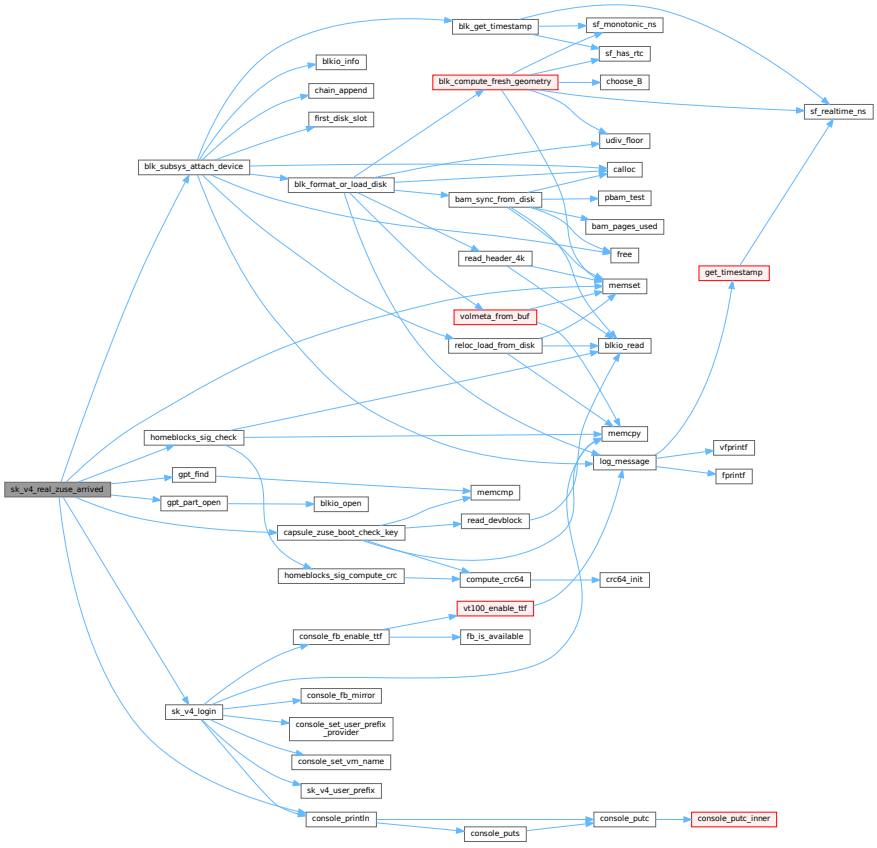
7.175.2.22 sk_v4_real_zuse_arrived()

```
static int sk_v4_real_zuse_arrived (
            sk_v4_drive_t * d,
            unsigned nparts )  [static]
```

References BLK_OK, blk_subsys_attach_device(), BLKIO_OK, capsule_zuse_boot_check_key(), console_println(), g_gpt_parts, g_marker, g_proxy_fallback, g_real_drive, gpt_find(), gpt_part_open(), gpt_type_starship_identity, sk_v4_marker_t::has_real, homeblocks_sig_check(), HOMEBLOCKS_SIG_START_FBLOCK, memset(), sk_v4_drive_t::part, sk_v4_drive_t::part_state, sk_v4_marker_t::present, sk_v4_drive_t::raw, sk_v4_marker_t::real, sk_v4_login(), and sk_v4_drive_t::used.

Referenced by sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

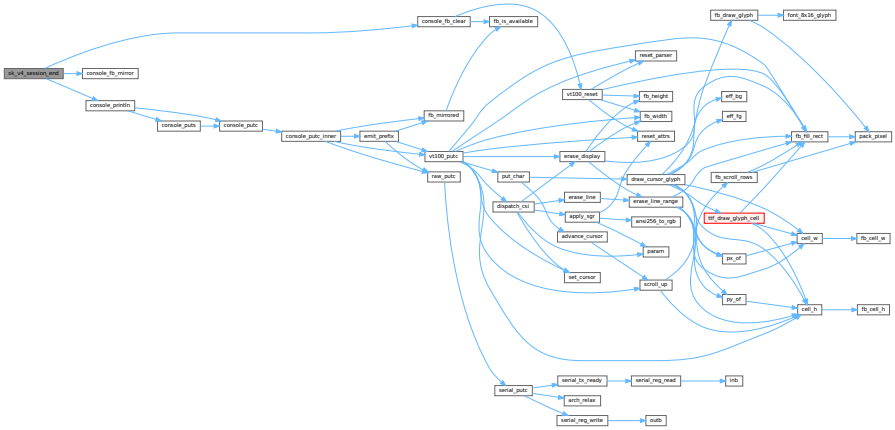
7.175.2.23 sk_v4_session_end()

```
static void sk_v4_session_end (
            void  )  [static]
```

References console_fb_clear(), console_fb_mirror(), console_println(), g_zuse_dev, and g_zuse_session.

Referenced by sk_v4_drive_left().

Here is the call graph for this function:



Here is the caller graph for this function:

7.175.2.24 sk_v4_user_prefix()

```
static const char * sk_v4_user_prefix (
            void  )  [static]
```

References g_zuse_session.

Referenced by sk_v4_login().

Here is the caller graph for this function:

7.175.3 Variable Documentation

7.175.3.1 count

```
uint32_t count
```

7.175.3.2 g_boot_disk

```
const BootDisk* g_boot_disk  [static]
```

Referenced by sk_v4_identity_boot(), and sk_v4_is_this_stick().

7.175.3.3 g_gpt_parts

```
gpt_part_t g_gpt_parts[16u]  [static]
```

Referenced by sk_v4_drive_arrived(), sk_v4_grow_stick(), sk_v4_is_this_stick(), and sk_v4_real_zuse_arrived().

7.175.3.4 g_last_beat

```
uint64_t g_last_beat  [static]
```

Referenced by sk_v4_identity_beat(), sk_v4_identity_boot(), and sk_v4_identity_check_now().

7.175.3.5 g_marker

```
sk_v4_marker_t g_marker  [static]
```

Referenced by sk_v4_drive_arrived(), sk_v4_drive_left(), sk_v4_genesis(), sk_v4_identity_marker(), sk_v4_identity_set_marker(), and sk_v4_real_zuse_arrived().

7.175.3.6 g_mint_waiting

```
int g_mint_waiting  [static]
```

Referenced by sk_v4_drive_arrived(), and sk_v4_identity_offer_wait().

7.175.3.7 g_offered

```
sk_v4_drive_t* g_offered  [static]
```

Referenced by sk_v4_drive_arrived(), sk_v4_drive_left(), sk_v4_identity_offer_wait(), sk_v4_identity_offered_present(), sk_v4_identity_recheck_offered(), and sk_v4_identity_release_offered().

7.175.3.8 g_pending_real

```
sk_v4_drive_t* g_pending_real  [static]
```

Referenced by sk_v4_drive_arrived(), and sk_v4_drive_left().

7.175.3.9 g_proxy_fallback

```
int g_proxy_fallback  [static]
```

Referenced by sk_v4_drive_arrived(), sk_v4_drive_left(), sk_v4_identity_wait(), and sk_v4_real_zuse_arrived().

7.175.3.10 g_real_drive

```
sk_v4_drive_t* g_real_drive  [static]
```

Referenced by sk_v4_drive_left(), and sk_v4_real_zuse_arrived().

7.175.3.11 g_spare

```
sk_v4_drive_t* g_spare  [static]
```

Referenced by sk_v4_drive_arrived(), sk_v4_drive_left(), sk_v4_identity_offer_wait(), and sk_v4_identity_release_offered().

7.175.3.12 g_stick_in

```
int g_stick_in  [static]
```

Referenced by sk_v4_drive_arrived(), sk_v4_drive_left(), sk_v4_identity_offer_wait(), sk_v4_identity_stick_in(), and sk_v4_is_this_stick().

7.175.3.13 [struct]

```
struct { ...  } g_usb[ 8u ]  [static]
```

Referenced by sk_v4_idle().

7.175.3.14 g_zuse_cert_installed

```
int g_zuse_cert_installed  [static]
```

Referenced by sk_v4_login().

7.175.3.15 g_zuse_dev

```
struct blkio_dev* g_zuse_dev  [static]
```

Referenced by sk_v4_drive_left(), sk_v4_login(), and sk_v4_session_end().

7.175.3.16 g_zuse_pubkey

```
uint8_t g_zuse_pubkey[32]  [static]
```

Referenced by sk_v4_login().

7.175.3.17 g_zuse_root_known

```
int g_zuse_root_known  [static]
```

Referenced by sk_v4_drive_arrived(), and sk_v4_login().

7.175.3.18 g_zuse_root_pubkey

```
uint8_t g_zuse_root_pubkey[32]  [static]
```

Referenced by sk_v4_drive_arrived(), and sk_v4_login().

7.175.3.19 g_zuse_seed

```
uint8_t g_zuse_seed[32]  [static]
```

Referenced by sk_v4_login().

7.175.3.20 g_zuse_session

```
int g_zuse_session  [static]
```

Referenced by sk_v4_drive_arrived(), sk_v4_identity_present(), sk_v4_identity_wait(), sk_v4_login(), sk_v4_session_end(), and sk_v4_user_prefix().

7.175.3.21 slots

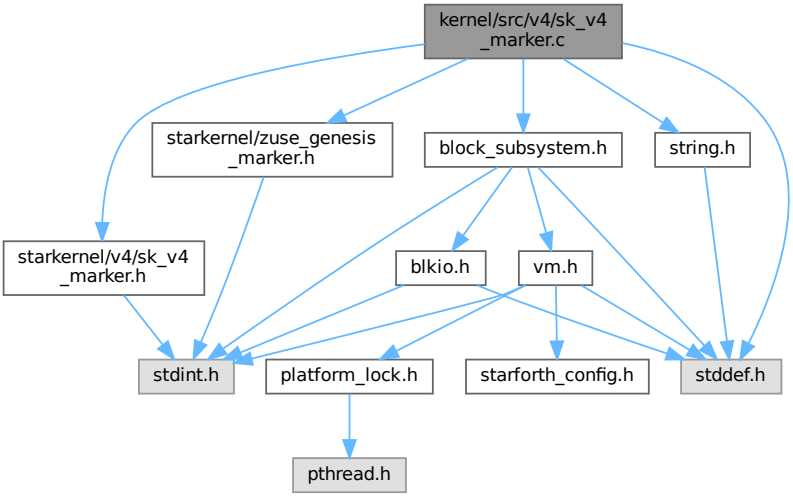
```
sk_v4_drive_t* slots
```

Referenced by sk_v4_idle().

7.176 kernel/src/v4/sk_v4_marker.c File Reference

```
#include "starkernel/v4/sk_v4_marker.h"
#include "starkernel/zuse_genesis_marker.h"
#include "block_subsystem.h"
#include <stddef.h>
#include <string.h>
```

Include dependency graph for sk_v4_marker.c:



Functions

- int sk_v4_marker_read (sk_v4_marker_t *m)
- int sk_v4_marker_write (const sk_v4_marker_t *m)

Variables

- union {
  uint8_t bytes [4096]
  zuse_genesis_marker_t v1
  zuse_genesis_marker_v2_t v2
  } g_buf

7.176.1 Function Documentation

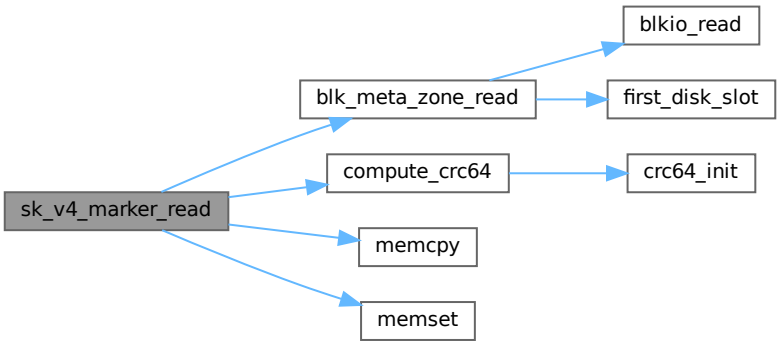
7.176.1.1 sk_v4_marker_read()

```
int sk_v4_marker_read (
            sk_v4_marker_t * m )
```

References blk_meta_zone_read(), compute_crc64(), g_buf, sk_v4_marker_t::has_real, memcpy(), memset(), sk_v4_marker_t::present, sk_v4_marker_t::proxy, sk_v4_marker_t::real, ZUSE_GENESIS_MARKER_MAGIC, ZUSE_GENESIS_MARKER_VERSION, and ZUSE_GENESIS_MARKER_VERSION_2.

Referenced by sk_v4_drive_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

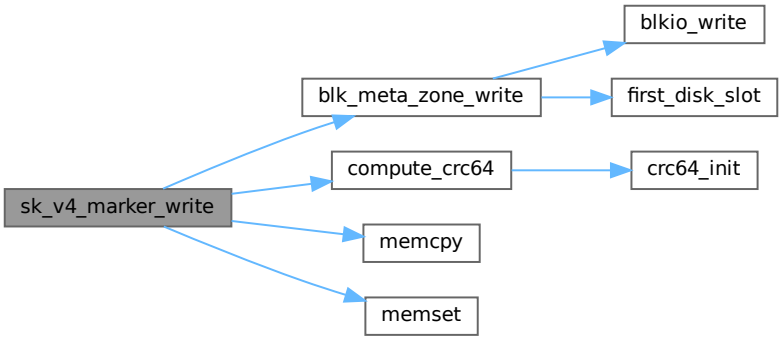
7.176.1.2 sk_v4_marker_write()

```
int sk_v4_marker_write (
            const sk_v4_marker_t * m )
```

References blk_meta_zone_write(), compute_crc64(), g_buf, sk_v4_marker_t::has_real, memcpy(), memset(), sk_v4_marker_t::proxy, sk_v4_marker_t::real, ZUSE_GENESIS_MARKER_MAGIC, and ZUSE_GENESIS_MARKER_VERSION_2.

Referenced by sk_v4_genesis(), and sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

7.176.2 Variable Documentation

7.176.2.1 bytes

```
uint8_t bytes[4096]
```

7.176.2.2 [union]

```
union { ... } g_buf [static]
```

Referenced by sk_v4_marker_read(), and sk_v4_marker_write().

7.176.2.3 v1

```
zuse_genesis_marker_t v1
```

Referenced by xxhash64().

7.176.2.4 v2

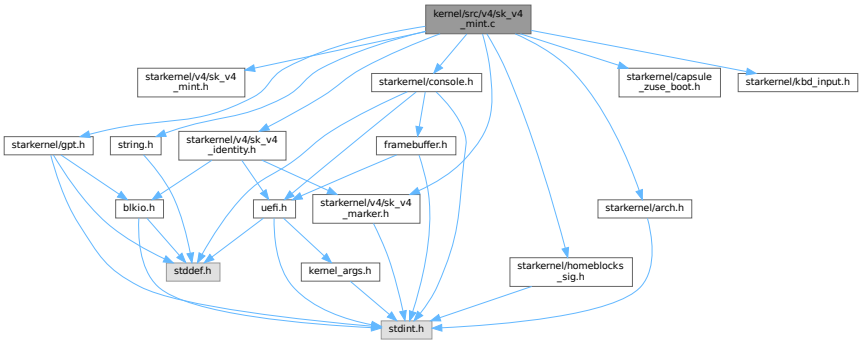
```
zuse_genesis_marker_v2_t v2
```

Referenced by rpi5_mailbox_get_framebuffer(), and xxhash64().

7.177 kernel/src/v4/sk_v4_mint.c File Reference

```
#include "starkernel/v4/sk_v4_mint.h"
#include "starkernel/v4/sk_v4_identity.h"
#include "starkernel/v4/sk_v4_marker.h"
#include "starkernel/capsule_zuse_boot.h"
#include "starkernel/homeblocks_sig.h"
#include "starkernel/kbd_input.h"
#include "starkernel/console.h"
#include "starkernel/arch.h"
#include "starkernel/gpt.h"
#include <string.h>
```

Include dependency graph for sk_v4_mint.c:



Macros

- #define MINT_MIN_BYTES (64ULL ∗ 1024ULL ∗ 1024ULL) /∗ a stick under 64 MiB is refused ∗/
- #define MINT_IDENT_SECTORS (500ULL ∗ 2048ULL) /∗ the identity partition: 500 MiB ∗/
- #define MINT_ZERO_FBLOCKS 32u /∗ its first 8 devblocks, zeroed ∗/

Functions

- static void say_size (uint64_t bytes)
- static int read_answer (char ∗buf, int max)
- static int failed (void)
- int sk_v4_mint_zuse (void)

7.177.1 Macro Definition Documentation

7.177.1.1 MINT_IDENT_SECTORS

```
#define MINT_IDENT_SECTORS (500ULL * 2048ULL) /* the identity partition:  500 MiB */
```

7.177.1.2 MINT_MIN_BYTES

```
#define MINT_MIN_BYTES (64ULL * 1024ULL * 1024ULL) /* a stick under 64 MiB is refused */
```

7.177.1.3 MINT_ZERO_FBLOCKS

```
#define MINT_ZERO_FBLOCKS 32u /* its first 8 devblocks, zeroed */
```

7.177.2 Function Documentation

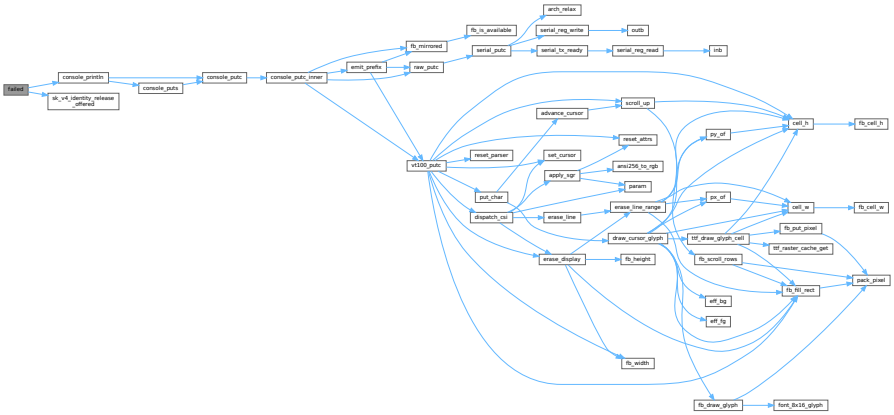
7.177.2.1 failed()

```
static int failed (
            void  )  [static]
```

References console_println(), and sk_v4_identity_release_offered().

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

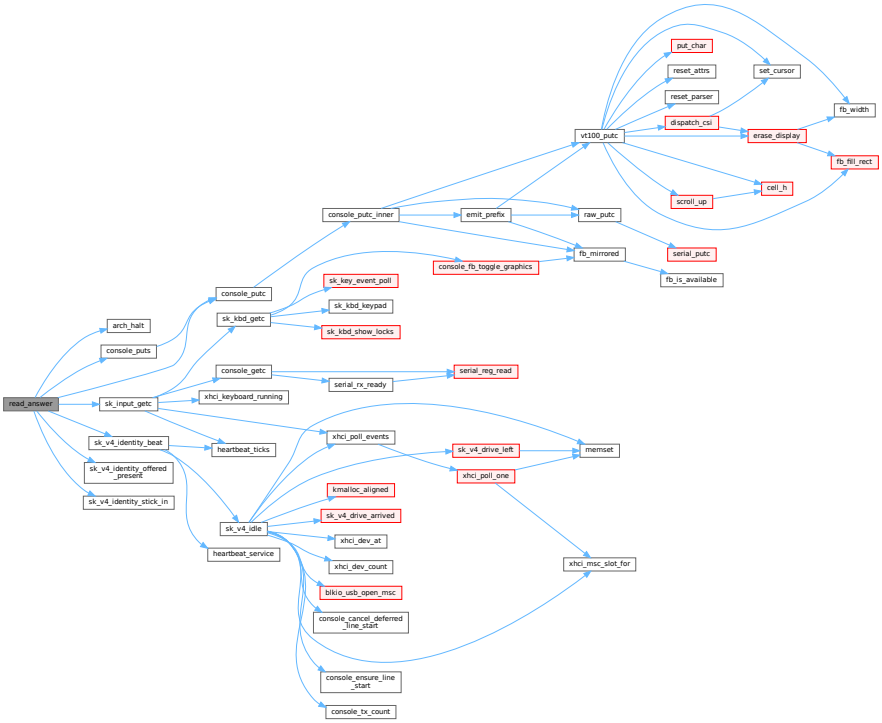
7.177.2.2 read_answer()

```
static int read_answer (
            char * buf,
            int max )  [static]
```

References arch_halt(), console_putc(), console_puts(), sk_input_getc(), sk_v4_identity_beat(), sk_v4_identity_offered_present(), and sk_v4_identity_stick_in().

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

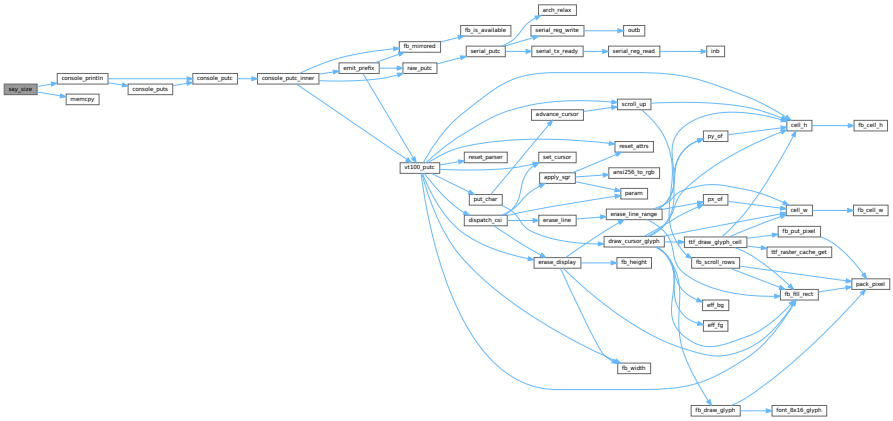
7.177.2.3 say_size()

```
static void say_size (
            uint64_t bytes )  [static]
```

References bytes, console_println(), and memcpy().

Referenced by sk_v4_mint_zuse().

Here is the call graph for this function:



Here is the caller graph for this function:

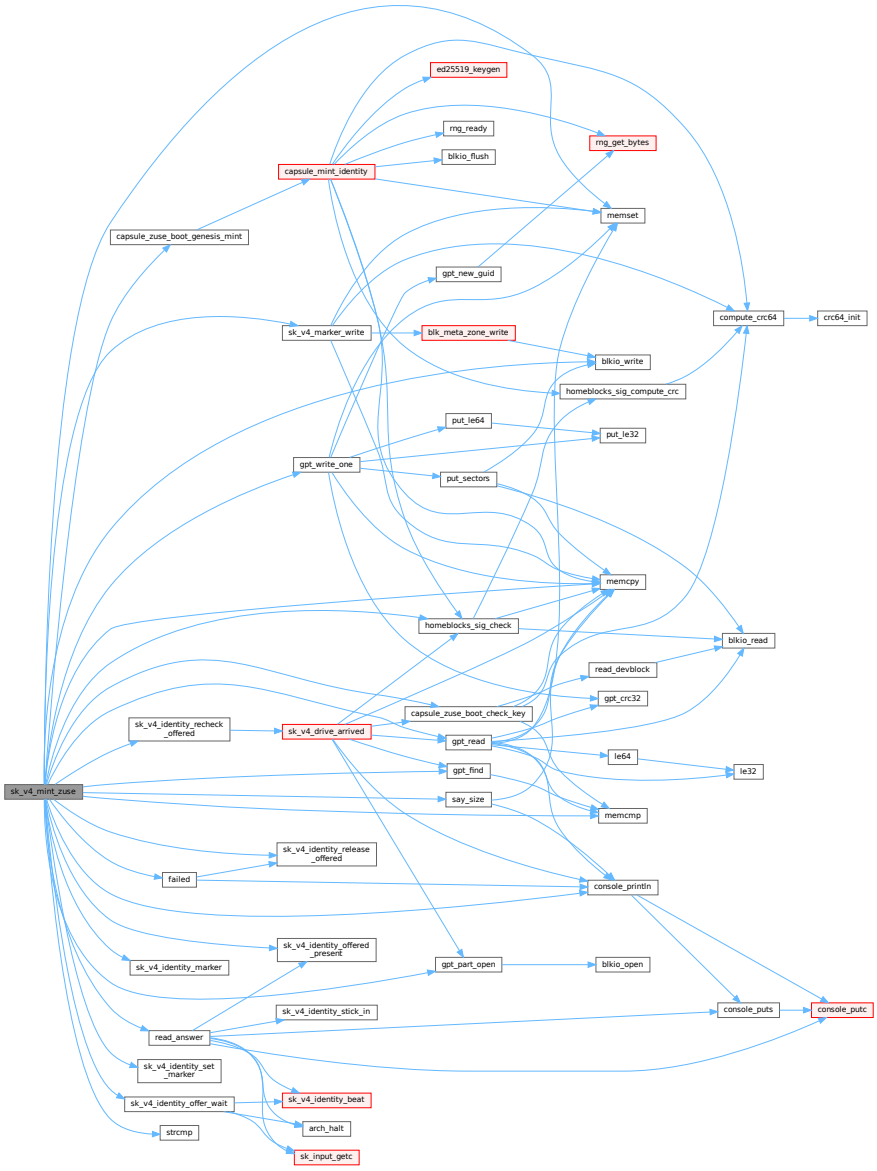
7.177.2.4 sk_v4_mint_zuse()

```
int sk_v4_mint_zuse (
            void  )
```

References BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_write(), bytes, capsule_zuse_boot_check_key(), capsule_zuse_boot_genesis_mint(), console_println(), failed(), GPT_BAD, gpt_find(), GPT_OK, gpt_part_open(), gpt_read(), gpt_type_starship_identity, gpt_write_one(), sk_v4_marker_t::has_real, homeblocks_sig_check(), HOMEBLOCKS_SIG_START_FBLOCK, memcmp(), memcpy(), memset(), MINT_IDENT_SECTORS, MINT_MIN_BYTES, MINT_ZERO_FBLOCKS, sk_v4_marker_t::present, read_answer(), sk_v4_marker_t::real, say_size(), sk_v4_identity_marker(), sk_v4_identity_offer_wait(), sk_v4_identity_offered_present(), sk_v4_identity_recheck_offered(), sk_v4_identity_release_offered(), sk_v4_identity_set_marker(), sk_v4_marker_write(), strcmp(), and zero.

Referenced by sk_v4_identity_wait(), and sk_v4_serve().

Here is the call graph for this function:



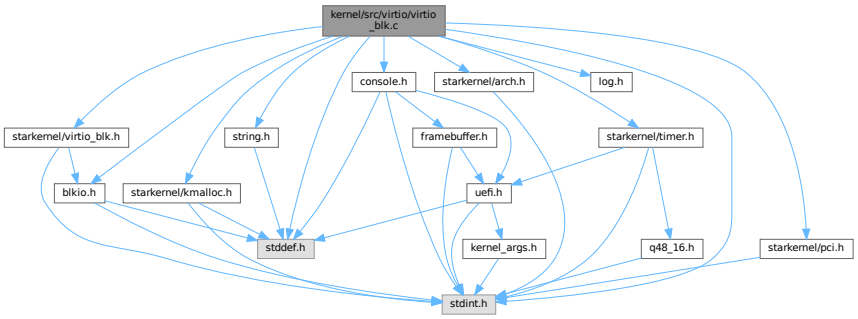
Here is the caller graph for this function:

7.178 kernel/src/virtio/virtio_blk.c File Reference

```
#include <stddef.h>
#include <stdint.h>
```

```
#include <string.h>
#include "starkernel/pci.h"
#include "starkernel/virtio_blk.h"
#include "starkernel/kmalloc.h"
#include "console.h"
#include "blkio.h"
#include "log.h"
#include "starkernel/arch.h"
#include "starkernel/timer.h"
```

Include dependency graph for virtio_blk.c:



Data Structures

- struct VirtioCommonCfg
- struct VirtqDesc
- struct VirtqAvail
- struct VirtqUsedElem
- struct VirtqUsed
- struct VirtBlkReq
- struct VirtBlkState

Macros

- #define VIRTIO_PCI_CAP_VENDOR_ID 0x09u
- #define VIRTIO_PCI_CAP_COMMON_CFG 1u
- #define VIRTIO_PCI_CAP_NOTIFY_CFG 2u
- #define VIRTIO_PCI_CAP_ISR_CFG 3u
- #define VIRTIO_PCI_CAP_DEVICE_CFG 4u
- #define VCAP_OFF_CAP_VNDR 0u
- #define VCAP_OFF_CAP_NEXT 1u
- #define VCAP_OFF_CAP_LEN 2u
- #define VCAP_OFF_CFG_TYPE 3u
- #define VCAP_OFF_BAR 4u
- #define VCAP_OFF_OFFSET 8u
- #define VCAP_OFF_LENGTH 12u
- #define VCAP_OFF_NOTIFY_MULT 16u
- #define VIRTIO_STATUS_ACKNOWLEDGE 0x01u
- #define VIRTIO_STATUS_DRIVER 0x02u
- #define VIRTIO_STATUS_DRIVER_OK 0x04u
- #define VIRTIO_STATUS_FEATURES_OK 0x08u

- #define VIRTIO_STATUS_FAILED 0x80u
- #define VIRTIO_F_VERSION_1 (1ULL << 32)
- #define VQUEUE_SIZE 4u /* power of 2; ≥ 3 for one in-flight chain */
- #define VRING_DESC_F_NEXT 1u
- #define VRING_DESC_F_WRITE 2u
- #define VBLK_IO_TIMEOUT_NS 5000000000ULL

Functions

- static void * walk_virtio_caps (const PciDevice *d, uint8_t cap_type, uint32_t *extra_out)
- static void wmb (void)
- static void rmb (void)
- static int vblk_io (int write, uint64_t sector, void *buf, uint32_t nbytes)
- static int vblk_open (blkio_dev_t *dev, const blkio_params_t *p)
- static int vblk_close (blkio_dev_t *dev)
- static int vblk_read (blkio_dev_t *dev, uint32_t fblock, void *dst)
- static int vblk_write (blkio_dev_t *dev, uint32_t fblock, const void *src)
- static int vblk_flush (blkio_dev_t *dev)
- static int vblk_info (blkio_dev_t *dev, blkio_info_t *out)
- static int vblk_init_device (const PciDevice *pci)
- int virtio_blk_find_artemis (blkio_dev_t *dev_out)

Variables

- static VirtBlkState g_vblk
- static int g_vblk_ready = 0
- static int g_vblk_dead = 0
- static const blkio_vtable_t g_vblk_vtable

7.178.1 Macro Definition Documentation

7.178.1.1 VBLK_IO_TIMEOUT_NS

```
#define VBLK_IO_TIMEOUT_NS 5000000000ULL
```

7.178.1.2 VCAP_OFF_BAR

```
#define VCAP_OFF_BAR 4u
```

7.178.1.3 VCAP_OFF_CAP_LEN

```
#define VCAP_OFF_CAP_LEN 2u
```

7.178.1.4 VCAP_OFF_CAP_NEXT

```
#define VCAP_OFF_CAP_NEXT 1u
```

7.178.1.5 VCAP_OFF_CAP_VNDR

```
#define VCAP_OFF_CAP_VNDR 0u
```

7.178.1.6 VCAP_OFF_CFG_TYPE

```
#define VCAP_OFF_CFG_TYPE 3u
```

7.178.1.7 VCAP_OFF_LENGTH

```
#define VCAP_OFF_LENGTH 12u
```

7.178.1.8 VCAP_OFF_NOTIFY_MULT

```
#define VCAP_OFF_NOTIFY_MULT 16u
```

7.178.1.9 VCAP_OFF_OFFSET

```
#define VCAP_OFF_OFFSET 8u
```

7.178.1.10 VIRTIO_F_VERSION_1

```
#define VIRTIO_F_VERSION_1 (1ULL << 32)
```

7.178.1.11 VIRTIO_PCI_CAP_COMMON_CFG

```
#define VIRTIO_PCI_CAP_COMMON_CFG 1u
```

7.178.1.12 VIRTIO_PCI_CAP_DEVICE_CFG

```
#define VIRTIO_PCI_CAP_DEVICE_CFG 4u
```

7.178.1.13 VIRTIO_PCI_CAP_ISR_CFG

```
#define VIRTIO_PCI_CAP_ISR_CFG 3u
```

7.178.1.14 VIRTIO_PCI_CAP_NOTIFY_CFG

```
#define VIRTIO_PCI_CAP_NOTIFY_CFG 2u
```

7.178.1.15 VIRTIO_PCI_CAP_VENDOR_ID

```
#define VIRTIO_PCI_CAP_VENDOR_ID 0x09u
```

7.178.1.16 VIRTIO_STATUS_ACKNOWLEDGE

```
#define VIRTIO_STATUS_ACKNOWLEDGE 0x01u
```

7.178.1.17 VIRTIO_STATUS_DRIVER

```
#define VIRTIO_STATUS_DRIVER 0x02u
```

7.178.1.18 VIRTIO_STATUS_DRIVER_OK

```
#define VIRTIO_STATUS_DRIVER_OK 0x04u
```

7.178.1.19 VIRTIO_STATUS_FAILED

```
#define VIRTIO_STATUS_FAILED 0x80u
```

7.178.1.20 VIRTIO_STATUS_FEATURES_OK

```
#define VIRTIO_STATUS_FEATURES_OK 0x08u
```

7.178.1.21 VQUEUE_SIZE

```
#define VQUEUE_SIZE 4u /* power of 2; ≥ 3 for one in-flight chain */
```

7.178.1.22 VRING_DESC_F_NEXT

```
#define VRING_DESC_F_NEXT 1u
```

7.178.1.23 VRING_DESC_F_WRITE

```
#define VRING_DESC_F_WRITE 2u
```

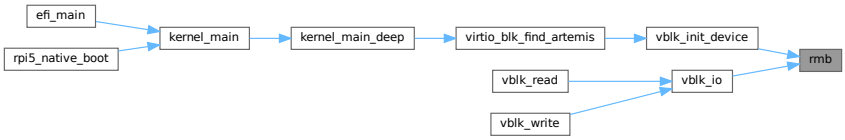
7.178.2 Function Documentation

7.178.2.1 rmb()

```
static void rmb (
            void  )  [inline], [static]
```

Referenced by vblk_init_device(), and vblk_io().

Here is the caller graph for this function:



7.178.2.2 vblk_close()

```
static int vblk_close (
            blkio_dev_t * dev )  [static]
```

References BLKIO_OK.

7.178.2.3 vblk_flush()

```
static int vblk_flush (
            blkio_dev_t * dev )  [static]
```

References BLKIO_OK.

7.178.2.4 vblk_info()

```
static int vblk_info (
            blkio_dev_t * dev,
            blkio_info_t * out )  [static]
```

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, VirtBlkState::capacity_sectors, blkio_info_t::forth_block_size, blkio_info_t::phys_sector_size, blkio_info_t::phys_size_bytes, blkio_info_t::read_only, blkio_dev::state, blkio_info_t::total_blocks, VirtBlkState::total_forth_blocks, and VBLK_SECTOR_SIZE.

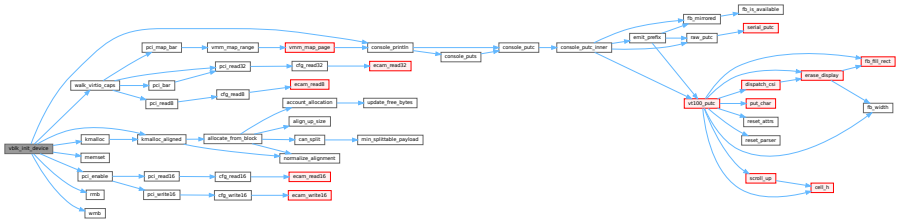
7.178.2.5 vblk_init_device()

```
static int vblk_init_device (
            const PciDevice * pci )  [static]
```

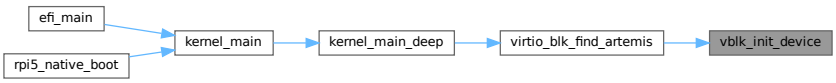
References VirtBlkState::avail, VirtBlkState::avail_idx, BLKIO_FORTH_BLOCK_SIZE, VirtBlkState::capacity_sectors, VirtBlkState::common, VirtioCommonCfg::config_msix_vector, console_println(), VirtBlkState::data_buf, VirtBlkState::desc, VirtioCommonCfg::device_status, VirtioCommonCfg::driver_feature, VirtioCommonCfg::driver_feature_select, g_vblk, kmalloc(), kmalloc_aligned(), VirtBlkState::last_used_idx, memset(), VirtBlkState::notify, VirtBlkState::notify_off_mult, pci_enable(), VirtioCommonCfg::queue_desc, VirtioCommonCfg::queue_device, VirtioCommonCfg::queue_driver, VirtioCommonCfg::queue_enable, VirtioCommonCfg::queue_msix_vector, VirtioCommonCfg::queue_notify_off, VirtBlkState::queue_notify_off, VirtioCommonCfg::queue_select, VirtioCommonCfg::queue_size, VirtBlkState::req_buf, rmb(), VirtBlkState::status_buf, VirtBlkState::total_forth_blocks, VirtBlkState::used, VBLK_ARTEMIS_SECTORS, VBLK_SECTORS_PER_BLOK, VIRTIO_F_VERSION_1, VIRTIO_PCI_CAP_COMMON_CFG, VIRTIO_PCI_CAP_DEVICE_CFG, VIRTIO_PCI_CAP_NOTIFY_CFG, VIRTIO_STATUS_ACKNOWLEDGE, VIRTIO_STATUS_DRIVER, VIRTIO_STATUS_DRIVER_OK, VIRTIO_STATUS_FAILED, VIRTIO_STATUS_FEATURES_OK, VQUEUE_SIZE, walk_virtio_caps(), and wmb().

Referenced by virtio_blk_find_artemis().

Here is the call graph for this function:



Here is the caller graph for this function:



7.178.2.6 vblk_io()

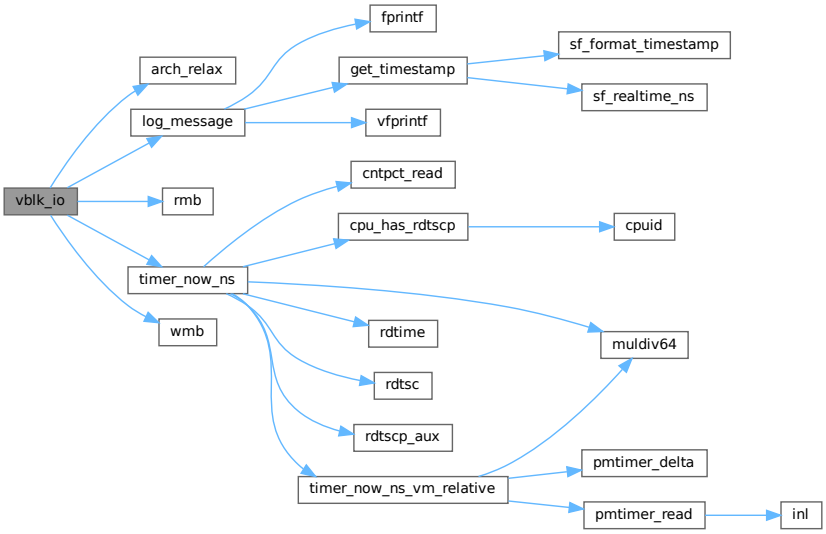
```
static int vblk_io (
            int write,
            uint64_t sector,
            void * buf,
            uint32_t nbytes )  [static]
```

References VirtqDesc::addr, arch_relax(), VirtBlkState::avail, VirtBlkState::avail_idx, BLKIO_EIO, BLKIO_OK, VirtBlkState::desc, VirtqDesc::flags, g_vblk, g_vblk_dead, VirtqAvail::idx, VirtqUsed::idx, VirtBlkState::last_used_idx, VirtqDesc::len, LOG_ERROR, log_message(), VirtqDesc::next, VirtBlkState::notify, VirtBlkState::notify_off_mult, VirtBlkState::queue_notify_off, VirtBlkState::req_buf, VirtBlkReq::reserved, VirtqAvail::ring, rmb(), VirtBlkReq::sector, VirtBlkState::status_buf, timer_now_ns(), VirtBlkReq::type, VirtBlkState::used, VBLK_IO_TIMEOUT_NS,

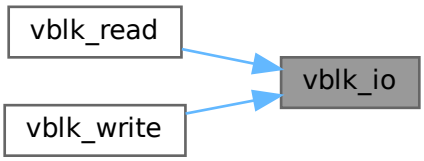
VIRTIO_BLK_S_OK, VIRTIO_BLK_T_IN, VIRTIO_BLK_T_OUT, VQUEUE_SIZE, VRING_DESC_F_NEXT, VRING_DESC_F_WRITE, and wmb().

Referenced by vblk_read(), and vblk_write().

Here is the call graph for this function:



Here is the caller graph for this function:



7.178.2.7 vblk_open()

```
static int vblk_open (
            blkio_dev_t * dev,
            const blkio_params_t * p )  [static]
```

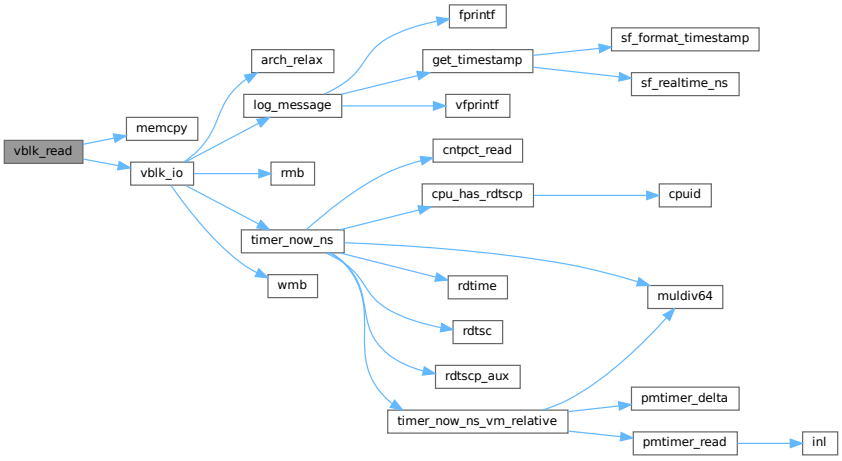
References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_dev::forth_block_size, g_vblk, blkio_dev::state, blkio_dev::total_blocks, and VirtBlkState::total_forth_blocks.

7.178.2.8 vblk_read()

```
static int vblk_read (
            blkio_dev_t * dev,
            uint32_t fblock,
            void * dst )  [static]
```

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, VirtBlkState::data_buf, memcpy(), blkio_dev::state, VirtBlkState::total_forth_blocks, vblk_io(), and VBLK_SECTORS_PER_BLOK.

Here is the call graph for this function:

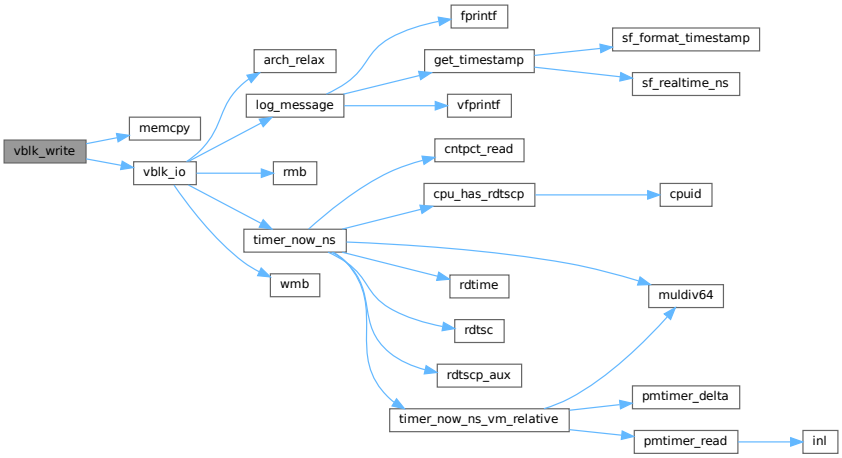


7.178.2.9 vblk_write()

```
static int vblk_write (
            blkio_dev_t * dev,
            uint32_t fblock,
            const void * src )  [static]
```

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, VirtBlkState::data_buf, memcpy(), blkio_dev::state, VirtBlkState::total_forth_blocks, vblk_io(), and VBLK_SECTORS_PER_BLOK.

Here is the call graph for this function:



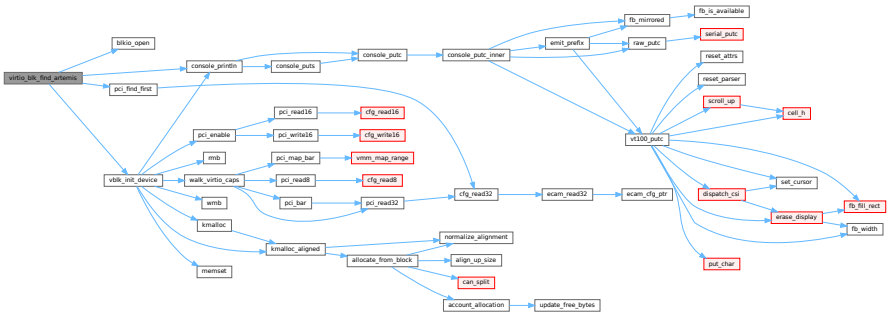
7.178.2.10 virtio_blk_find_artemis()

```
int virtio_blk_find_artemis (
            blkio_dev_t * dev_out )
```

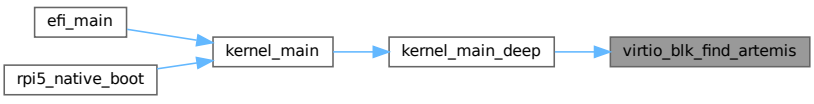
References BLKIO_FORTH_BLOCK_SIZE, blkio_open(), console_println(), blkio_params_t::forth_block_size, g_vblk, g_vblk_ready, g_vblk_vtable, blkio_params_t::opaque, pci_find_first(), blkio_params_t::total_blocks, VirtBlkState::total_forth_blocks, vblk_init_device(), VIRTIO_BLK_DEVICE_LEGACY, VIRTIO_BLK_DEVICE_MODERN, and VIRTIO_PCI_VENDOR_ID.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



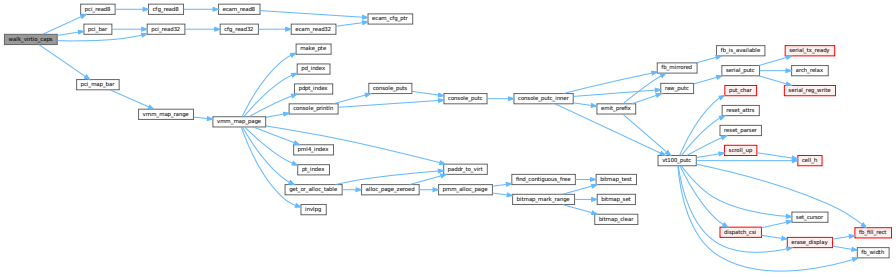
7.178.2.11 walk_virtio_caps()

```
static void * walk_virtio_caps (
            const PciDevice * d,
            uint8_t cap_type,
            uint32_t * extra_out )  [static]
```

References limit, pci_bar(), PCI_CFG_CAP_PTR, pci_map_bar(), pci_read32(), pci_read8(), VCAP_OFF_BAR, VCAP_OFF_CAP_NEXT, VCAP_OFF_CAP_VNDR, VCAP_OFF_CFG_TYPE, VCAP_OFF_LENGTH, VCAP_OFF_NOTIFY_MULT, VCAP_OFF_OFFSET, VIRTIO_PCI_CAP_NOTIFY_CFG, and VIRTIO_PCI_CAP_VENDOR_ID.

Referenced by vblk_init_device().

Here is the call graph for this function:



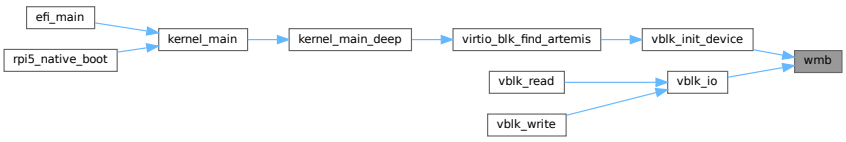
Here is the caller graph for this function:

7.178.2.12 wmb()

```
static void wmb (
            void  )  [inline], [static]
```

Referenced by vblk_init_device(), and vblk_io().

Here is the caller graph for this function:



7.178.3 Variable Documentation

7.178.3.1 g_vblk

```
VirtBlkState g_vblk  [static]
```

Referenced by vblk_init_device(), vblk_io(), vblk_open(), and virtio_blk_find_artemis().

7.178.3.2 g_vblk_dead

```
int g_vblk_dead = 0  [static]
```

Referenced by vblk_io().

7.178.3.3 g_vblk_ready

```
int g_vblk_ready = 0  [static]
```

Referenced by virtio_blk_find_artemis().

7.178.3.4 g_vblk_vtable

```
const blkio_vtable_t g_vblk_vtable  [static]
```

Initial value:

```
= {
    .open  = vblk_open,
    .close = vblk_close,
    .read  = vblk_read,
    .write = vblk_write,
    .flush = vblk_flush,
    .info  = vblk_info
}
```

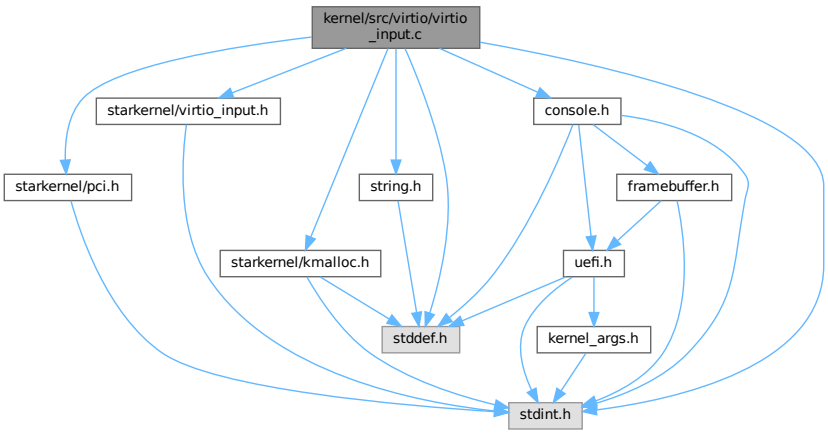
Referenced by virtio_blk_find_artemis().

7.179 kernel/src/virtio/virtio_input.c File Reference

```
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include "starkernel/pci.h"
#include "starkernel/virtio_input.h"
#include "starkernel/kmalloc.h"
```

```
#include "console.h"
```

Include dependency graph for virtio_input.c:



Data Structures

- struct VirtioCommonCfg
- struct VirtqDesc
- struct VirtqAvail
- struct VirtqUsedElem
- struct VirtqUsed
- struct DiagEvent
- struct VirtInputState

Macros

- #define VIRTIO_PCI_CAP_VENDOR_ID 0x09u
- #define VIRTIO_PCI_CAP_COMMON_CFG 1u
- #define VIRTIO_PCI_CAP_NOTIFY_CFG 2u
- #define VIRTIO_PCI_CAP_ISR_CFG 3u
- #define VCAP_OFF_CAP_VNDR 0u
- #define VCAP_OFF_CAP_NEXT 1u
- #define VCAP_OFF_CFG_TYPE 3u
- #define VCAP_OFF_BAR 4u
- #define VCAP_OFF_OFFSET 8u
- #define VCAP_OFF_LENGTH 12u
- #define VCAP_OFF_NOTIFY_MULT 16u
- #define VIRTIO_STATUS_ACKNOWLEDGE 0x01u
- #define VIRTIO_STATUS_DRIVER 0x02u
- #define VIRTIO_STATUS_DRIVER_OK 0x04u
- #define VIRTIO_STATUS_FEATURES_OK 0x08u
- #define VIRTIO_STATUS_FAILED 0x80u
- #define VIRTIO_F_VERSION_1 (1ULL << 32)
- #define EVENTQ_SIZE 8u /* power of 2; headroom for several queued events */
- #define VRING_DESC_F_WRITE 2u
- #define DIAG_RING_SIZE 16u

Functions

- static void diag_push (uint16_t code, uint32_t value)
- int virtio_input_pop_event (uint16_t *code, uint32_t *value)
- static void * walk_virtio_caps (const PciDevice *d, uint8_t cap_type, uint32_t *extra_out)
- static void wmb (void)
- static int enable_interrupt_route (const PciDevice *pci)
- static int vinput_init_device (const PciDevice *pci)
- int virtio_input_find_keyboard (void)
- void virtio_input_isr (void)

Variables

- static DiagEvent g_diag_ring [16u]
- static volatile uint32_t g_diag_head = 0
- static volatile uint32_t g_diag_tail = 0
- volatile uint32_t g_virtio_input_isr_count = 0
- static VirtInputState g_vinput
- static int g_vinput_ready = 0
- uint32_t g_virtio_input_plic_source = 0
- uint32_t g_virtio_input_gic_intid = 0

7.179.1 Macro Definition Documentation

7.179.1.1 DIAG_RING_SIZE

```
#define DIAG_RING_SIZE 16u
```

7.179.1.2 EVENTQ_SIZE

```
#define EVENTQ_SIZE 8u /* power of 2; headroom for several queued events */
```

7.179.1.3 VCAP_OFF_BAR

```
#define VCAP_OFF_BAR 4u
```

7.179.1.4 VCAP_OFF_CAP_NEXT

```
#define VCAP_OFF_CAP_NEXT 1u
```

7.179.1.5 VCAP_OFF_CAP_VNDR

```
#define VCAP_OFF_CAP_VNDR 0u
```

7.179.1.6 VCAP_OFF_CFG_TYPE

```
#define VCAP_OFF_CFG_TYPE 3u
```

7.179.1.7 VCAP_OFF_LENGTH

```
#define VCAP_OFF_LENGTH 12u
```

7.179.1.8 VCAP_OFF_NOTIFY_MULT

```
#define VCAP_OFF_NOTIFY_MULT 16u
```

7.179.1.9 VCAP_OFF_OFFSET

```
#define VCAP_OFF_OFFSET 8u
```

7.179.1.10 VIRTIO_F_VERSION_1

```
#define VIRTIO_F_VERSION_1 (1ULL << 32)
```

7.179.1.11 VIRTIO_PCI_CAP_COMMON_CFG

```
#define VIRTIO_PCI_CAP_COMMON_CFG 1u
```

7.179.1.12 VIRTIO_PCI_CAP_ISR_CFG

```
#define VIRTIO_PCI_CAP_ISR_CFG 3u
```

7.179.1.13 VIRTIO_PCI_CAP_NOTIFY_CFG

```
#define VIRTIO_PCI_CAP_NOTIFY_CFG 2u
```

7.179.1.14 VIRTIO_PCI_CAP_VENDOR_ID

```
#define VIRTIO_PCI_CAP_VENDOR_ID 0x09u
```

7.179.1.15 VIRTIO_STATUS_ACKNOWLEDGE

```
#define VIRTIO_STATUS_ACKNOWLEDGE 0x01u
```

7.179.1.16 VIRTIO_STATUS_DRIVER

```
#define VIRTIO_STATUS_DRIVER 0x02u
```

7.179.1.17 VIRTIO_STATUS_DRIVER_OK

```
#define VIRTIO_STATUS_DRIVER_OK 0x04u
```

7.179.1.18 VIRTIO_STATUS_FAILED

```
#define VIRTIO_STATUS_FAILED 0x80u
```

7.179.1.19 VIRTIO_STATUS_FEATURES_OK

```
#define VIRTIO_STATUS_FEATURES_OK 0x08u
```

7.179.1.20 VRING_DESC_F_WRITE

```
#define VRING_DESC_F_WRITE 2u
```

7.179.2 Function Documentation

7.179.2.1 diag_push()

```
static void diag_push (
            uint16_t code,
            uint32_t value )  [static]
```

References DiagEvent::code, DIAG_RING_SIZE, g_diag_head, g_diag_ring, g_diag_tail, and DiagEvent::value.

Referenced by virtio_input_isr().

Here is the caller graph for this function:

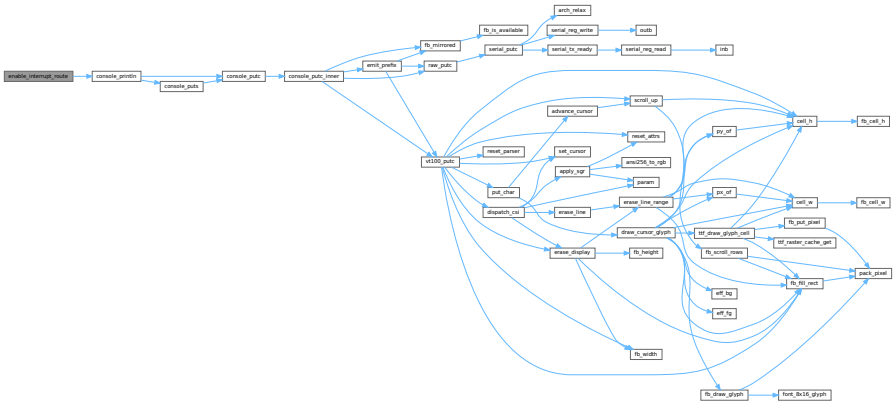
7.179.2.2 enable_interrupt_route()

```
static int enable_interrupt_route (
            const PciDevice * pci )  [static]
```

References console_println().

Referenced by vinput_init_device().

Here is the call graph for this function:



Here is the caller graph for this function:

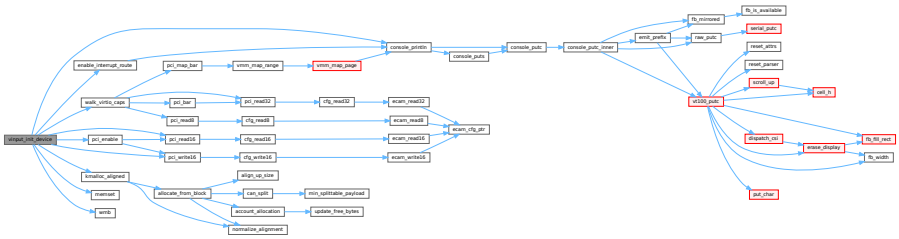
7.179.2.3 vinput_init_device()

```
static int vinput_init_device (
            const PciDevice * pci )  [static]
```

References VirtqDesc::addr, VirtInputState::avail, VirtInputState::avail_idx, VirtInputState::bufs, VirtInputState::common, VirtioCommonCfg::config_msix_vector, console_println(), VirtInputState::desc, VirtioCommonCfg::device_status, VirtioCommonCfg::driver_feature, VirtioCommonCfg::driver_feature_select, enable_interrupt_route(), EVENTQ_SIZE, VirtqDesc::flags, g_vinput, VirtqAvail::idx, VirtInputState::isr_status, kmalloc_aligned(), VirtInputState::last_used_idx, VirtqDesc::len, memset(), VirtqDesc::next, VirtInputState::notify, VirtInputState::notify_off_mult, VirtioCommonCfg::num_queues, PCI_CFG_COMMAND, PCI_CMD_INTX_DISABLE, pci_enable(), pci_read16(), pci_write16(), VirtioCommonCfg::queue_desc, VirtioCommonCfg::queue_device, VirtioCommonCfg::queue_driver, VirtioCommonCfg::queue_enable, VirtioCommonCfg::queue_ms VirtioCommonCfg::queue_notify_off, VirtInputState::queue_notify_off, VirtioCommonCfg::queue_select, VirtioCommonCfg::queue_si VirtqAvail::ring, VirtInputState::used, VIRTIO_F_VERSION_1, VIRTIO_PCI_CAP_COMMON_CFG, VIRTIO_PCI_CAP_ISR_CFG, VIRTIO_PCI_CAP_NOTIFY_CFG, VIRTIO_STATUS_ACKNOWLEDGE, VIRTIO_STATUS_DRIVER, VIRTIO_STATUS_DRIVER_OK, VIRTIO_STATUS_FAILED, VIRTIO_STATUS_FEATURES_OK, VRING_DESC_F_WRITE, walk_virtio_caps(), and wmb().

Referenced by virtio_input_find_keyboard().

Here is the call graph for this function:



Here is the caller graph for this function:

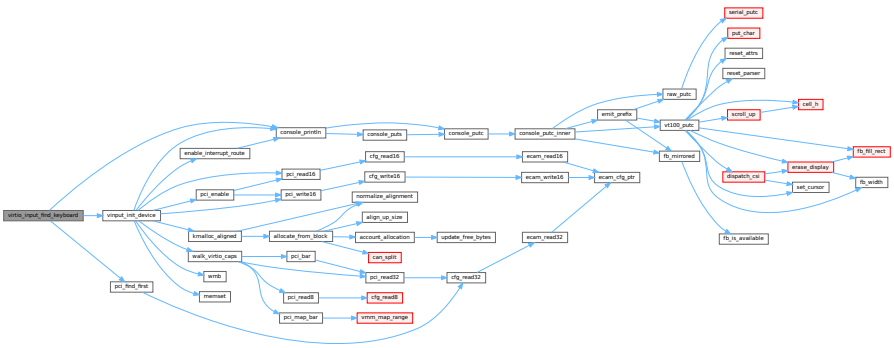
7.179.2.4 virtio_input_find_keyboard()

```
int virtio_input_find_keyboard (
            void  )
```

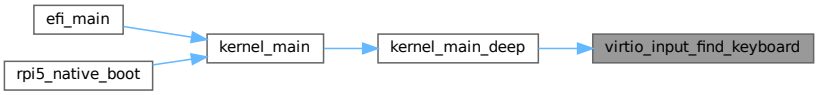
References console_println(), g_vinput_ready, pci_find_first(), vinput_init_device(), VIRTIO_INPUT_PCI_DEVICE_ID, and VIRTIO_INPUT_PCI_VENDOR_ID.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



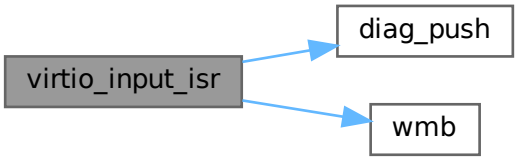
7.179.2.5 virtio_input_isr()

```
void virtio_input_isr (
            void  )
```

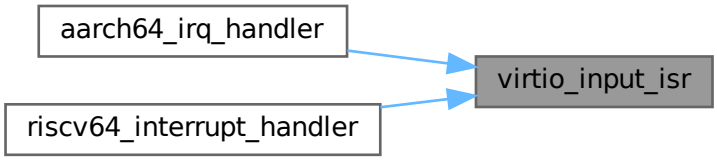
References VirtInputState::avail, VirtInputState::avail_idx, VirtInputState::bufs, VirtioInputEvent::code, diag_push(), EVENTQ_SIZE, g_vinput, g_vinput_ready, g_virtio_input_isr_count, VirtqUsedElem::id, VirtqAvail::idx, VirtqUsed::idx, VirtInputState::isr_status, VirtInputState::last_used_idx, VirtInputState::notify, VirtInputState::notify_off_mult, VirtInputState::queue_notify_off, VirtqAvail::ring, VirtqUsed::ring, VirtioInputEvent::type, VirtInputState::used, VirtioInputEvent::value, VIRTIO_INPUT_EV_KEY, and wmb().

Referenced by aarch64_irq_handler(), and riscv64_interrupt_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



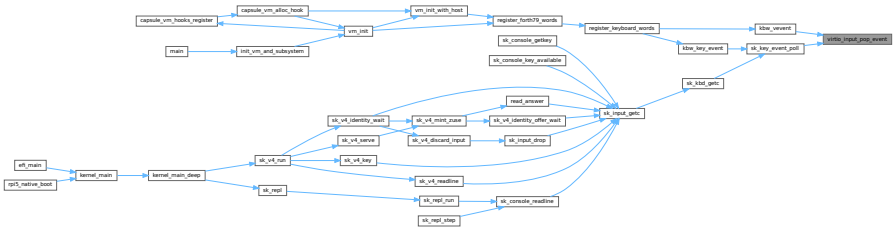
7.179.2.6 virtio_input_pop_event()

```
int virtio_input_pop_event (
            uint16_t * code,
            uint32_t * value )
```

References DiagEvent::code, DIAG_RING_SIZE, g_diag_head, g_diag_ring, g_diag_tail, and DiagEvent::value.

Referenced by kbw_vevent(), and sk_key_event_poll().

Here is the caller graph for this function:



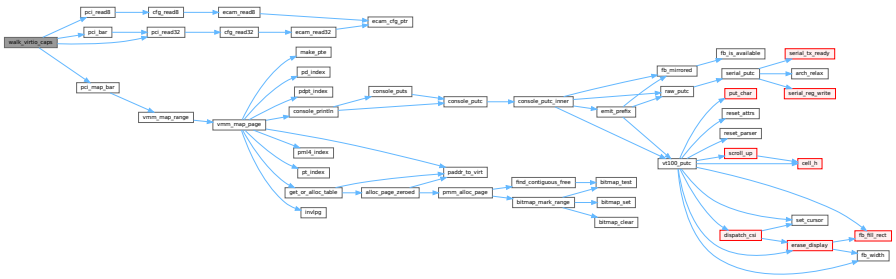
7.179.2.7 walk_virtio_caps()

```
static void * walk_virtio_caps (
            const PciDevice * d,
            uint8_t cap_type,
            uint32_t * extra_out )  [static]
```

References limit, pci_bar(), PCI_CFG_CAP_PTR, pci_map_bar(), pci_read32(), pci_read8(), VCAP_OFF_BAR, VCAP_OFF_CAP_NEXT, VCAP_OFF_CAP_VNDR, VCAP_OFF_CFG_TYPE, VCAP_OFF_LENGTH, VCAP_OFF_NOTIFY_MULT, VCAP_OFF_OFFSET, VIRTIO_PCI_CAP_NOTIFY_CFG, and VIRTIO_PCI_CAP_VENDOR_ID.

Referenced by vinput_init_device().

Here is the call graph for this function:



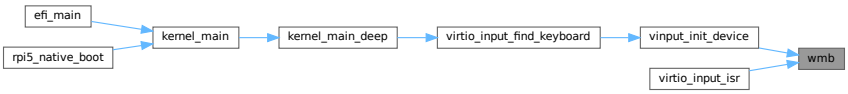
Here is the caller graph for this function:

7.179.2.8 wmb()

```
static void wmb (
            void  )  [inline], [static]
```

Referenced by vinput_init_device(), and virtio_input_isr().

Here is the caller graph for this function:



7.179.3 Variable Documentation

7.179.3.1 g_diag_head

```
volatile uint32_t g_diag_head = 0  [static]
```

Referenced by diag_push(), and virtio_input_pop_event().

7.179.3.2 g_diag_ring

```
DiagEvent g_diag_ring[16u]  [static]
```

Referenced by diag_push(), and virtio_input_pop_event().

7.179.3.3 g_diag_tail

```
volatile uint32_t g_diag_tail = 0  [static]
```

Referenced by diag_push(), and virtio_input_pop_event().

7.179.3.4 g_vinput

```
VirtInputState g_vinput  [static]
```

Referenced by vinput_init_device(), and virtio_input_isr().

7.179.3.5 g_vinput_ready

```
int g_vinput_ready = 0  [static]
```

Referenced by virtio_input_find_keyboard(), and virtio_input_isr().

7.179.3.6 g_virtio_input_gic_intid

```
uint32_t g_virtio_input_gic_intid = 0
```

Referenced by aarch64_irq_handler().

7.179.3.7 g_virtio_input_isr_count

```
volatile uint32_t g_virtio_input_isr_count = 0
```

Referenced by kbw_vdebug(), and virtio_input_isr().

7.179.3.8 g_virtio_input_plic_source

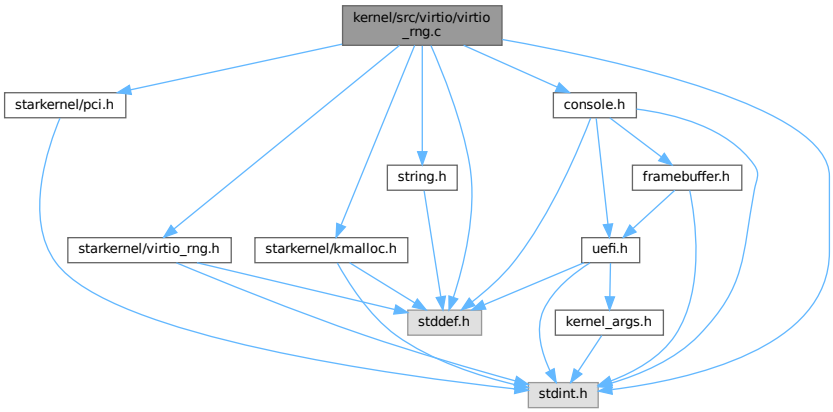
```
uint32_t g_virtio_input_plic_source = 0
```

Referenced by riscv64_interrupt_handler().

7.180 kernel/src/virtio/virtio_rng.c File Reference

```
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include "starkernel/pci.h"
#include "starkernel/virtio_rng.h"
#include "starkernel/kmalloc.h"
#include "console.h"
```

Include dependency graph for virtio_rng.c:



Data Structures

- struct VirtioCommonCfg
- struct VirtqDesc
- struct VirtqAvail
- struct VirtqUsedElem
- struct VirtqUsed
- struct VirtRngState

Macros

- #define VIRTIO_PCI_CAP_VENDOR_ID 0x09u
- #define VIRTIO_PCI_CAP_COMMON_CFG 1u
- #define VIRTIO_PCI_CAP_NOTIFY_CFG 2u
- #define VCAP_OFF_CAP_VNDR 0u
- #define VCAP_OFF_CAP_NEXT 1u
- #define VCAP_OFF_CFG_TYPE 3u
- #define VCAP_OFF_BAR 4u
- #define VCAP_OFF_OFFSET 8u
- #define VCAP_OFF_LENGTH 12u
- #define VCAP_OFF_NOTIFY_MULT 16u
- #define VIRTIO_STATUS_ACKNOWLEDGE 0x01u
- #define VIRTIO_STATUS_DRIVER 0x02u
- #define VIRTIO_STATUS_DRIVER_OK 0x04u
- #define VIRTIO_STATUS_FEATURES_OK 0x08u
- #define VIRTIO_STATUS_FAILED 0x80u
- #define VIRTIO_F_VERSION_1 (1ULL << 32)
- #define VQUEUE_SIZE 1u
- #define VRING_DESC_F_WRITE 2u
- #define VRNG_BUF_SIZE 64u /* bytes requested from the device per round */

Functions

- static void * walk_virtio_caps (const PciDevice *d, uint8_t cap_type, uint32_t *extra_out)
- static void wmb (void)
- static void rmb (void)
- static int vrng_request (uint32_t *bytes_out)
- static int vrng_init_device (const PciDevice *pci)
- int virtio_rng_init (void)
- int virtio_rng_ready (void)
- int virtio_rng_get_bytes (uint8_t *buf, size_t n)

Variables

- static VirtRngState g_vrng
- static int g_vrng_ready = 0

7.180.1 Macro Definition Documentation

7.180.1.1 VCAP_OFF_BAR

```
#define VCAP_OFF_BAR 4u
```

7.180.1.2 VCAP_OFF_CAP_NEXT

```
#define VCAP_OFF_CAP_NEXT 1u
```

7.180.1.3 VCAP_OFF_CAP_VNDR

```
#define VCAP_OFF_CAP_VNDR 0u
```

7.180.1.4 VCAP_OFF_CFG_TYPE

```
#define VCAP_OFF_CFG_TYPE 3u
```

7.180.1.5 VCAP_OFF_LENGTH

```
#define VCAP_OFF_LENGTH 12u
```

7.180.1.6 VCAP_OFF_NOTIFY_MULT

```
#define VCAP_OFF_NOTIFY_MULT 16u
```

7.180.1.7 VCAP_OFF_OFFSET

```
#define VCAP_OFF_OFFSET 8u
```

7.180.1.8 VIRTIO_F_VERSION_1

```
#define VIRTIO_F_VERSION_1 (1ULL << 32)
```

7.180.1.9 VIRTIO_PCI_CAP_COMMON_CFG

```
#define VIRTIO_PCI_CAP_COMMON_CFG 1u
```

7.180.1.10 VIRTIO_PCI_CAP_NOTIFY_CFG

```
#define VIRTIO_PCI_CAP_NOTIFY_CFG 2u
```

7.180.1.11 VIRTIO_PCI_CAP_VENDOR_ID

```
#define VIRTIO_PCI_CAP_VENDOR_ID 0x09u
```

7.180.1.12 VIRTIO_STATUS_ACKNOWLEDGE

```
#define VIRTIO_STATUS_ACKNOWLEDGE 0x01u
```

7.180.1.13 VIRTIO_STATUS_DRIVER

```
#define VIRTIO_STATUS_DRIVER 0x02u
```

7.180.1.14 VIRTIO_STATUS_DRIVER_OK

```
#define VIRTIO_STATUS_DRIVER_OK 0x04u
```

7.180.1.15 VIRTIO_STATUS_FAILED

```
#define VIRTIO_STATUS_FAILED 0x80u
```

7.180.1.16 VIRTIO_STATUS_FEATURES_OK

```
#define VIRTIO_STATUS_FEATURES_OK 0x08u
```

7.180.1.17 VQUEUE_SIZE

```
#define VQUEUE_SIZE 1u
```

7.180.1.18 VRING_DESC_F_WRITE

```
#define VRING_DESC_F_WRITE 2u
```

7.180.1.19 VRNG_BUF_SIZE

```
#define VRNG_BUF_SIZE 64u /* bytes requested from the device per round */
```

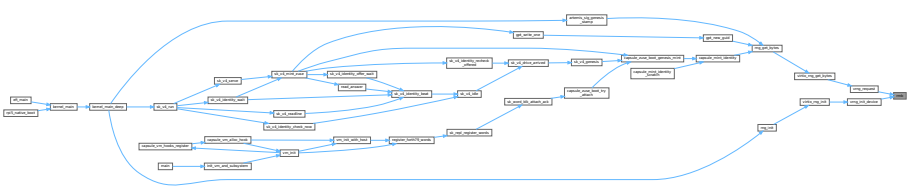
7.180.2 Function Documentation

7.180.2.1 rmb()

```
static void rmb (
            void  )  [inline], [static]
```

Referenced by vrng_init_device(), and vrng_request().

Here is the caller graph for this function:



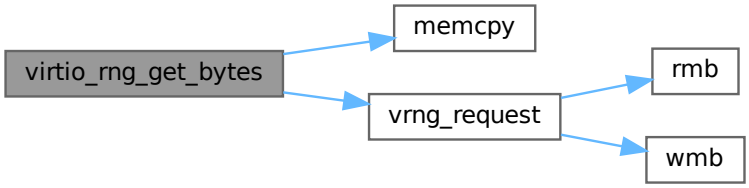
7.180.2.2 virtio_rng_get_bytes()

```
int virtio_rng_get_bytes (
            uint8_t * buf,
            size_t n )
```

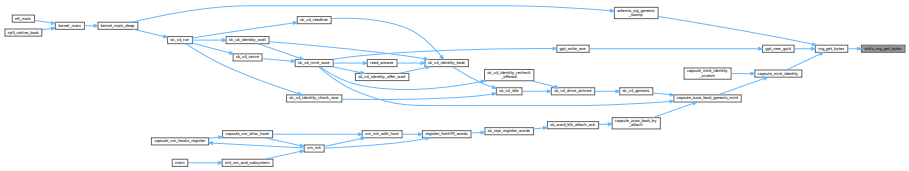
References VirtRngState::data_buf, g_vrng, g_vrng_ready, memcpy(), and vrng_request().

Referenced by rng_get_bytes().

Here is the call graph for this function:



Here is the caller graph for this function:



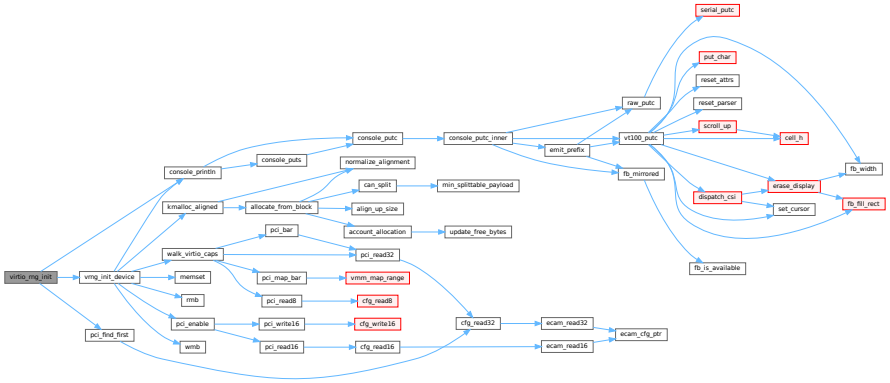
7.180.2.3 virtio_rng_init()

```
int virtio_rng_init (
            void  )
```

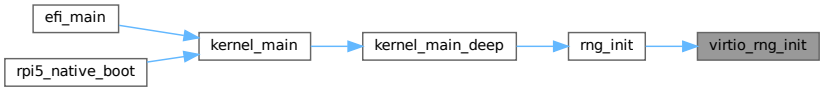
References console_println(), g_vrng_ready, pci_find_first(), VIRTIO_RNG_DEVICE_LEGACY, VIRTIO_RNG_DEVICE_MODERN, VIRTIO_RNG_PCI_VENDOR_ID, and vrng_init_device().

Referenced by rng_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.180.2.4 virtio_rng_ready()

```
int virtio_rng_ready (
            void  )
```

References g_vrng_ready.

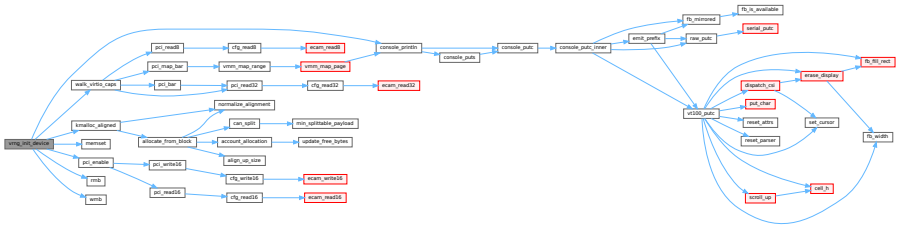
7.180.2.5 vrng_init_device()

```
static int vrng_init_device (
            const PciDevice * pci )  [static]
```

References VirtRngState::avail, VirtRngState::avail_idx, VirtRngState::common, VirtioCommonCfg::config_msix_vector, console_println(), VirtRngState::data_buf, VirtRngState::desc, VirtioCommonCfg::device_status, VirtioCommonCfg::driver_feature, VirtioCommonCfg::driver_feature_select, g_vrng, kmalloc_aligned(), VirtRngState::last_used_idx, memset(), VirtRngState::notify, VirtRngState::notify_off_mult, pci_enable(), VirtioCommonCfg::queue_desc, VirtioCommonCfg::queue_device, VirtioCommonCfg::queue_driver, VirtioCommonCfg::queue_enable, VirtioCommonCfg::queue_msix_vector, VirtioCommonCfg::queue_notify_off, VirtRngState::queue_notify_off, VirtioCommonCfg::queue_select, VirtioCommonCfg::queue_siz rmb(), VirtRngState::used, VIRTIO_F_VERSION_1, VIRTIO_PCI_CAP_COMMON_CFG, VIRTIO_PCI_CAP_NOTIFY_CFG, VIRTIO_STATUS_ACKNOWLEDGE, VIRTIO_STATUS_DRIVER, VIRTIO_STATUS_DRIVER_OK, VIRTIO_STATUS_FAILED, VIRTIO_STATUS_FEATURES_OK, VQUEUE_SIZE, VRNG_BUF_SIZE, walk_virtio_caps(), and wmb().

Referenced by virtio_rng_init().

Here is the call graph for this function:



Here is the caller graph for this function:

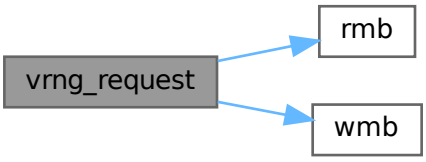
7.180.2.6 vrng_request()

```
static int vrng_request (
            uint32_t * bytes_out ) [static]
```

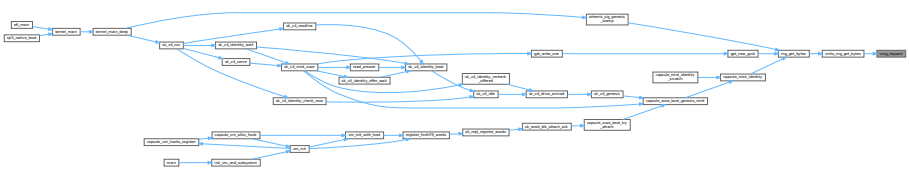
References VirtqDesc::addr, VirtRngState::avail, VirtRngState::avail_idx, VirtRngState::data_buf, VirtRngState::desc, VirtqDesc::flags, g_vrng, VirtqAvail::idx, VirtqUsed::idx, VirtRngState::last_used_idx, VirtqDesc::len, VirtqUsedElem::len, VirtqDesc::next, VirtRngState::notify, VirtRngState::notify_off_mult, VirtRngState::queue_notify_off, VirtqAvail::ring, VirtqUsed::ring, rmb(), VirtRngState::used, VQUEUE_SIZE, VRING_DESC_F_WRITE, VRNG_BUF_SIZE, and wmb().

Referenced by virtio_rng_get_bytes().

Here is the call graph for this function:



Here is the caller graph for this function:



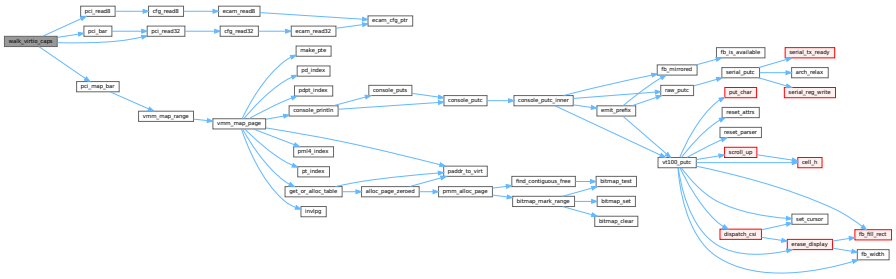
7.180.2.7 walk_virtio_caps()

```
static void * walk_virtio_caps (
            const PciDevice * d,
            uint8_t cap_type,
            uint32_t * extra_out )  [static]
```

References limit, pci_bar(), PCI_CFG_CAP_PTR, pci_map_bar(), pci_read32(), pci_read8(), VCAP_OFF_BAR, VCAP_OFF_CAP_NEXT, VCAP_OFF_CAP_VNDR, VCAP_OFF_CFG_TYPE, VCAP_OFF_LENGTH, VCAP_OFF_NOTIFY_MULT, VCAP_OFF_OFFSET, VIRTIO_PCI_CAP_NOTIFY_CFG, and VIRTIO_PCI_CAP_VENDOR_ID.

Referenced by vrng_init_device().

Here is the call graph for this function:



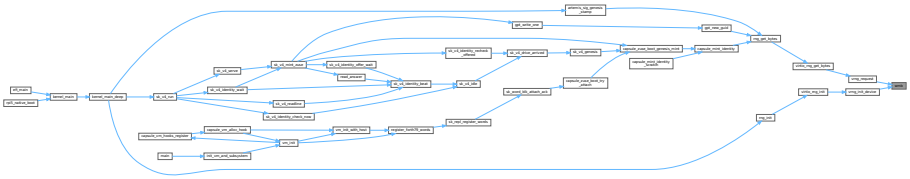
Here is the caller graph for this function:

7.180.2.8 wmb()

```
static void wmb (
            void  )  [inline], [static]
```

Referenced by vrng_init_device(), and vrng_request().

Here is the caller graph for this function:



7.180.3 Variable Documentation

7.180.3.1 g_vrng

```
VirtRngState g_vrng  [static]
```

Referenced by virtio_rng_get_bytes(), vrng_init_device(), and vrng_request().

7.180.3.2 g_vrng_ready

```
int g_vrng_ready = 0  [static]
```

Referenced by virtio_rng_get_bytes(), virtio_rng_init(), and virtio_rng_ready().

7.181 kernel/src/vm/arena.c File Reference

7.182 kernel/src/vm/bootstrap/sk_vm_bootstrap.c File Reference

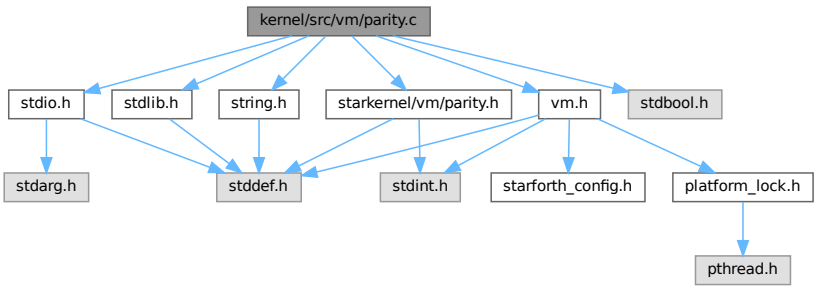
7.183 kernel/src/vm/host/shim.c File Reference

7.184 kernel/src/vm/kernel_hermes.c File Reference

7.185 kernel/src/vm/parity.c File Reference

```
#include "starkernel/vm/parity.h"
#include "vm.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
```

Include dependency graph for parity.c:



Macros

- #define SK_PARITY_DEBUG 0

  parity.c - Parity packet collection and canonical hash
- #define sk_parity_debug_log_msg(msg) do { (void)(msg); } while (0)
- #define sk_parity_debug_check_ptr(vm, label, entry, idx, hptr, xt, ptr) do { (void)(vm); (void)(label); (void)(entry); (void)(idx); (void)(hptr); (void)(xt); (void)(ptr); } while (0)
- #define MAX_DICT_ENTRIES 2048

Functions

- static void print_u32 (uint32_t val)

  Print a `uint32_t` value as decimal to the kernel console or stdout.
- static void print_hex64 (uint64_t val)

  Print a `uint64_t` value as a 16-digit hex string with "0x" prefix.
- static void print_str (const char *s)

  Print a null-terminated string to the kernel console or stdout.
- static void print_nl (void)

  Print a newline to the kernel console or stdout.
- uint64_t fnv1a_64 (const uint8_t *data, size_t len, uint64_t hash)

  FNV-1a 64-bit incremental hash over a byte buffer.
- static uint64_t fnv1a_64_u8 (uint8_t byte, uint64_t hash)

  FNV-1a 64-bit incremental hash over a single byte.
- static uint64_t fnv1a_64_u32 (uint32_t val, uint64_t hash)

  FNV-1a 64-bit incremental hash over a 32-bit value (little-endian).
- uint32_t sk_dict_word_count (struct VM *vm)

  Count the number of entries in the VM dictionary.
- static uint64_t hash_colon_body (struct VM *vm, DictEntry *entry, uint32_t word_index, uint64_t hash)

  Hash the compiled body of a colon definition as a sequence of word IDs.
- uint64_t sk_dict_canonical_hash (struct VM *vm)

  Compute the canonical FNV-1a 64-bit hash of the VM dictionary.
- void sk_parity_collect (struct VM *vm, ParityPacket *out)

  Collect M7 parity data from the VM into a `ParityPacket`.
- void sk_parity_print (const ParityPacket *pkt)

  Print a `ParityPacket` to the kernel console or stdout.

7.185.1 Macro Definition Documentation

7.185.1.1 MAX_DICT_ENTRIES

```
#define MAX_DICT_ENTRIES 2048
```

7.185.1.2 SK_PARITY_DEBUG

```
#define SK_PARITY_DEBUG 0
```

parity.c - Parity packet collection and canonical hash

Implements M7 parity validation for hosted vs kernel comparison.

M7 Normative Rules enforced here:

- Rule 1: word_id is monotonic creation index
- Rule 2: Colon bodies hashed as word_id sequence
- Rule 3: Dictionary traversal in creation order

7.185.1.3 sk_parity_debug_check_ptr

```
#define sk_parity_debug_check_ptr(
              vm,
              label,
              entry,
              idx,
              hptr,
              xt,
              ptr )  do { (void)(vm); (void)(label); (void)(entry); (void)(idx); (void)(hptr);
(void)(xt); (void)(ptr); } while (0)
```

7.185.1.4 sk_parity_debug_log_msg

```
#define sk_parity_debug_log_msg(
              msg ) do { (void)(msg); } while (0)
```

7.185.2 Function Documentation

7.185.2.1 fnv1a_64()

```
uint64_t fnv1a_64 (
             const uint8_t * data,
             size_t len,
             uint64_t hash )
```

FNV-1a 64-bit incremental hash over a byte buffer.

Updates a running FNV-1a hash by folding `len` bytes from `data` into `hash`. The caller supplies the current hash state so that multiple buffers can be chained: pass `FNV1A_64_OFFSET_BASIS` for the first call and the returned value as `hash` for subsequent calls.

Algorithm: for each byte `b`, `hash` ^= b then `hash` *= FNV1A_64_PRIME.

Parameters

data	Pointer to the byte buffer to hash.
len	Number of bytes to consume.
hash	Running hash state; seed with `FNV1A_64_OFFSET_BASIS`.

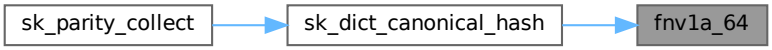
Returns

Updated hash state after processing all `len` bytes.

References FNV1A_64_PRIME.

Referenced by sk_dict_canonical_hash().

Here is the caller graph for this function:



7.185.2.2 fnv1a_64_u32()

```
static uint64_t fnv1a_64_u32 (
            uint32_t val,
            uint64_t hash )  [static]
```

FNV-1a 64-bit incremental hash over a 32-bit value (little-endian).

Folds all four bytes of `val` into `hash` in little-endian order (least-significant byte first) via four `fnv1a_64_u8()` calls. Used to hash `word_id` fields — which are `uint32_t` — into the canonical dictionary hash in a byte-order-stable way.

Parameters

val	32-bit value to hash.
hash	Running FNV-1a hash state.

Returns

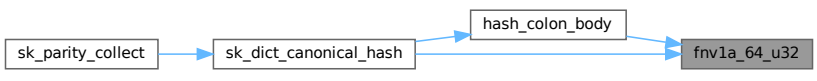
Updated hash state after consuming all four bytes.

References fnv1a_64_u8().

Referenced by hash_colon_body(), and sk_dict_canonical_hash().

Here is the call graph for this function:

Here is the caller graph for this function:



7.185.2.3 fnv1a_64_u8()

```
static uint64_t fnv1a_64_u8 (
            uint8_t byte,
            uint64_t hash )  [static]
```

FNV-1a 64-bit incremental hash over a single byte.

Convenience wrapper for folding one byte into a running FNV-1a hash state. Used by `fnv1a_64_u32()` to hash multi-byte values byte-by-byte in a defined endian order.

Parameters

byte	Single byte to fold into the hash.
hash	Running hash state.

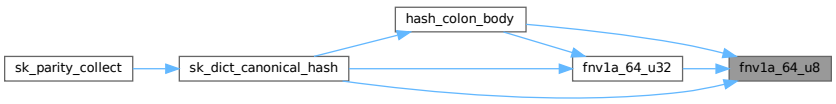
Returns

Updated hash state.

References FNV1A_64_PRIME.

Referenced by fnv1a_64_u32(), hash_colon_body(), and sk_dict_canonical_hash().

Here is the caller graph for this function:



7.185.2.4 hash_colon_body()

```
static uint64_t hash_colon_body (
            struct VM * vm,
            DictEntry * entry,
            uint32_t word_index,
            uint64_t hash )  [static]
```

Hash the compiled body of a colon definition as a sequence of word IDs.

Implements M7 Rule 2: colon bodies are hashed as their `word_id` sequence, not as raw code pointers, ensuring the hash is stable across address-space layout changes (ASLR, relocation, different build addresses).

Traversal reads each cell from the threaded body in order:

- A non-NULL `DictEntry` pointer → its `word_id` is hashed.
- An `EXIT` entry → its `word_id` is hashed and traversal stops.
- A `LIT` entry → its `word_id` is hashed and the following cell (the literal value) is hashed as eight individual bytes (little-endian).
- A `NULL` cell → traversal stops (end-of-body sentinel).

Returns `hash` unchanged if `entry` lacks the `WORD_COMPILED` flag or has no valid data-field. Capped at 1024 cells per body as a safety limit.

Parameters

vm	Active VM instance.
entry	Dictionary entry whose body will be hashed; must have `WORD_COMPILED` set.
word_index	Creation-order ordinal of `entry` (used by debug mode).
hash	Running FNV-1a hash state from the caller.

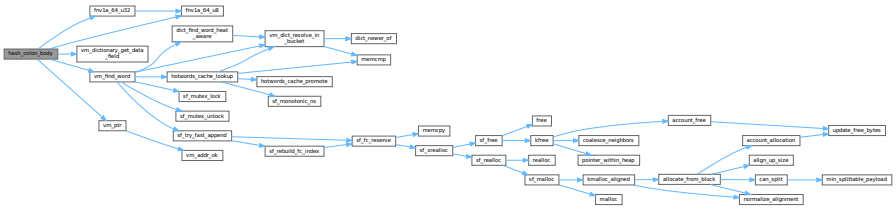
Returns

Updated hash state after processing the colon body.

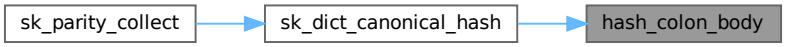
References entry, fnv1a_64_u32(), fnv1a_64_u8(), DictEntry::func, VM::memory, sk_parity_debug_check_ptr, sk_parity_debug_log_msg, vm_dictionary_get_data_field(), vm_find_word(), VM_MEMORY_SIZE, vm_ptr(), WORD_COMPILED, and DictEntry::word_id.

Referenced by sk_dict_canonical_hash().

Here is the call graph for this function:



Here is the caller graph for this function:



7.185.2.5 print_hex64()

```
static void print_hex64 (
            uint64_t val )  [static]
```

Print a `uint64_t` value as a 16-digit hex string with "0x" prefix.

Formats `val` as "0xNNNNNNNNNNNNNNNN" using a fixed-size local buffer (no libc required) and emits it via `console_puts()` in kernel builds or `printf()` in hosted builds. Used by `sk_parity_print()` to format hash and address fields in parity output lines.

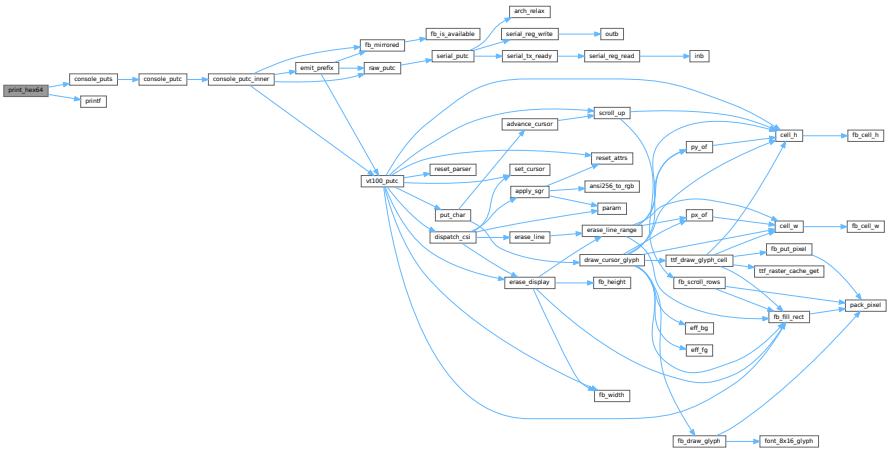
Parameters

val	64-bit value to print as lowercase hex.

References console_puts(), and printf().

Referenced by sk_parity_print().

Here is the call graph for this function:



Here is the caller graph for this function:

7.185.2.6 print_nl()

```
static void print_nl (
            void  )  [static]
```

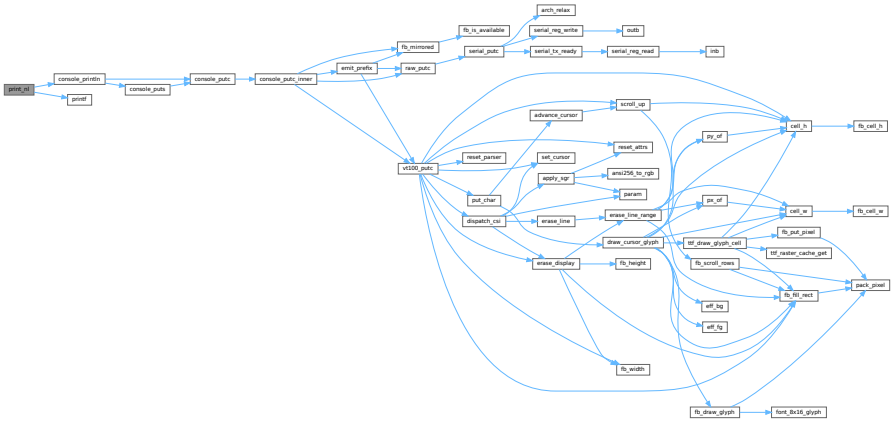
Print a newline to the kernel console or stdout.

Emits a line terminator via `console_println("")` in kernel builds or `printf("\n")` in hosted builds. Paired with `print_str()` to avoid scattering `ifdef` guards through parity formatting code.

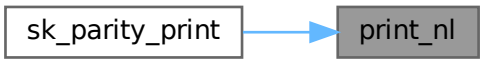
References console_println(), and printf().

Referenced by sk_parity_print().

Here is the call graph for this function:



Here is the caller graph for this function:



7.185.2.7 print_str()

```
static void print_str (
            const char * s )  [static]
```

Print a null-terminated string to the kernel console or stdout.

Emits `s` via `console_puts()` in kernel builds or `printf()` in hosted builds. Serves as a single dispatch point so parity output compiles cleanly in both environments without scattering `ifdef` guards through the formatting code.

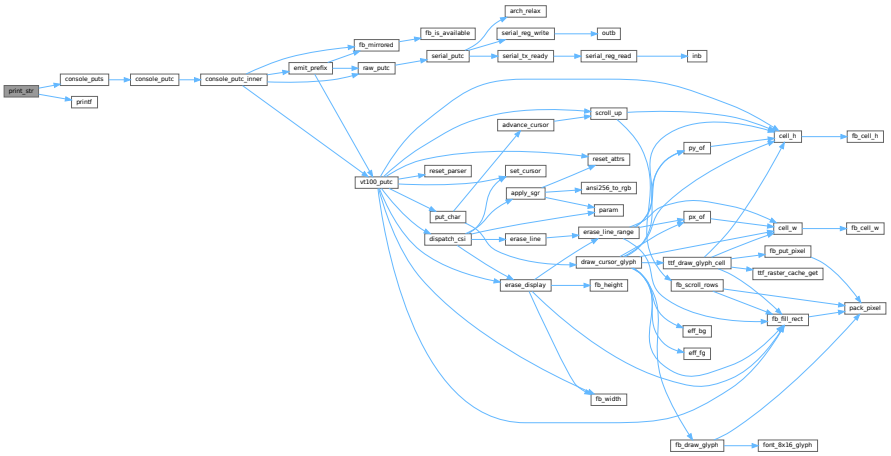
Parameters

s	Null-terminated string to print.

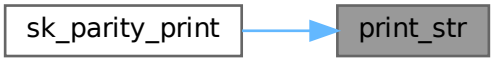
References console_puts(), and printf().

Referenced by sk_parity_print().

Here is the call graph for this function:



Here is the caller graph for this function:



7.185.2.8 print_u32()

```
static void print_u32 (
            uint32_t val )  [static]
```

Print a `uint32_t` value as decimal to the kernel console or stdout.

Converts `val` to a decimal ASCII string using a local buffer (no libc required) and emits it via `console_puts()` in kernel builds or `printf()` in hosted builds. Used by `sk_parity_print()` to format count and ID fields in parity output lines.

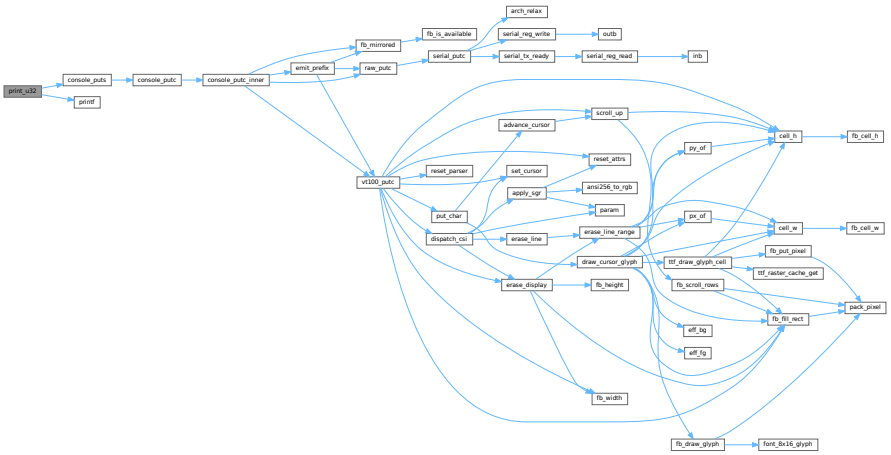
Parameters

val	32-bit unsigned value to print.

References console_puts(), and printf().

Referenced by sk_parity_print().

Here is the call graph for this function:



Here is the caller graph for this function:

7.185.2.9 sk_dict_canonical_hash()

```
uint64_t sk_dict_canonical_hash (
            struct VM * vm )
```

Compute the canonical FNV-1a 64-bit hash of the VM dictionary.

Implements M7 Rule 3: dictionary traversal is in creation order (oldest word first). Because `vm->latest` is the newest word, the function first builds a reverse array of entries and then hashes them in reverse-array order (oldest → newest).

For each entry the following structural fields are folded into the hash:

1. `flags` (1 byte)
2. `name_len` (1 byte)
3. `name` (name_len bytes)
4. `acl_default` (1 byte)
5. `word_id` (4 bytes, little-endian)
6. Colon body (via `hash_colon_body()`) if `WORD_COMPILED` is set

Pointer-valued fields (`func`, `link`) are intentionally excluded so the hash is stable across different build addresses and address-space layouts.

Traversal is capped at `MAX_DICT_ENTRIES` (2048) entries to guard against corruption.

Parameters

vm	Active VM whose dictionary will be hashed; returns 0 if NULL.

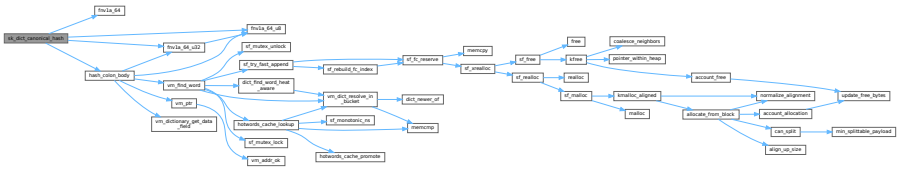
Returns

64-bit FNV-1a canonical hash of the dictionary structure.

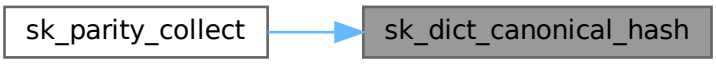
References DictEntry::acl_default, count, DictEntry::flags, fnv1a_64(), FNV1A_64_OFFSET_BASIS, fnv1a_64_u32(), fnv1a_64_u8(), DictEntry::func, hash_colon_body(), VM::latest, MAX_DICT_ENTRIES, DictEntry::name, DictEntry::name_len, sk_parity_debug_check_ptr, sk_parity_debug_log_msg, WORD_COMPILED, and DictEntry::word_id.

Referenced by sk_parity_collect().

Here is the call graph for this function:



Here is the caller graph for this function:



7.185.2.10 sk_dict_word_count()

```
uint32_t sk_dict_word_count (
            struct VM * vm )
```

Count the number of entries in the VM dictionary.

Walks the linked list starting at `vm->latest`, following each entry's `link` pointer until `NULL`. Stops early and returns `MAX_DICT_ENTRIES` if the list exceeds that limit, preventing runaway traversal on a corrupt dictionary.

Parameters

vm	Active VM whose dictionary will be counted; returns 0 if NULL.

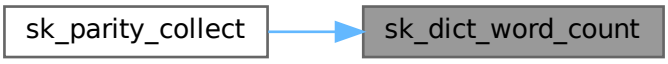
Returns

Number of dictionary entries found, capped at `MAX_DICT_ENTRIES`.

References count, VM::latest, and MAX_DICT_ENTRIES.

Referenced by sk_parity_collect().

Here is the caller graph for this function:



7.185.2.11 sk_parity_collect()

```
void sk_parity_collect (
            struct VM * vm,
            ParityPacket * out )
```

Collect M7 parity data from the VM into a `ParityPacket`.

Populates `out` with a snapshot of the VM's structural state for offline determinism verification. The packet contains:

- M7.1a fields: `word_count`, `here_offset`, `latest_word_id`, and the canonical dictionary hash (`header_hash64`) computed by `sk_dict_canonical_hash()`.
- M7.1b fields: test result counters (`tests_total` / `tests_passed` / `tests_failed` / `tests_skipped` / `tests_errors`) populated from `global_test_stats` when `STARFORTH_ENABLE_TESTS` is defined.

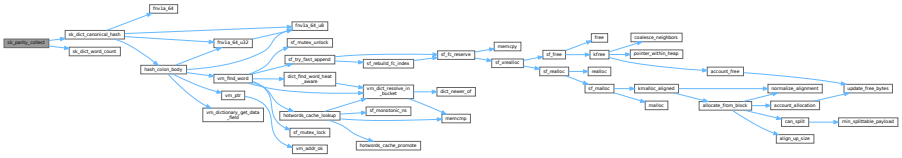
If `vm` is `NULL`, sets `out->bootstrap_result` to `SK_BOOTSTRAP_INIT_FAIL` and returns with all other fields zeroed. If `out` is `NULL` the function is a no-op.

Parameters

vm	Active VM to collect parity data from; may be `NULL`.
out	Output `ParityPacket` to fill; caller must allocate.

References ParityPacket::bootstrap_result, global_test_stats, ParityPacket::header_hash64, VM::here, ParityPacket::here_offset, VM::latest, ParityPacket::latest_word_id, SK_BOOTSTRAP_INIT_FAIL, SK_BOOTSTRAP_OK, sk_dict_canonical_hash(), sk_dict_word_count(), sk_parity_debug_log_msg, ParityPacket::tests_errors, ParityPacket::tests_failed, ParityPacket::tests_passed, ParityPacket::tests_skipped, ParityPacket::tests_total, TestStats::total_error, TestStats::total_fail, TestStats::total_pass, TestStats::total_skip, TestStats::total_tests, ParityPacket::window_hash64, ParityPacket::word_count, and DictEntry::word_id.

Here is the call graph for this function:



7.185.2.12 sk_parity_print()

```
void sk_parity_print (
            const ParityPacket * pkt )
```

Print a ParityPacket to the kernel console or stdout.

Emits two or three structured lines that can be captured in the QEMU serial log for offline determinism verification:

- "PARITY:M7.1a word_count=N here=0xH latest_id=N hash=0xH" — always emitted; contains the structural dictionary fingerprint.
- "PARITY:M7.1b tests=N pass=N fail=N skip=N err=N" — emitted only when pkt->tests_total > 0; contains POST result counts.
- "PARITY:OK" or "PARITY:FAIL code=N" — result verdict line.

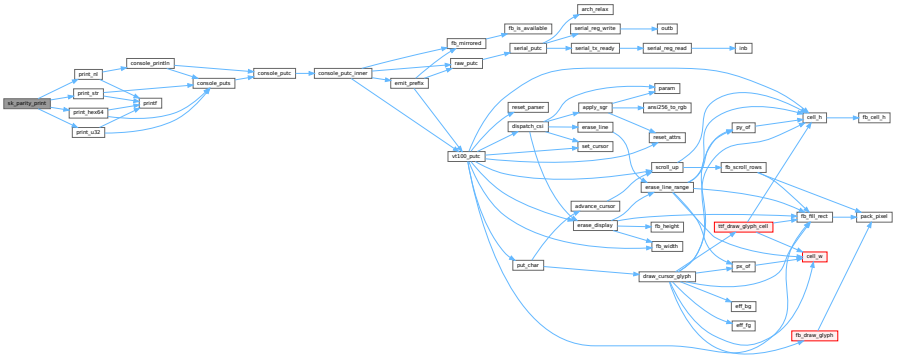
All output goes through print_str() / print_u32() / print_hex64() / print_nl() so the function compiles cleanly in both kernel and hosted builds. No-op if pkt is NULL.

Parameters

pkt	Pointer to the ParityPacket to display; may be NULL.

References ParityPacket::bootstrap_result, ParityPacket::header_hash64, ParityPacket::here_offset, ParityPacket::latest_word_id, print_hex64(), print_nl(), print_str(), print_u32(), SK_BOOTSTRAP_OK, ParityPacket::tests_errors, ParityPacket::tests_failed, ParityPacket::tests_passed, ParityPacket::tests_skipped, ParityPacket::tests_total, and ParityPacket::word_count.

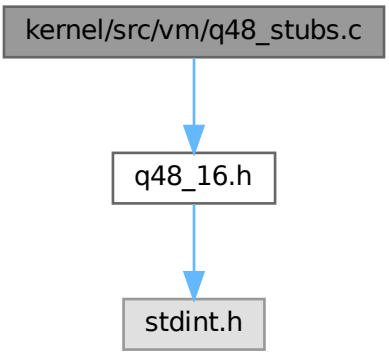
Here is the call graph for this function:



7.186 kernel/src/vm/q48_stubs.c File Reference

```
#include "q48_16.h"
```

Include dependency graph for q48_stubs.c:



Functions

- q48_16_t q48_from_double (double d)
  Convert a `double` to Q48.16 format.
- double q48_to_double (q48_16_t q)
  Convert a Q48.16 value to `double`.

7.186.1 Function Documentation

7.186.1.1 q48_from_double()

```
q48_16_t q48_from_double (
            double d )
```

Convert a `double` to Q48.16 format.

Multiplies `d` by 65536.0 and rounds to nearest. Intended for testing and diagnostics only — not called in the inference engine hot path to avoid floating-point dependency.

Parameters

d	Value to convert

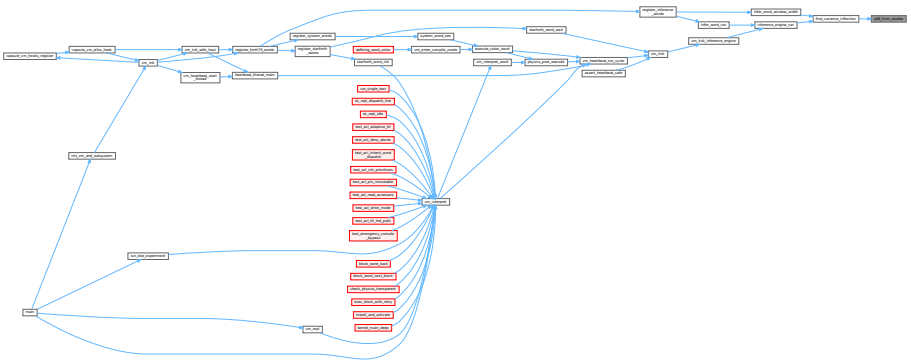
Returns

Nearest Q48.16 representation of `d`

References zero.

Referenced by find_variance_inflection().

Here is the caller graph for this function:



7.186.1.2 q48_to_double()

```
double q48_to_double (
            q48_16_t q )
```

Convert a Q48.16 value to `double`.

Divides `q` by 65536.0. Used for logging and RStudio dashboard output; not called in the inference engine hot path.

Parameters

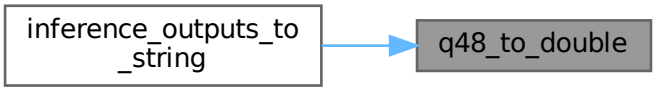
q	Value in Q48.16 format

Returns

Equivalent `double` value

Referenced by inference_outputs_to_string().

Here is the caller graph for this function:

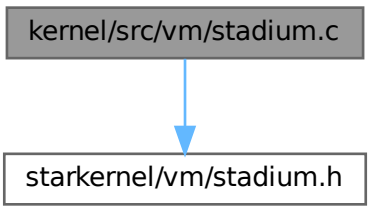


7.187 kernel/src/vm/session.c File Reference

7.188 kernel/src/vm/stadium.c File Reference

```
#include "starkernel/vm/stadium.h"
```

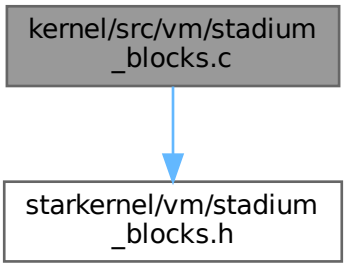
Include dependency graph for stadium.c:



7.189 kernel/src/vm/stadium_blocks.c File Reference

```
#include "starkernel/vm/stadium_blocks.h"
```

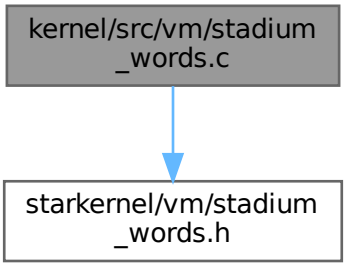
Include dependency graph for stadium_blocks.c:



7.190 kernel/src/vm/stadium_words.c File Reference

```
#include "starkernel/vm/stadium_words.h"
```

Include dependency graph for stadium_words.c:



7.191 kernel/src/vm/switch.c File Reference

7.192 kernel/src/vm/vm_core.c File Reference

```
#include "../../../v3/include/vm.h"
#include "../../../v3/include/inference_engine.h"
#include "../../../v3/include/log.h"
#include "../../../v3/include/word_registry.h"
#include "../../../v3/include/vm_debug.h"
#include "../../../v3/include/profiler.h"
#include "../../../v3/include/platform_time.h"
#include "../../../v3/include/physics_metadata.h"
#include "../../../v3/include/physics_hotwords_cache.h"
#include "../../../v3/include/physics_pipelining_metrics.h"
#include "../../../v3/include/rolling_window_of_truth.h"
#include "../../../v3/include/dictionary_heat_optimization.h"
#include "../../../v3/include/ssm_jacquard.h"
#include "../../../v3/include/vm_host.h"
#include "word_source/include/vocabulary_words.h"
#include "vm_internal.h"
#include <stdlib.h>
#include <errno.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>
```

Include dependency graph for vm_core.c:

Macros

- #define vm_check_arena(tag) ((void)0)

Functions

- static void * hosted_alloc (size_t size, size_t align)
- static void hosted_free (void *ptr)
- static uint64_t hosted_time_ns (void)
- static int hosted_mutex_init (void *mutex)
- static int hosted_mutex_lock (void *mutex)
- static int hosted_mutex_unlock (void *mutex)
- static void hosted_mutex_destroy (void *mutex)
- static int hosted_puts (const char *str)
- static int hosted_putc (int c)
- static void hosted_panic (const char *message)
- static bool hosted_xt_is_executable (const void *ptr)
- static bool hosted_xt_entry_owned (const void *ptr, size_t bytes)
- void vm_reset_hosted_fake_ns (VM *vm)
- static void vm_assert_interpreter_enabled (VM *vm, const char *caller)
- void vm_enable_interpreter (VM *vm)
- const VMHostServices * vm_default_host_services (void)
- const VMHostServices * vm_host (const VM *vm)
- void * vm_host_alloc (VM *vm, size_t size, size_t align)
- void * vm_host_calloc (VM *vm, size_t n, size_t size)
- void vm_host_free (VM *vm, void *ptr)
- uint64_t vm_monotonic_ns (VM *vm)
- unsigned vm_get_base (const VM *vm)

  Read the current number base from the VM.
- void vm_set_base (VM *vm, unsigned b)

  Set the VM number base and update both the cell and the host mirror.
- int vm_zuse_cert_install (VM *vm, const uint8_t seed[32], const uint8_t pubkey[32])

  One-time write of the Zuse cert (seed + derived pubkey) – blows the fuse.
- void vm_cleanup (VM *vm)

  Clean up and free VM resources.
- int vm_parse_word (VM *vm, char *word, size_t max_len)

  Parse the next whitespace-delimited token from the input buffer.
- int vm_parse_number (VM *vm, const char *s, cell_t *out)

  Attempt to parse a string as an integer in the current number base.
- void vm_enter_compile_mode (VM *vm, const char *name, size_t len)

  Begin a colon definition and switch the VM to compile mode.
- void vm_compile_word (VM *vm, DictEntry *entry)

  Append a word reference (DictEntry) to the current threaded body.*
- void vm_compile_literal (VM *vm, cell_t value)

  Compile a literal value into the current threaded body (or push it).
- void vm_compile_call (VM *vm, word_func_t func)

  Compile a call to a word identified by its C function pointer.
- void vm_compile_exit (VM *vm)

  Compile an `EXIT` reference into the current threaded body.
- void vm_exit_compile_mode (VM *vm)

  Finalise a colon definition and return to interpret mode.
- static void acl_recheck (VM *vm, DictEntry *entry)

- void execute_colon_word (VM *vm)

  Inner interpreter for threaded colon definitions.
- void vm_interpret_word (VM *vm, const char *word_str, size_t len)

  Interpret or compile a single word token.
- VM * vm_log_attributed_vm (void)
- int sk_vm_at_outermost_interpret (void)
- void vm_interpret (VM *vm, const char *input)
- int vm_addr_ok (struct VM *vm, vaddr_t addr, size_t len)

  Validate that a VM address range lies within `vm->memory`.
- uint8_t * vm_ptr (struct VM *vm, vaddr_t addr)

  Return a host C pointer into VM memory at `addr`.
- uint8_t vm_load_u8 (struct VM *vm, vaddr_t addr)

  Load an unsigned byte from VM memory.
- void vm_store_u8 (struct VM *vm, vaddr_t addr, uint8_t v)

  Store an unsigned byte to VM memory.
- cell_t vm_load_cell (struct VM *vm, vaddr_t addr)

  Load an aligned cell from VM memory.
- void vm_store_cell (struct VM *vm, vaddr_t addr, cell_t v)

  Store an aligned cell to VM memory.
- void vm_make_immediate (VM *vm)

  Mark the most recently created word as IMMEDIATE (FORTH-79).

Variables

- static uint64_t hosted_fake_ns = 0
- static const VMHostServices hosted_services
- static VM * g_log_attrib_vm
- static int g_vm_interpret_depth

7.192.1 Macro Definition Documentation

7.192.1.1 vm_check_arena

```
#define vm_check_arena(
             tag ) ((void)0)
```

7.192.2 Function Documentation

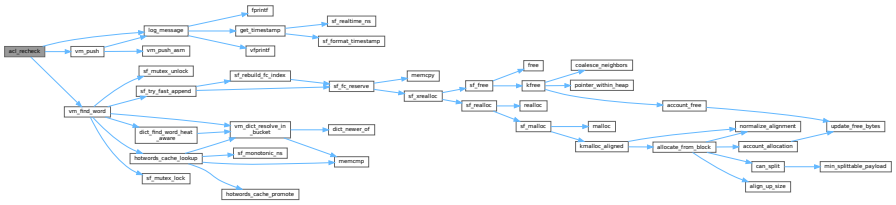
7.192.2.1 acl_recheck()

```
static void acl_recheck (
            VM * vm,
            DictEntry * entry )  [static]
```

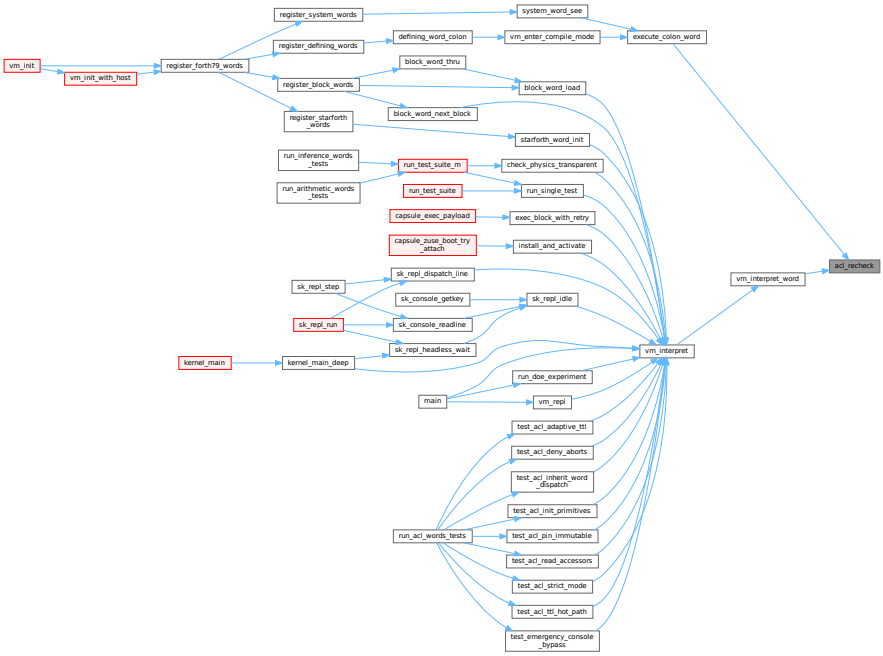
References ACL_TTL_OPEN, VM::current_executing_entry, VM::emergency_console, entry, VM::error, DictEntry::func, log_message(), LOG_WARN, vm_find_word(), and vm_push().

Referenced by execute_colon_word(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.2 execute_colon_word()

```
void execute_colon_word (
            VM * vm )
```

Inner interpreter for threaded colon definitions.

Runs the body of a colon-defined word whose `DictEntry` points to a sequence of `DictEntry*` cells in VM memory (indirect threaded code). The interpreter pointer (IP) is maintained on the return stack so that runtime control-flow words ((BRANCH), (0BRANCH), (DO), etc.) can redirect execution by modifying the top-of-return-stack IP.

Execution loop per iteration:

1. Read `DictEntry* w` from `*ip`.
2. Run physics pre-execute hooks (heat, decay, pipelining).

3. ACL hot-path check (decrement TTL; call `acl_recheck()` on zero).
4. Push `ip+1` onto the return stack as the resume point.
5. Call `w->func(vm)`.
6. Run physics post-execute hooks.
7. Check `vm->exit_colon` (set by `EXIT`): pop and discard the saved IP, then return to caller.
8. Check `vm->abort_requested`: unwind and return immediately.
9. Pop the (possibly modified) IP from the return stack and repeat.

`vm->ecw_nesting` is incremented on entry and decremented on every return path to enable recursive colon calls.

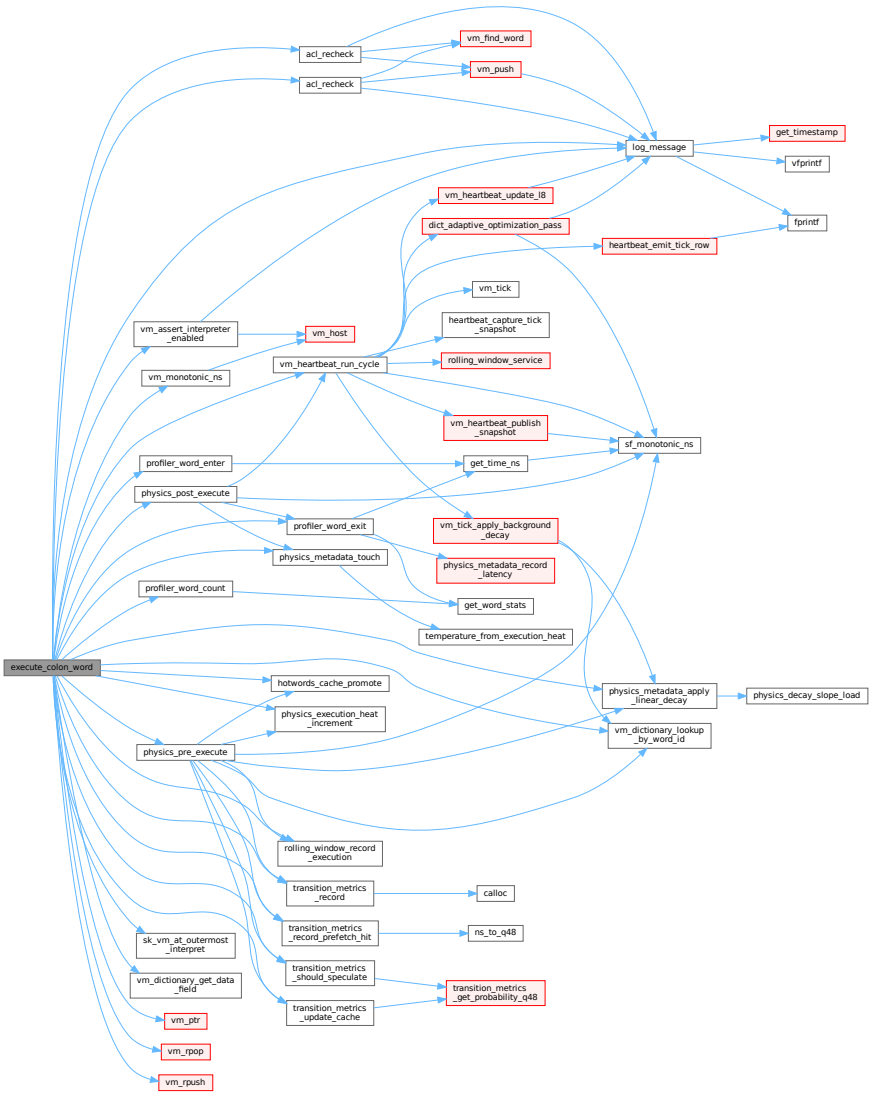
Parameters

vm	VM instance; `vm->current_executing_entry` must point to the `DictEntry` being executed.

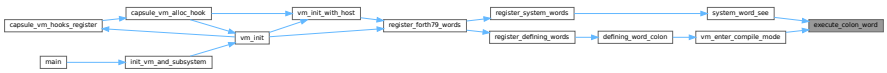
References VM::abort_requested, DictEntry::acl_allow, acl_recheck(), acl_recheck(), DictEntry::acl_ttl, HeartbeatState::check_counte VM::current_executing_entry, DICTIONARY_SIZE, VM::dsp, VM::ecw_nesting, VM::emergency_console, ENABLE_HOTWORDS_CACHE, ENABLE_PIPELINING, entry, VM::error, DictEntry::execution_heat, VM::exit_colon, DictEntry::func, g_sk_fault_word, VM::heartbeat, HEARTBEAT_CHECK_FREQUENCY, VM::hotwords_cache, hotwords_cache_promote(), HOTWORDS_EXECUTION_HEAT_THRESHOLD, DictPhysics::last_active_ns, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, WordTransitionMetrics::most_likely_next_word_id, DictEntry::name, DictEntry::name_len, DictEntry::physics, physics_execution_heat_increment(), physics_metadata_apply_linear_decay(), physics_metadata_touch(), physics_post_execute(), physics_pre_execute(), VM::pipeline_metrics, WordTransitionMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, profiler_word_count(), profiler_word_enter(), profiler_word_exit(), VM::rolling_window, rolling_window_record_execution(), VM::rsp, sk_vm_at_outermost_interpret(), HeartbeatState::tick_count, DictEntry::transition_metrics, transition_metrics_record(), transition_metrics_record_prefetch_hit(), transition_metrics_should_speculate(), transition_metrics_update_cache(), vm_assert_interpreter_enabled(), vm_dictionary_get_data_field(), vm_dictionary_lookup_by_word_id(), vm_heartbeat_run_cycle(), vm_monotonic_ns(), vm_ptr(), vm_rpop(), vm_rpush(), word_id, DictEntry::word_id, HeartbeatState::words_executed, and HeartbeatState::worker.

Referenced by system_word_see(), and vm_enter_compile_mode().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.3 hosted_alloc()

```
static void * hosted_alloc (
            size_t size,
            size_t align )  [static]
```

References malloc().

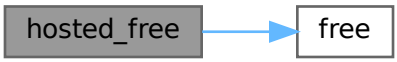
Here is the call graph for this function:

7.192.2.4 hosted_free()

```
static void hosted_free (
            void * ptr )  [static]
```

References free().

Here is the call graph for this function:



7.192.2.5 hosted_mutex_destroy()

```
static void hosted_mutex_destroy (
            void * mutex )  [static]
```

References sf_mutex_destroy().

Here is the call graph for this function:

7.192.2.6 hosted_mutex_init()

```
static int hosted_mutex_init (
            void * mutex )  [static]
```

References sf_mutex_init().

Here is the call graph for this function:

7.192.2.7 hosted_mutex_lock()

```
static int hosted_mutex_lock (
            void * mutex )  [static]
```

References sf_mutex_lock().

Here is the call graph for this function:

7.192.2.8 hosted_mutex_unlock()

```
static int hosted_mutex_unlock (
            void * mutex )  [static]
```

References sf_mutex_unlock().

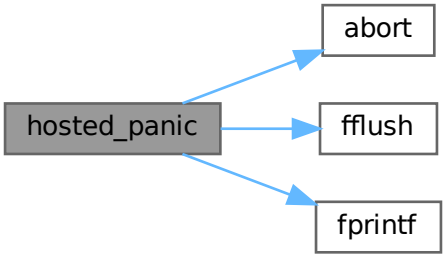
Here is the call graph for this function:

7.192.2.9 hosted_panic()

```
static void hosted_panic (
            const char * message )  [static]
```

References abort(), fflush(), fprintf(), and stderr.

Here is the call graph for this function:



7.192.2.10 hosted_putc()

```
static int hosted_putc (
            int c )  [static]
```

References fputc(), and stdout.

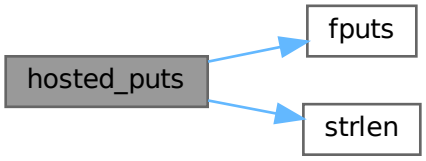
Here is the call graph for this function:

7.192.2.11 hosted_puts()

```
static int hosted_puts (
            const char * str )  [static]
```

References fputs(), stdout, and strlen().

Here is the call graph for this function:

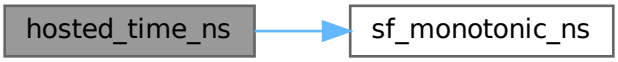


7.192.2.12 hosted_time_ns()

```
static uint64_t hosted_time_ns (
            void  )  [static]
```

References hosted_fake_ns, and sf_monotonic_ns().

Here is the call graph for this function:



7.192.2.13 hosted_xt_entry_owned()

```
static bool hosted_xt_entry_owned (
            const void * ptr,
            size_t bytes )  [static]
```

References bytes.

7.192.2.14 hosted_xt_is_executable()

```
static bool hosted_xt_is_executable (
            const void * ptr )  [static]
```

7.192.2.15 sk_vm_at_outermost_interpret()

```
int sk_vm_at_outermost_interpret (
            void  )
```

References g_vm_interpret_depth.

Referenced by execute_colon_word().

Here is the caller graph for this function:

7.192.2.16 vm_addr_ok()

```
int vm_addr_ok (
            struct VM * vm,
            vaddr_t addr,
            size_t len )
```

Validate that a VM address range lies within `vm->memory`.

Checks that [addr, addr+len) is fully contained within the `VM_MEMORY_SIZE`-byte flat address space. The check is done with unsigned arithmetic to avoid UB on overflow: if `len` alone exceeds `VM_MEMORY_SIZE` the range is immediately rejected.

Parameters

vm	VM instance; returns 0 if NULL or memory not allocated.
addr	Start of the range (inclusive).
len	Number of bytes in the range.

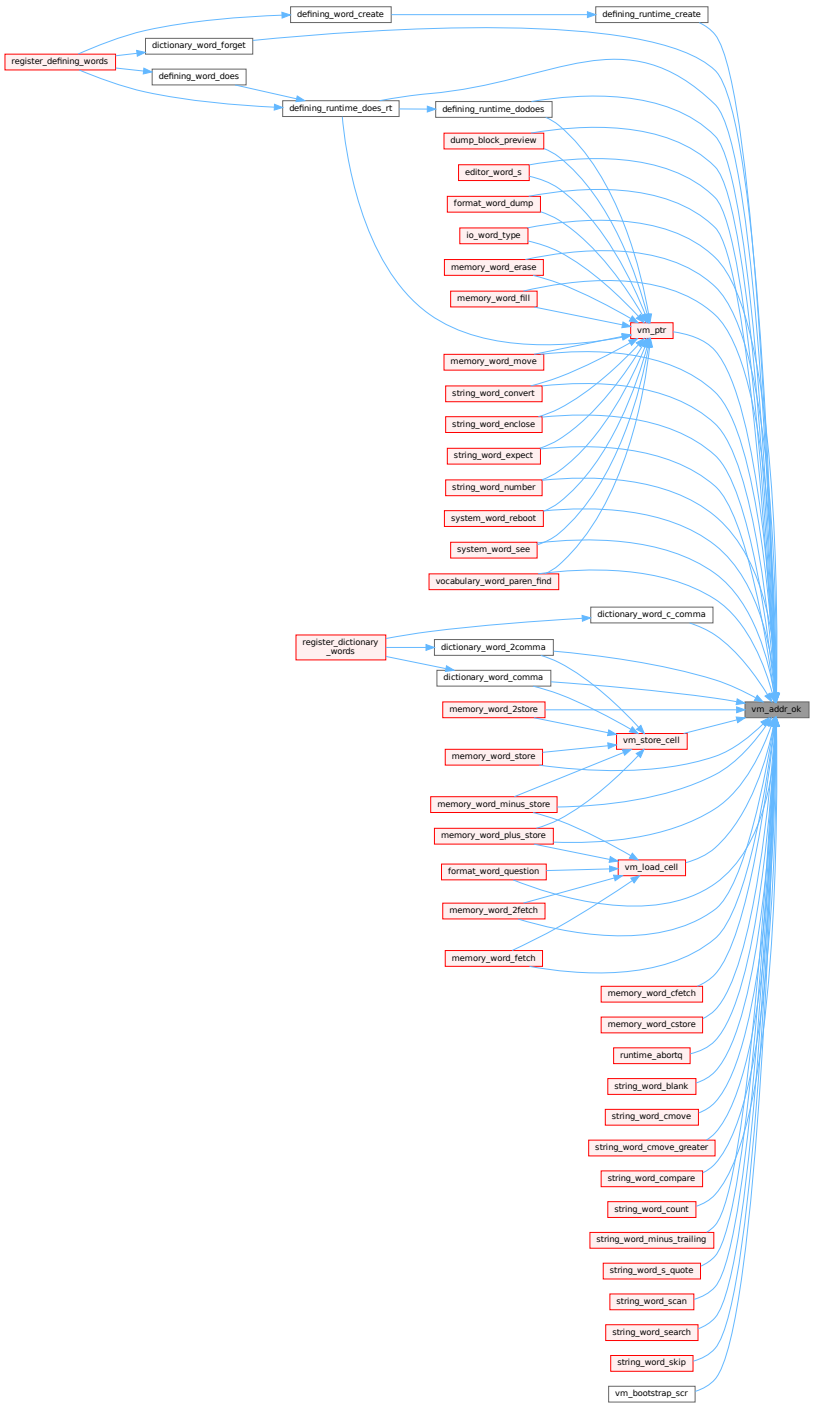
Returns

1 if the full range [addr, addr+len) is in bounds; 0 otherwise.

References VM::memory, and VM_MEMORY_SIZE.

Referenced by defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_does_rt(), dictionary_word_2comma(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_forget(), dump_block_preview(), editor_word_s(), format_word_dump(), format_word_question(), io_word_type(), memory_word_2fetch(), memory_word_2store(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), runtime_abortq(), string_word_blank(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_minus_trailing(), string_word_number(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), system_word_reboot(), system_word_see(), vm_bootstrap_scr(), vm_load_cell(), vm_ptr(), vm_store_cell(), and vocabulary_word_paren_find().

Here is the caller graph for this function:



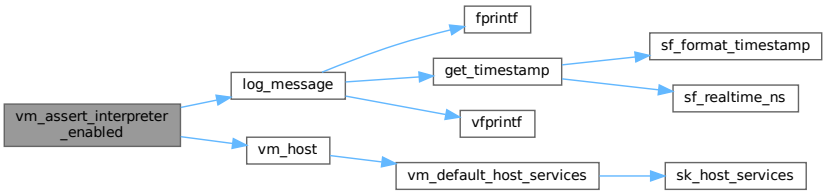
7.192.2.17 vm_assert_interpreter_enabled()

```
static void vm_assert_interpreter_enabled (
            VM * vm,
            const char * caller )  [static]
```

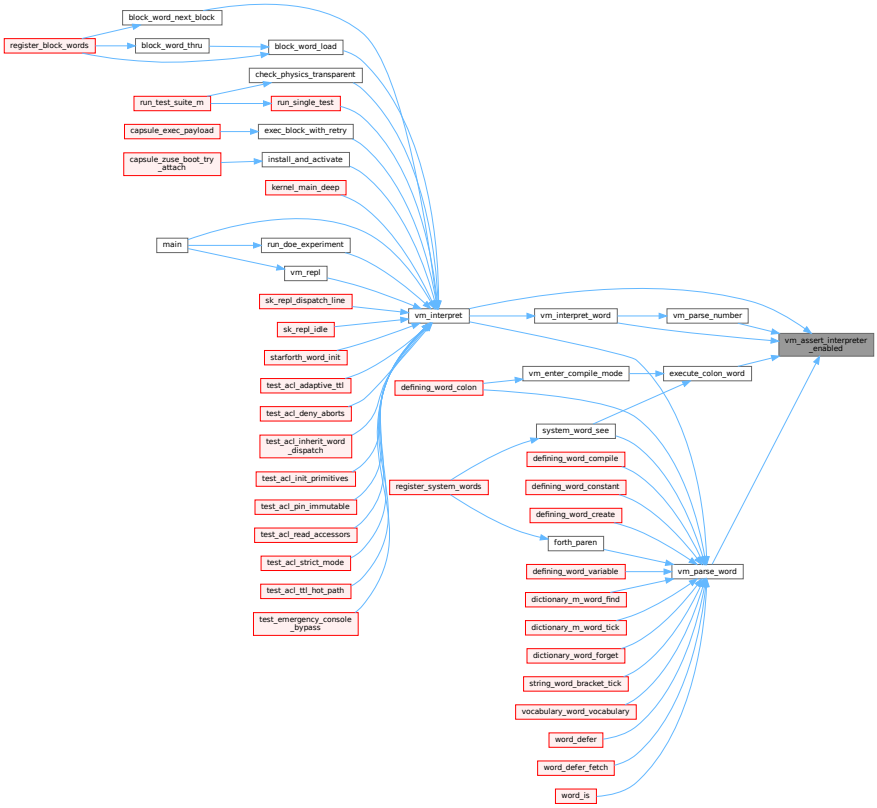
References VM::error, VM::interpreter_enabled, LOG_ERROR, log_message(), and vm_host().

Referenced by execute_colon_word(), vm_interpret(), vm_interpret_word(), vm_parse_number(), and vm_parse_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.18 vm_cleanup()

```
void vm_cleanup (
            VM * vm )
```

Clean up and free VM resources.

Frees allocated memory and resets VM state.

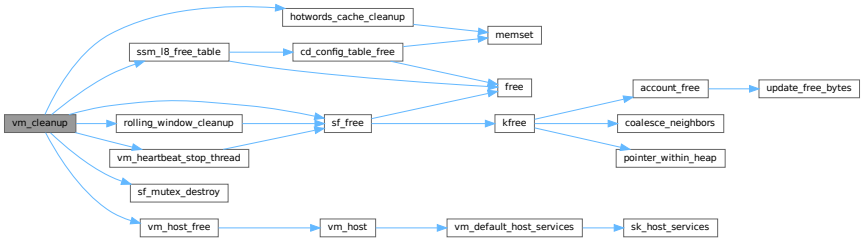
Parameters

vm	Pointer to VM structure to clean up

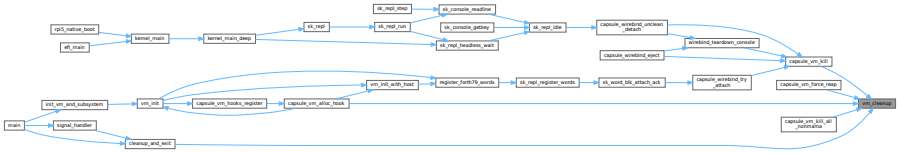
References VM::dict_lock, VM::heartbeat, VM::here, VM::hotwords_cache, hotwords_cache_cleanup(), VM::latest, DictEntry::link, VM::memory, VM::rolling_window, rolling_window_cleanup(), sf_free(), sf_mutex_destroy(), VM::ssm_config, ssm_l8_free_table(), VM::ssm_l8_state, HeartbeatWorker::stop_requested, DictEntry::transition_metrics, VM::tuning_lock, vm_heartbeat_stop_thread(), vm_host_free(), and HeartbeatState::worker.

Referenced by capsule_vm_alloc_hook(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_vm_kill_all_nonmama(), and cleanup_and_exit().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.19 vm_compile_call()

```
void vm_compile_call (
            VM * vm,
            word_func_t func )
```

Compile a call to a word identified by its C function pointer.

Looks up the `DictEntry` whose `func` field matches `func` via `vm_dictionary_find_by_func()`, then delegates to `vm_compile_word()`. Used by generating words (e.g., `DOES>`) that need to embed a specific runtime routine into a compiled definition without knowing its name.

Sets `vm->error` if not in compile mode or if no matching entry is found.

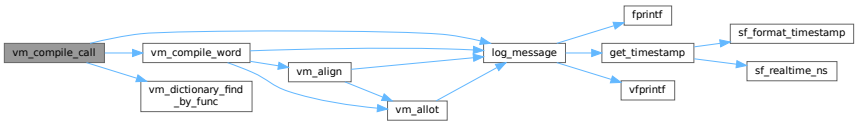
Parameters

vm	VM instance in compile mode.
func	C function pointer identifying the word to compile.

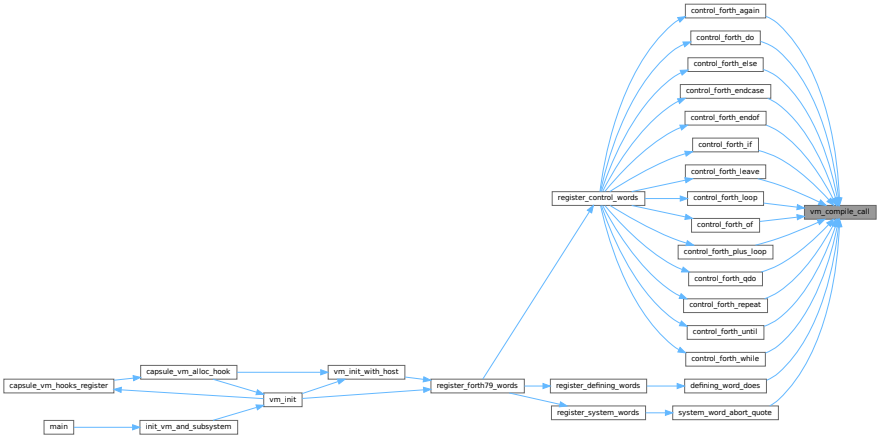
References entry, VM::error, func, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_compile_word(), and vm_dictionary_find_by_func().

Referenced by control_forth_again(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_if(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_until(), control_forth_while(), defining_word_does(), and system_word_abort_quote().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.20 vm_compile_exit()

```
void vm_compile_exit (
            VM * vm )
```

Compile an `EXIT` reference into the current threaded body.

Appends the `EXIT` word entry so that when the threaded body is executed, `execute_colon_word()` sets `vm->exit_colon` and unwinds the inner loop. Used by control-flow generating words (e.g., `IF` optimisation paths) that need an explicit early-return.

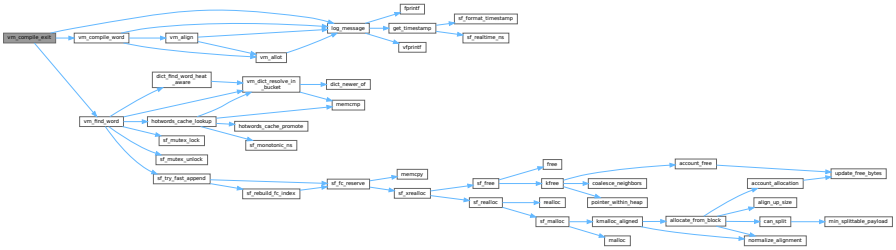
Sets `vm->error` if not in compile mode or if `EXIT` is not found.

Parameters

vm	VM instance in compile mode.

References VM::error, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_compile_word(), and vm_find_word().

Here is the call graph for this function:



7.192.2.21 vm_compile_literal()

```
void vm_compile_literal (
            VM * vm,
            cell_t value )
```

Compile a literal value into the current threaded body (or push it).

In compile mode, appends the `LIT` word reference followed by the literal value so the inner interpreter will push `value` at runtime. In interpret mode, pushes `value` directly onto the data stack.

Sets `vm->error` if `LIT` cannot be found in the dictionary.

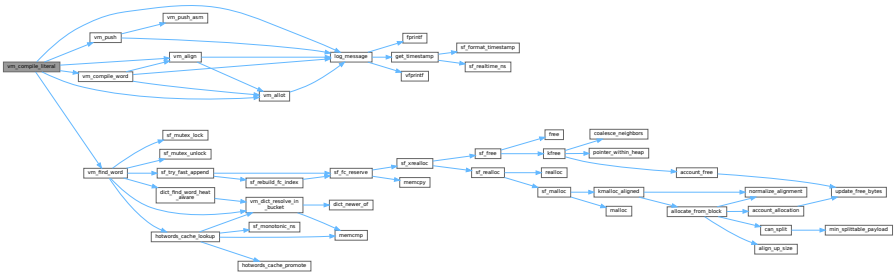
Parameters

vm	VM instance.
value	Cell value to compile or push.

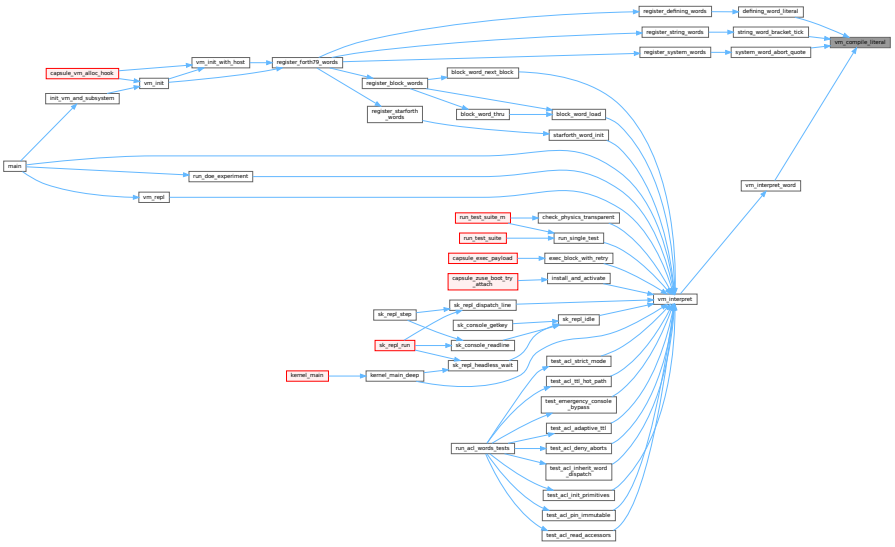
References VM::error, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_align(), vm_allot(), vm_compile_word(), vm_find_word(), and vm_push().

Referenced by defining_word_literal(), string_word_bracket_tick(), system_word_abort_quote(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.22 vm_compile_word()

```
void vm_compile_word (
            VM * vm,
            DictEntry * entry )
```

Append a word reference (DictEntry*) to the current threaded body.

Aligns `HERE`, then allots one `cell_t` slot and stores the `DictEntry*` as a cell. This is the fundamental compilation step used by the inner interpreter: each slot in the threaded body is a pointer to the `DictEntry` of the word to call.

Sets `vm->error` if the VM is not in compile mode, `entry` is NULL, or allotment fails.

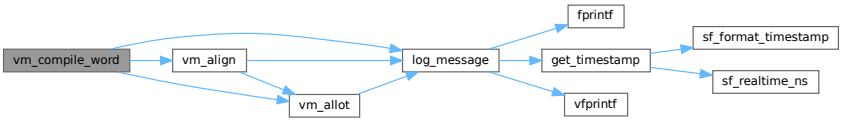
Parameters

vm	VM instance in compile mode.
entry	Dictionary entry of the word to compile; must not be NULL.

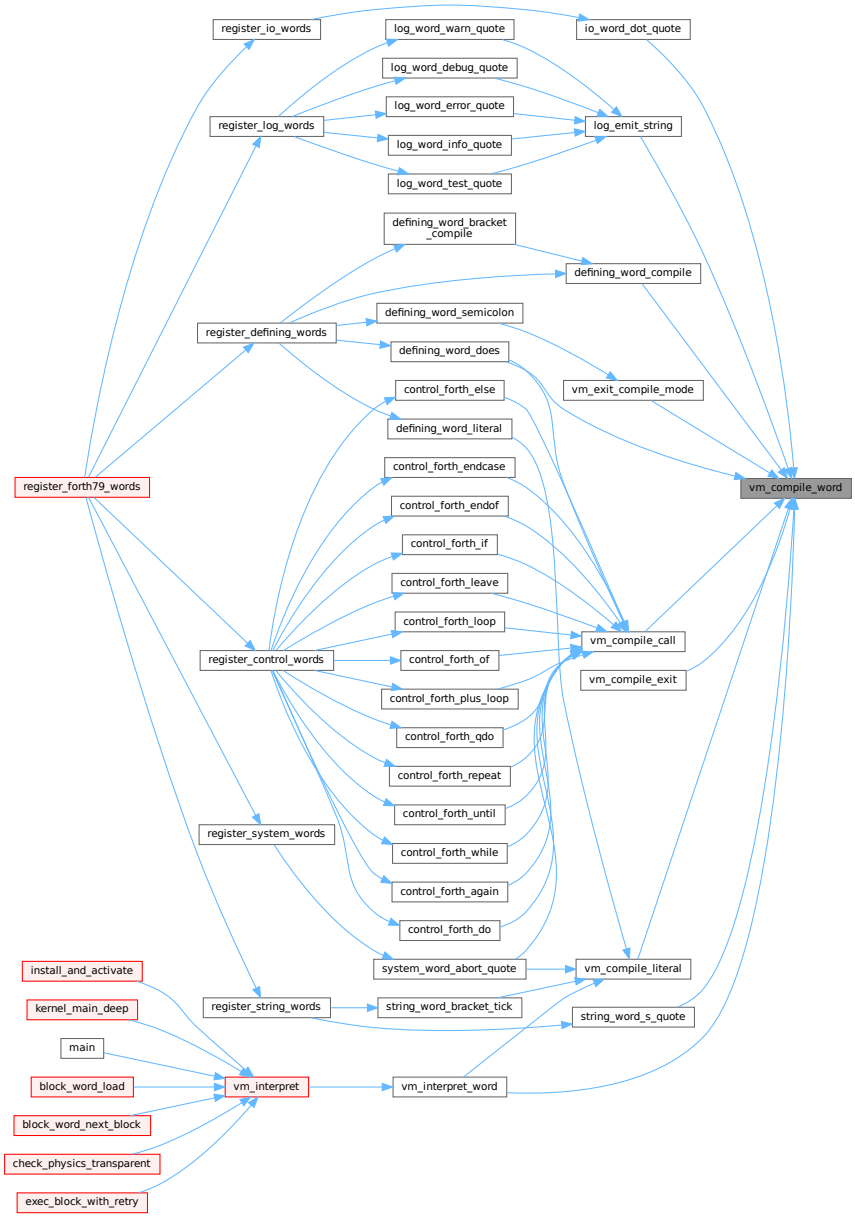
References entry, VM::error, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_align(), and vm_allot().

Referenced by defining_word_compile(), defining_word_does(), io_word_dot_quote(), log_emit_string(), string_word_s_quote(), vm_compile_call(), vm_compile_exit(), vm_compile_literal(), vm_exit_compile_mode(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



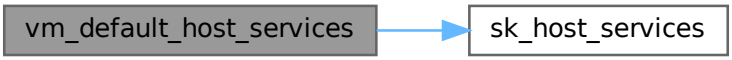
7.192.2.23 vm_default_host_services()

```
const VMHostServices * vm_default_host_services (
            void  )
```

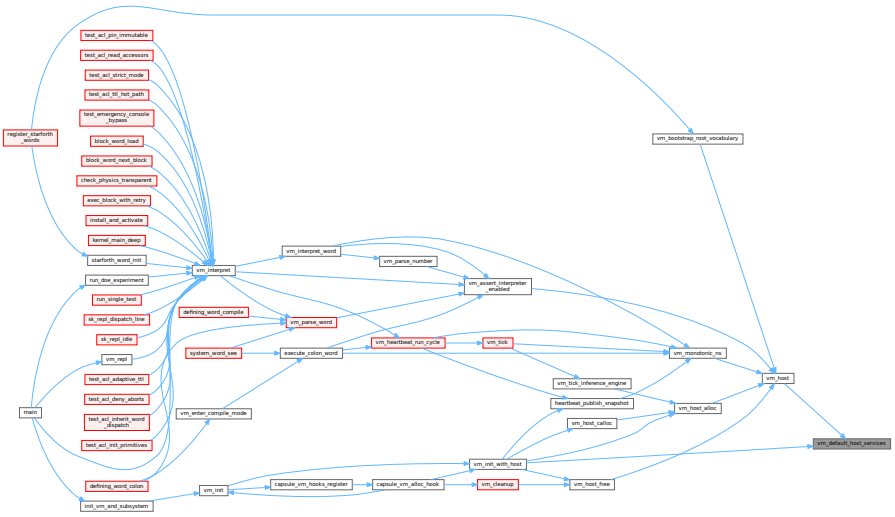
References hosted_services, and sk_host_services().

Referenced by vm_host(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



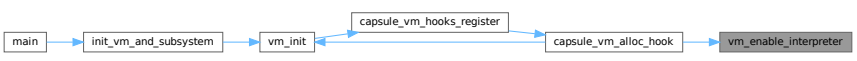
7.192.2.24 vm_enable_interpreter()

```
void vm_enable_interpreter (
            VM * vm )
```

References VM::interpreter_enabled.

Referenced by capsule_vm_alloc_hook().

Here is the caller graph for this function:



7.192.2.25 vm_enter_compile_mode()

```
void vm_enter_compile_mode (
            VM * vm,
            const char * name,
            size_t len )
```

Begin a colon definition and switch the VM to compile mode.

Creates a new dictionary entry for `name` via `vm_create_word()`, marks it `WORD_SMUDGED` so it is invisible to lookups during compilation, sets the VM mode to `MODE_COMPILE`, and writes the current `HERE` address into the entry's data field as the start of the threaded body.

The SMUDGE flag is cleared by `vm_exit_compile_mode()` when the semicolon is encountered.

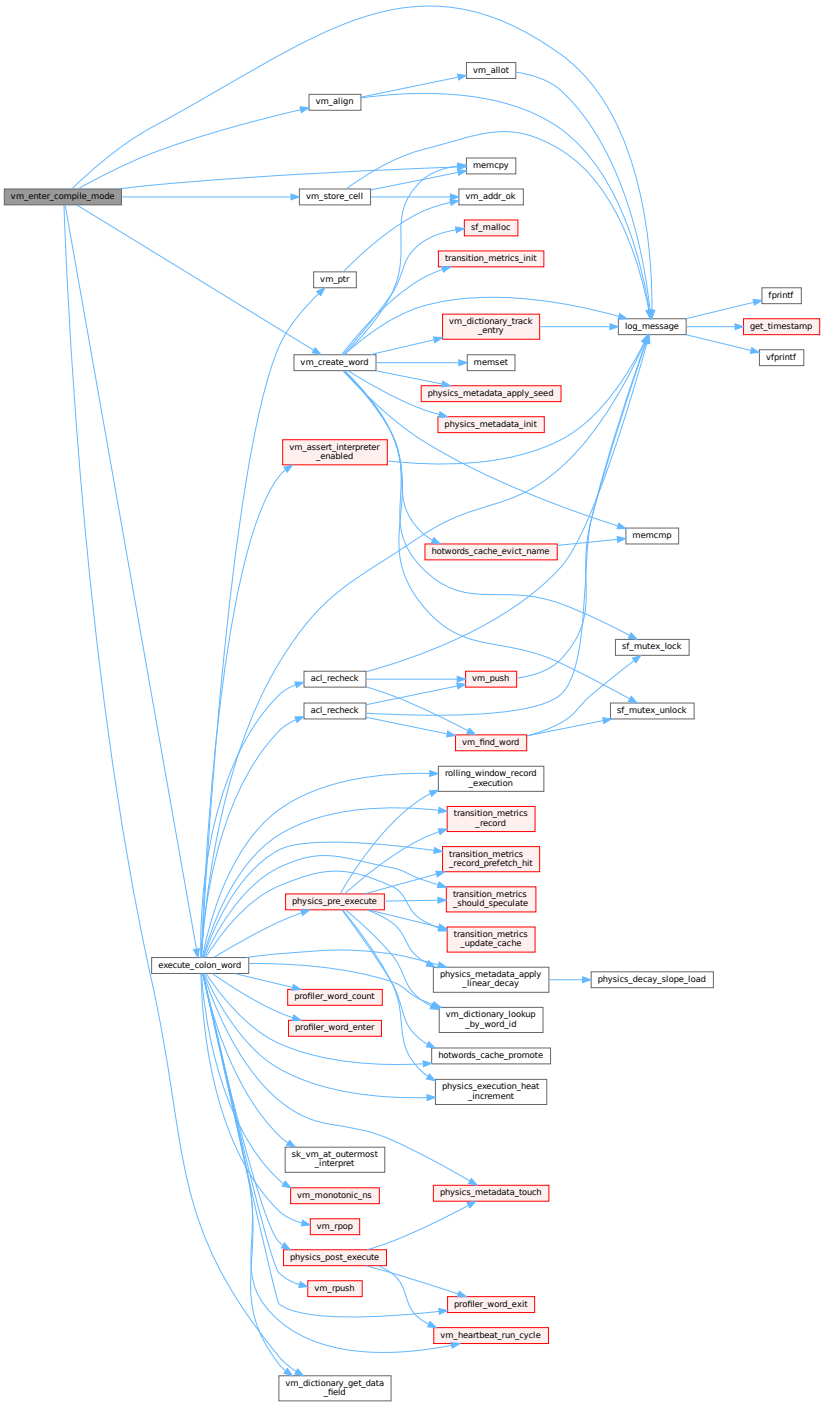
Parameters

vm	VM instance.
name	Word name string (does not need to be NUL-terminated).
len	Length of `name`; clamped to `WORD_NAME_MAX`.

References VM::compiling_word, VM::current_word_name, VM::error, execute_colon_word(), DictEntry::flags, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), memcpy(), VM::mode, MODE_COMPILE, name, VM::state_addr, VM::state_var, vm_align(), vm_create_word(), vm_dictionary_get_data_field(), vm_store_cell(), WORD_NAME_MAX, and WORD_SMUDGED.

Referenced by defining_word_colon().

Here is the call graph for this function:



Here is the caller graph for this function:

7.192.2.26 vm_exit_compile_mode()

```
void vm_exit_compile_mode (
            VM * vm )
```

Finalise a colon definition and return to interpret mode.

Appends a final `EXIT` reference, clears the `WORD_SMUDGED` flag so the new word becomes visible in the dictionary, and sets `WORD_COMPILED`. Also computes the word's mass (header + body bytes) for the physics engine and calls `physics_metadata_refresh_state()`.

Resets `vm->mode` to `MODE_INTERPRET`, clears `vm->state_var`, and NULLs `vm->compiling_word`.

Sets `vm->error` if `vm` has no active `compiling_word` or if `EXIT` cannot be found.

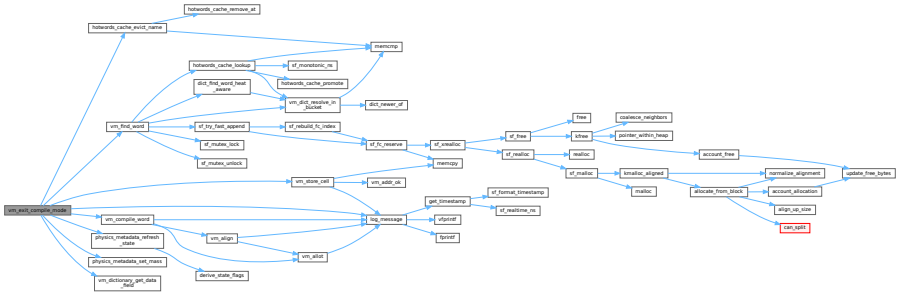
Parameters

vm	VM instance currently in compile mode.

References DictEntry::acl_allow, DictEntry::acl_ttl, ACL_TTL_OPEN, VM::compiling_word, VM::error, DictEntry::flags, VM::here, VM::hotwords_cache, hotwords_cache_evict_name(), LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_INTERPRET, DictEntry::name, DictEntry::name_len, physics_metadata_refresh_state(), physics_metadata_set_mass(), VM::state_addr, VM::state_var, vm_compile_word(), vm_dictionary_get_data_field(), vm_find_word(), vm_store_cell(), WORD_COMPILED, and VM::zuse_session.

Referenced by defining_word_semicolon().

Here is the call graph for this function:



Here is the caller graph for this function:

7.192.2.27 vm_get_base()

```
unsigned vm_get_base (
            const VM * vm )
```

Read the current number base from the VM.

Prefers the cell stored at `vm->base_addr` (the canonical FORTH `BASE` variable) if it is aligned and in the valid range [2, 36]. Falls back to the host-mirror `vm->base` if the cell is out of range, and finally returns 10 if both are invalid or `vm` is NULL.

The double-read (cell + mirror) keeps the VM consistent when C code modifies `vm->base` directly without going through `vm_set_base()`.

Parameters

vm	VM instance to query (may be NULL; returns 10 in that case).

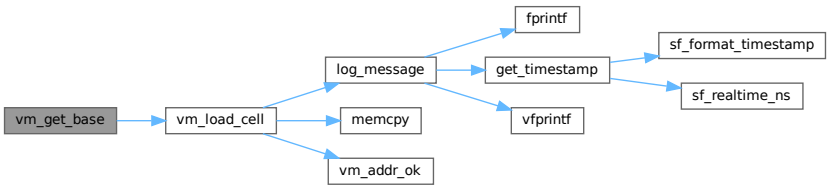
Returns

Current number base in range [2, 36].

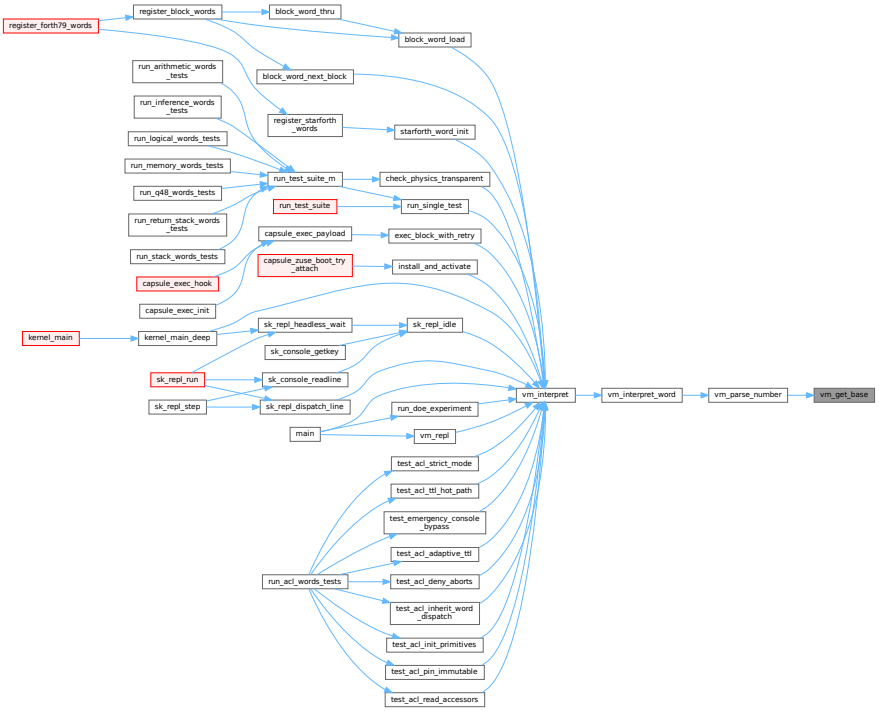
References VM::base, VM::base_addr, vm_load_cell(), and VM_MEMORY_SIZE.

Referenced by vm_parse_number().

Here is the call graph for this function:



Here is the caller graph for this function:



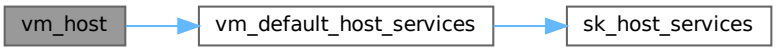
7.192.2.28 vm_host()

```
const VMHostServices * vm_host (
            const VM * vm )
```

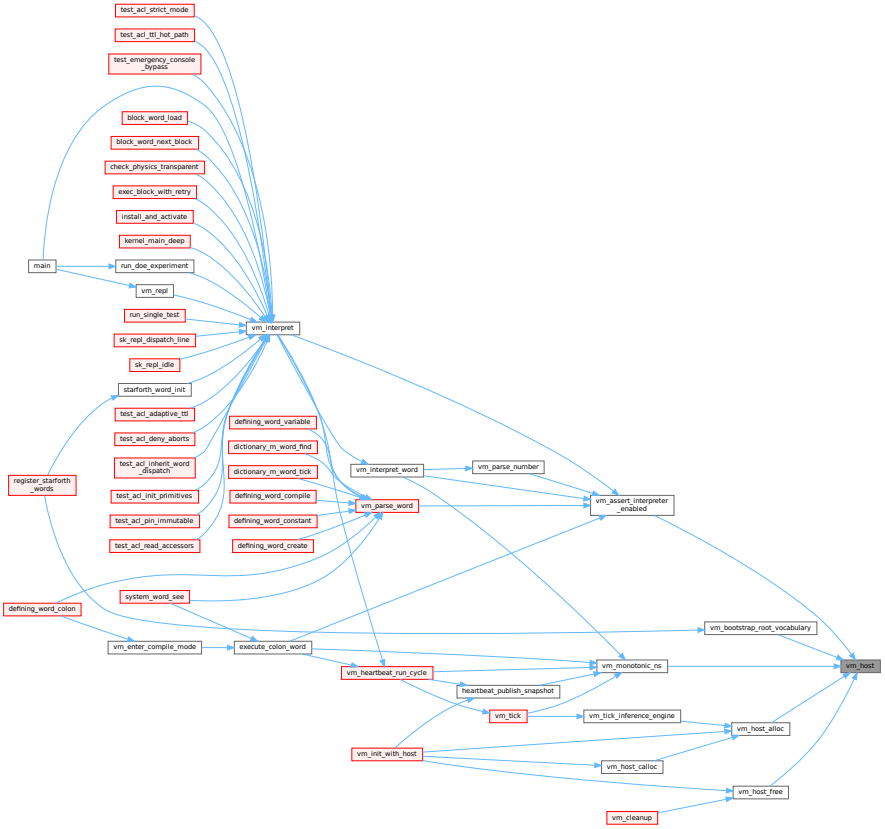
References VM::host, and vm_default_host_services().

Referenced by vm_assert_interpreter_enabled(), vm_bootstrap_root_vocabulary(), vm_host_alloc(), vm_host_free(), and vm_monotonic_ns().

Here is the call graph for this function:



Here is the caller graph for this function:



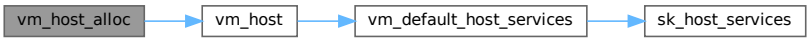
7.192.2.29 vm_host_alloc()

```
void * vm_host_alloc (
            VM * vm,
            size_t size,
            size_t align )
```

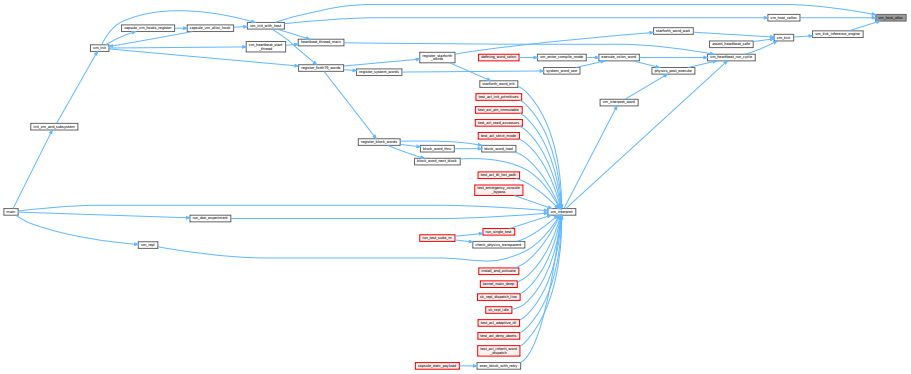
References VMHostServices::alloc, and vm_host().

Referenced by vm_host_calloc(), vm_init_with_host(), and vm_tick_inference_engine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.30 vm_host_calloc()

```
void * vm_host_calloc (
            VM * vm,
            size_t n,
            size_t size )
```

References memset(), and vm_host_alloc().

Referenced by vm_init_with_host().

Here is the call graph for this function:

Here is the caller graph for this function:

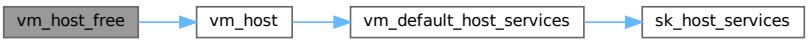
7.192.2.31 vm_host_free()

```
void vm_host_free (
            VM * vm,
            void * ptr )
```

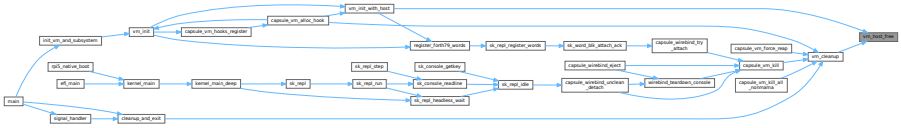
References VMHostServices::free, and vm_host().

Referenced by vm_cleanup(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



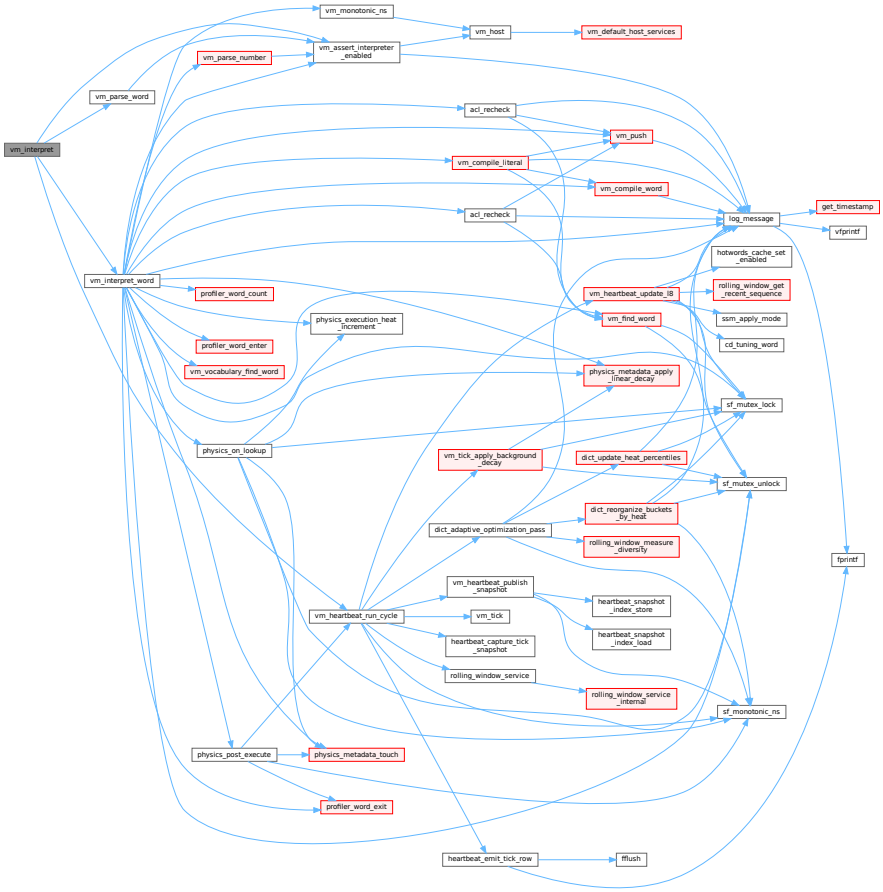
7.192.2.32 vm_interpret()

```
void vm_interpret (
            VM * vm,
            const char * input )
```

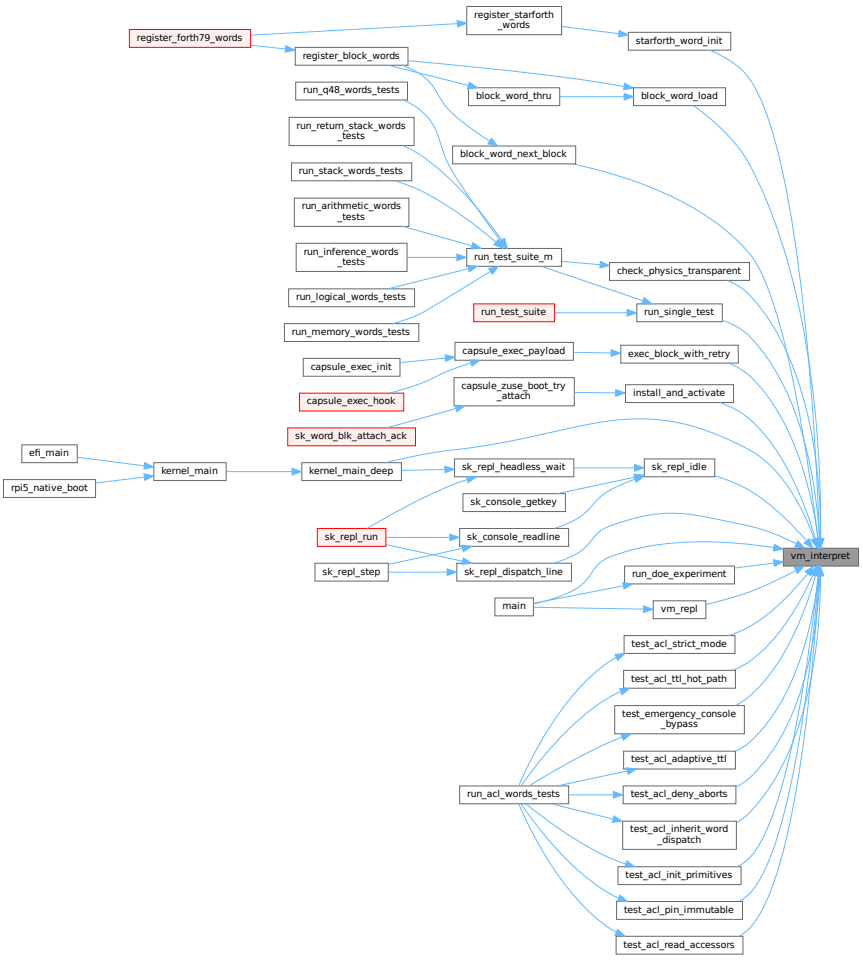
References VM::abort_requested, HeartbeatState::check_counter, VM::error, g_log_attrib_vm, g_vm_interpret_depth, g_vm_interpret_depth, VM::heartbeat, HEARTBEAT_CHECK_FREQUENCY, VM::input_buffer, INPUT_BUFFER_SIZE, VM::input_length, VM::input_pos, vm_assert_interpreter_enabled(), vm_heartbeat_run_cycle(), vm_interpret_word(), vm_parse_word(), and HeartbeatState::worker.

Referenced by block_word_load(), block_word_next_block(), check_physics_transparent(), exec_block_with_retry(), install_and_activate(), kernel_main_deep(), main(), run_doe_experiment(), run_single_test(), sk_repl_dispatch_line(), sk_repl_idle(), starforth_word_init(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_bypass(), and vm_repl().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.33 vm_interpret_word()

```
void vm_interpret_word (
            VM * vm,
            const char * word_str,
            size_t len )
```

Interpret or compile a single word token.

This is the outer interpreter's per-token dispatch:

1. **Dictionary lookup**: vocabulary-aware search, falling back to the canonical flat dictionary. Physics heat is recorded via `physics_on_lookup()`.
2. **Compile mode, non-immediate**: append the entry to the current threaded body via `vm_compile_word()`.
3. **Interpret mode, or immediate in compile mode**: ACL check (hot-path TTL decrement / cold-path `acl_recheck()`), then call `entry->func()`.
4. **Number literal**: if no dictionary match, attempt `vm_parse_number()`. In compile mode, compile as a literal; in interpret mode, push directly.
5. **Error**: unknown token — sets `vm->error`.

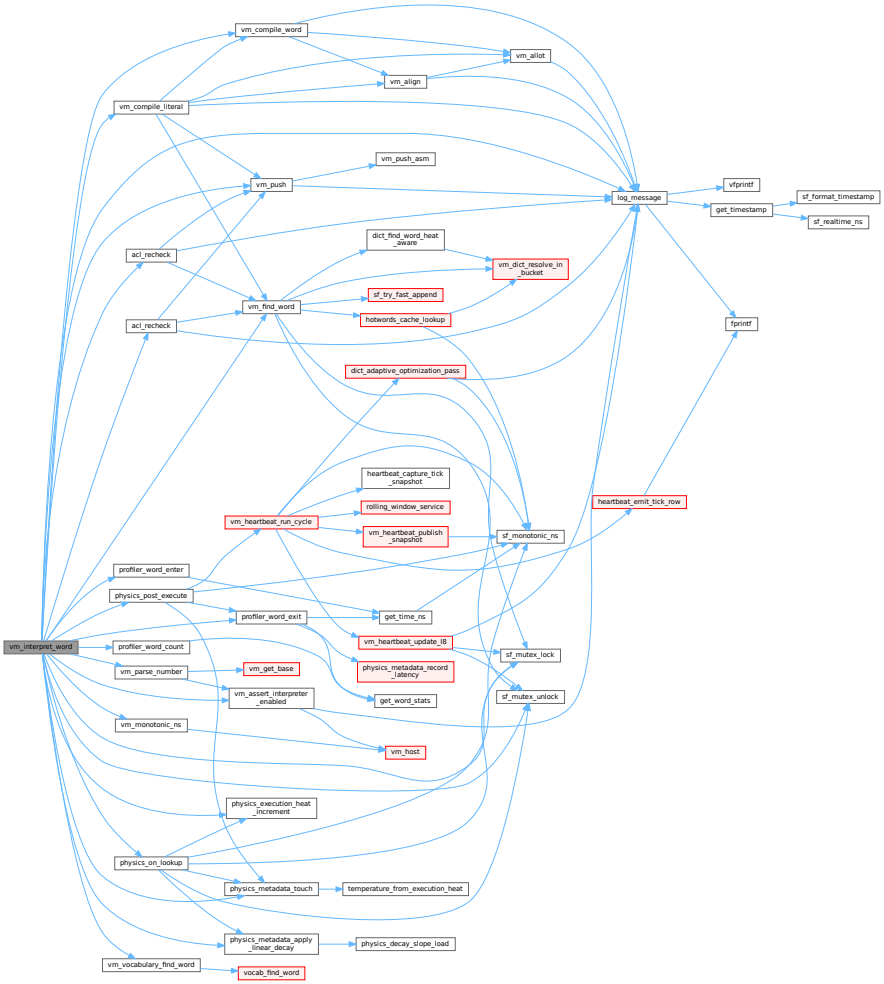
Parameters

vm	VM instance.
word_str	Token string (need not be NUL-terminated).
len	Length of `word_str` in bytes.

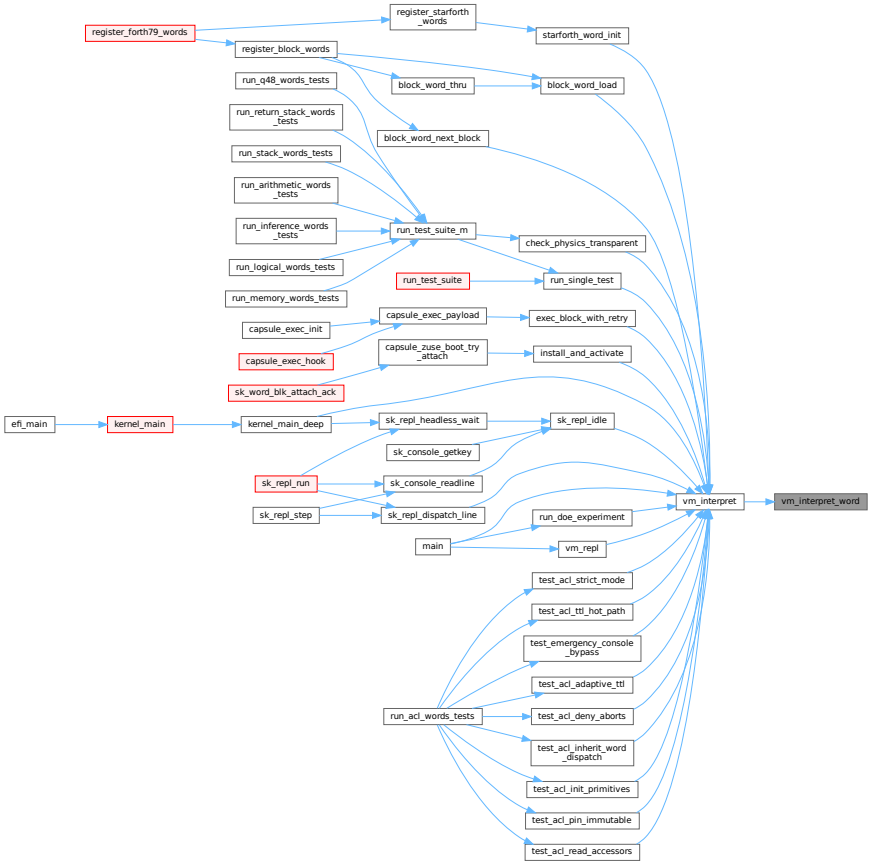
References acl_recheck(), acl_recheck(), VM::current_executing_entry, VM::dict_lock, VM::dsp, VM::emergency_console, entry, VM::error, DictEntry::execution_heat, DictEntry::flags, g_sk_fault_word, VM::heartbeat, DictPhysics::last_active_ns, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, VM::mode, MODE_COMPILE, name, DictEntry::physics, physics_execution_heat_increment(), physics_metadata_apply_linear_decay(), physics_metadata_touch(), physics_on_lookup(), physics_post_execute(), profiler_word_count(), profiler_word_enter(), profiler_word_exit(), VM::rsp, sf_mutex_lock(), sf_mutex_unlock(), HeartbeatState::tick_count, vm_assert_interpreter_enabled(), vm_compile_literal(), vm_compile_word(), vm_find_word(), vm_monotonic_ns(), vm_parse_number(), vm_push(), vm_vocabulary_find_word(), DictEntry::word_id, WORD_IMMEDIATE, and HeartbeatState::words_executed.

Referenced by vm_interpret().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.34 vm_load_cell()

```
cell_t vm_load_cell (
            struct VM * vm,
            vaddr_t addr )
```

Load an aligned cell from VM memory.

Validates that `addr` is within bounds for a full `cell_t` read and that it is naturally aligned (addr % sizeof(cell_t) == 0). Uses `memcpy()` rather than a direct pointer dereference to avoid strict aliasing and alignment UB. Sets `vm->error` and returns 0 on failure.

Parameters

vm	VM instance.
addr	Aligned VM byte offset to read.

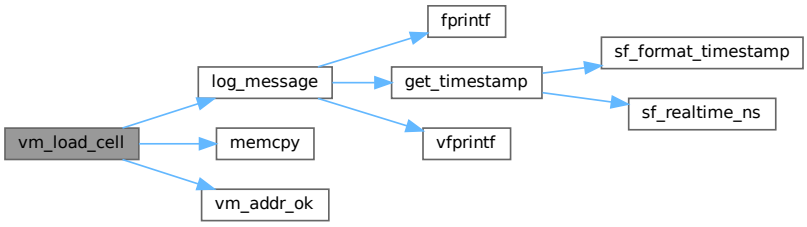
Returns

Cell value at `addr`, or 0 on alignment or bounds error.

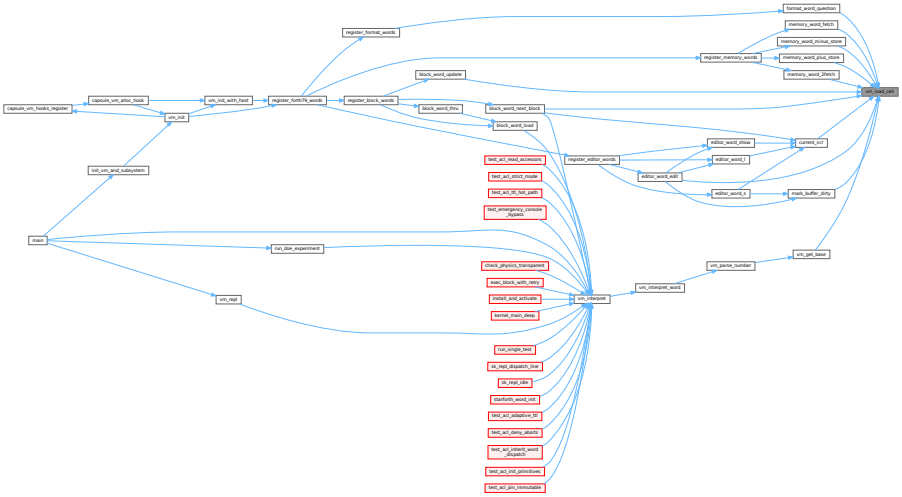
References VM::error, LOG_ERROR, log_message(), memcpy(), VM::memory, and vm_addr_ok().

Referenced by block_word_next_block(), block_word_update(), current_scr(), editor_word_edit(), format_word_question(), mark_buffer_dirty(), memory_word_2fetch(), memory_word_fetch(), memory_word_minus_store(), memory_word_plus_store(), and vm_get_base().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.35 vm_load_u8()

```
uint8_t vm_load_u8 (
            struct VM * vm,
            vaddr_t addr )
```

Load an unsigned byte from VM memory.

Resolves `addr` via `vm_ptr()`. Sets `vm->error` and returns 0 if the address is out of bounds.

Parameters

vm	VM instance.
addr	VM byte offset to read.

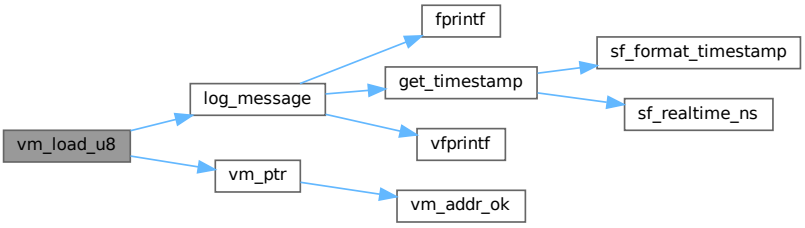
Returns

Byte value at `addr`, or 0 on error.

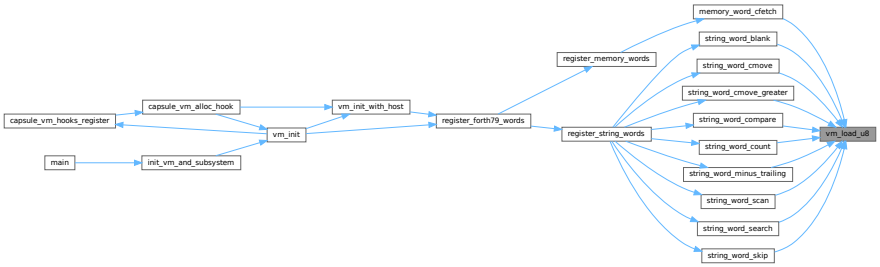
References VM::error, LOG_ERROR, log_message(), and vm_ptr().

Referenced by memory_word_cfetch(), string_word_blank(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_count(), string_word_minus_trailing(), string_word_scan(), string_word_search(), and string_word_skip().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.36 vm_log_attributed_vm()

```
VM * vm_log_attributed_vm (
            void  )
```

References g_log_attrib_vm.

7.192.2.37 vm_make_immediate()

```
void vm_make_immediate (
            VM * vm )
```

Mark the most recently created word as IMMEDIATE (FORTH-79).

Sets `WORD_IMMEDIATE` on `vm->latest` so the word executes during compilation rather than being compiled into the current definition. Also calls `physics_metadata_refresh_state()` to update the physics engine's record of the entry's flags.

Sets `vm->error` if `vm` has no `latest` entry (i.e., the dictionary is empty or no word has been created yet).

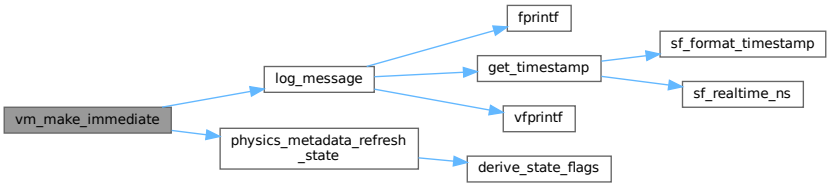
Parameters

vm	VM instance.

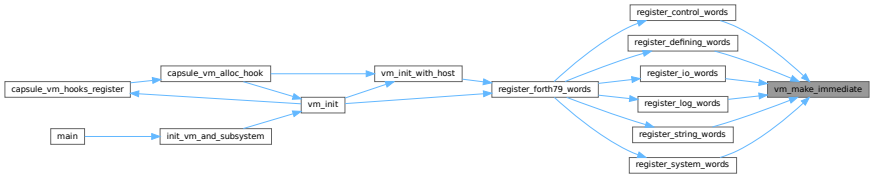
References VM::error, DictEntry::flags, VM::latest, LOG_DEBUG, LOG_ERROR, log_message(), DictEntry::name, DictEntry::name_len, physics_metadata_refresh_state(), and WORD_IMMEDIATE.

Referenced by register_control_words(), register_defining_words(), register_io_words(), register_log_words(), register_string_words(), and register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:



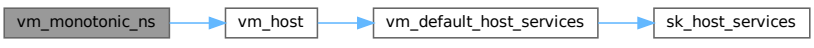
7.192.2.38 vm_monotonic_ns()

```
uint64_t vm_monotonic_ns (
            VM * vm )
```

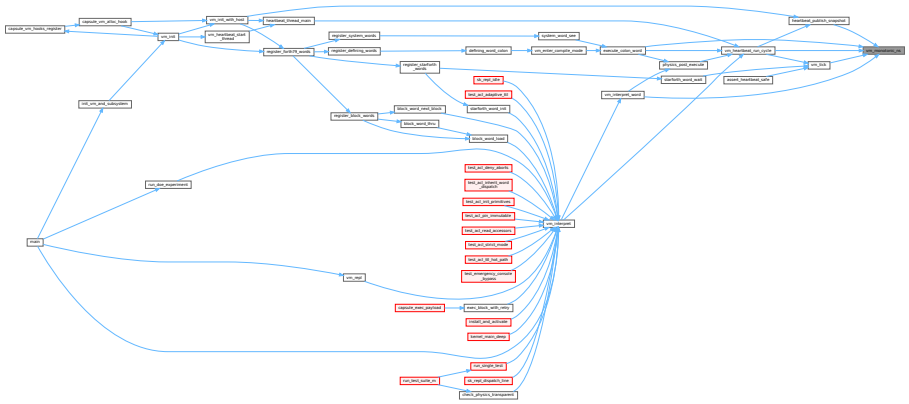
References VMHostServices::monotonic_ns, and vm_host().

Referenced by execute_colon_word(), heartbeat_publish_snapshot(), vm_heartbeat_run_cycle(), vm_interpret_word(), and vm_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.39 vm_parse_number()

```
int vm_parse_number (
            VM * vm,
            const char * s,
            cell_t * out )
```

Attempt to parse a string as an integer in the current number base.

Reads the current base from `vm_get_base()`, then converts each character of `s` (case-insensitive, digits 0–9 and letters A–Z/a–z) to a digit value. Returns 0 immediately on any character that is not a valid digit in the current base, or on an empty string after the optional sign.

Handles an optional leading `'+'` or `'-'` sign. The result is a full-width `cell_t` (sign extended for negative values).

Parameters

vm	VM instance (used for base only; may be NULL, in which case base defaults to 10).
s	NUL-terminated string to parse.
out	Pointer to receive the parsed value on success.

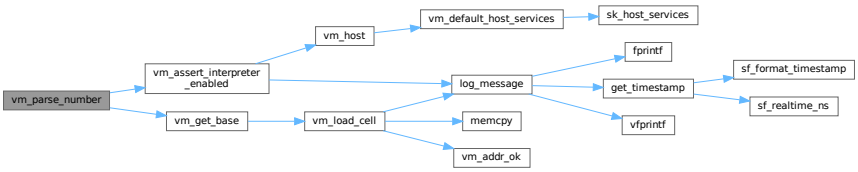
Returns

1 on successful parse, 0 on any parse failure.

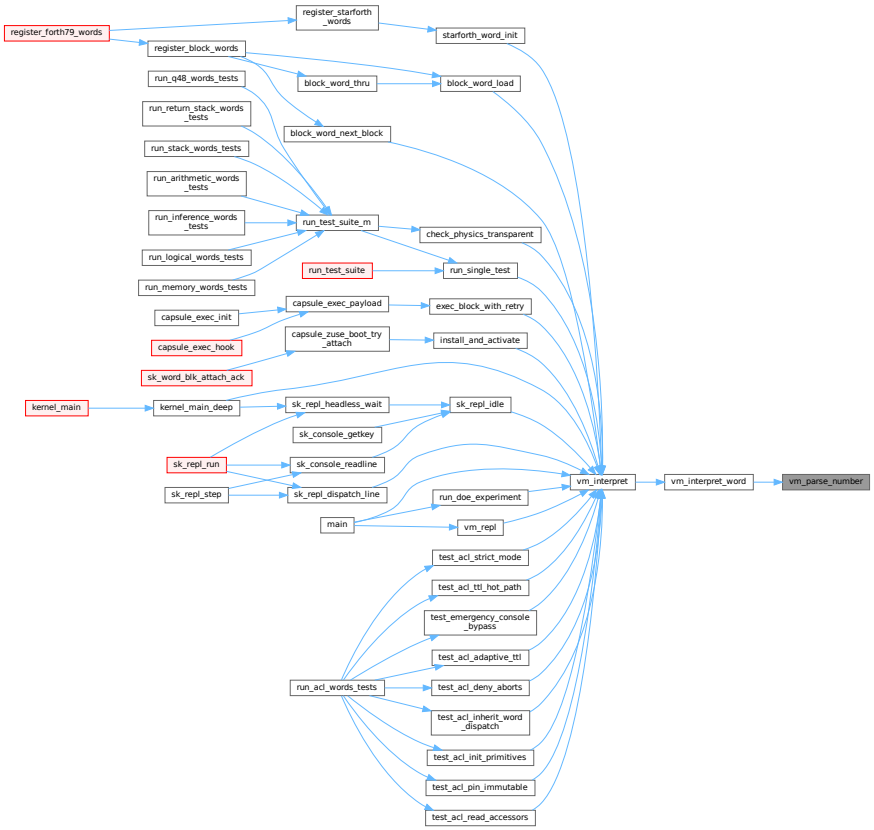
References base, vm_assert_interpreter_enabled(), and vm_get_base().

Referenced by vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.40 vm_parse_word()

```
int vm_parse_word (
            VM * vm,
            char * word,
            size_t max_len )
```

Parse the next whitespace-delimited token from the input buffer.

Skips leading whitespace (space, tab, CR, LF), then extracts characters until the next whitespace or end of buffer, storing up to `max_len` ? 1 characters in `word` followed by a NUL terminator. Advances `vm->input_pos` past the consumed token.

Parameters

vm	VM instance; `vm->input_buffer` / `input_pos` / `input_length` must be populated (typically by `vm_interpret()`).
word	Output buffer for the parsed token.
max_len	Size of `word` in bytes (including NUL terminator).

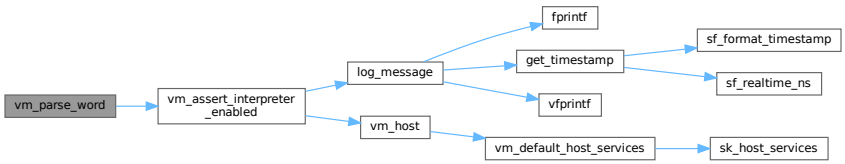
Returns

Number of characters written (excluding NUL), or 0 if the buffer is exhausted or `vm` / `word` is NULL.

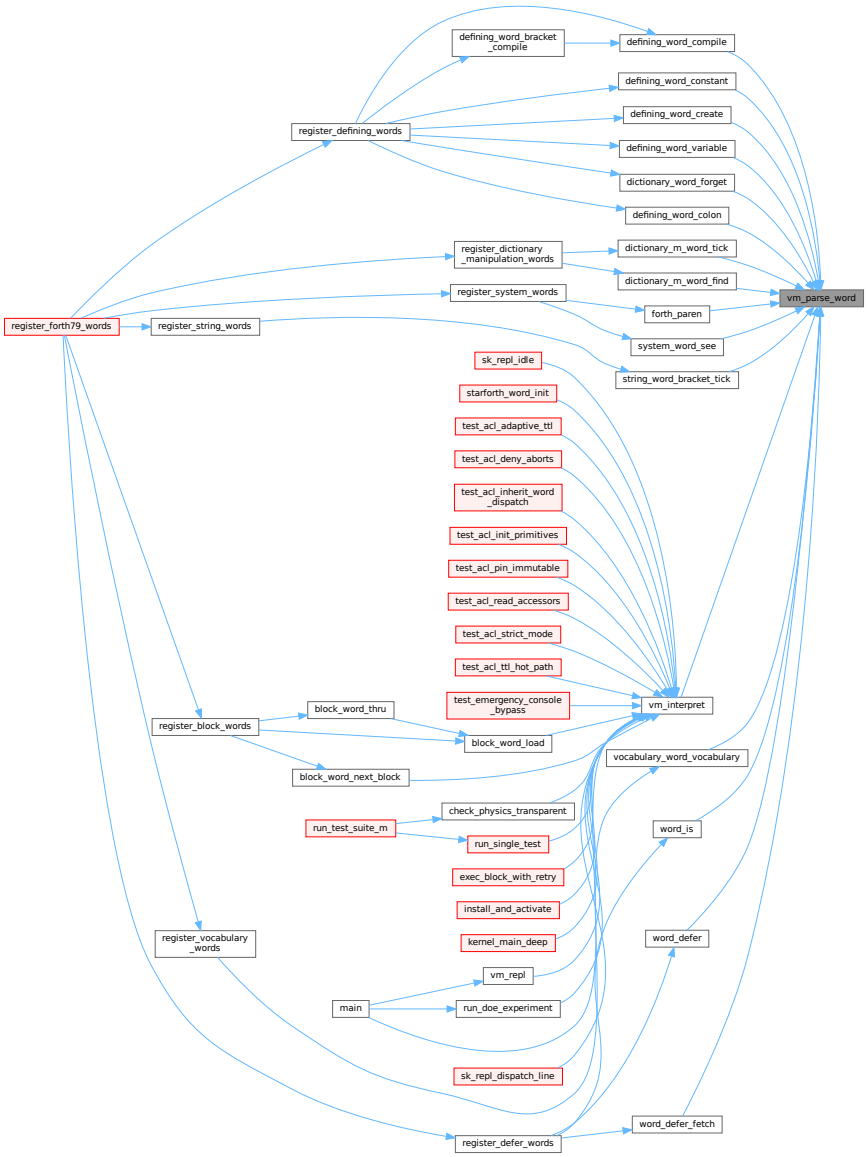
References VM::input_buffer, VM::input_length, VM::input_pos, and vm_assert_interpreter_enabled().

Referenced by defining_word_colon(), defining_word_compile(), defining_word_constant(), defining_word_create(), defining_word_variable(), dictionary_m_word_find(), dictionary_m_word_tick(), dictionary_word_forget(), forth_paren(), string_word_bracket_tick(), system_word_see(), vm_interpret(), vocabulary_word_vocabulary(), word_defer(), word_defer_fetch(), and word_is().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.41 vm_ptr()

```
uint8_t * vm_ptr (
            struct VM * vm,
            vaddr_t addr )
```

Return a host C pointer into VM memory at `addr`.

Validates the address with `vm_addr_ok()` (minimum 1-byte range) and returns `NULL` if the address is out of bounds or `vm` is uninitialised. The returned pointer is valid for at least one byte; callers that need multi-byte access should perform their own range check first.

Parameters

vm	VM instance.
addr	VM byte offset to dereference.

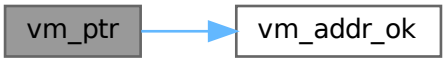
Returns

Host pointer into `vm->memory`, or NULL on bounds failure.

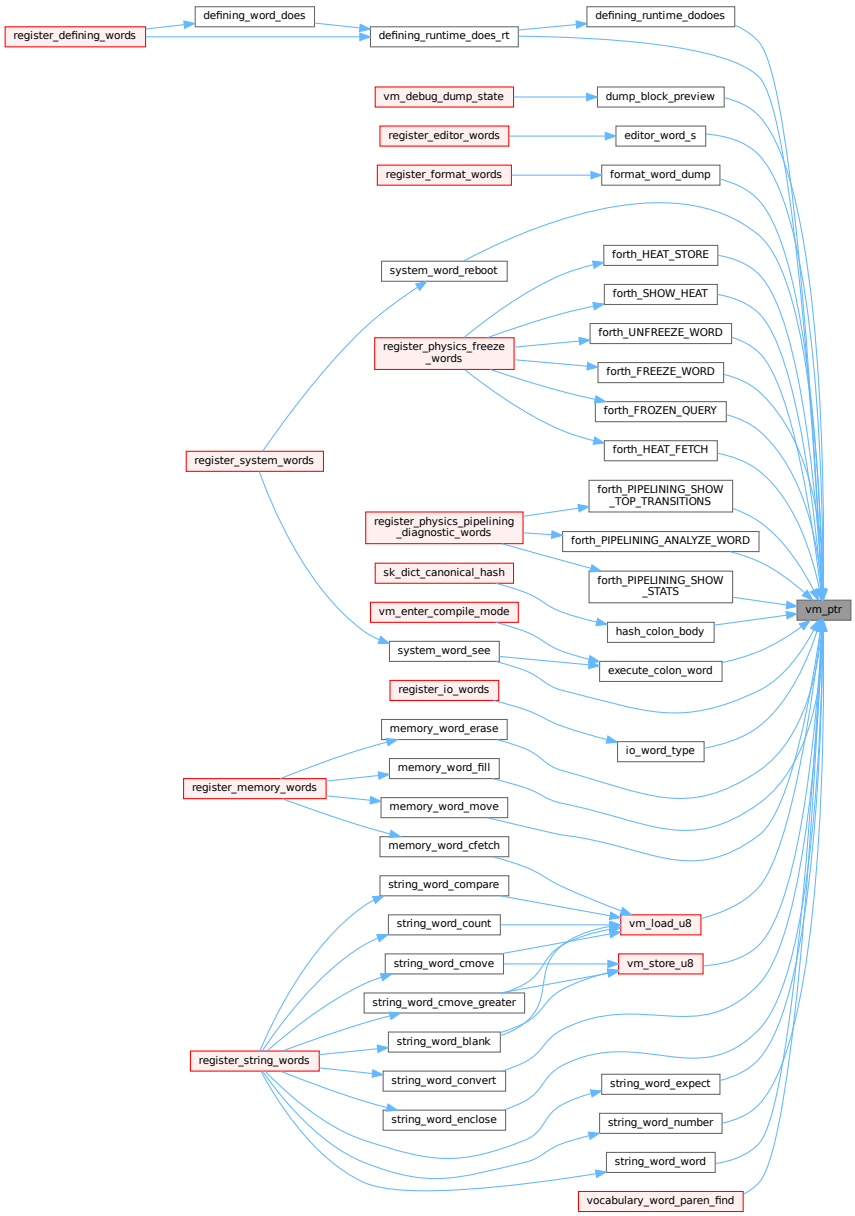
References VM::memory, and vm_addr_ok().

Referenced by defining_runtime_dodoes(), defining_runtime_does_rt(), dump_block_preview(), editor_word_s(), execute_colon_word(), format_word_dump(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_STORE(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TO forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), hash_colon_body(), io_word_type(), memory_word_erase(), memory_word_fill(), memory_word_move(), string_word_convert(), string_word_enclose(), string_word_expect(), string_word_number(), string_word_word(), system_word_reboot(), system_word_see(), vm_load_u8(), vm_store_u8(), and vocabulary_word_paren_find().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.42 vm_reset_hosted_fake_ns()

```
void vm_reset_hosted_fake_ns (
            VM * vm )
```

References VM::host, hosted_fake_ns, hosted_services, and PARITY_MODE.

Referenced by vm_init_with_host().

Here is the caller graph for this function:

7.192.2.43 vm_set_base()

```
void vm_set_base (
            VM * vm,
            unsigned b )
```

Set the VM number base and update both the cell and the host mirror.

Clamps `b` to [2, 36] (replacing out-of-range values with 10), then writes to the FORTH `BASE` variable cell at `vm->base_addr` and to the host-mirror field `vm->base` so both sources agree.

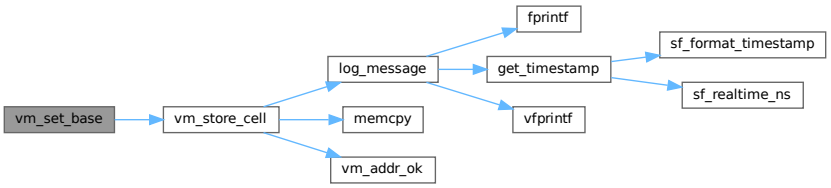
Parameters

vm	VM instance to update.
b	Desired number base; clamped to [2, 36].

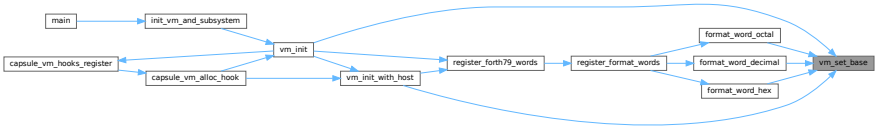
References VM::base, VM::base_addr, and vm_store_cell().

Referenced by format_word_decimal(), format_word_hex(), format_word_octal(), vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.44 vm_store_cell()

```
void vm_store_cell (
            struct VM * vm,
            vaddr_t addr,
            cell_t v )
```

Store an aligned cell to VM memory.

Validates alignment and bounds, then copies `v` into VM memory via `memcpy()`. Sets `vm->error` on alignment or bounds failure.

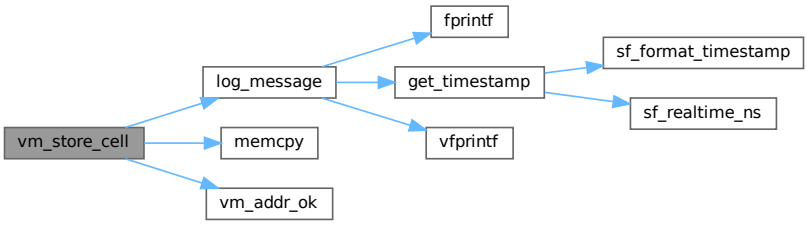
Parameters

vm	VM instance.
addr	Aligned VM byte offset to write.
v	Cell value to store.

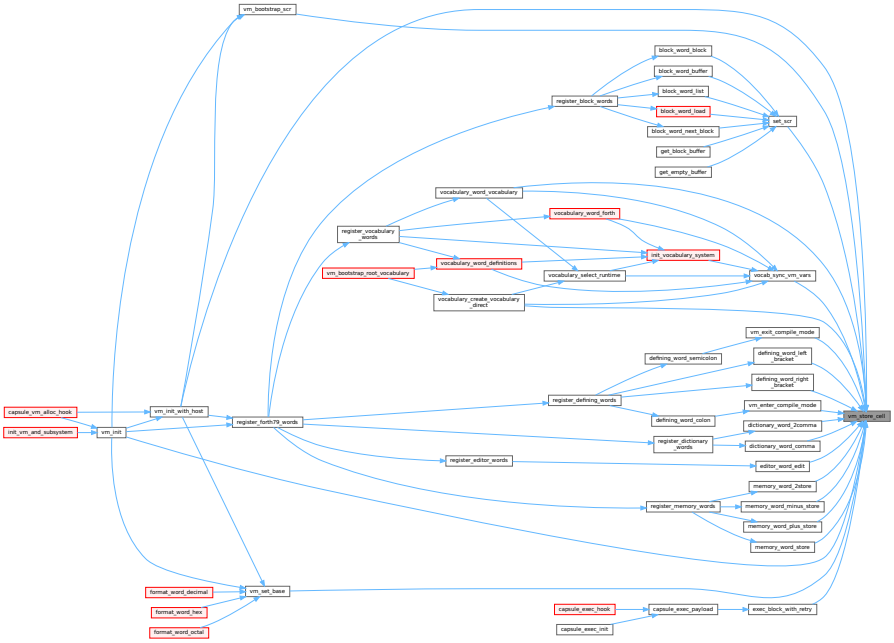
References VM::error, LOG_ERROR, log_message(), memcpy(), VM::memory, and vm_addr_ok().

Referenced by defining_word_left_bracket(), defining_word_right_bracket(), dictionary_word_2comma(), dictionary_word_comma(), editor_word_edit(), exec_block_with_retry(), memory_word_2store(), memory_word_minus_store(), memory_word_plus_store(), memory_word_store(), set_scr(), vm_bootstrap_scr(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_init(), vm_init_with_host(), vm_set_base(), vocab_sync_vm_vars(), vocabulary_create_vocabulary_dire and vocabulary_word_vocabulary().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.45 vm_store_u8()

```
void vm_store_u8 (
            struct VM * vm,
            vaddr_t addr,
            uint8_t v )
```

Store an unsigned byte to VM memory.

Resolves `addr` via `vm_ptr()`. Sets `vm->error` if the address is out of bounds; otherwise writes `v`.

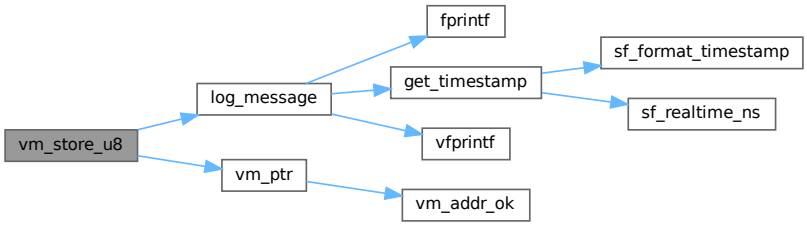
Parameters

vm	VM instance.
addr	VM byte offset to write.
v	Byte value to store.

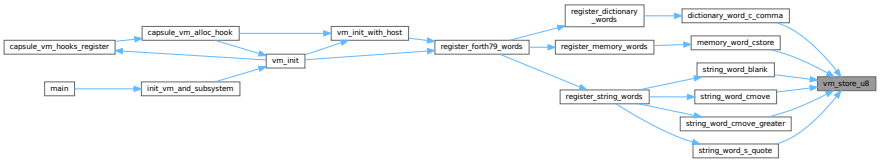
References VM::error, LOG_ERROR, log_message(), and vm_ptr().

Referenced by dictionary_word_c_comma(), memory_word_cstore(), string_word_blank(), string_word_cmove(), string_word_cmove_greater(), and string_word_s_quote().

Here is the call graph for this function:



Here is the caller graph for this function:



7.192.2.46 vm_zuse_cert_install()

```
int vm_zuse_cert_install (
            VM * vm,
            const uint8_t seed[32],
            const uint8_t pubkey[32] )
```

One-time write of the Zuse cert (seed + derived pubkey) – blows the fuse.

Deliberately not backed by a dictionary CONSTANT: ACL-PIN only blocks redefinition (vm_create_word shadowing), not a >BODY-then-store on the word's data field, so a pinned CONSTANT is not actually immutable against FORTH-level tampering. Keeping the value in a VM struct field with no corresponding FORTH store word closes that path entirely – see FABRIC-2.md's Milestone 4 mint-then-pin writeup for the finding.

Parameters

vm	VM instance.
seed	Ed25519 seed, 32 bytes (the private identity).
pubkey	Ed25519 public key derived from seed, 32 bytes.

Returns

0 on success; -1 if already installed (fuse already blown).

References memcpy(), VM::zuse_cert_installed, VM::zuse_cert_pubkey, and VM::zuse_cert_seed.

Referenced by install_and_activate().

Here is the call graph for this function:

Here is the caller graph for this function:

7.192.3 Variable Documentation

7.192.3.1 g_log_attrib_vm

```
VM* g_log_attrib_vm  [static]
```

Referenced by vm_interpret(), and vm_log_attributed_vm().

7.192.3.2 g_vm_interpret_depth

```
int g_vm_interpret_depth  [static]
```

Referenced by sk_vm_at_outermost_interpret(), and vm_interpret().

7.192.3.3 hosted_fake_ns

```
uint64_t hosted_fake_ns = 0  [static]
```

Referenced by hosted_time_ns(), and vm_reset_hosted_fake_ns().

7.192.3.4 hosted_services

```
const VMHostServices hosted_services  [static]
```

Initial value:

```
= {
    .alloc = hosted_alloc,
    .free = hosted_free,
    .monotonic_ns = hosted_time_ns,
    .mutex_init = hosted_mutex_init,
    .mutex_lock = hosted_mutex_lock,
    .mutex_unlock = hosted_mutex_unlock,
    .mutex_destroy = hosted_mutex_destroy,
    .puts = hosted_puts,
    .putc = hosted_putc,
    .is_executable_ptr = hosted_xt_is_executable,
    .owns_xt_entry = hosted_xt_entry_owned,
    .panic = hosted_panic,
    .parity_mode =  0 ,
    .verbose = 0
}
```

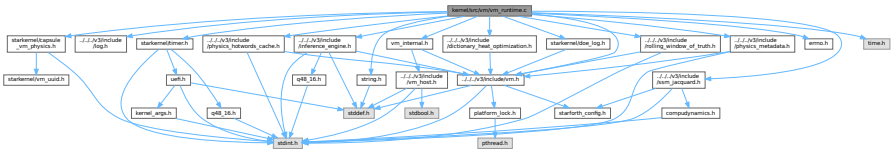
Referenced by vm_default_host_services(), and vm_reset_hosted_fake_ns().

7.193 kernel/src/vm/vm_log_buffer.c File Reference

7.194 kernel/src/vm/vm_runtime.c File Reference

```
#include "../../../v3/include/vm.h"
#include "../../../v3/include/log.h"
#include "../../../v3/include/rolling_window_of_truth.h"
#include "../../../v3/include/dictionary_heat_optimization.h"
#include "../../../v3/include/inference_engine.h"
#include "../../../v3/include/physics_metadata.h"
#include "../../../v3/include/physics_hotwords_cache.h"
#include "starkernel/doe_log.h"
#include "../../../v3/include/ssm_jacquard.h"
#include "vm_internal.h"
#include "starkernel/capsule_vm_physics.h"
#include "starkernel/timer.h"
#include <errno.h>
#include <string.h>
#include <time.h>
```

Include dependency graph for vm_runtime.c:



Functions

- uint32_t heartbeat_snapshot_index_load (const volatile uint32_t *ptr)
- void heartbeat_snapshot_index_store (volatile uint32_t *ptr, uint32_t value)
- void heartbeat_publish_snapshot (VM *vm)
- void vm_tick (VM *vm)

  Central heartbeat dispatcher for time-driven tuning operations.
- void vm_tick_window_tuner (VM *vm)

  Loop #5: Context-aware window tuning via binary chop search.
- void vm_tick_slope_validator (VM *vm)

  Loop #3: Heat decay slope validation via periodic measurement.
- static void vm_tick_apply_background_decay (VM *vm, uint64_t now_ns)
- static void vm_heartbeat_update_l8 (VM *vm)

  L8 FINAL INTEGRATION: Jacquard mode selector heartbeat update.
- void vm_heartbeat_run_cycle (VM *vm)
- void * heartbeat_thread_main (void *arg)
- void vm_snapshot_read (const VM *vm, HeartbeatSnapshot *out_snapshot)
- void vm_tick_inference_engine (VM *vm)

  Phase 2: Unified Inference Engine - Adaptive Window & Decay Slope Tuning.

7.194.1 Function Documentation

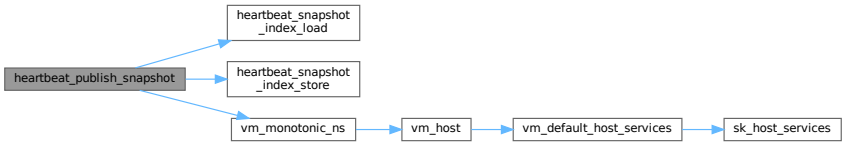
7.194.1.1 heartbeat_publish_snapshot()

```
void heartbeat_publish_snapshot (
            VM * vm )
```

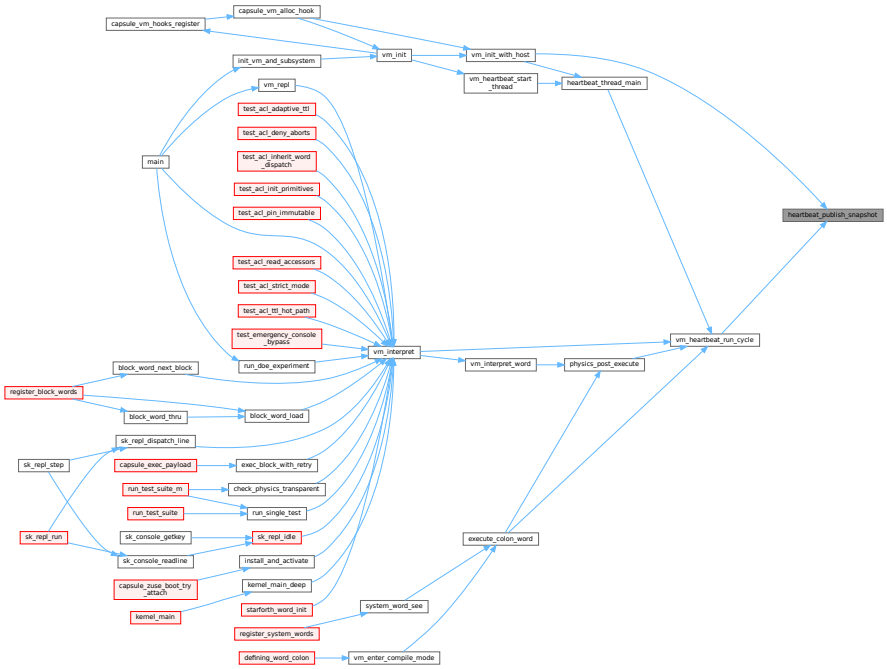
References HeartbeatSnapshot::decay_slope_q48, VM::decay_slope_q48, RollingWindowOfTruth::effective_window_size, VM::heartbeat, heartbeat_snapshot_index_load(), heartbeat_snapshot_index_store(), HeartbeatSnapshot::hot_word_count, VM::hot_word_count_at_check, HeartbeatSnapshot::published_ns, HeartbeatSnapshot::published_tick, VM::rolling_window, HeartbeatState::snapshot_index, HeartbeatState::snapshots, HeartbeatSnapshot::stale_word_count, VM::stale_word_count_at_chec HeartbeatState::tick_count, HeartbeatSnapshot::total_heat, VM::total_heat_at_last_check, vm_monotonic_ns(), and HeartbeatSnapshot::window_width.

Referenced by vm_heartbeat_run_cycle(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:

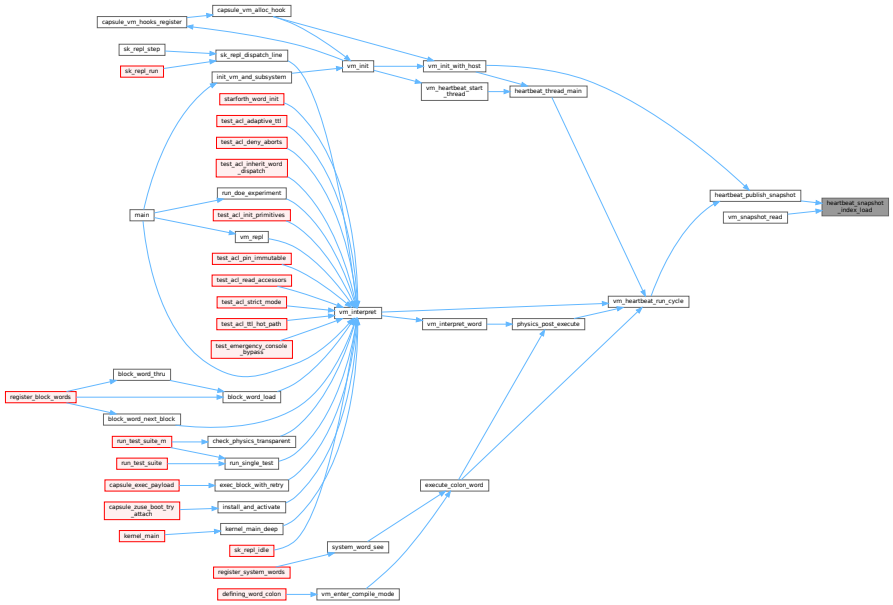


7.194.1.2 heartbeat_snapshot_index_load()

```
uint32_t heartbeat_snapshot_index_load (
            const volatile uint32_t * ptr )
```

Referenced by heartbeat_publish_snapshot(), and vm_snapshot_read().

Here is the caller graph for this function:

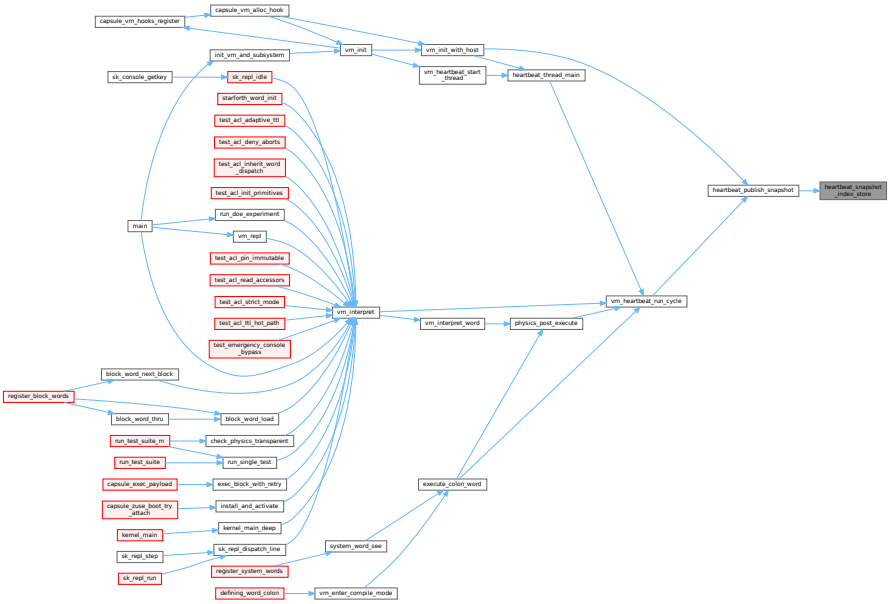


7.194.1.3 heartbeat_snapshot_index_store()

```
void heartbeat_snapshot_index_store (
            volatile uint32_t * ptr,
            uint32_t value )
```

Referenced by heartbeat_publish_snapshot().

Here is the caller graph for this function:



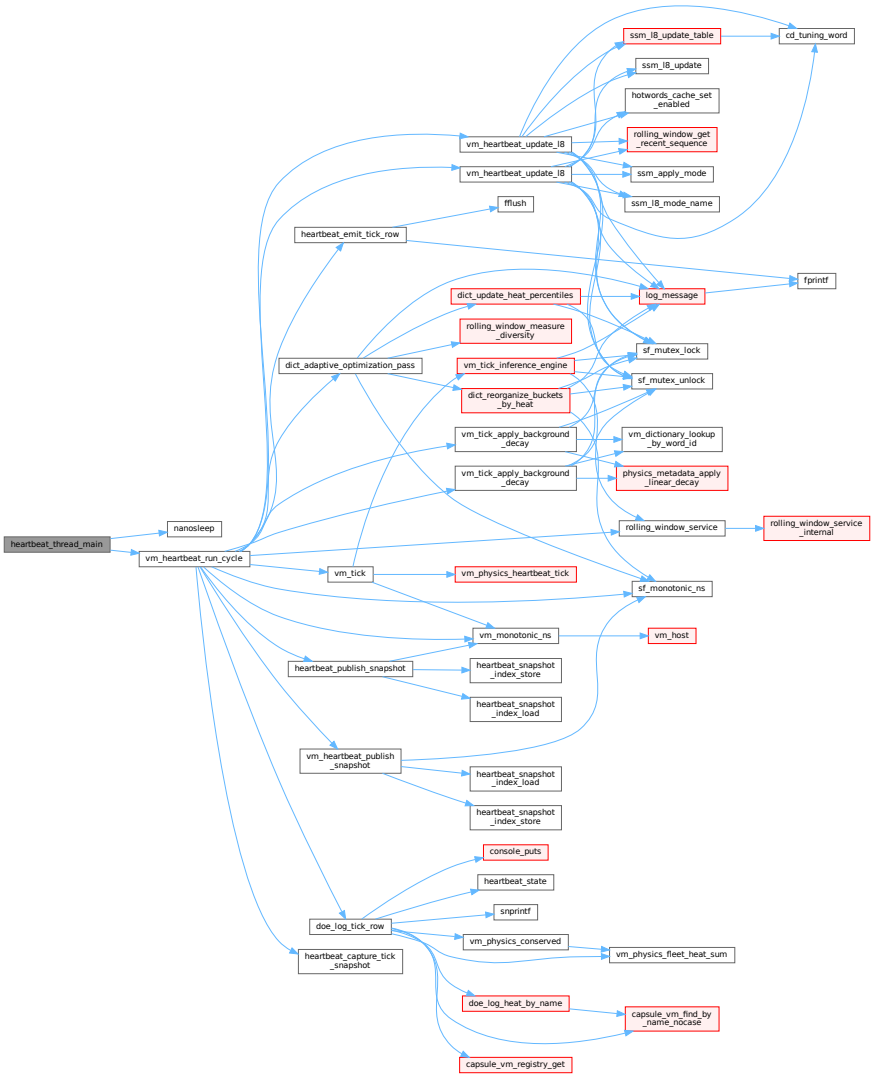
7.194.1.4 heartbeat_thread_main()

```
void * heartbeat_thread_main (
            void * arg )
```

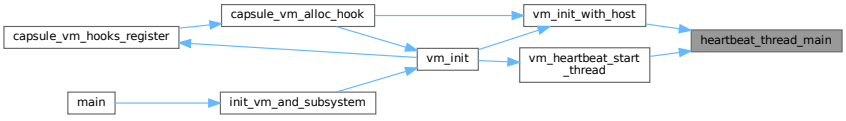
References errno, VM::heartbeat, HEARTBEAT_TICK_NS, nanosleep(), HeartbeatWorker::running, HeartbeatWorker::stop_requeste
HeartbeatWorker::tick_ns, timespec::tv_nsec, timespec::tv_sec, vm_heartbeat_run_cycle(), and HeartbeatState::worker.

Referenced by vm_heartbeat_start_thread(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



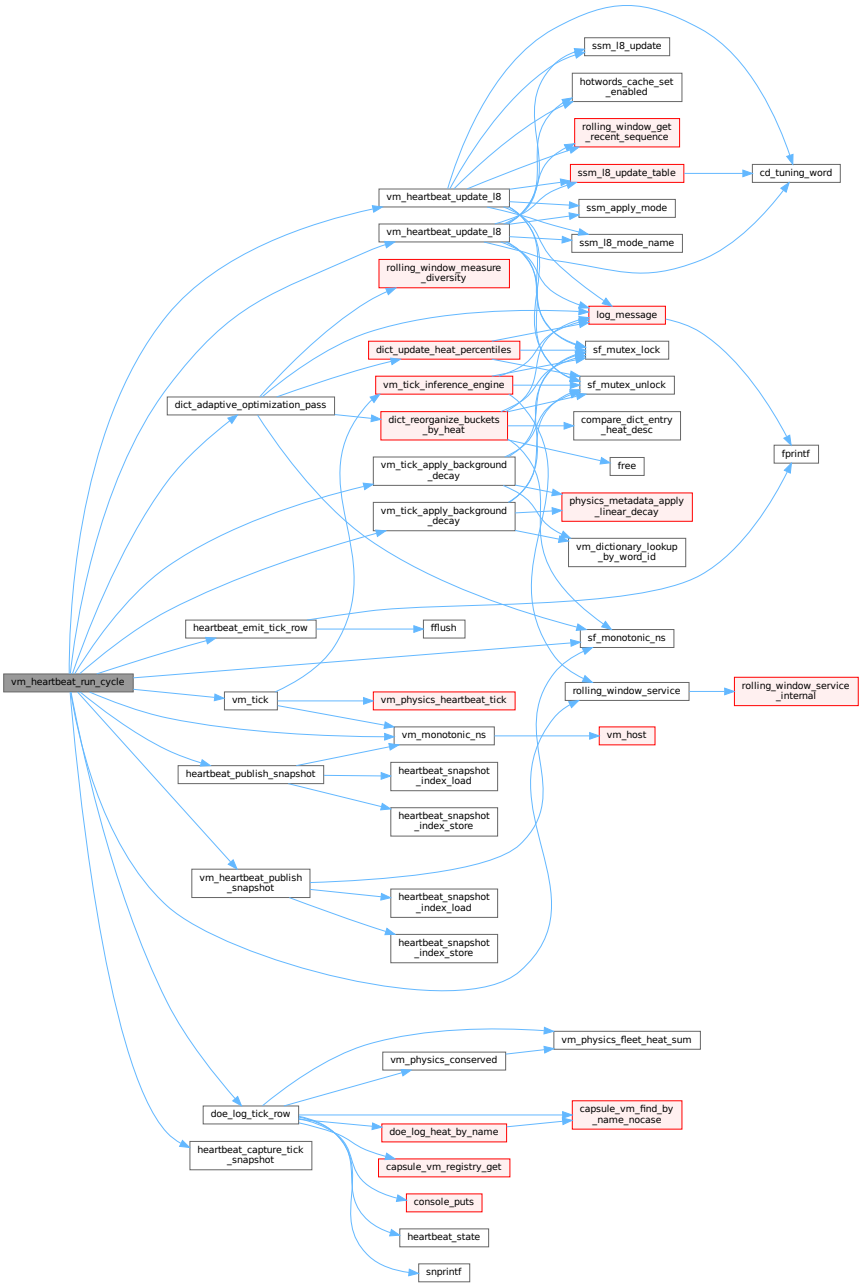
7.194.1.5 vm_heartbeat_run_cycle()

```
void vm_heartbeat_run_cycle (
            VM * vm )
```

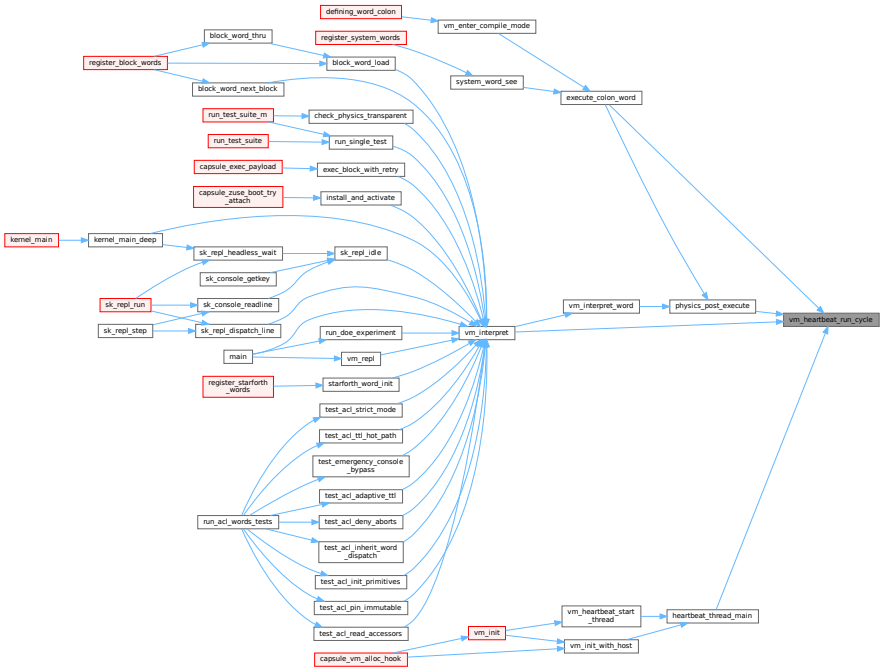
References dict_adaptive_optimization_pass(), doe_log_tick_row(), VM::heartbeat, heartbeat_capture_tick_snapshot(), heartbeat_emit_tick_row(), HeartbeatState::heartbeat_enabled, heartbeat_publish_snapshot(), VM::rolling_window, rolling_window_service(), sf_monotonic_ns(), vm_heartbeat_publish_snapshot(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_monotonic_ns(), vm_tick(), vm_tick_apply_background_decay(), and vm_tick_apply_background_dec

Referenced by execute_colon_word(), heartbeat_thread_main(), physics_post_execute(), and vm_interpret().

Here is the call graph for this function:



Here is the caller graph for this function:



7.194.1.6 vm_heartbeat_update_l8()

```
static void vm_heartbeat_update_l8 (
            VM * vm )  [static]
```

L8 FINAL INTEGRATION: Jacquard mode selector heartbeat update.

Collects metrics from L1-L7 physics layers and feeds them to the L8 Jacquard mode selector. L8 then chooses the optimal configuration mode based on workload characteristics and applies it to the runtime.

This is the SOLE policy engine - all adaptive decisions flow through L8.

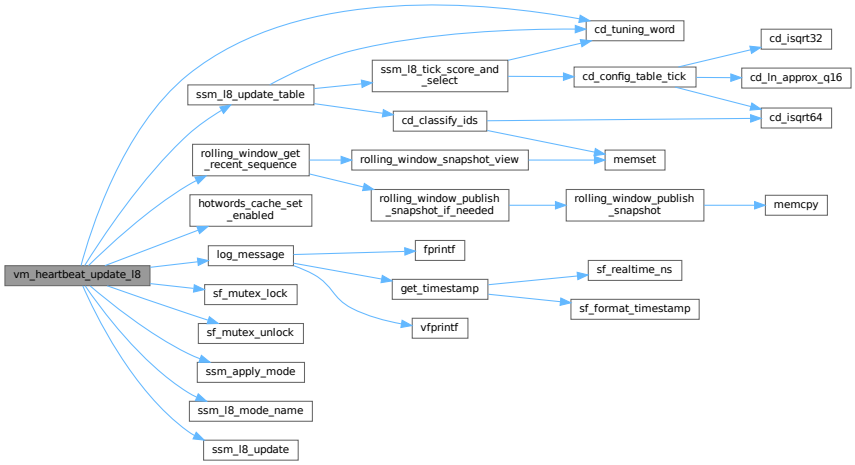
Parameters

vm	Pointer to VM instance

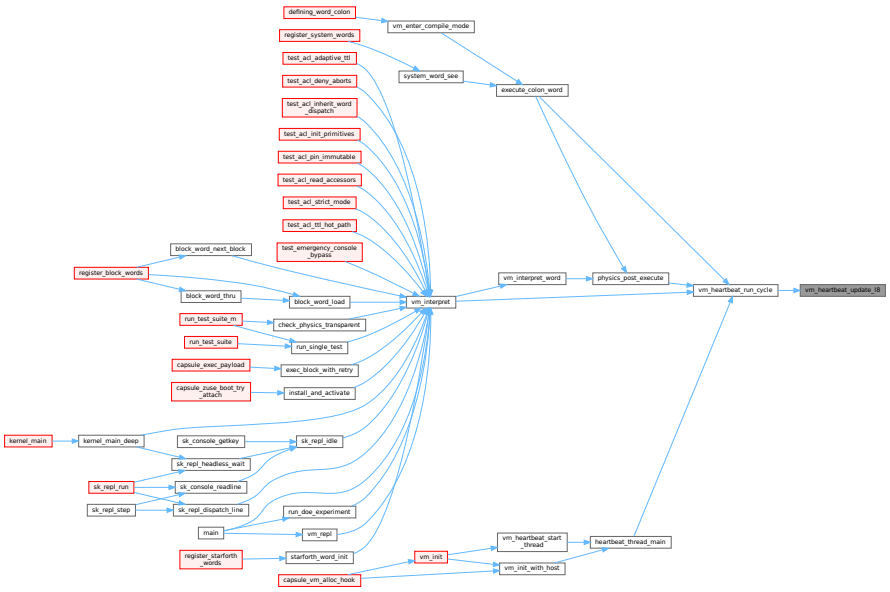
References CD_MAX_CLASSIFY_DEPTH, cd_tuning_word(), CDConfigTable::current_config, ssm_l8_state_t::current_mode, CDConfigTable::current_regime, ssm_l8_metrics_t::cv, VM::decay_slope_q48, InferenceOutputs::early_exited, RollingWindowOfTruth::effective_window_size, ssm_l8_metrics_t::entropy, VM::heartbeat, VM::hotwords_cache, hotwords_cache_set_enabled(), CDTuning::id_window_depth, ssm_l8_metrics_t::inference_early_exited, ssm_l8_metrics_t::inference, ssm_config_t::L1_heat_tracking, ssm_config_t::L2_rolling_window, ssm_config_t::L3_linear_decay, ssm_config_t::L4_pipelining, ssm_config_t::L5_window_inference, ssm_config_t::L6_decay_inference, ssm_config_t::L7_adaptive_heartrate, VM::last_inference_outputs, HeartbeatState::last_inference_tick, LOG_INFO, log_message(), CDConfigTable::num_configs, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, CDConfigTable::regime_scores, VM::rolling_window, rolling_window_get_recent_sequence(), ROLLING_WINDOW_SIZE, sf_mutex_lock(), sf_mutex_unlock(), ssm_apply_mode(), VM::ssm_config, ssm_l8_mode_name(), VM::ssm_l8_state, ssm_l8_update(), ssm_l8_update_table(), ssm_l8_metrics_t::stability_score, ssm_l8_state_t::table, ssm_l8_metrics_t::temporal_decay, HeartbeatState::tick_count, RollingWindowOfTruth::total_executions, and VM::tuning_lock.

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:

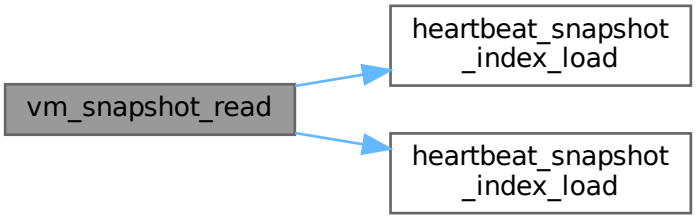


7.194.1.7 vm_snapshot_read()

```
void vm_snapshot_read (
            const VM * vm,
            HeartbeatSnapshot * out_snapshot )
```

References VM::heartbeat, heartbeat_snapshot_index_load(), heartbeat_snapshot_index_load(), HeartbeatState::snapshot_index, and HeartbeatState::snapshots.

Here is the call graph for this function:



7.194.1.8 vm_tick()

```
void vm_tick (
            VM * vm )
```

Central heartbeat dispatcher for time-driven tuning operations.

Aggregates all periodic optimization tasks (Loop #3 and Loop #5) into one place. Designed as plugin architecture - new tuning operations can be added as plugins.

Options for integration:

- Synchronous (now): Called from main execution loop, every N executions
- Background thread (future): Runs in separate thread, decoupled from VM execution

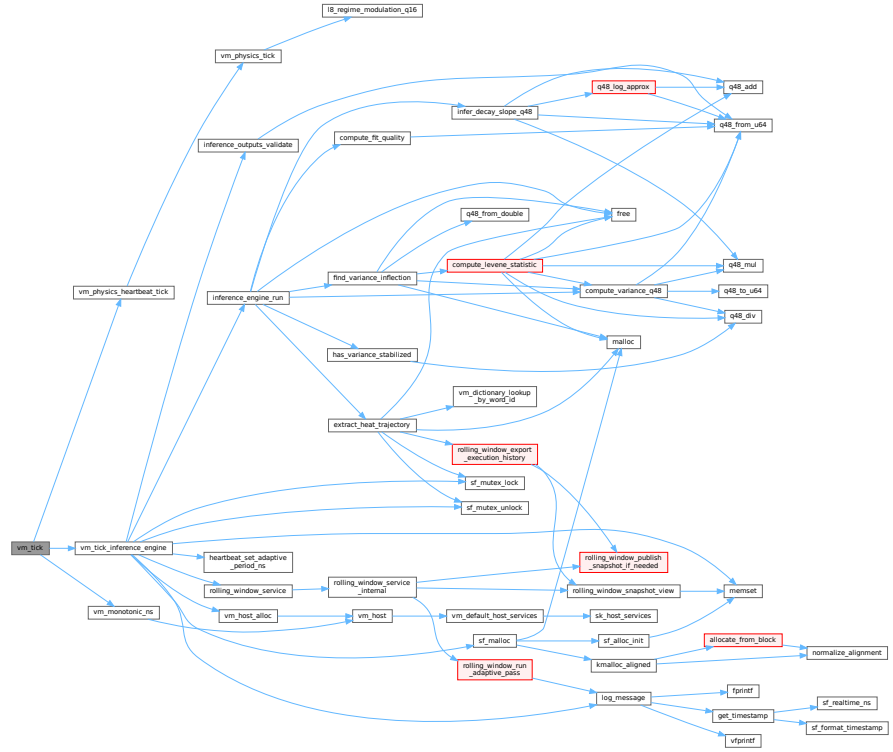
Parameters

vm	Pointer to VM instance

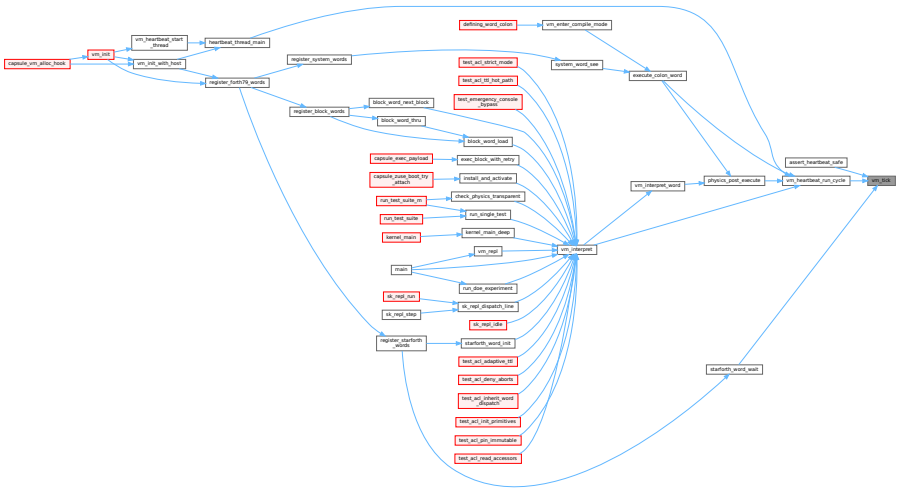
References VM::heartbeat, HeartbeatState::heartbeat_enabled, HEARTBEAT_INFERENCE_FREQUENCY, HeartbeatState::last_inference_tick, HeartbeatState::tick_count, vm_monotonic_ns(), vm_physics_heartbeat_tick(), and vm_tick_inference_engine().

Referenced by assert_heartbeat_safe(), starforth_word_wait(), and vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



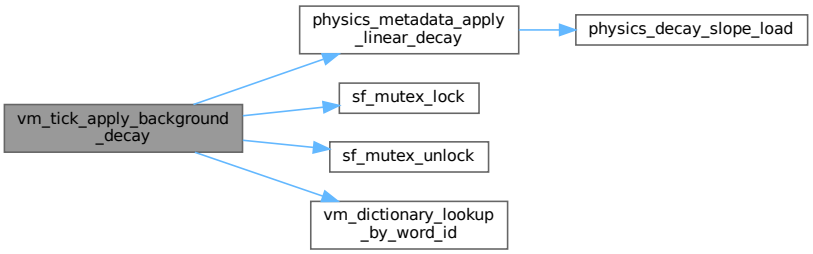
7.194.1.9 vm_tick_apply_background_decay()

```
static void vm_tick_apply_background_decay (
            VM * vm,
            uint64_t now_ns )  [static]
```

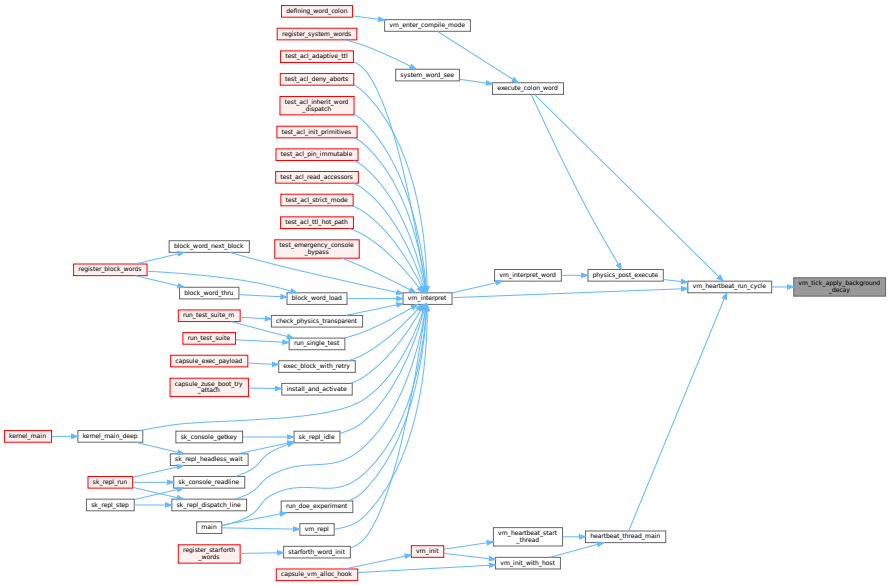
References VM::dict_lock, VM::heartbeat, HEARTBEAT_DECAY_BATCH, VM::heartbeat_decay_cursor_id, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, VM::latest, DictEntry::link, DictEntry::physics, physics_metadata_apply_line sf_mutex_lock(), sf_mutex_unlock(), HeartbeatState::tick_count, vm_dictionary_lookup_by_word_id(), DictEntry::word_id, and WORD_ID_INVALID.

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.194.1.10 vm_tick_inference_engine()

```
void vm_tick_inference_engine (
            VM * vm )
```

Phase 2: Unified Inference Engine - Adaptive Window & Decay Slope Tuning.

Coordinates inference on rolling window of truth to determine:

- Optimal adaptive window width (via variance inflection detection)
- Optimal decay slope (via exponential regression on heat trajectory)

Uses ANOVA early-exit to skip full inference when variance is stable (<5% change). All math uses Q48.16 fixed-point (integer-only, no floating-point).

Replaces legacy vm_tick_window_tuner() and vm_tick_slope_validator().

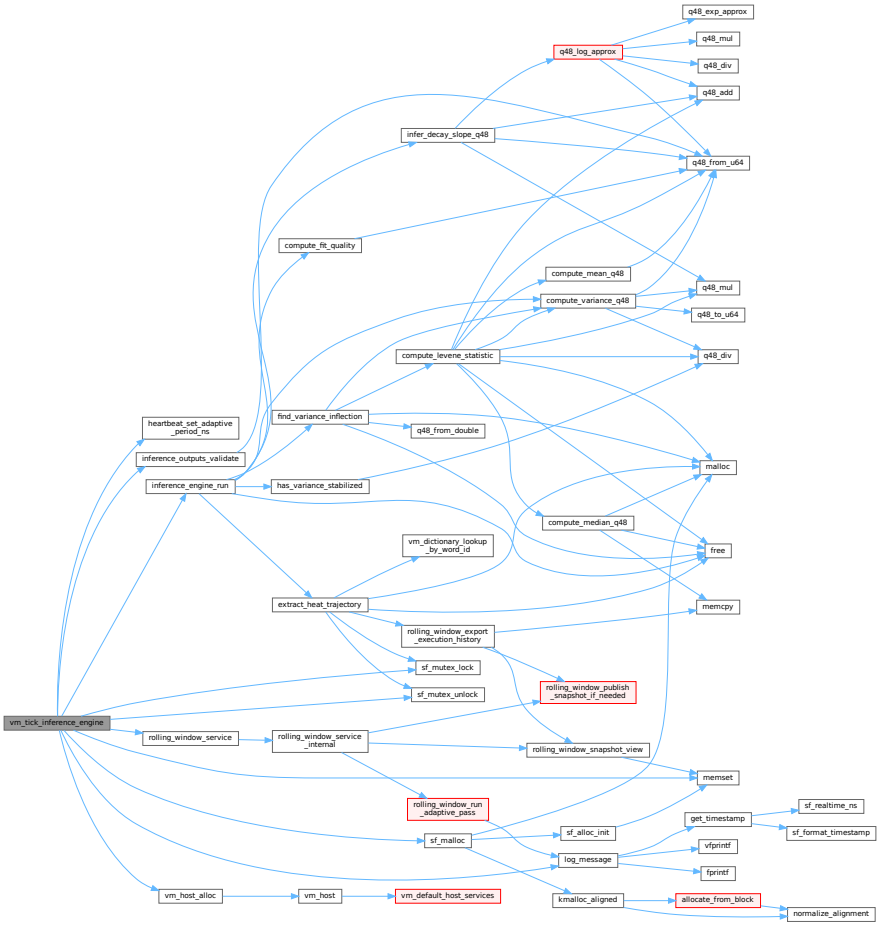
Parameters

vm	Pointer to VM instance

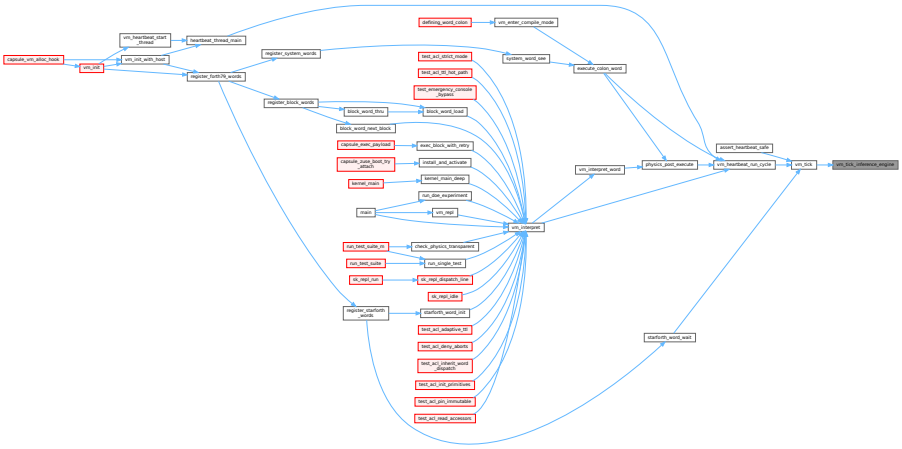
References InferenceOutputs::adaptive_decay_slope, InferenceOutputs::adaptive_window_width, VM::decay_slope_q48, VM::dict_lock, HeartbeatState::early_exit_count, InferenceOutputs::early_exited, RollingWindowOfTruth::effective_window_size, VM::heartbeat, HeartbeatState::heartbeat_enabled, heartbeat_set_adaptive_period_ns(), HEARTBEAT_TICK_NS, HOTWORDS_EXECUTION_HEAT_THRESHOLD, inference_engine_run(), inference_outputs_validate(), HeartbeatState::inference_, RollingWindowOfTruth::is_warm, VM::last_inference_outputs, HeartbeatState::last_inference_tick, VM::latest, LOG_DEBUG, LOG_ERROR, LOG_INFO, log_message(), memset(), VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempt, PipelineGlobalMetrics::prefetch_hits, VM::rolling_window, rolling_window_service(), sf_malloc(), sf_mutex_lock(), sf_mutex_unlock(), InferenceOutputs::slope_fit_quality_q48, VM::stale_word_count_at_check, HeartbeatState::tick_count, HeartbeatWorker::tick_ns, HeartbeatState::tick_target_ns, RollingWindowOfTruth::total_executions, VM::total_heat_at_last_check, VM::tuning_lock, vm_host_alloc(), InferenceInputs::window, RollingWindowOfTruth::window_pos, InferenceOutputs::window_variance, word_count, VM::word_count_at_check, and HeartbeatState::worker.

Referenced by vm_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.194.1.11 vm_tick_slope_validator()

```
void vm_tick_slope_validator (
```

```
VM * vm )
```

Loop #3: Heat decay slope validation via periodic measurement.

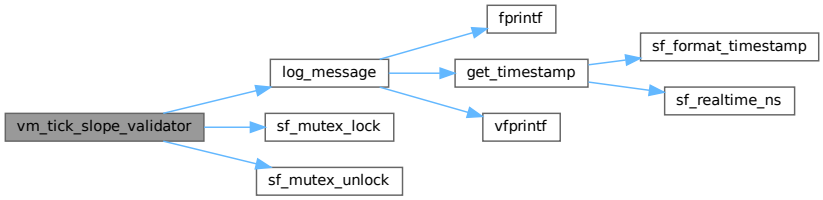
Validates that linear decay is actually helping optimize dictionary caching. Measures stale word ratio, hot word count, and average heat distribution.

Parameters

vm	Pointer to VM instance

References VM::decay_slope_direction, VM::decay_slope_q48, VM::dict_lock, VM::hot_word_count_at_check, HOTWORDS_EXECUTION_HEAT_THRESHOLD, VM::latest, LOG_DEBUG, LOG_INFO, log_message(), sf_mutex_lock(), sf_mutex_unlock(), VM::stale_word_count_at_check, VM::total_heat_at_last_check, VM::tuning_lock, word_count, and VM::word_count_at_check.

Here is the call graph for this function:



7.194.1.12 vm_tick_window_tuner()

```
void vm_tick_window_tuner (
VM * vm )
```

Loop #5: Context-aware window tuning via binary chop search.

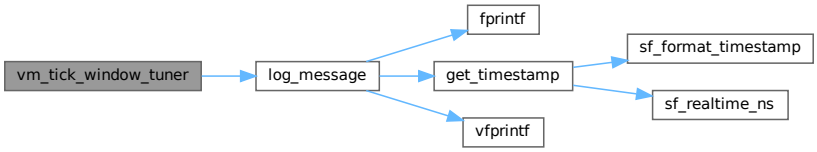
Uses prefetch accuracy to guide window size adaptation. Binary search converges on optimal effective_window_↩ size for current workload.

Parameters

vm	Pointer to VM instance

References ADAPTIVE_MIN_WINDOW_SIZE, RollingWindowOfTruth::effective_window_size, ENABLE_PIPELINING, RollingWindowOfTruth::is_warm, PipelineGlobalMetrics::last_checked_accuracy, PipelineGlobalMetrics::last_checked_window_size, LOG_DEBUG, LOG_INFO, log_message(), VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::rolling_window, ROLLING_WINDOW_SIZE, and PipelineGlobalMetrics::window_tuning_ch

Here is the call graph for this function:

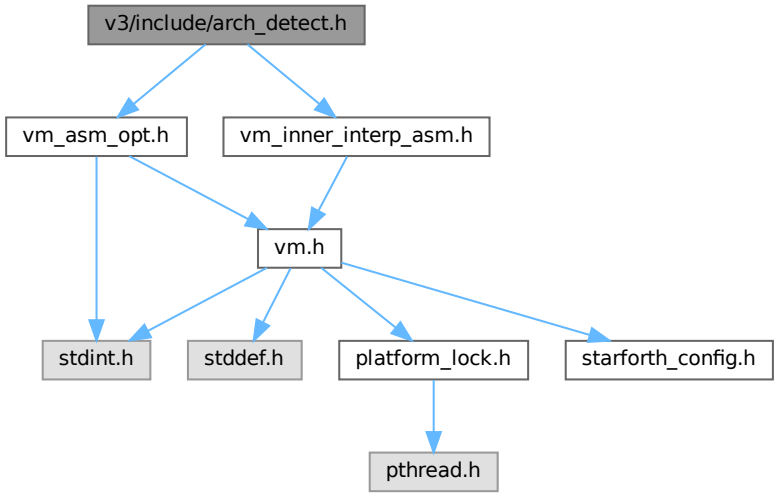


7.195 v3/include/arch_detect.h File Reference

Architecture Detection and Unified API for StarForth.

```
#include "vm_asm_opt.h"
#include "vm_inner_interp_asm.h"
```

Include dependency graph for arch_detect.h:



Macros

- #define ARCH_X86_64 0
- #define ARCH_ARM64 0
- #define ARCH_RISCV64 0
- #define ARCH_NAME "Unknown"
- #define vm_push_opt(vm, val) vm_push_asm(vm, val)
- #define vm_pop_opt(vm) vm_pop_asm(vm)
- #define vm_rpush_opt(vm, val) vm_rpush_asm(vm, val)
- #define vm_rpop_opt(vm) vm_rpop_asm(vm)
- #define vm_add_overflow_opt(a, b, res) vm_add_check_overflow(a, b, res)

- #define vm_sub_overflow_opt(a, b, res) vm_sub_check_overflow(a, b, res)
- #define vm_mul_double_opt(a, b, hi, lo) vm_mul_double(a, b, hi, lo)
- #define vm_divmod_opt(dividend, divisor, quot, rem) vm_divmod(dividend, divisor, quot, rem)
- #define vm_strcmp_opt(s1, s2, len) vm_strcmp_asm(s1, s2, len)
- #define vm_memcpy_opt(dst, src, len) vm_memcpy_asm(dst, src, len)
- #define vm_memzero_opt(dst, len) vm_memzero_asm(dst, len)
- #define vm_min_opt(a, b) vm_min_asm(a, b)
- #define vm_max_opt(a, b) vm_max_asm(a, b)
- #define vm_abs_opt(a) vm_abs_asm(a)
- #define vm_clz_opt(x) vm_clz(x)
- #define vm_ctz_opt(x) vm_ctz(x)
- #define vm_popcnt_opt(x) vm_popcnt(x)
- #define vm_prefetch_opt(addr) vm_prefetch(addr)
- #define vm_prefetch_stream_opt(addr) vm_prefetch_stream(addr)
- #define NEXT_OPT() NEXT_ASM()
- #define vm_setup_registers_opt(vm) vm_setup_registers(vm)
- #define vm_save_registers_opt(vm) vm_save_registers(vm)
- #define PRIM_DUP_OPT() PRIM_DUP()
- #define PRIM_DROP_OPT() PRIM_DROP()
- #define PRIM_SWAP_OPT() PRIM_SWAP()
- #define PRIM_PLUS_OPT() PRIM_PLUS()
- #define PRIM_MINUS_OPT() PRIM_MINUS()
- #define PRIM_STAR_OPT() PRIM_STAR()
- #define PRIM_FETCH_OPT() PRIM_FETCH()
- #define PRIM_STORE_OPT() PRIM_STORE()
- #define PRIM_TO_R_OPT() PRIM_TO_R()
- #define PRIM_R_FROM_OPT() PRIM_R_FROM()
- #define PRIM_R_FETCH_OPT() PRIM_R_FETCH()
- #define PRIM_BRANCH_OPT() PRIM_BRANCH()
- #define PRIM_ZBRANCH_OPT() PRIM_ZBRANCH()

Functions

- static int vm_has_sse42_opt (void)
- static const char * vm_get_arch_name (void)
- static int vm_has_asm_opt (void)
- static int vm_has_direct_threading (void)
- static const char * vm_get_build_info (void)

7.195.1 Detailed Description

Architecture Detection and Unified API for StarForth.

Automatically selects x86_64 or ARM64 optimizations and provides a unified interface for architecture-specific optimizations.

This work is released under license terms in LICENSE file. No warranty. Use at your own risk.

7.195.2 Macro Definition Documentation

7.195.2.1 ARCH_ARM64

```
#define ARCH_ARM64 0
```

7.195.2.2 ARCH_NAME

```
#define ARCH_NAME "Unknown"
```

7.195.2.3 ARCH_RISCV64

```
#define ARCH_RISCV64 0
```

7.195.2.4 ARCH_X86_64

```
#define ARCH_X86_64 0
```

7.195.2.5 NEXT_OPT

```
#define NEXT_OPT( ) NEXT_ASM()
```

7.195.2.6 PRIM_BRANCH_OPT

```
#define PRIM_BRANCH_OPT( ) PRIM_BRANCH()
```

7.195.2.7 PRIM_DROP_OPT

```
#define PRIM_DROP_OPT( ) PRIM_DROP()
```

7.195.2.8 PRIM_DUP_OPT

```
#define PRIM_DUP_OPT( ) PRIM_DUP()
```

7.195.2.9 PRIM_FETCH_OPT

```
#define PRIM_FETCH_OPT( ) PRIM_FETCH()
```

7.195.2.10 PRIM_MINUS_OPT

```
#define PRIM_MINUS_OPT( ) PRIM_MINUS()
```

7.195.2.11 PRIM_PLUS_OPT

```
#define PRIM_PLUS_OPT( ) PRIM_PLUS()
```

7.195.2.12 PRIM_R_FETCH_OPT

```
#define PRIM_R_FETCH_OPT( ) PRIM_R_FETCH()
```

7.195.2.13 PRIM_R_FROM_OPT

```
#define PRIM_R_FROM_OPT( ) PRIM_R_FROM()
```

7.195.2.14 PRIM_STAR_OPT

```
#define PRIM_STAR_OPT( ) PRIM_STAR()
```

7.195.2.15 PRIM_STORE_OPT

```
#define PRIM_STORE_OPT( ) PRIM_STORE()
```

7.195.2.16 PRIM_SWAP_OPT

```
#define PRIM_SWAP_OPT( ) PRIM_SWAP()
```

7.195.2.17 PRIM_TO_R_OPT

```
#define PRIM_TO_R_OPT( ) PRIM_TO_R()
```

7.195.2.18 PRIM_ZBRANCH_OPT

```
#define PRIM_ZBRANCH_OPT( ) PRIM_ZBRANCH()
```

7.195.2.19 vm_abs_opt

```
#define vm_abs_opt(
              a ) vm_abs_asm(a)
```

7.195.2.20 vm_add_overflow_opt

```
#define vm_add_overflow_opt(
              a,
              b,
              res ) vm_add_check_overflow(a, b, res)
```

7.195.2.21 vm_clz_opt

```
#define vm_clz_opt(
              x ) vm_clz(x)
```

7.195.2.22 vm_ctz_opt

```
#define vm_ctz_opt(
              x ) vm_ctz(x)
```

7.195.2.23 vm_divmod_opt

```
#define vm_divmod_opt(
              dividend,
              divisor,
              quot,
              rem ) vm_divmod(dividend, divisor, quot, rem)
```

7.195.2.24 vm_max_opt

```
#define vm_max_opt(
              a,
              b ) vm_max_asm(a, b)
```

7.195.2.25 vm_memcpy_opt

```
#define vm_memcpy_opt(
              dst,
              src,
              len ) vm_memcpy_asm(dst, src, len)
```

7.195.2.26 vm_memzero_opt

```
#define vm_memzero_opt(
              dst,
              len ) vm_memzero_asm(dst, len)
```

7.195.2.27 vm_min_opt

```
#define vm_min_opt(
              a,
              b ) vm_min_asm(a, b)
```

7.195.2.28 vm_mul_double_opt

```
#define vm_mul_double_opt(
              a,
              b,
              hi,
              lo ) vm_mul_double(a, b, hi, lo)
```

7.195.2.29 vm_pop_opt

```
#define vm_pop_opt(
              vm ) vm_pop_asm(vm)
```

7.195.2.30 vm_popcnt_opt

```
#define vm_popcnt_opt(
              x ) vm_popcnt(x)
```

7.195.2.31 vm_prefetch_opt

```
#define vm_prefetch_opt(
              addr ) vm_prefetch(addr)
```

7.195.2.32 vm_prefetch_stream_opt

```
#define vm_prefetch_stream_opt(
              addr ) vm_prefetch_stream(addr)
```

7.195.2.33 vm_push_opt

```
#define vm_push_opt(
              vm,
              val ) vm_push_asm(vm, val)
```

7.195.2.34 vm_rpop_opt

```
#define vm_rpop_opt(
              vm ) vm_rpop_asm(vm)
```

7.195.2.35 vm_rpush_opt

```
#define vm_rpush_opt(
              vm,
              val ) vm_rpush_asm(vm, val)
```

7.195.2.36 vm_save_registers_opt

```
#define vm_save_registers_opt(
              vm ) vm_save_registers(vm)
```

7.195.2.37 vm_setup_registers_opt

```
#define vm_setup_registers_opt(
              vm ) vm_setup_registers(vm)
```

7.195.2.38 vm_strcmp_opt

```
#define vm_strcmp_opt(
              s1,
              s2,
              len ) vm_strcmp_asm(s1, s2, len)
```

7.195.2.39 vm_sub_overflow_opt

```
#define vm_sub_overflow_opt(
              a,
              b,
              res ) vm_sub_check_overflow(a, b, res)
```

7.195.3 Function Documentation

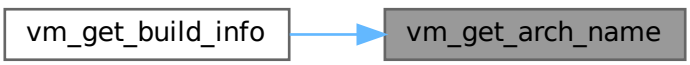
7.195.3.1 vm_get_arch_name()

```
static const char * vm_get_arch_name (
            void  )  [inline], [static]
```

References ARCH_NAME.

Referenced by vm_get_build_info().

Here is the caller graph for this function:

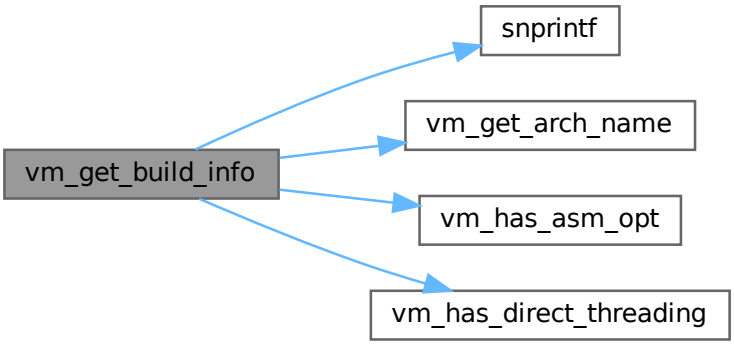


7.195.3.2 vm_get_build_info()

```
static const char * vm_get_build_info (
            void  )  [inline], [static]
```

References snprintf(), vm_get_arch_name(), vm_has_asm_opt(), and vm_has_direct_threading().

Here is the call graph for this function:



7.195.3.3 vm_has_asm_opt()

```
static int vm_has_asm_opt (
            void  )  [inline], [static]
```

Referenced by vm_get_build_info().

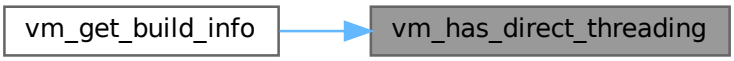
Here is the caller graph for this function:

7.195.3.4 vm_has_direct_threading()

```
static int vm_has_direct_threading (
            void  )  [inline], [static]
```

Referenced by vm_get_build_info().

Here is the caller graph for this function:

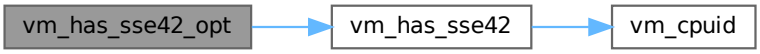


7.195.3.5 vm_has_sse42_opt()

```
static int vm_has_sse42_opt (
            void  )  [inline], [static]
```

References vm_has_sse42().

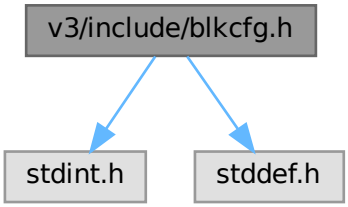
Here is the call graph for this function:



7.196 v3/include/blkcfg.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for blkcfg.h:



Data Structures

- struct blkcfg_t

Macros

- #define BLKCFG_DEFAULT_FBS 1024u /* default Forth Block Size in bytes */
- #define BLKCFG_PATH_MAX 4096

Enumerations

- enum blkcfg_backend_t { BLKCFG_BACKEND_NONE = 0 , BLKCFG_BACKEND_FILE = 1 , BLKCFG_BACKEND_RAM = 2 }

Functions

- int blkcfg_init_from_options (blkcfg_t *cfg, const char *disk_img_path, uint64_t ram_mb, uint32_t fbs, uint8_t read_only)

  Initialize a block configuration from CLI-style options.
- int blkcfg_format_summary (const blkcfg_t *cfg, char *dst, size_t dst_cap)

  Format a single-line summary into 'dst' (capacity 'dst_cap' bytes).

Variables

- blkcfg_t g_blkcfg

7.196.1 Macro Definition Documentation

7.196.1.1 BLKCFG_DEFAULT_FBS

```
#define BLKCFG_DEFAULT_FBS 1024u /* default Forth Block Size in bytes */
```

7.196.1.2 BLKCFG_PATH_MAX

```
#define BLKCFG_PATH_MAX 4096
```

7.196.2 Enumeration Type Documentation

7.196.2.1 blkcfg_backend_t

```
enum blkcfg_backend_t
```

Enumerator

BLKCFG_BACKEND_NONE	
BLKCFG_BACKEND_FILE	
BLKCFG_BACKEND_RAM	

7.196.3 Function Documentation

7.196.3.1 blkcfg_format_summary()

```
int blkcfg_format_summary (
            const blkcfg_t * cfg,
            char * dst,
            size_t dst_cap )
```

Format a single-line summary into 'dst' (capacity 'dst_cap' bytes).

Example: "backend=FILE path=./disks/os.img fbs=1024 ro=0" "backend=RAM size=1024MB fbs=1024 blocks=1048576 ro=0"

Returns number of bytes that would have been written (like snprintf).

7.196.3.2 blkcfg_init_from_options()

```
int blkcfg_init_from_options (
            blkcfg_t * cfg,
            const char * disk_img_path,
            uint64_t ram_mb,
            uint32_t fbs,
            uint8_t read_only )
```

Initialize a block configuration from CLI-style options.

Inputs:

- disk_img_path: if non-NULL and non-empty => choose FILE backend
- ram_mb: megabytes for RAM fallback (only used if no disk image)
- fbs: forth block size (bytes). If 0 => BLKCFG_DEFAULT_FBS
- read_only: intended read-only flag (advisory)

Behavior:

- FILE: record path, set blocks/size_bytes = 0 (unknown until attached)
- RAM : compute size_bytes = ram_mb∗MiB; blocks = size_bytes/fbs (floored)

Returns: 0 on success; -1 on invalid args.

7.196.4 Variable Documentation

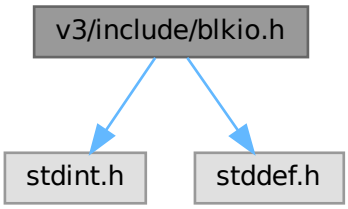
7.196.4.1 g_blkcfg

```
blkcfg_t g_blkcfg  [extern]
```

7.197 v3/include/blkio.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for blkio.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct blkio_params_t
- struct blkio_info_t
- struct blkio_vtable_t
- struct blkio_dev

Macros

- #define BLKIO_FORTH_BLOCK_SIZE 1024u

Typedefs

- typedef struct blkio_dev blkio_dev_t

Enumerations

- enum {
  BLKIO_OK = 0 , BLKIO_EINVAL = -1 , BLKIO_ENOSUP = -2 , BLKIO_EIO = -3 ,
  BLKIO_ENOSPACE = -4 , BLKIO_ECLOSED = -5 }

Functions

- static int blkio_open (blkio_dev_t *dev, const blkio_vtable_t *vt, const blkio_params_t *p)
- static int blkio_close (blkio_dev_t *dev)
- static int blkio_read (blkio_dev_t *dev, uint32_t fblock, void *dst)
- static int blkio_write (blkio_dev_t *dev, uint32_t fblock, const void *src)
- static int blkio_flush (blkio_dev_t *dev)
- static int blkio_info (blkio_dev_t *dev, blkio_info_t *out)
- static void blkio_bzero (void *p, uint32_t fbs)

7.197.1 Macro Definition Documentation

7.197.1.1 BLKIO_FORTH_BLOCK_SIZE

```
#define BLKIO_FORTH_BLOCK_SIZE 1024u
```

7.197.2 Typedef Documentation

7.197.2.1 blkio_dev_t

```
typedef struct blkio_dev blkio_dev_t
```

7.197.3 Enumeration Type Documentation

7.197.3.1 anonymous enum

```
anonymous enum
```

Enumerator

BLKIO_OK	
BLKIO_EINVAL	
BLKIO_ENOSUP	
BLKIO_EIO	
BLKIO_ENOSPACE	
BLKIO_ECLOSED	

7.197.4 Function Documentation

7.197.4.1 blkio_bzero()

```
static void blkio_bzero (
            void * p,
            uint32_t fbs )  [inline], [static]
```

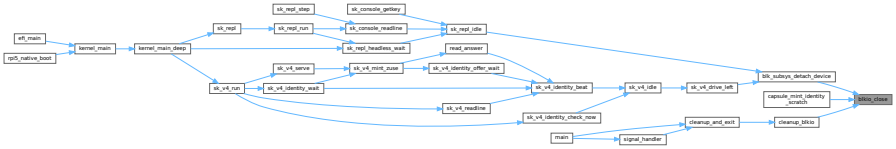
7.197.4.2 blkio_close()

```
static int blkio_close (
            blkio_dev_t * dev )  [inline], [static]
```

References BLKIO_ECLOSED, blkio_vtable_t::close, blkio_dev::is_open, and blkio_dev::vt.

Referenced by blk_subsys_detach_device(), capsule_mint_identity_scratch(), and cleanup_blkio().

Here is the caller graph for this function:



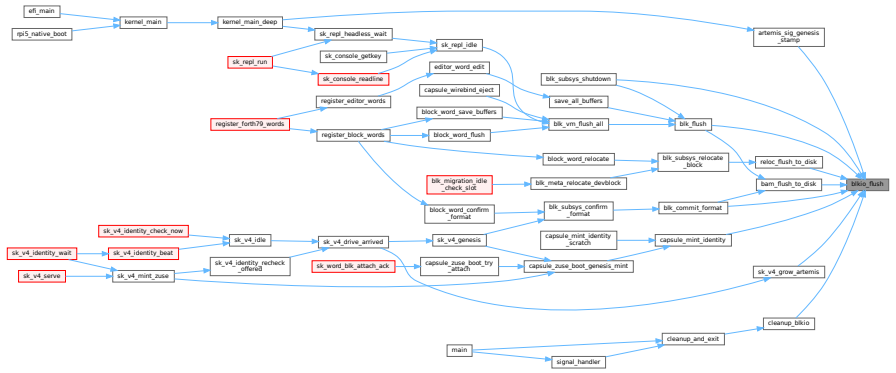
7.197.4.3 blkio_flush()

```
static int blkio_flush (
            blkio_dev_t * dev )  [inline], [static]
```

References BLKIO_ECLOSED, blkio_vtable_t::flush, blkio_dev::is_open, and blkio_dev::vt.

Referenced by artemis_sig_genesis_stamp(), bam_flush_to_disk(), blk_commit_format(), blk_flush(), blk_subsys_shutdown(), capsule_mint_identity(), cleanup_blkio(), reloc_flush_to_disk(), and sk_v4_grow_artemis().

Here is the caller graph for this function:



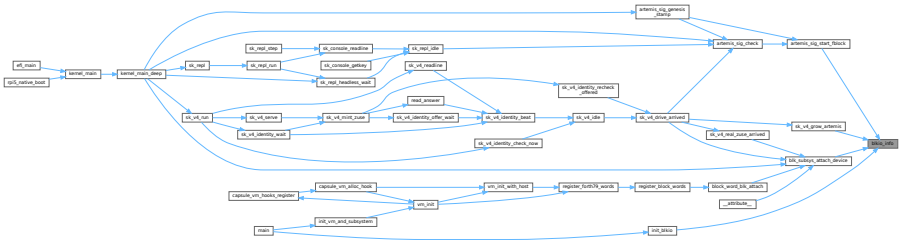
7.197.4.4 blkio_info()

```
static int blkio_info (
            blkio_dev_t * dev,
            blkio_info_t * out )  [inline], [static]
```

References BLKIO_EINVAL, BLKIO_OK, blkio_info_t::forth_block_size, blkio_dev::forth_block_size, blkio_vtable_t::info, blkio_dev::is_open, blkio_info_t::total_blocks, blkio_dev::total_blocks, and blkio_dev::vt.

Referenced by artemis_sig_start_fblock(), blk_subsys_attach_device(), init_blkio(), and sk_v4_grow_artemis().

Here is the caller graph for this function:



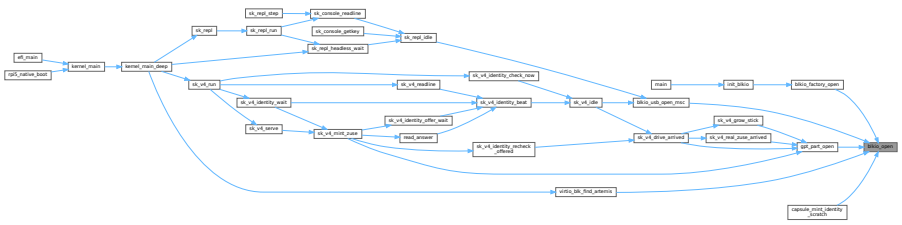
7.197.4.5 blkio_open()

```
static int blkio_open (
            blkio_dev_t * dev,
            const blkio_vtable_t * vt,
            const blkio_params_t * p )  [inline], [static]
```

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_vtable_t::close, blkio_vtable_t::flush, blkio_params_t::forth_block_size, blkio_dev::forth_block_size, blkio_vtable_t::info, blkio_dev::is_open, blkio_vtable_t::open, blkio_vtable_t::read, blkio_dev::state, blkio_params_t::total_blocks, blkio_dev::total_blocks, blkio_dev::vt, and blkio_vtable_t::write.

Referenced by blkio_factory_open(), blkio_usb_open_msc(), capsule_mint_identity_scratch(), gpt_part_open(), and virtio_blk_find_artemis().

Here is the caller graph for this function:



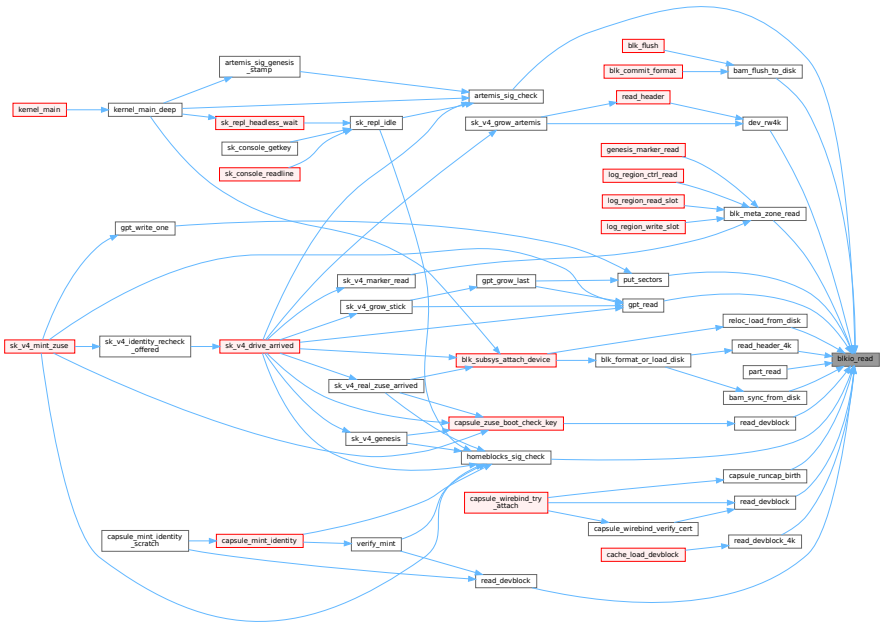
7.197.4.6 blkio_read()

```
static int blkio_read (
            blkio_dev_t * dev,
            uint32_t fblock,
            void * dst )  [inline], [static]
```

References BLKIO_EINVAL, blkio_dev::is_open, blkio_vtable_t::read, blkio_dev::total_blocks, and blkio_dev::vt.

Referenced by artemis_sig_check(), bam_flush_to_disk(), bam_sync_from_disk(), blk_meta_zone_read(), capsule_runcap_birth(), dev_rw4k(), gpt_read(), homeblocks_sig_check(), part_read(), put_sectors(), read_devblock(), read_devblock(), read_devblock_4k(), read_header_4k(), and reloc_load_from_disk().

Here is the caller graph for this function:



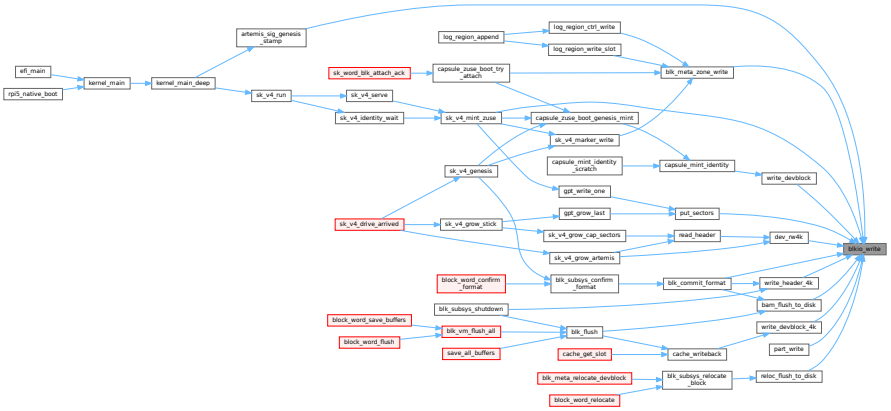
7.197.4.7 blkio_write()

```
static int blkio_write (
            blkio_dev_t * dev,
            uint32_t fblock,
            const void * src )  [inline], [static]
```

References BLKIO_EINVAL, blkio_dev::is_open, blkio_dev::total_blocks, blkio_dev::vt, and blkio_vtable_t::write.

Referenced by artemis_sig_genesis_stamp(), bam_flush_to_disk(), blk_commit_format(), blk_meta_zone_write(), dev_rw4k(), part_write(), put_sectors(), reloc_flush_to_disk(), sk_v4_mint_zuse(), write_devblock(), write_devblock_4k(), and write_header_4k().

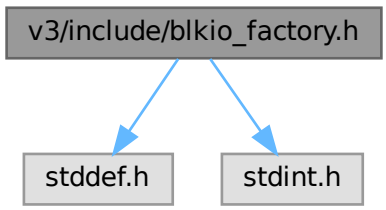
Here is the caller graph for this function:



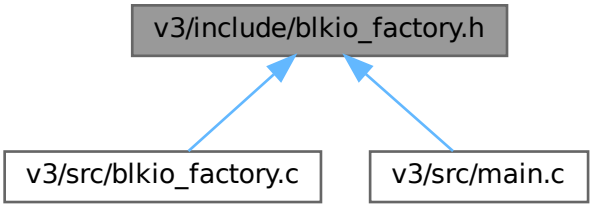
7.198 v3/include/blkio_factory.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for blkio_factory.h:



This graph shows which files directly or indirectly include this file:



Functions

- int blkio_factory_open (struct blkio_dev *dev, const char *disk_img_path, uint8_t read_only, uint32_t total_blocks, uint32_t fbs, void *state_mem, size_t state_len, uint8_t *ram_base, uint32_t ram_blocks, uint8_t *out_used_file)

  Open a block device using either a file-backed image (if disk_img_path is non-empty) or a RAM-backed buffer (fallback).

7.198.1 Function Documentation

7.198.1.1 blkio_factory_open()

```
int blkio_factory_open (
            blkio_dev_t * dev,
            const char * disk_img_path,
            uint8_t read_only,
            uint32_t total_blocks,
            uint32_t fbs,
            void * state_mem,
            size_t state_len,
            uint8_t * ram_base,
            uint32_t ram_blocks,
            uint8_t * out_used_file )
```

Open a block device using either a file-backed image (if disk_img_path is non-empty) or a RAM-backed buffer (fallback).

Parameters:

- dev : output device object (caller-allocated)
- disk_img_path : path to raw image; if NULL/empty => RAM fallback
- read_only : 1 = open read-only; 0 = read/write
- total_blocks : FILE: 0 => derive from file size; >0 => size hint RAM : must match ram_blocks
- fbs : forth block size in bytes (0 => default per blkio.h)
- state_mem : caller-provided backend state buffer (opaque to caller)
- state_len : size of state_mem in bytes
- ram_base : RAM storage base pointer (RAM backend only)
- ram_blocks : number of Forth blocks in ram_base (RAM backend only)
- out_used_file : optional; set to 1 if FILE backend selected, else 0

Returns: 0 on success; negative blkio error code on failure.

Open a block device using either a file-backed image (if disk_img_path is non-empty) or a RAM-backed buffer (fallback).

Selects the file backend when `disk_img_path` is non-empty, otherwise falls back to the RAM backend using `ram_base` / `ram_blocks`.

File backend behaviour:

- Creates the file if it does not exist (unless `read_only` is set).
- Derives `total_blocks` from the file size when `total_blocks` == 0.
- Sets `*out_used_file` = 1 on success.

RAM backend behaviour:

- `ram_base` must point to at least `ram_blocks * fbs` bytes of caller-owned memory that outlives the device.

Parameters

dev	Output device handle to populate
disk_img_path	Path to disk image file, or NULL/empty for RAM
read_only	1 = open read-only, 0 = read-write
total_blocks	Desired block count (0 = derive from file size)
fbs	FORTH block size in bytes (0 = use `BLKIO_FORTH_BLOCK_SIZE`)
state_mem	Caller-allocated state buffer (must be large enough)
state_len	Size of `state_mem` in bytes
ram_base	Base of RAM block array (used only when no disk path)
ram_blocks	Number of blocks in `ram_base`
out_used_file	Optional output: set to 1 if the file backend was used

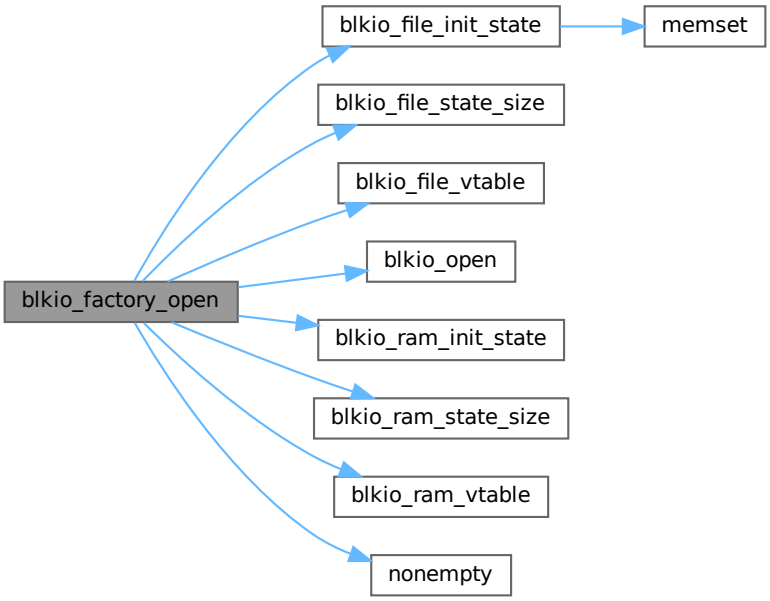
Returns

`BLKIO_OK` on success, or a `BLKIO_E*` error code

References BLKIO_EINVAL, BLKIO_ENOSPACE, blkio_file_init_state(), blkio_file_state_size(), blkio_file_vtable(), BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_open(), blkio_ram_init_state(), blkio_ram_state_size(), blkio_ram_vtable(), blkio_params_t::forth_block_size, nonempty(), and ram_base.

Referenced by init_blkio().

Here is the call graph for this function:

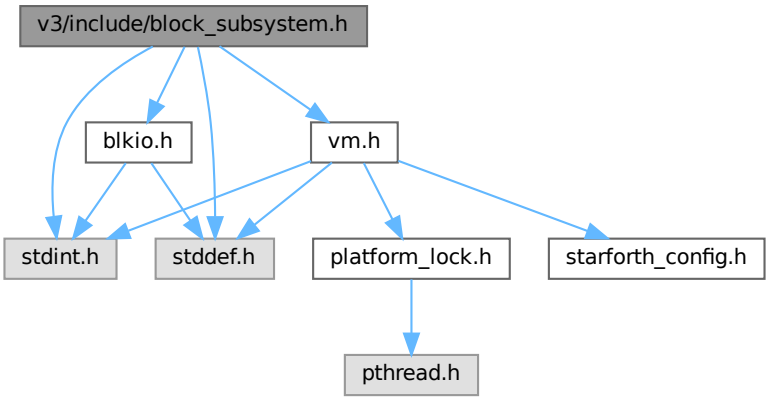


Here is the caller graph for this function:

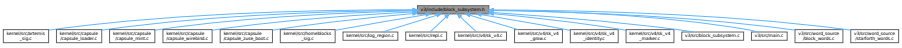
7.199 v3/include/block_subsystem.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "vm.h"
#include "blkio.h"
```

Include dependency graph for block_subsystem.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct blk_volume_meta_t
- struct blk_meta_t
- struct blk_bam_entry_t

Macros

- #define BLK_FORTH_SIZE 1024u /∗ Forth block size ∗/
- #define BLK_RAM_BLOCKS 2080u /∗ Physical RAM blocks (user-visible LBN 0..2047) ∗/
- #define BLK_DISK_START 2080u /∗ Device-backed Forth blocks start at LBN 2080 ∗/
- #define BLK_DEVICE_SECTOR 4096u /∗ Physical “devblock” size (4×1 KiB blkio units) ∗/
- #define BLK_PACK_RATIO 3u /∗ 3× 1 KiB data per 4 KiB devblock (plus 1 KiB metadata) ∗/
- #define BLK_META_TOTAL 1024u /∗ Last 1 KiB in a 4 KiB devblock is metadata ∗/
- #define BLK_META_PER_BLOCK 341u /∗ 341×3 ∼= 1023, padded to 1024 ∗/
- #define BLK_FORTH_SYS_RESERVED 32u /∗ RAM physical blocks 0..31 hidden; user LBN 0 = PBN 32 ∗/
- #define BLK_DISK_SYS_RESERVED 32u /∗ First 32 blocks of each disk reserved (byte-aligned BAM offset) ∗/
- #define BLK_META_FENCE_INIT
- #define BLK_FLAG_CLAIMED (1ull << 0) /∗ owned, per BMAPFMT's owner_fp ∗/
- #define BLK_FLAG_MIGRATING (1ull << 1) /∗ mid-migration; serves MIGSM ∗/
- #define BLK_FLAG_STALE (1ull << 2) /∗ interrupted flush; serves UNCLEAN ∗/

Enumerations

- enum {
  BLK_OK = 0 , BLK_EINVAL = -1 , BLK_ERANGE = -2 , BLK_EIO = -3 ,
  BLK_ENODEV = -4 , BLK_ERESERVED = -5 , BLK_EDIRTY = -6 , BLK_ENOMEM = -7 ,
  BLK_ENOSPC = -8 }
- enum { BLK_FMT_FORMATTED = 0 , BLK_FMT_PROVISIONAL = 1 }

Functions

- int blk_subsys_init (uint8_t *ram_base, size_t ram_size)
- int blk_subsys_attach_device (struct blkio_dev *dev)
- int blk_subsys_detach_device (struct blkio_dev *dev)
- uint64_t blk_subsys_epoch (void)
- int blk_subsys_relocate_block (uint32_t home_lbn, uint32_t target_lbn)
- int blk_get_device_range (struct blkio_dev *dev, uint32_t *out_start_lbn, uint32_t *out_count)
- const void * blk_lbn_device_handle (uint32_t lbn)
- int blk_get_device_free_blocks (struct blkio_dev *dev, uint64_t *out_free, uint64_t *out_total)
- int blk_get_first_disk_range (uint32_t *out_start_lbn, uint32_t *out_count)
- int blk_firsttouch_claim (const uint8_t owner_fp[8], uint32_t count, uint32_t *out_chain_head)
- int blk_owner_has_claim (const uint8_t owner_fp[8])
- int blk_meta_relocate_devblock (uint32_t home_devblock, uint32_t target_devblock)
- void blk_migration_idle_check (void)
- int blk_subsys_shutdown (void)
- uint8_t * blk_get_buffer (uint32_t block_num, int writable)
- uint8_t * blk_get_empty_buffer (uint32_t block_num)
- int blk_flush (uint32_t block_num)
- int blk_update (uint32_t block_num)
- int blk_get_volume_meta (blk_volume_meta_t *meta)
- int blk_set_volume_meta (const blk_volume_meta_t *meta)
- int blk_meta_zone_read (uint32_t devblock_from_top, uint8_t buf[4096])
- int blk_meta_zone_write (uint32_t devblock_from_top, const uint8_t buf[4096])
- uint64_t compute_crc64 (const uint8_t *data, size_t len)
- int blk_is_valid (uint32_t block_num)
- uint32_t blk_get_total_blocks (void)
- int blk_get_meta (uint32_t block_num, blk_meta_t *meta)
- int blk_set_meta (uint32_t block_num, const blk_meta_t *meta)
- int blk_owner_fp_get (uint32_t block_num, uint8_t out_fp[8])
- int blk_owner_fp_set (uint32_t block_num, const uint8_t fp[8])
- int blk_acl_allow_get (uint32_t block_num, uint8_t *out_allow)
- int blk_acl_allow_set (uint32_t block_num, uint8_t allow)
- int blk_acl_ttl_get (uint32_t block_num, uint32_t *out_ttl)
- int blk_acl_ttl_set (uint32_t block_num, uint32_t ttl)
- int blk_flags_get (uint32_t block_num, uint64_t *out_flags)
- int blk_flags_set (uint32_t block_num, uint64_t flags)
- int blk_is_allocated (uint32_t block_num)
- int blk_mark_allocated (uint32_t block_num)
- int blk_mark_free (uint32_t block_num)
- int blk_allocate (uint32_t *block_num)
- int blk_subsys_add_raw_device (uint8_t *buf, uint32_t nblocks)
- int blk_subsys_confirm_format (uint32_t lbn)
- void blk_volume_compute_totals (blk_volume_meta_t *m)
- void blk_subsys_set_format_bam_for (uint64_t total_devblocks)

7.199.1 Macro Definition Documentation

7.199.1.1 BLK_DEVICE_SECTOR

```
#define BLK_DEVICE_SECTOR 4096u /* Physical "devblock" size (4×1 KiB blkio units) */
```

7.199.1.2 BLK_DISK_START

```
#define BLK_DISK_START 2080u /* Device-backed Forth blocks start at LBN 2080 */
```

7.199.1.3 BLK_DISK_SYS_RESERVED

```
#define BLK_DISK_SYS_RESERVED 32u /* First 32 blocks of each disk reserved (byte-aligned BAM
offset) */
```

7.199.1.4 BLK_FLAG_CLAIMED

```
#define BLK_FLAG_CLAIMED (1ull << 0) /* owned, per BMAPFMT's owner_fp */
```

7.199.1.5 BLK_FLAG_MIGRATING

```
#define BLK_FLAG_MIGRATING (1ull << 1) /* mid-migration; serves MIGSM */
```

7.199.1.6 BLK_FLAG_STALE

```
#define BLK_FLAG_STALE (1ull << 2) /* interrupted flush; serves UNCLEAN */
```

7.199.1.7 BLK_FORTH_SIZE

```
#define BLK_FORTH_SIZE 1024u /* Forth block size */
```

7.199.1.8 BLK_FORTH_SYS_RESERVED

```
#define BLK_FORTH_SYS_RESERVED 32u /* RAM physical blocks 0..31 hidden; user LBN 0 = PBN 32 */
```

7.199.1.9 BLK_META_FENCE_INIT

```
#define BLK_META_FENCE_INIT
```

Value:

```
128u  /* Starting size (4 KiB devblocks -- 512 KiB) of the
* top-of-device system-metadata fence, grows downward
* from here. */
```

7.199.1.10 BLK_META_PER_BLOCK

```
#define BLK_META_PER_BLOCK 341u /* 341×3 ∼= 1023, padded to 1024 */
```

7.199.1.11 BLK_META_TOTAL

```
#define BLK_META_TOTAL 1024u /* Last 1 KiB in a 4 KiB devblock is metadata */
```

7.199.1.12 BLK_PACK_RATIO

```
#define BLK_PACK_RATIO 3u /* 3× 1 KiB data per 4 KiB devblock (plus 1 KiB metadata) */
```

7.199.1.13 BLK_RAM_BLOCKS

```
#define BLK_RAM_BLOCKS 2080u /* Physical RAM blocks (user-visible LBN 0..2047) */
```

7.199.2 Enumeration Type Documentation

7.199.2.1 anonymous enum

```
anonymous enum
```

Enumerator

BLK_OK	
BLK_EINVAL	
BLK_ERANGE	
BLK_EIO	
BLK_ENODEV	
BLK_ERESERVED	
BLK_EDIRTY	
BLK_ENOMEM	
BLK_ENOSPC	

7.199.2.2 anonymous enum

```
anonymous enum
```

Enumerator

BLK_FMT_FORMATTED	
BLK_FMT_PROVISIONAL	

7.199.3 Function Documentation

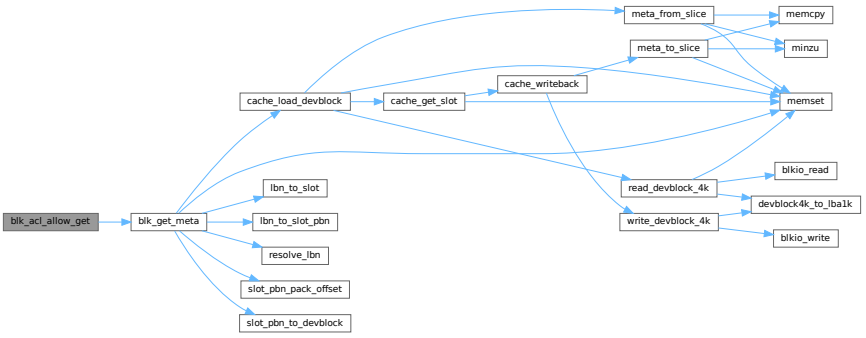
7.199.3.1 blk_acl_allow_get()

```
int blk_acl_allow_get (
            uint32_t block_num,
            uint8_t * out_allow )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, and meta.

Referenced by block_word_acl_allow_fetch().

Here is the call graph for this function:



Here is the caller graph for this function:

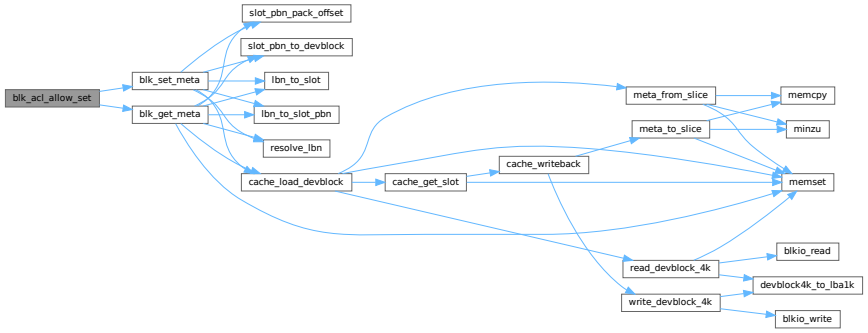
7.199.3.2 blk_acl_allow_set()

```
int blk_acl_allow_set (
            uint32_t block_num,
            uint8_t allow )
```

References blk_get_meta(), BLK_OK, blk_set_meta(), and meta.

Referenced by block_word_acl_allow_store().

Here is the call graph for this function:



Here is the caller graph for this function:

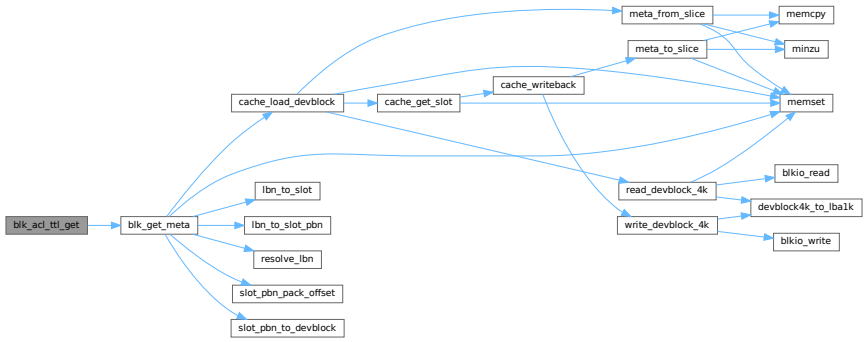
7.199.3.3 blk_acl_ttl_get()

```
int blk_acl_ttl_get (
            uint32_t block_num,
            uint32_t * out_ttl )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, and meta.

Referenced by block_word_acl_ttl_fetch().

Here is the call graph for this function:



Here is the caller graph for this function:

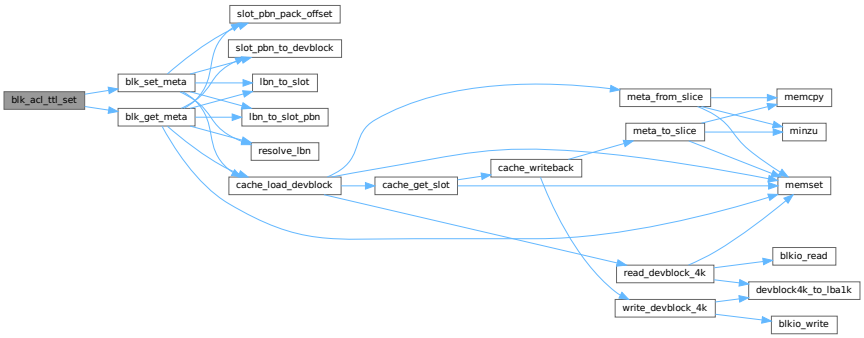
7.199.3.4 blk_acl_ttl_set()

```
int blk_acl_ttl_set (
            uint32_t block_num,
            uint32_t ttl )
```

References blk_get_meta(), BLK_OK, blk_set_meta(), and meta.

Referenced by block_word_acl_ttl_store().

Here is the call graph for this function:



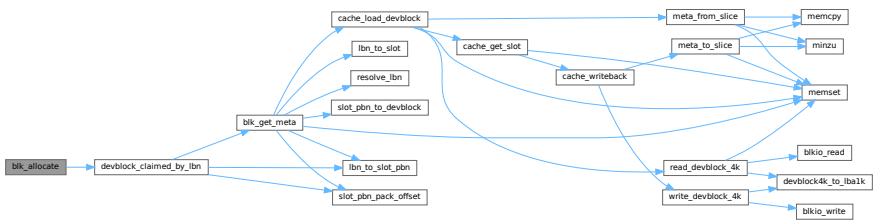
Here is the caller graph for this function:

7.199.3.5 blk_allocate()

```
int blk_allocate (
            uint32_t * block_num )
```

References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_dev_slot::bam_dirty, BLK_EINVAL, BLK_ENODEV, BLK_ERANGE, BLK_FMT_PROVISIONAL, BLK_OK, blk_dev_slot::dev, devblock_claimed_by_lbn(), blk_volume_meta_t::first_free, blk_dev_slot::format_state, blk_volume_meta_t::free_blocks, g, blk_volume_meta_t::last_allocated, limit, blk_dev_slot::next, blk_dev_slot::raw_base, blk_dev_slot::start_lbn, blk_dev_slot::user_blocks, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Here is the call graph for this function:



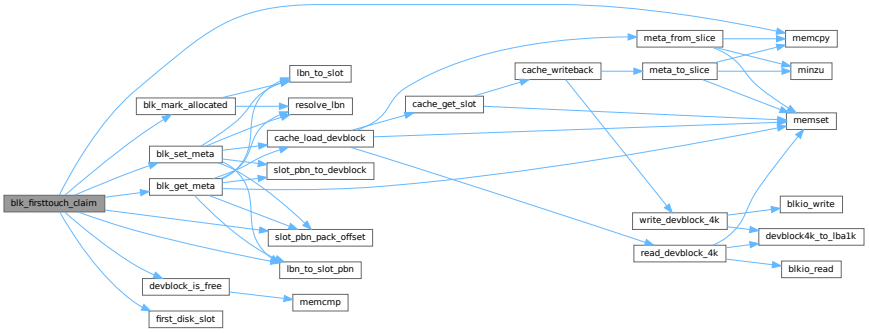
7.199.3.6 blk_firsttouch_claim()

```
int blk_firsttouch_claim (
            const uint8_t owner_fp[8],
            uint32_t count,
            uint32_t * out_chain_head )
```

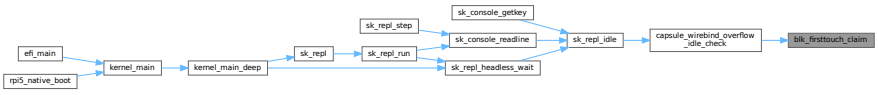
References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ENOSPC, BLK_FIRSTTOUCH_MAX_CLAIM, BLK_FLAG_CLAIMED, blk_get_meta(), blk_mark_allocated(), BLK_OK, BLK_PACK_RATIO, blk_set_meta(), count, devblock_is_free(), first_disk_slot(), blk_volume_meta_t::flags, lbn_to_slot_pbn(), memcpy(), meta, slot_pbn_pack_offset(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by capsule_wirebind_overflow_idle_check().

Here is the call graph for this function:



Here is the caller graph for this function:

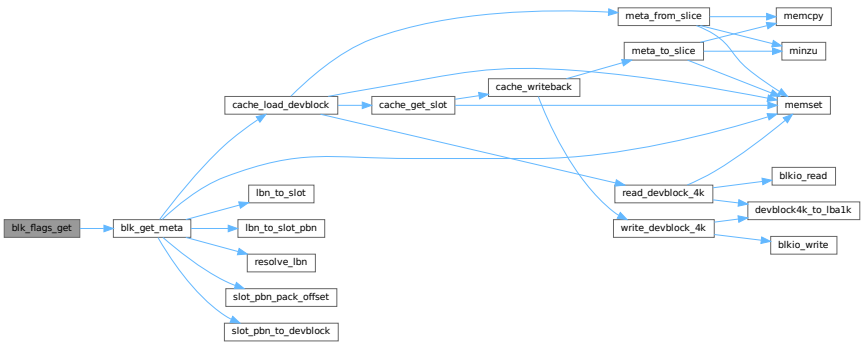


7.199.3.7 blk_flags_get()

```
int blk_flags_get (
            uint32_t block_num,
            uint64_t * out_flags )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, blk_volume_meta_t::flags, and meta.

Here is the call graph for this function:

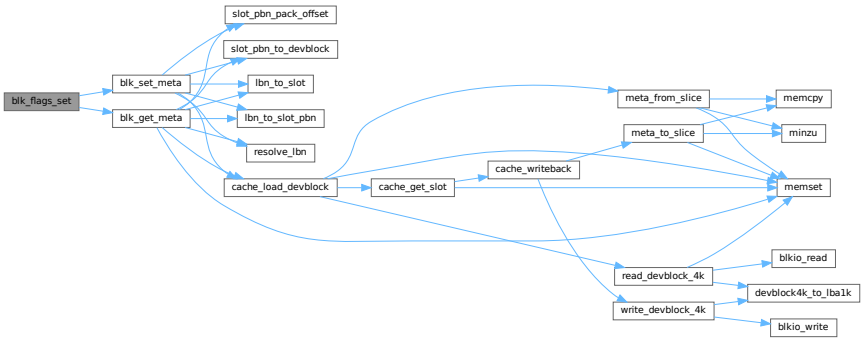


7.199.3.8 blk_flags_set()

```
int blk_flags_set (
            uint32_t block_num,
            uint64_t flags )
```

References blk_get_meta(), BLK_OK, blk_set_meta(), blk_volume_meta_t::flags, flags, and meta.

Here is the call graph for this function:



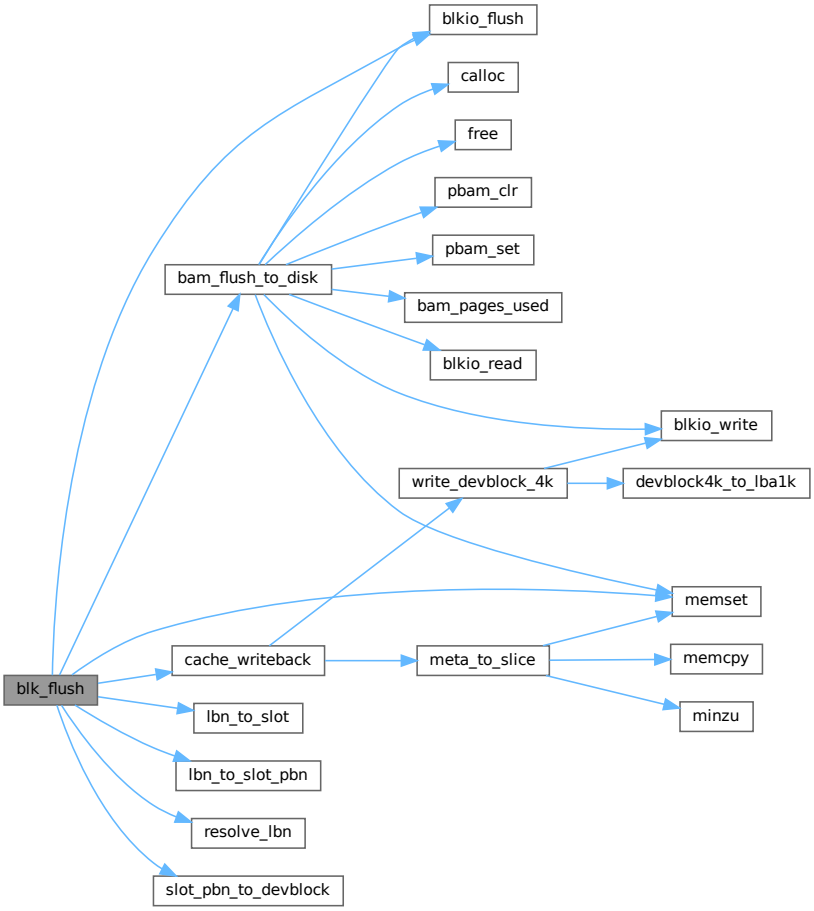
7.199.3.9 blk_flush()

```
int blk_flush (
            uint32_t block_num )
```

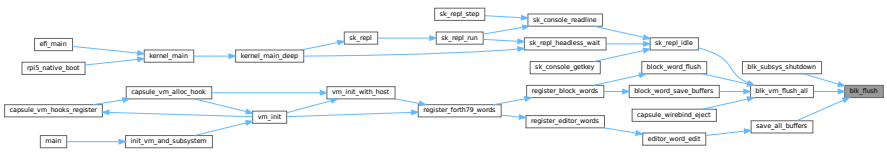
References blk_dev_slot::bam, bam_flush_to_disk(), BLK_ENODEV, BLK_ERANGE, BLK_FORTH_SYS_RESERVED, BLK_OK, BLK_RAM_BLOCKS, blkio_flush(), blk_dev_slot::cache, cache_writeback(), blk_dev_slot::dev, cache_slot_t::devblock, blk_bam_entry_t::dirty, cache_slot_t::dirty, DISK_CACHE_SLOTS, g, lbn_to_slot(), lbn_to_slot_pbn(), memset(), cache_slot_t::meta_dirty, blk_dev_slot::next, blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_to_devblock(), blk_dev_slot::start_lbn, blk_dev_slot::user_blocks, and cache_slot_t::valid.

Referenced by blk_subsys_shutdown(), blk_vm_flush_all(), and save_all_buffers().

Here is the call graph for this function:



Here is the caller graph for this function:



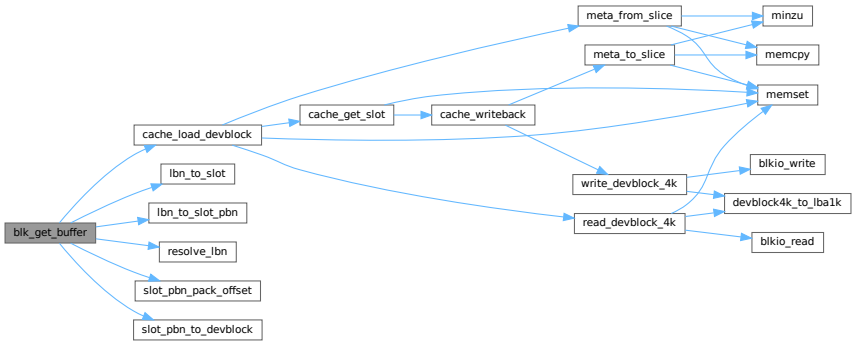
7.199.3.10 blk_get_buffer()

```
uint8_t * blk_get_buffer (
            uint32_t block_num,
            int writable )
```

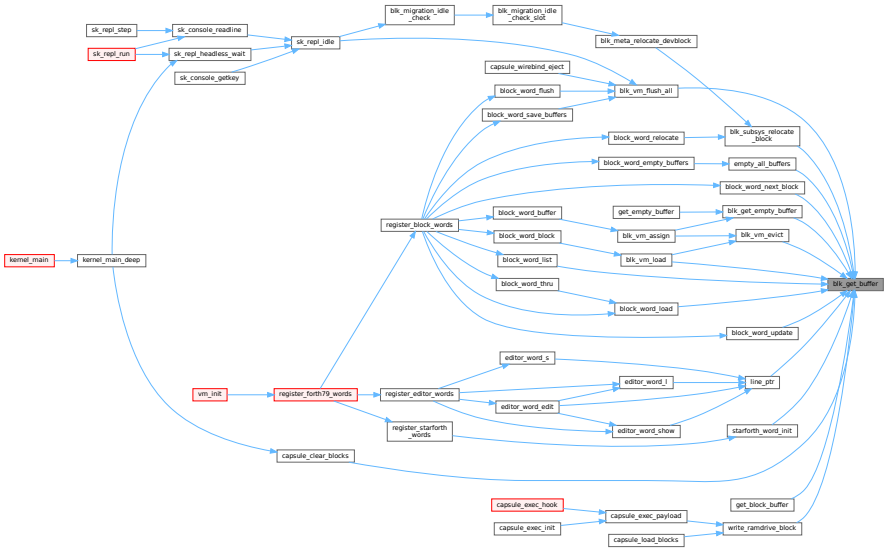
References blk_bam_entry_t::allocated, blk_dev_slot::bam, BLK_FMT_PROVISIONAL, BLK_FORTH_SIZE, BLK_FORTH_SYS_RESERVED, BLK_RAM_BLOCKS, cache_load_devblock(), cache_slot_t::data, blk_bam_entry_t::dirty, cache_slot_t::dirty, blk_dev_slot::format_state, g, lbn_to_slot(), lbn_to_slot_pbn(), blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_pack_offset(), slot_pbn_to_devblock(), and blk_dev_slot::start_lbn.

Referenced by blk_get_empty_buffer(), blk_subsys_relocate_block(), blk_vm_evict(), blk_vm_flush_all(), blk_vm_load(), block_word_list(), block_word_load(), block_word_next_block(), block_word_update(), capsule_clear_blocks(), empty_all_buffers(), get_block_buffer(), line_ptr(), starforth_word_init(), and write_ramdrive_block().

Here is the call graph for this function:



Here is the caller graph for this function:



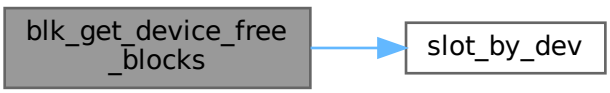
7.199.3.11 blk_get_device_free_blocks()

```
int blk_get_device_free_blocks (
            struct blkio_dev * dev,
            uint64_t * out_free,
            uint64_t * out_total )
```

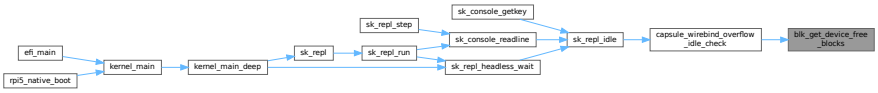
References BLK_EINVAL, BLK_ENODEV, BLK_OK, blk_volume_meta_t::free_blocks, blk_dev_slot::raw_base, slot_by_dev(), blk_volume_meta_t::total_blocks, and blk_dev_slot::vol_meta.

Referenced by capsule_wirebind_overflow_idle_check().

Here is the call graph for this function:



Here is the caller graph for this function:



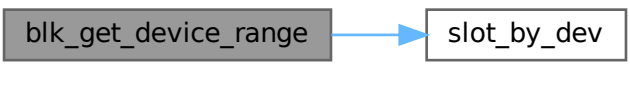
7.199.3.12 blk_get_device_range()

```
int blk_get_device_range (
            struct blkio_dev * dev,
            uint32_t * out_start_lbn,
            uint32_t * out_count )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, slot_by_dev(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by sk_v4_genesis().

Here is the call graph for this function:



Here is the caller graph for this function:

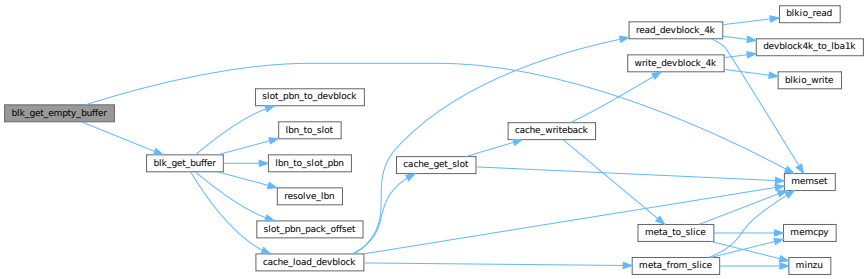
7.199.3.13 blk_get_empty_buffer()

```
uint8_t * blk_get_empty_buffer (
            uint32_t block_num )
```

References BLK_FORTH_SIZE, blk_get_buffer(), and memset().

Referenced by blk_vm_assign(), and get_empty_buffer().

Here is the call graph for this function:



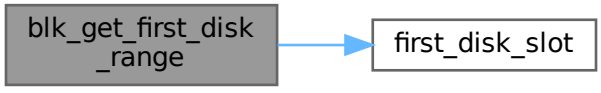
Here is the caller graph for this function:

7.199.3.14 blk_get_first_disk_range()

```
int blk_get_first_disk_range (
            uint32_t * out_start_lbn,
            uint32_t * out_count )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, first_disk_slot(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Here is the call graph for this function:



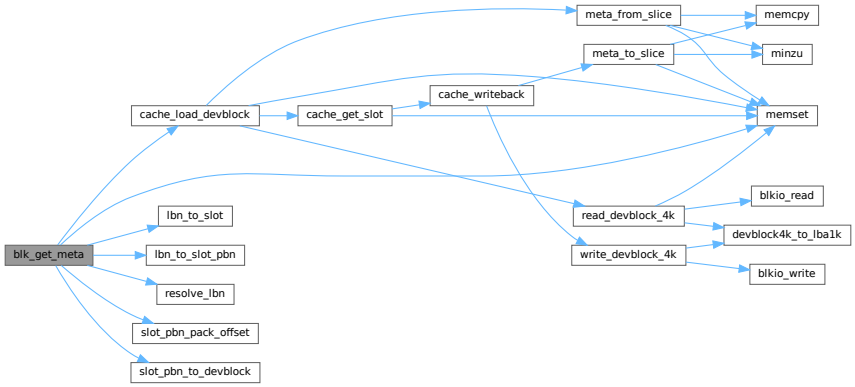
7.199.3.15 blk_get_meta()

```
int blk_get_meta (
            uint32_t block_num,
            blk_meta_t * meta )
```

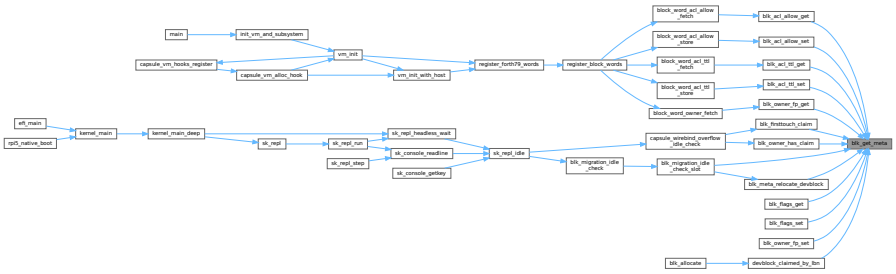
References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ERANGE, BLK_OK, cache_load_devblock(), g, lbn_to_slot(), lbn_to_slot_pbn(), blk_volume_meta_t::magic, memset(), meta, cache_slot_t::meta, blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_pack_offset(), and slot_pbn_to_devblock().

Referenced by blk_acl_allow_get(), blk_acl_allow_set(), blk_acl_ttl_get(), blk_acl_ttl_set(), blk_firsttouch_claim(), blk_flags_get(), blk_flags_set(), blk_meta_relocate_devblock(), blk_migration_idle_check_slot(), blk_owner_fp_get(), blk_owner_fp_set(), blk_owner_has_claim(), and devblock_claimed_by_lbn().

Here is the call graph for this function:



Here is the caller graph for this function:



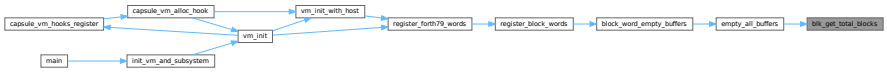
7.199.3.16 blk_get_total_blocks()

```
uint32_t blk_get_total_blocks (
            void  )
```

References g.

Referenced by empty_all_buffers().

Here is the caller graph for this function:

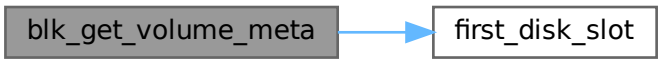


7.199.3.17 blk_get_volume_meta()

```
int blk_get_volume_meta (
            blk_volume_meta_t * meta )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, first_disk_slot(), meta, and blk_dev_slot::vol_meta.

Here is the call graph for this function:

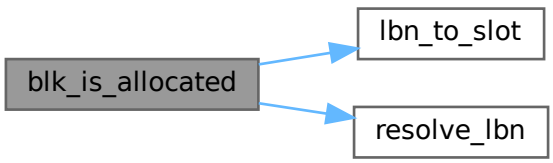


7.199.3.18 blk_is_allocated()

```
int blk_is_allocated (
            uint32_t block_num )
```

References blk_bam_entry_t::allocated, blk_dev_slot::bam, BLK_EINVAL, BLK_ENODEV, g, lbn_to_slot(), resolve_lbn(), and blk_dev_slot::start_lbn.

Here is the call graph for this function:



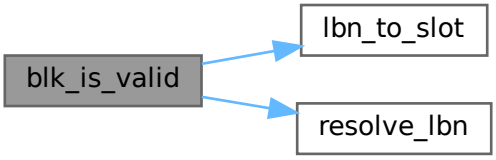
7.199.3.19 blk_is_valid()

```
int blk_is_valid (
            uint32_t block_num )
```

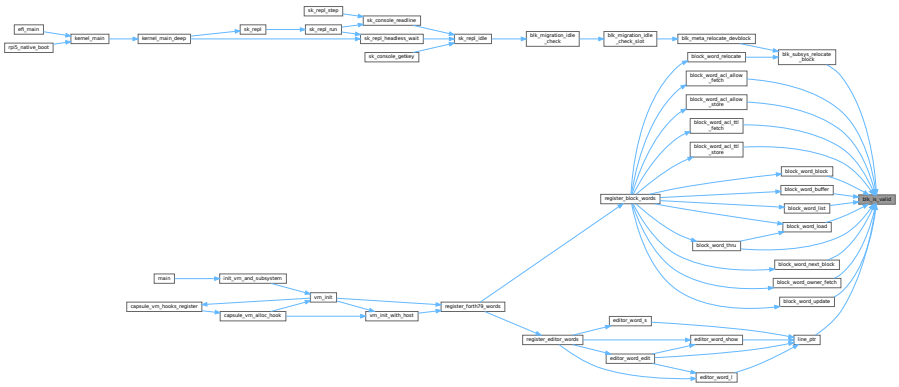
References g, lbn_to_slot(), and resolve_lbn().

Referenced by blk_subsys_relocate_block(), block_word_acl_allow_fetch(), block_word_acl_allow_store(), block_word_acl_ttl_fetch(), block_word_acl_ttl_store(), block_word_block(), block_word_buffer(), block_word_list(), block_word_load(), block_word_next_block(), block_word_owner_fetch(), block_word_thru(), block_word_update(), and line_ptr().

Here is the call graph for this function:



Here is the caller graph for this function:



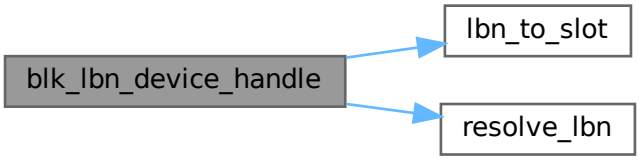
7.199.3.20 blk_lbn_device_handle()

```
const void * blk_lbn_device_handle (
            uint32_t lbn )
```

References lbn_to_slot(), and resolve_lbn().

Referenced by blk_vm_is_cross_device_copy().

Here is the call graph for this function:



Here is the caller graph for this function:

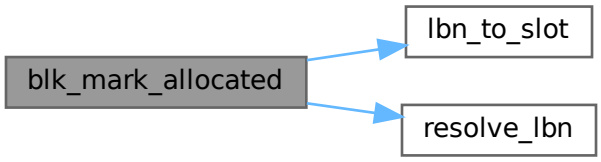
7.199.3.21 blk_mark_allocated()

```
int blk_mark_allocated (
            uint32_t block_num )
```

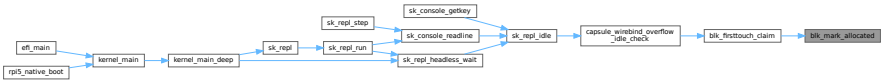
References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_dev_slot::bam_dirty, BLK_EINVAL, BLK_ENODEV, BLK_ERANGE, BLK_ERESERVED, BLK_FMT_PROVISIONAL, BLK_OK, blk_dev_slot::format_state, blk_volume_meta_t::free_blocg, lbn_to_slot(), blk_dev_slot::raw_base, resolve_lbn(), blk_dev_slot::start_lbn, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Referenced by blk_firsttouch_claim().

Here is the call graph for this function:



Here is the caller graph for this function:



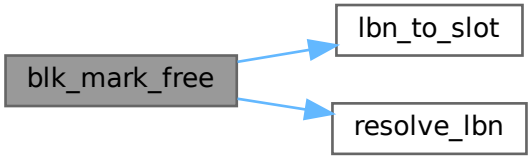
7.199.3.22 blk_mark_free()

```
int blk_mark_free (
            uint32_t block_num )
```

References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_dev_slot::bam_dirty, BLK_EINVAL, BLK_ENODEV, BLK_ERANGE, BLK_ERESERVED, BLK_FMT_PROVISIONAL, BLK_OK, blk_volume_meta_t::first_free, blk_dev_slot::format_state, blk_volume_meta_t::free_blocks, g, lbn_to_slot(), blk_dev_slot::raw_base, resolve_lbn(), blk_dev_slot::start_lbn, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Referenced by blk_subsys_relocate_block().

Here is the call graph for this function:



Here is the caller graph for this function:

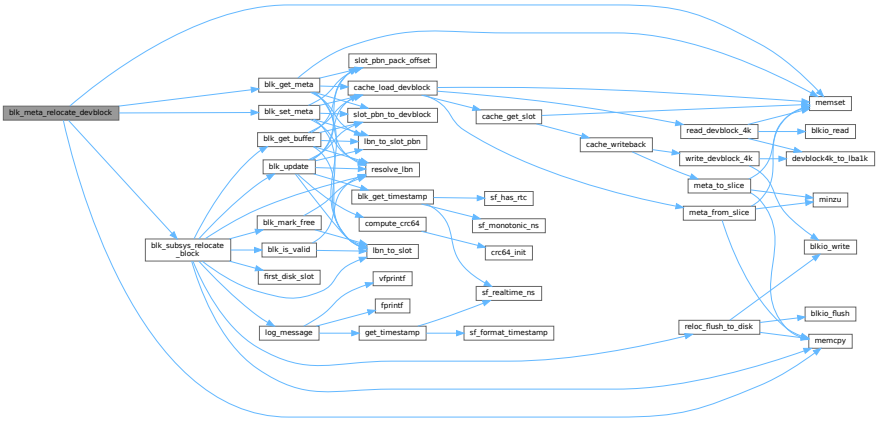
7.199.3.23 blk_meta_relocate_devblock()

```
int blk_meta_relocate_devblock (
            uint32_t home_devblock,
            uint32_t target_devblock )
```

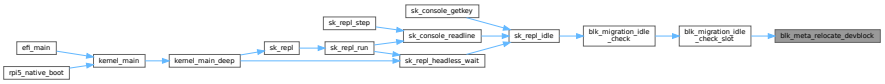
References blk_meta_t::acl_allow, blk_meta_t::acl_ttl, BLK_EINVAL, BLK_EIO, BLK_FLAG_CLAIMED, BLK_FLAG_MIGRATING, blk_get_meta(), BLK_OK, BLK_PACK_RATIO, blk_set_meta(), blk_subsys_relocate_block(), blk_meta_t::flags, memcpy(), memset(), and blk_meta_t::owner_fp.

Referenced by blk_migration_idle_check_slot().

Here is the call graph for this function:



Here is the caller graph for this function:



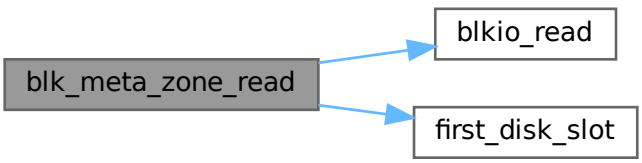
7.199.3.24 blk_meta_zone_read()

```
int blk_meta_zone_read (
            uint32_t devblock_from_top,
            uint8_t buf[4096] )
```

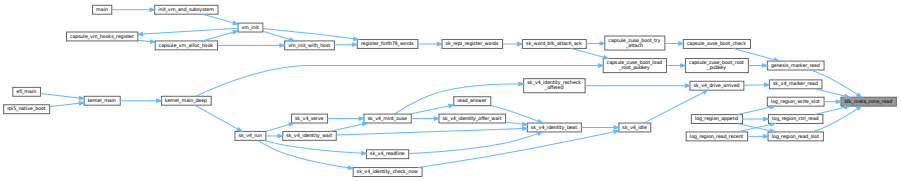
References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_OK, BLKIO_OK, blkio_read(), blk_dev_slot::dev, first_disk_slot(), blk_volume_meta_t::meta_fence_blocks, blk_volume_meta_t::total_devblocks, and blk_dev_slot::vol_meta.

Referenced by genesis_marker_read(), log_region_ctrl_read(), log_region_read_slot(), log_region_write_slot(), and sk_v4_marker_read().

Here is the call graph for this function:



Here is the caller graph for this function:



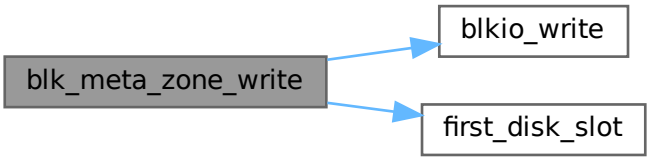
7.199.3.25 blk_meta_zone_write()

```
int blk_meta_zone_write (
            uint32_t devblock_from_top,
            const uint8_t buf[4096] )
```

References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_OK, BLKIO_OK, blkio_write(), blk_dev_slot::dev, first_disk_slot(), blk_volume_meta_t::meta_fence_blocks, blk_volume_meta_t::total_devblocks, and blk_dev_slot::vol_meta.

Referenced by capsule_zuse_boot_try_attach(), log_region_ctrl_write(), log_region_write_slot(), and sk_v4_marker_write().

Here is the call graph for this function:



Here is the caller graph for this function:

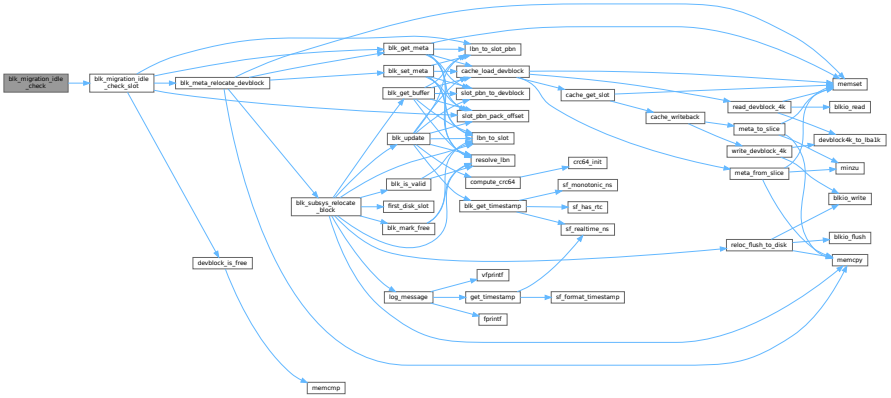
7.199.3.26 blk_migration_idle_check()

```
void blk_migration_idle_check (
            void  )
```

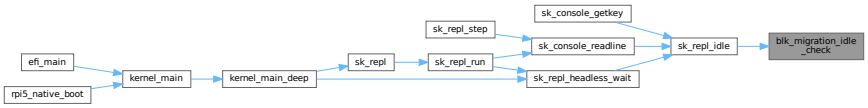
References blk_migration_idle_check_slot(), blk_dev_slot::dev, g, and blk_dev_slot::next.

Referenced by sk_repl_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



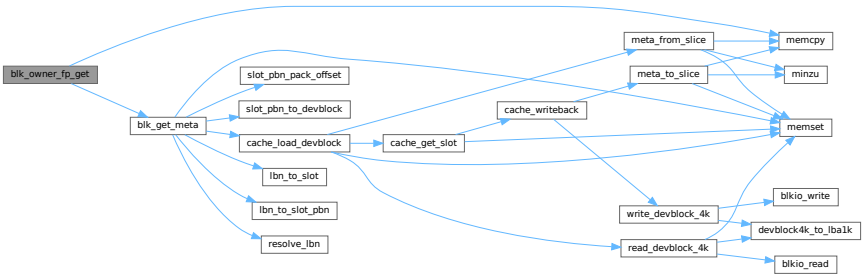
7.199.3.27 blk_owner_fp_get()

```
int blk_owner_fp_get (
            uint32_t block_num,
            uint8_t out_fp[8] )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, memcpy(), and meta.

Referenced by block_word_owner_fetch().

Here is the call graph for this function:



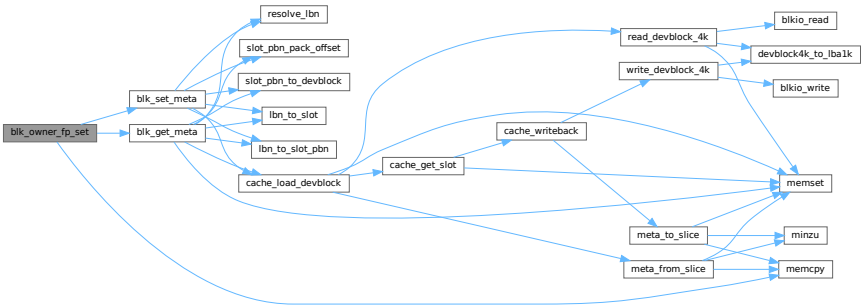
Here is the caller graph for this function:

7.199.3.28 blk_owner_fp_set()

```
int blk_owner_fp_set (
            uint32_t block_num,
            const uint8_t fp[8] )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, blk_set_meta(), memcpy(), and meta.

Here is the call graph for this function:



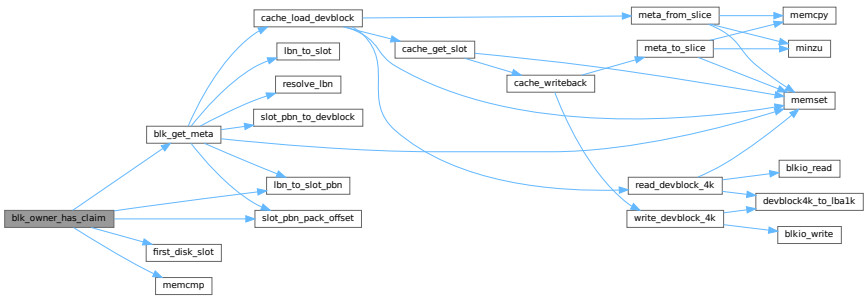
7.199.3.29 blk_owner_has_claim()

```
int blk_owner_has_claim (
            const uint8_t owner_fp[8] )
```

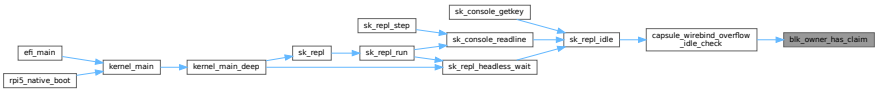
References BLK_FLAG_CLAIMED, blk_get_meta(), BLK_OK, first_disk_slot(), blk_volume_meta_t::flags, lbn_to_slot_pbn(), memcmp(), meta, slot_pbn_pack_offset(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by capsule_wirebind_overflow_idle_check().

Here is the call graph for this function:



Here is the caller graph for this function:



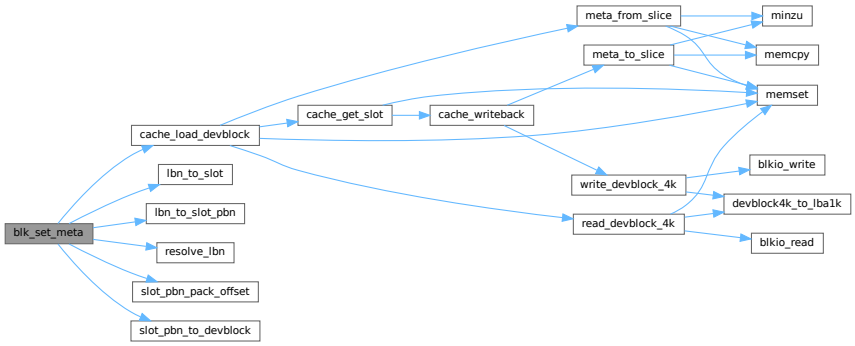
7.199.3.30 blk_set_meta()

```
int blk_set_meta (
            uint32_t block_num,
            const blk_meta_t * meta )
```

References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ERANGE, BLK_FLAG_CLAIMED, BLK_OK, cache_load_devblock(), blk_volume_meta_t::flags, g, blk_dev_slot::has_ever_claimed, lbn_to_slot(), lbn_to_slot_pbn(), meta, cache_slot_t::meta, cache_slot_t::meta_dirty, blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_pack_offset(), and slot_pbn_to_devblock().

Referenced by blk_acl_allow_set(), blk_acl_ttl_set(), blk_firsttouch_claim(), blk_flags_set(), blk_meta_relocate_devblock(), and blk_owner_fp_set().

Here is the call graph for this function:



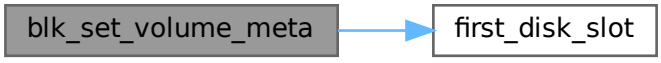
Here is the caller graph for this function:

7.199.3.31 blk_set_volume_meta()

```
int blk_set_volume_meta (
            const blk_volume_meta_t * meta )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, first_disk_slot(), meta, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Here is the call graph for this function:

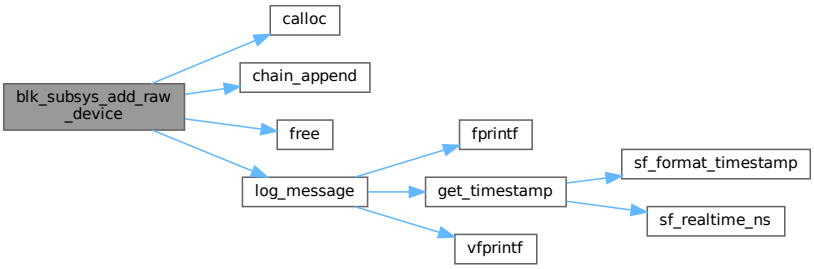


7.199.3.32 blk_subsys_add_raw_device()

```
int blk_subsys_add_raw_device (
            uint8_t * buf,
            uint32_t nblocks )
```

References blk_dev_slot::bam, BLK_EINVAL, BLK_ENODEV, BLK_ENOMEM, BLK_OK, calloc(), chain_append(), free(), g, LOG_INFO, log_message(), blk_dev_slot::raw_base, blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Here is the call graph for this function:



7.199.3.33 blk_subsys_attach_device()

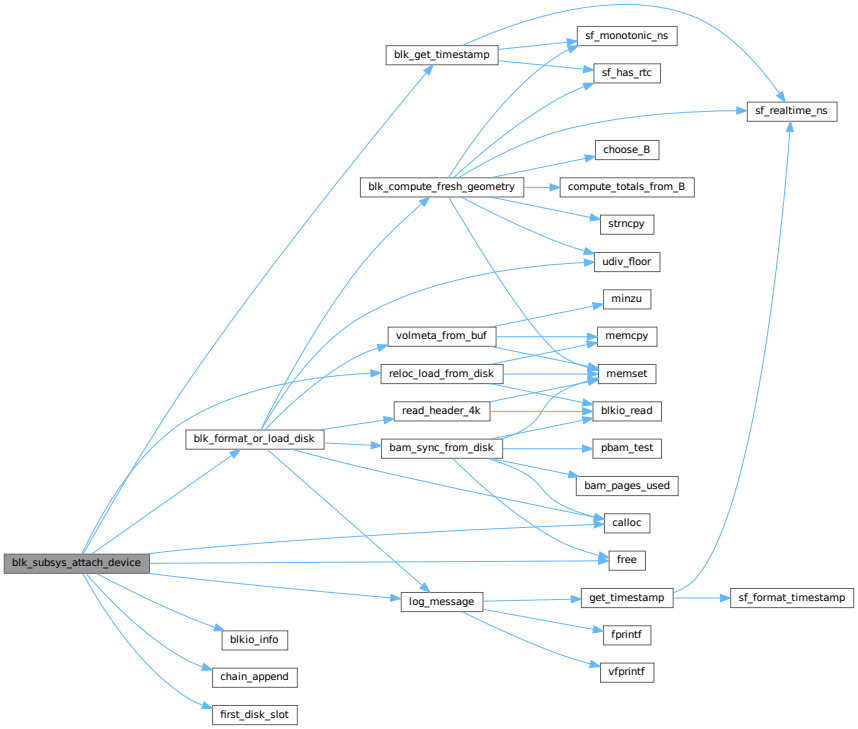
```
int blk_subsys_attach_device (
            struct blkio_dev * dev )
```

References blk_dev_slot::bam, blk_volume_meta_t::bam_devblocks, BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ENOMEM, BLK_FMT_FORMATTED, blk_format_or_load_disk(), blk_get_timestamp(), BLK_OK, blkio_info(),

BLKIO_OK, calloc(), chain_append(), blk_dev_slot::dev, blk_volume_meta_t::devblock_base, first_disk_slot(), blk_dev_slot::format_state, free(), blk_volume_meta_t::free_blocks, g, blk_volume_meta_t::label, LOG_INFO, log_message(), blk_volume_meta_t::meta_fence_blocks, blk_volume_meta_t::mounted_time, reloc_load_from_disk(), blk_dev_slot::start_lbn, blk_dev_slot::total_blkio_blocks_1k, blkio_info_t::total_blocks, blk_volume_meta_t::total_blocks, blk_volume_meta_t::total_devblocks, blk_dev_slot::user_blocks, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Referenced by __attribute__(), block_word_blk_attach(), kernel_main_deep(), sk_v4_drive_arrived(), and sk_v4_real_zuse_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

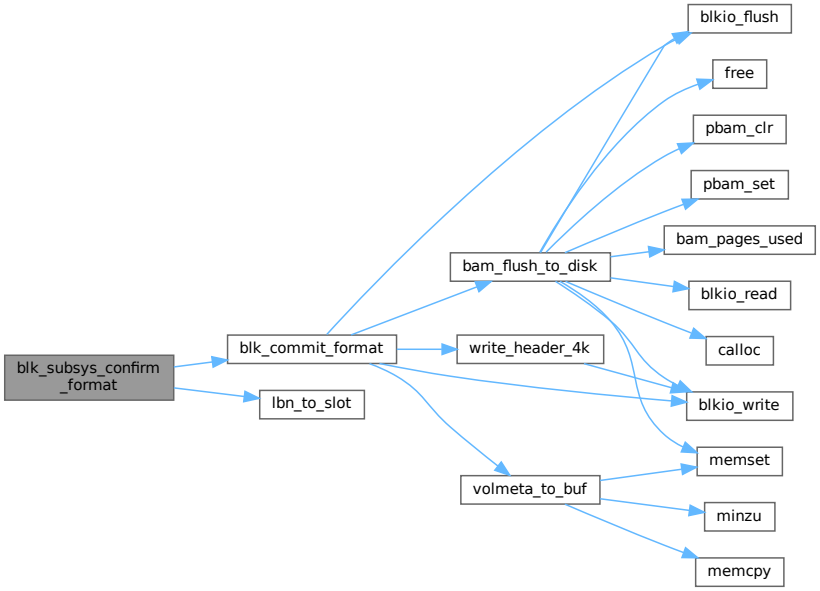
7.199.3.34 blk_subsys_confirm_format()

```
int blk_subsys_confirm_format (
            uint32_t lbn )
```

References blk_commit_format(), BLK_ENODEV, BLK_ERANGE, blk_dev_slot::dev, g, and lbn_to_slot().

Referenced by block_word_confirm_format(), and sk_v4_genesis().

Here is the call graph for this function:



Here is the caller graph for this function:

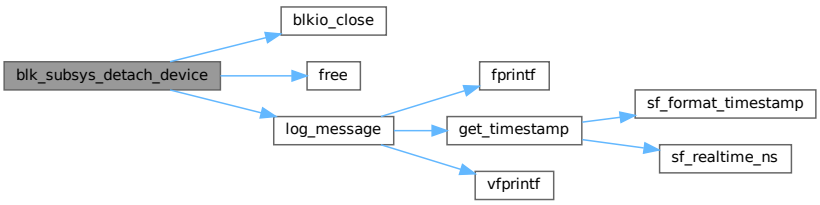
7.199.3.35 blk_subsys_detach_device()

```
int blk_subsys_detach_device (
            struct blkio_dev * dev )
```

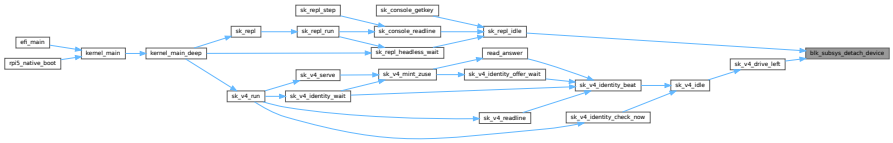
References blk_dev_slot::bam, BLK_EINVAL, BLK_ENODEV, BLK_OK, blkio_close(), blk_dev_slot::dev, free(), g, blk_volume_meta_t::label, LOG_INFO, log_message(), LOG_WARN, blk_dev_slot::next, blk_dev_slot::start_lbn, blk_dev_slot::user_blocks, and blk_dev_slot::vol_meta.

Referenced by sk_repl_idle(), and sk_v4_drive_left().

Here is the call graph for this function:



Here is the caller graph for this function:



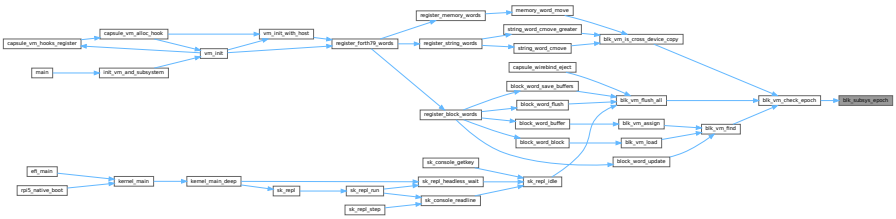
7.199.3.36 blk_subsys_epoch()

```
uint64_t blk_subsys_epoch (
            void  )
```

References g.

Referenced by blk_vm_check_epoch().

Here is the caller graph for this function:



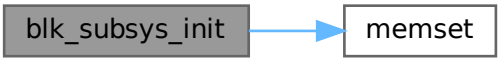
7.199.3.37 blk_subsys_init()

```
int blk_subsys_init (
            uint8_t * ram_base,
            size_t ram_size )
```

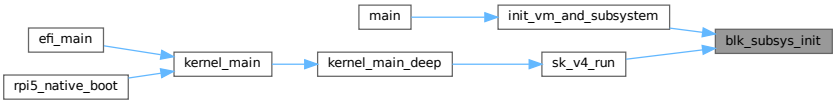
References BLK_EINVAL, BLK_FORTH_SIZE, BLK_FORTH_SYS_RESERVED, BLK_OK, BLK_RAM_BLOCKS, g, memset(), ram_base, and ram_size.

Referenced by init_vm_and_subsystem(), and sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



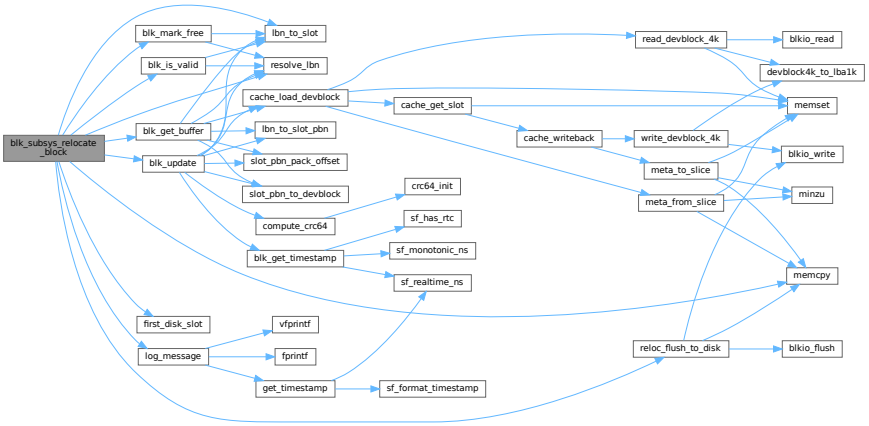
7.199.3.38 blk_subsys_relocate_block()

```
int blk_subsys_relocate_block (
            uint32_t home_lbn,
            uint32_t target_lbn )
```

References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ENOMEM, BLK_ERANGE, BLK_FORTH_SIZE, blk_get_buffer(), blk_is_valid(), blk_mark_free(), BLK_OK, BLK_RELOC_MAX_ENTRIES, blk_update(), first_disk_slot(), g, lbn_to_slot(), LOG_INFO, log_message(), LOG_WARN, memcpy(), reloc_flush_to_disk(), and resolve_lbn().

Referenced by blk_meta_relocate_devblock(), and block_word_relocate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.199.3.39 blk_subsys_set_format_bam_for()

```
void blk_subsys_set_format_bam_for (
            uint64_t total_devblocks )
```

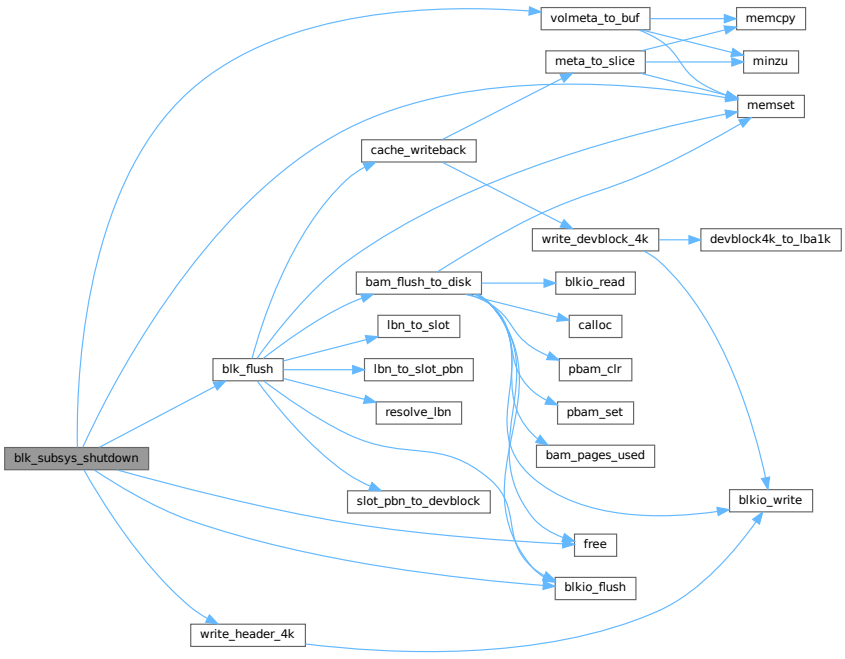
References g_format_bam_for.

7.199.3.40 blk_subsys_shutdown()

```
int blk_subsys_shutdown (
            void  )
```

References blk_dev_slot::bam, BLK_DEVICE_SECTOR, blk_flush(), BLK_FMT_FORMATTED, BLK_OK, blkio_flush(), blk_dev_slot::dev, blk_dev_slot::format_state, free(), g, memset(), blk_dev_slot::next, blk_dev_slot::vol_meta, blk_dev_slot::vol_meta_dirty, volmeta_to_buf(), and write_header_4k().

Here is the call graph for this function:



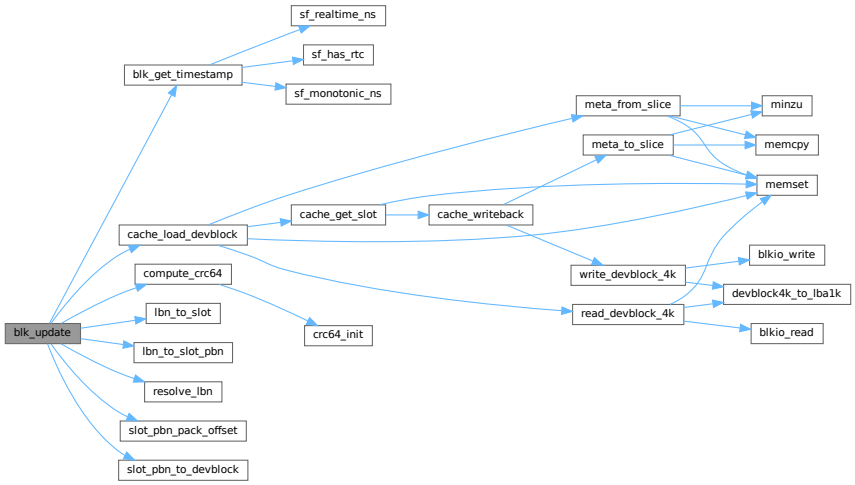
7.199.3.41 blk_update()

```
int blk_update (
            uint32_t block_num )
```

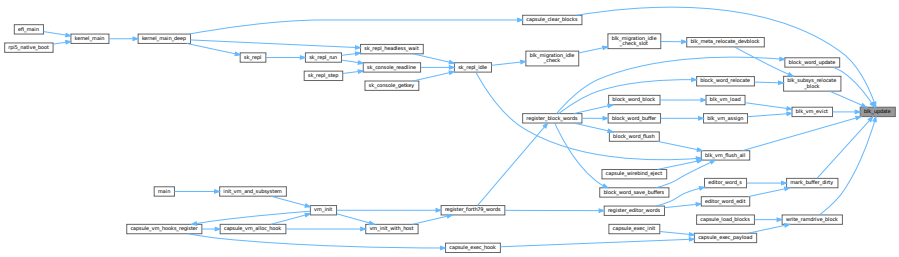
References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_dev_slot::bam_dirty, BLK_EIO, BLK_ENODEV, BLK_ERANGE, BLK_ERESERVED, BLK_FMT_PROVISIONAL, BLK_FORTH_SIZE, BLK_FORTH_SYS_RESERVED, blk_get_timestamp(), BLK_OK, BLK_RAM_BLOCKS, cache_load_devblock(), blk_meta_t::checksum, compute_crc64(), blk_meta_t::created_time, cache_slot_t::data, blk_bam_entry_t::dirty, cache_slot_t::dirty, blk_dev_slot::format_state, blk_volume_meta_t::free_blocks, g, lbn_to_slot(), lbn_to_slot_pbn(), blk_meta_t::magic, cache_slot_t::meta, cache_slot_t::meta_dirty, blk_meta_t::modified_time, blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_pack_offset(), slot_pbn_to_devblock(), blk_dev_slot::start_lbn, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Referenced by blk_subsys_relocate_block(), blk_vm_evict(), blk_vm_flush_all(), block_word_update(), capsule_clear_blocks(), mark_buffer_dirty(), and write_ramdrive_block().

Here is the call graph for this function:



Here is the caller graph for this function:



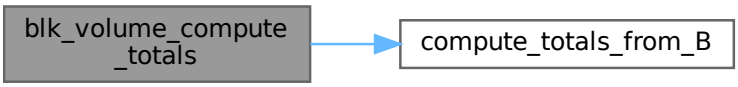
7.199.3.42 blk_volume_compute_totals()

```
void blk_volume_compute_totals (
            blk_volume_meta_t * m )
```

References compute_totals_from_B().

Referenced by sk_v4_grow_artemis().

Here is the call graph for this function:



Here is the caller graph for this function:

7.199.3.43 compute_crc64()

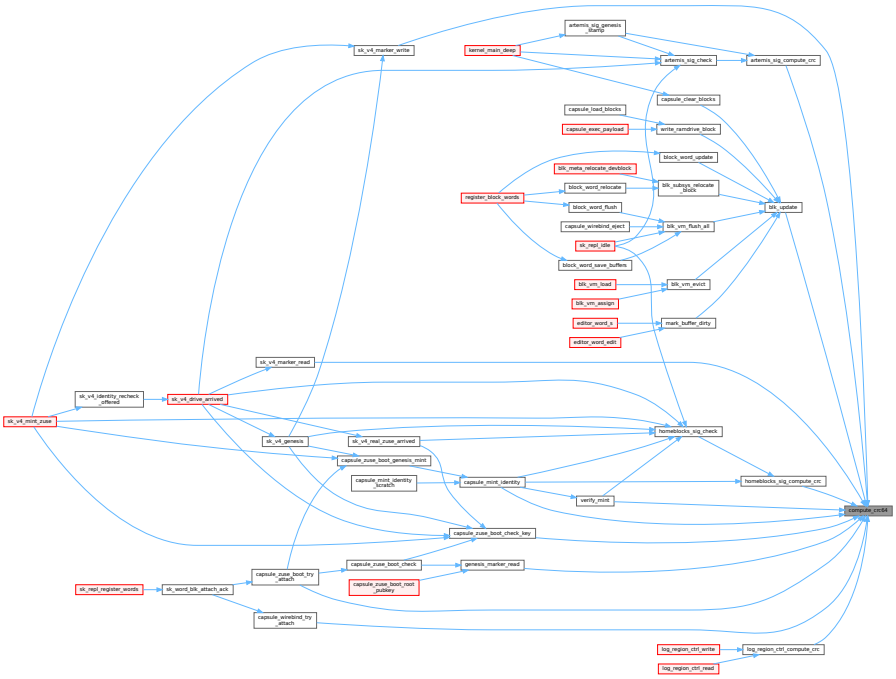
```
uint64_t compute_crc64 (
            const uint8_t * data,
            size_t len )
```

References crc64_init(), crc64_inited, and crc64_table.

Referenced by artemis_sig_compute_crc(), blk_update(), capsule_mint_identity(), capsule_wirebind_try_attach(), capsule_zuse_boot_check_key(), capsule_zuse_boot_try_attach(), genesis_marker_read(), homeblocks_sig_compute_crc(), log_region_ctrl_compute_crc(), sk_v4_marker_read(), sk_v4_marker_write(), and verify_mint().

Here is the call graph for this function:

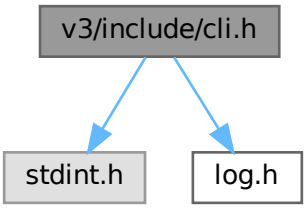
Here is the caller graph for this function:



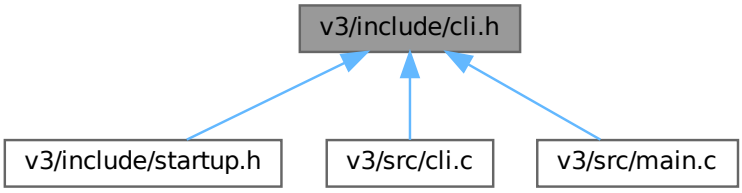
7.200 v3/include/cli.h File Reference

```
#include <stdint.h>
#include "log.h"
```

Include dependency graph for cli.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct CLIConfig

  Configuration structure from command-line arguments Consolidates all parsed CLI options into a single, cleanly organized struct.

Enumerations

- enum HeartbeatLogMode { HEARTBEAT_LOG_OFF = 0 , HEARTBEAT_LOG_SUMMARY = 1 , HEARTBEAT_LOG_FULL = 2 }

  Heartbeat logging modes for DoE experiments.

Functions

- CLIConfig cli_parse (int argc, char ∗argv[ ])

  Parse command-line arguments into CLIConfig.
- void cli_print_usage (const char ∗program_name)

  Print usage/help information.

7.200.1 Enumeration Type Documentation

7.200.1.1 HeartbeatLogMode

```
enum HeartbeatLogMode
```

Heartbeat logging modes for DoE experiments.

Enumerator

HEARTBEAT_LOG_OFF	
HEARTBEAT_LOG_SUMMARY	
HEARTBEAT_LOG_FULL	

7.200.2 Function Documentation

7.200.2.1 cli_parse()

```
CLIConfig cli_parse (
            int argc,
            char * argv[] )
```

Parse command-line arguments into CLIConfig.

Parameters

argc	Argument count
argv	Argument vector

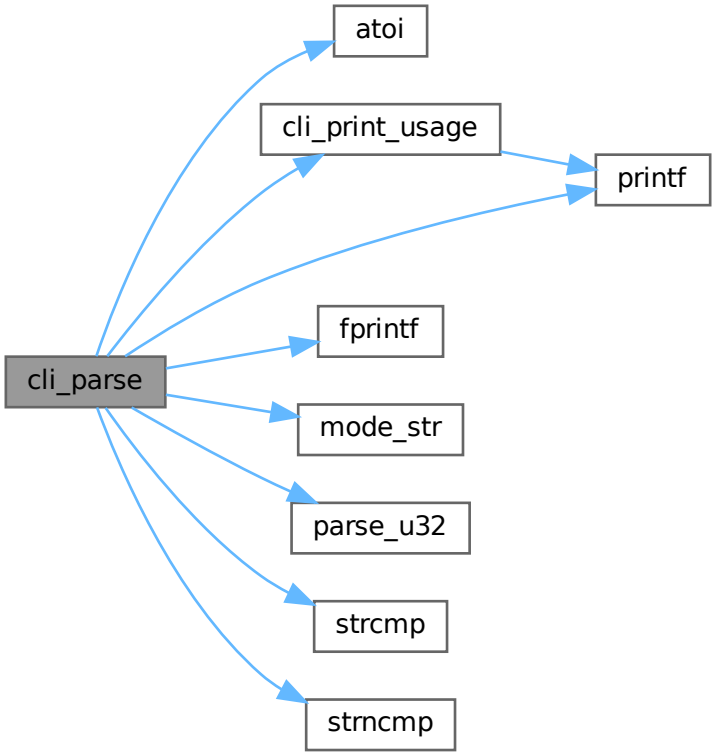
Returns

CLIConfig struct with parsed options, or exits on –version/–help

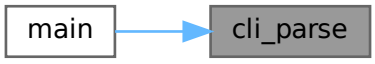
References atoi(), CLIConfig::benchmark, CLIConfig::benchmark_iterations, CLIConfig::break_me, cli_print_usage(), CLIConfig::disk_img_path, CLIConfig::doe_experiment, CLIConfig::fail_fast, fprintf(), HEARTBEAT_LOG_FULL, CLIConfig::heartbeat_log_mode, HEARTBEAT_LOG_OFF, HEARTBEAT_LOG_SUMMARY, LOG_DEBUG, LOG_ERROR, LOG_INFO, CLIConfig::log_level, CLIConfig::log_level_explicitly_set, LOG_NONE, LOG_TEST, LOG_WARN, mode_str(), parse_u32(), printf(), CLIConfig::ram_disk_mb, CLIConfig::run_tests, CLIConfig::run_tests_flag_observed, CLIConfig::script_mode, STARFORTH_VERSION_FULL, stderr, strcmp(), and strncmp().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.200.2.2 cli_print_usage()

```
void cli_print_usage (
            const char * program_name )
```

Print usage/help information.

Parameters

program_name	Name of the executable

References printf().

Referenced by cli_parse().

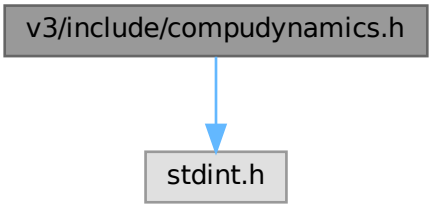
Here is the call graph for this function:

Here is the caller graph for this function:

7.201 v3/include/compudynamics.h File Reference

```
#include <stdint.h>
```

Include dependency graph for compudynamics.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct CDRegimeMetrics
- struct CDTuning
- struct CDConfigEntry
- struct CDConfigTable

Macros

- #define CD_MAX_CLASSIFY_DEPTH 64u

Functions

- CDRegimeMetrics cd_classify_ids (const uint32_t *ids, uint32_t count, uint32_t locality_lookback)

  Classify a recent entity-ID touch sequence into diversity/volatility/ locality, purely from execution counts – no wall-clock input anywhere.
- CDTuning cd_tuning_word (void)
- CDTuning cd_tuning_vm (void)
- int cd_config_table_init (CDConfigTable *table, const CDTuning *tuning)

  Allocate a config table sized for tuning->num_configs / num_regimes.
- void cd_config_table_free (CDConfigTable *table)

  Free all memory owned by a config table.
- void cd_config_table_seed (CDConfigTable *table, const CDTuning *tuning, const uint32_t *initial_scores, uint8_t initial_config)

  Seed every (regime, config) score with initial_scores[config], and set the starting active config to initial_config.
- uint8_t cd_config_table_tick (CDConfigTable *table, const CDTuning *tuning, uint8_t regime, uint32_t anova_stability_q16, int use_joint_convergence, uint32_t current_window, uint32_t current_locality_q16)

  Score the currently-active config from this tick's outcome and UCB1-reselect this tick's config, for the given regime.
- void cd_config_table_force (CDConfigTable *table, uint8_t config_idx)

  Force the table onto an externally-chosen config (as if UCB1 had picked it).

7.201.1 Macro Definition Documentation

7.201.1.1 CD_MAX_CLASSIFY_DEPTH

```
#define CD_MAX_CLASSIFY_DEPTH 64u
```

7.201.2 Function Documentation

7.201.2.1 cd_classify_ids()

```
CDRegimeMetrics cd_classify_ids (
            const uint32_t * ids,
            uint32_t count,
            uint32_t locality_lookback )
```

Classify a recent entity-ID touch sequence into diversity/volatility/ locality, purely from execution counts – no wall-clock input anywhere.

Parameters

ids	Recent entity IDs, oldest first, chronological order
count	Number of valid entries in ids (clamped internally to CD_MAX_CLASSIFY_DEPTH)
locality_lookback	How many preceding touches count as "recent" for the locality measure (clamped to count)

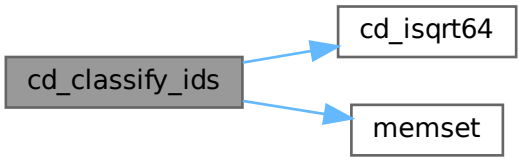
Returns

Classified regime metrics; all-zero if count == 0

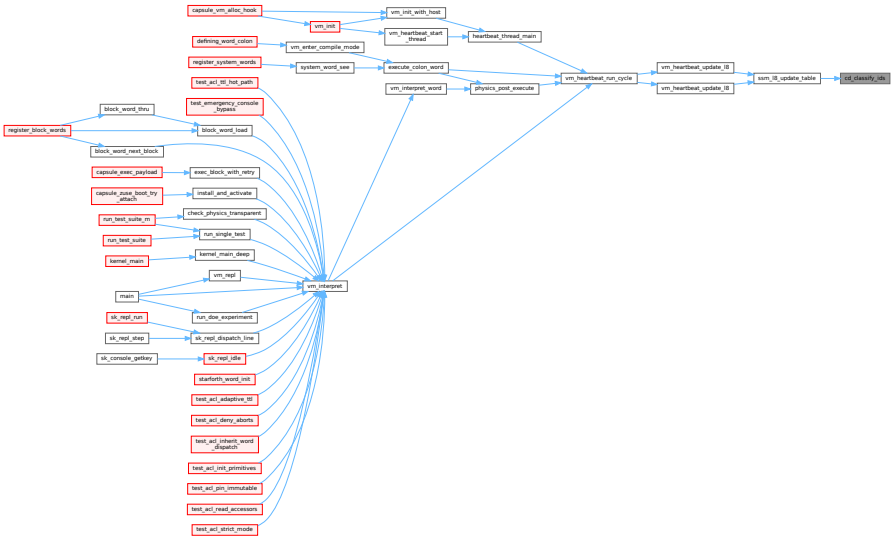
References cd_isqrt64(), CD_MAX_CLASSIFY_DEPTH, count, CDRegimeMetrics::diversity_q16, CDRegimeMetrics::locality_q16, memset(), and CDRegimeMetrics::volatility_q16.

Referenced by ssm_l8_update_table().

Here is the call graph for this function:



Here is the caller graph for this function:



7.201.2.2 cd_config_table_force()

```
void cd_config_table_force (
            CDConfigTable * table,
            uint8_t config_idx )
```

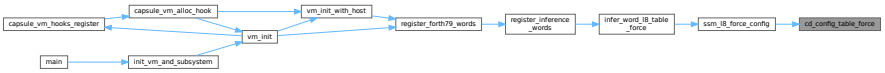
Force the table onto an externally-chosen config (as if UCB1 had picked it).

One-tick nudge, not a hold – see ssm_l8_force_config().

References CDConfigTable::current_config, CDConfigTable::entries, and CDConfigTable::num_configs.

Referenced by ssm_l8_force_config().

Here is the caller graph for this function:



7.201.2.3 cd_config_table_free()

```
void cd_config_table_free (
            CDConfigTable * table )
```

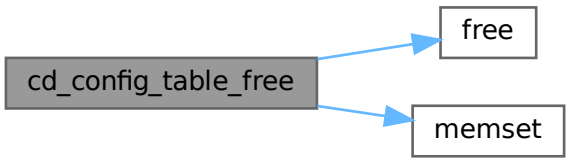
Free all memory owned by a config table.

Safe on a zeroed table.

References CDConfigTable::entries, free(), memset(), CDConfigTable::regime_scores, CDConfigTable::regime_ticks, and CDConfigTable::total_regime_ticks.

Referenced by cd_config_table_init(), and ssm_l8_free_table().

Here is the call graph for this function:



Here is the caller graph for this function:

7.201.2.4 cd_config_table_init()

```
int cd_config_table_init (
            CDConfigTable * table,
            const CDTuning * tuning )
```

Allocate a config table sized for tuning->num_configs / num_regimes.

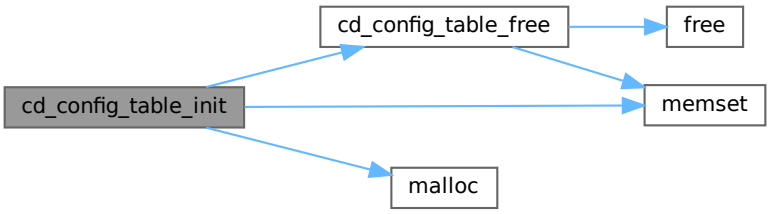
Returns

0 on success, -1 on malloc failure (table left zeroed, unusable)

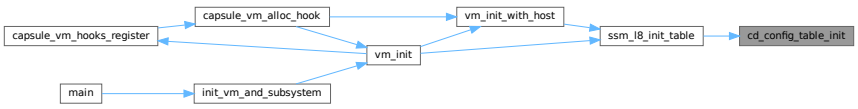
References cd_config_table_free(), CDConfigTable::entries, malloc(), memset(), CDTuning::num_configs, CDConfigTable::num_configs, CDTuning::num_regimes, CDConfigTable::num_regimes, CDConfigTable::regime_scores, CDConfigTable::regime_ticks, and CDConfigTable::total_regime_ticks.

Referenced by ssm_l8_init_table().

Here is the call graph for this function:



Here is the caller graph for this function:



7.201.2.5 cd_config_table_seed()

```
void cd_config_table_seed (
            CDConfigTable * table,
            const CDTuning * tuning,
            const uint32_t * initial_scores,
            uint8_t initial_config )
```

Seed every (regime, config) score with initial_scores[config], and set the starting active config to initial_config.

Parameters

initial_scores	Array of tuning->num_configs initial benefit scores

References CDConfigEntry::_pad, CDConfigEntry::benefit_score, CDConfigEntry::config_bits, CDConfigTable::current_config, CDConfigTable::current_regime, CDConfigTable::entries, CDConfigEntry::has_prev, CDConfigTable::num_configs, CDConfigTable::num_regimes, CDConfigEntry::prev_locality_q16, CDConfigEntry::prev_window, CDConfigTable::regime_scores, CDConfigTable::regime_ticks, CDConfigEntry::tick_count, and CDConfigTable::total_regime_ticks.

Referenced by ssm_l8_seed_from_doe().

Here is the caller graph for this function:

7.201.2.6 cd_config_table_tick()

```
uint8_t cd_config_table_tick (
            CDConfigTable * table,
            const CDTuning * tuning,
            uint8_t regime,
            uint32_t anova_stability_q16,
            int use_joint_convergence,
            uint32_t current_window,
            uint32_t current_locality_q16 )
```

Score the currently-active config from this tick's outcome and UCB1-reselect this tick's config, for the given regime.

Combined reward = joint_weight ∗ window/locality-convergence-stability

- anova_weight ∗ anova_stability_q16 Joint-convergence is skipped (falls back to ANOVA-only) on a config's first tick, or when use_joint_convergence is false (the caller's config bits don't include loops whose convergence this signal is meant to track – word level's L5/L6, see ssm_jacquard.c).

Parameters

table	Config table (must be initialised)
tuning	Tuning knobs matching this table's sizing
regime	Current regime index, < tuning->num_regimes
anova_stability_q16	Caller-supplied stability signal, Q16 [0,65536]
use_joint_convergence	Whether to include the window/locality signal
current_window	Caller's current window-width metric
current_locality_q16	Caller's current locality_q16 from cd_classify_ids

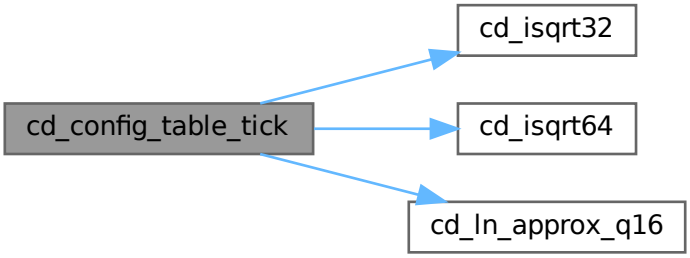
Returns

The newly-selected config's config_bits

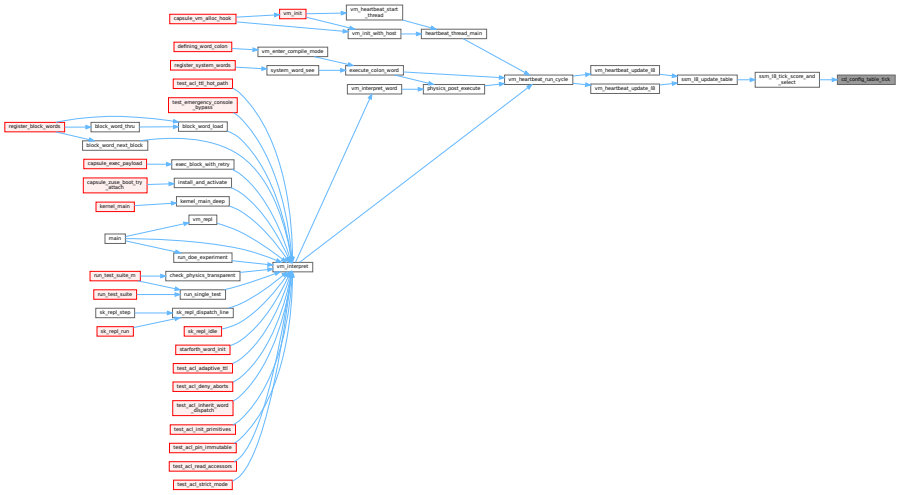
References CDTuning::anova_weight_q16, cd_isqrt32(), cd_isqrt64(), cd_ln_approx_q16(), CDConfigEntry::config_bits, CDConfigTable::current_config, CDConfigTable::current_regime, CDConfigTable::entries, entry, CDTuning::joint_weight_q16, CDConfigTable::num_configs, CDConfigTable::regime_scores, CDConfigTable::regime_ticks, CDTuning::reward_gain, CDTuning::score_decay_factor_q16, CDTuning::score_max, CDTuning::score_min, CDConfigTable::total_regime_ticks, and CDTuning::ucb_k.

Referenced by ssm_l8_tick_score_and_select().

Here is the call graph for this function:



Here is the caller graph for this function:



7.201.2.7 cd_tuning_vm()

```
CDTuning cd_tuning_vm (
            void  )
```

References CDTuning::anova_weight_q16, CDTuning::diversity_high_threshold_q16, CDTuning::id_window_depth, CDTuning::joint_weight_q16, CDTuning::locality_high_threshold_q16, CDTuning::locality_lookback, CDTuning::num_configs, CDTuning::num_regimes, CDTuning::reward_gain, CDTuning::score_decay_factor_q16, CDTuning::score_max, CDTuning::score_min, CDTuning::ucb_k, and CDTuning::volatility_high_threshold_q16.

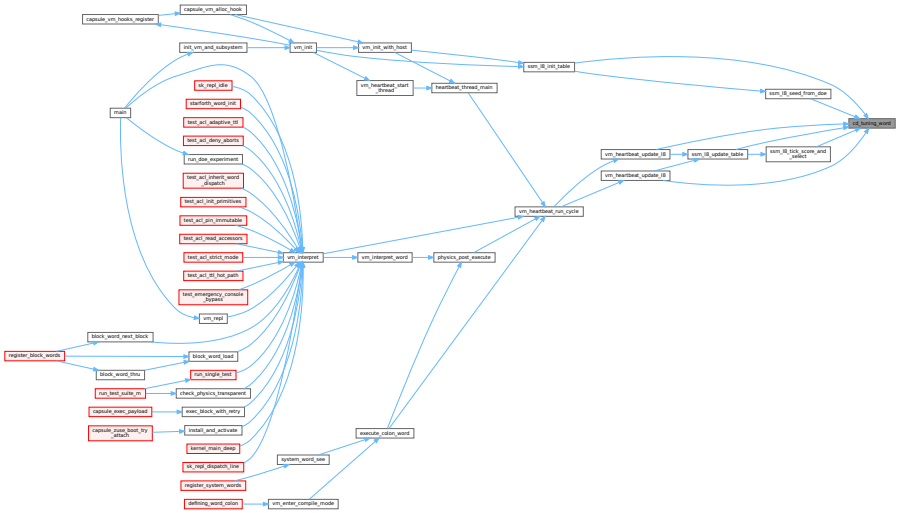
7.201.2.8 cd_tuning_word()

```
CDTuning cd_tuning_word (
            void  )
```

References CDTuning::anova_weight_q16, CDTuning::diversity_high_threshold_q16, CDTuning::id_window_depth, CDTuning::joint_weight_q16, CDTuning::locality_high_threshold_q16, CDTuning::locality_lookback, CDTuning::num_configs, CDTuning::num_regimes, CDTuning::reward_gain, CDTuning::score_decay_factor_q16, CDTuning::score_max, CDTuning::score_min, CDTuning::ucb_k, and CDTuning::volatility_high_threshold_q16.

Referenced by ssm_l8_init_table(), ssm_l8_seed_from_doe(), ssm_l8_tick_score_and_select(), ssm_l8_update_table(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

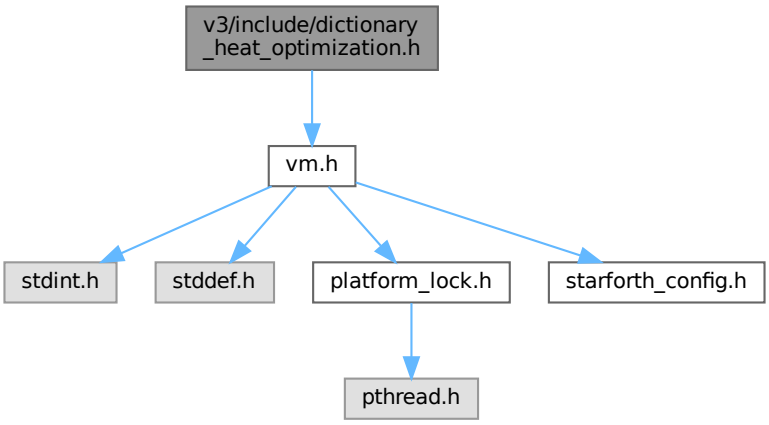
Here is the caller graph for this function:



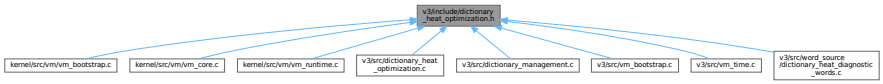
7.202 v3/include/dictionary_heat_optimization.h File Reference

```
#include "vm.h"
```

Include dependency graph for dictionary_heat_optimization.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define SF_FC_BUCKETS 256

Functions

- void dict_update_heat_percentiles (VM *vm)

  Calculate heat percentile thresholds for all words in dictionary.
- void dict_reorganize_buckets_by_heat (VM *vm)

  Reorganize dictionary buckets by inference-driven priorities.
- DictEntry * dict_find_word_heat_aware (VM *vm, const char *name, size_t len)

  Find word with heat-aware search strategy.
- void dict_adaptive_optimization_pass (VM *vm)

  Periodically reoptimize dictionary based on latest heat patterns.

Variables

- DictEntry ** sf_fc_list [256]
- size_t sf_fc_count [256]
- size_t sf_fc_cap [256]

7.202.1 Macro Definition Documentation

7.202.1.1 SF_FC_BUCKETS

```
#define SF_FC_BUCKETS 256
```

7.202.2 Function Documentation

7.202.2.1 dict_adaptive_optimization_pass()

```
void dict_adaptive_optimization_pass (
            VM * vm )
```

Periodically reoptimize dictionary based on latest heat patterns.

Called every ∼1 second or every N executions to adapt lookup strategy to current workload patterns from rolling window.

Periodically reoptimize dictionary based on latest heat patterns.

Called from the interpreter loop (or heartbeat) on every word dispatch; the 1 Hz gate (`now_ns - vm->last_bucket_reorg_ns` < 1,000,000,000) ensures the three expensive sub-steps run at a rate that is imperceptible to FORTH execution while still keeping the dictionary layout fresh:

1. **Update heat percentiles** — calls `dict_update_heat_percentiles()` to re-sample `execution_heat` across all words and compute the 25th / 50th / 75th thresholds used by the stratified lookup.
2. **Reorganize buckets** — calls `dict_reorganize_buckets_by_heat()` to sort each of the 256 first-character hash buckets so that the hottest words sit at the lowest array indices, shortening the average scan depth during the hot-tier pass.
3. **Select lookup strategy** — queries the rolling window for the current pattern-diversity score via `rolling_window_measure_diversity()`:
   - Diversity $>$ 70 (high churn in which words are called) $\rightarrow$ strategy 1 (`heat-aware` lookup, `dict_find_word_heat_aware()`), where the heat stratification most pays off.
   - Diversity $\leq$ 70 (stable repetitive pattern) $\rightarrow$ strategy 0 (naive newest-first bucket scan), which is already nearly O(1) for programs that call the same small word set repeatedly. The chosen strategy is stored in `vm->lookup_strategy` and read by the interpreter's dictionary lookup dispatch.

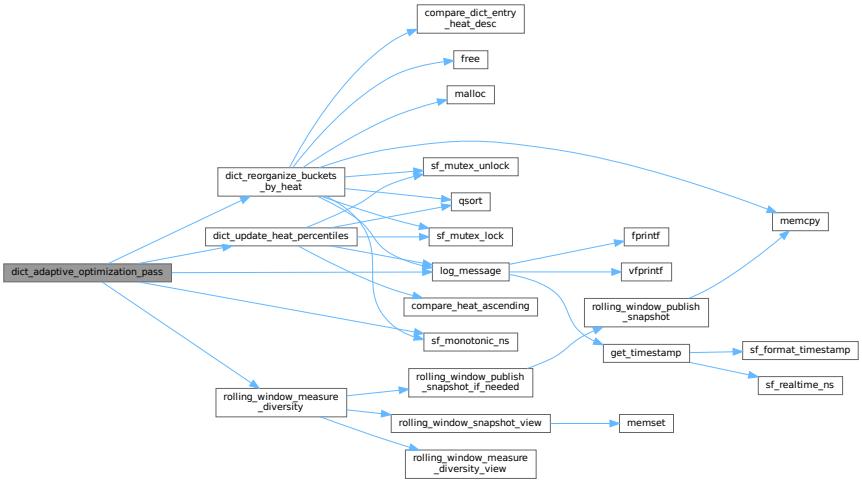
Parameters

vm	The VM to optimise. No-op if NULL or if less than 1 second has elapsed since the last pass (`vm->last_bucket_reorg_ns`).

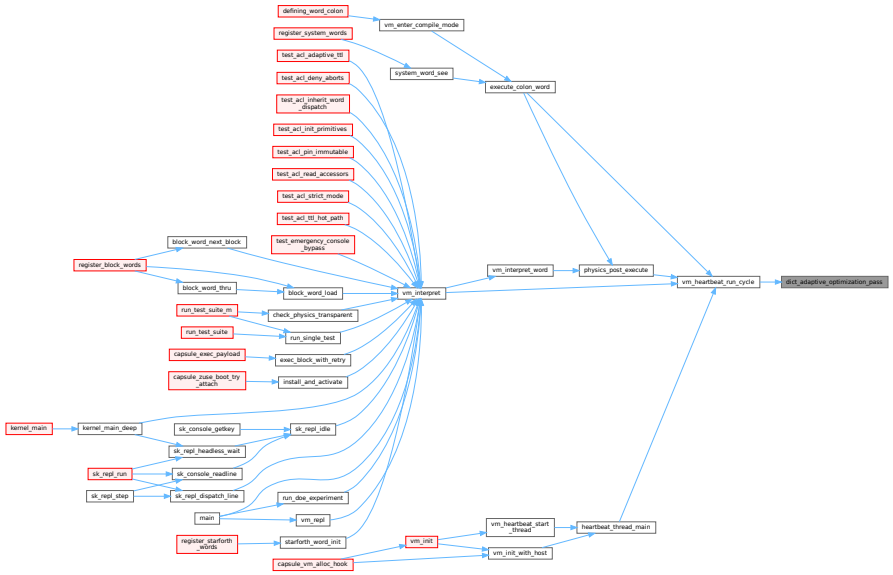
References dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), VM::last_bucket_reorg_ns, LOG_DEBUG, log_message(), VM::lookup_strategy, VM::rolling_window, rolling_window_measure_diversity(), and sf_monotonic_ns().

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.202.2.2 dict_find_word_heat_aware()

```
DictEntry * dict_find_word_heat_aware (
            VM * vm,
            const char * name,
            size_t len )
```

Find word with heat-aware search strategy.

Instead of newest-first, searches buckets in order:

1. Top 25% by heat (likely hits)
2. Middle 50% by heat
3. Bottom 25% (rarely executed)

Find word with heat-aware search strategy.

Searches the first-character bucket `sf_fc_list`[(unsigned char)name[0]] for a word whose `name` and `name_len` match `name` / `len`, scanning in three priority tiers ordered by execution heat:

1. **Hot pass** (heat $\geq$ `heat_threshold_75th`): words in the top 25% by execution frequency. Most FORTH programs spend the vast majority of ticks in a handful of hot primitives; checking this tier first yields a near-constant-time hit on steady-state workloads.
2. **Medium pass** (heat in [threshold_25th, threshold_75th)): the middle 50% of words. These are occasionally-called words that still appear often enough to warrant a dedicated pass before touching the long tail.
3. **Cold pass** (heat $<$ `heat_threshold_25th`): the rarely-executed bottom quartile. Words that have never been called also live here.

Within each pass, matching uses the same cheap-first filter chain as the standard lookup: `name_len` comparison → last-character byte check → full `memcmp`. The last-character pre-filter is skipped for single-character names.

Returns `NULL` only if the word is absent from the bucket; all three passes exhausted without a hit. Also returns `NULL` immediately if `vm`, `name`, or `len` are invalid.

Heat thresholds are updated lazily by `dict_adaptive_optimization_pass()` (≈1 Hz); stale thresholds degrade to a full three-pass scan without affecting correctness.

Parameters

vm	The VM; provides `heat_threshold_*` fields and the bucket table.
name	Word name to search for (not NUL-terminated; length in `len`).
len	Length of `name` in bytes; must be ≥ 1.

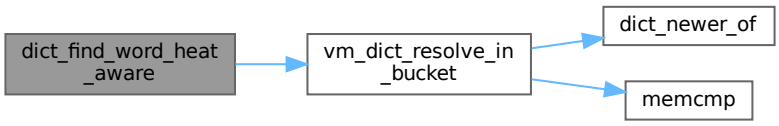
Returns

Pointer to the matching `DictEntry`, or `NULL` if not found.

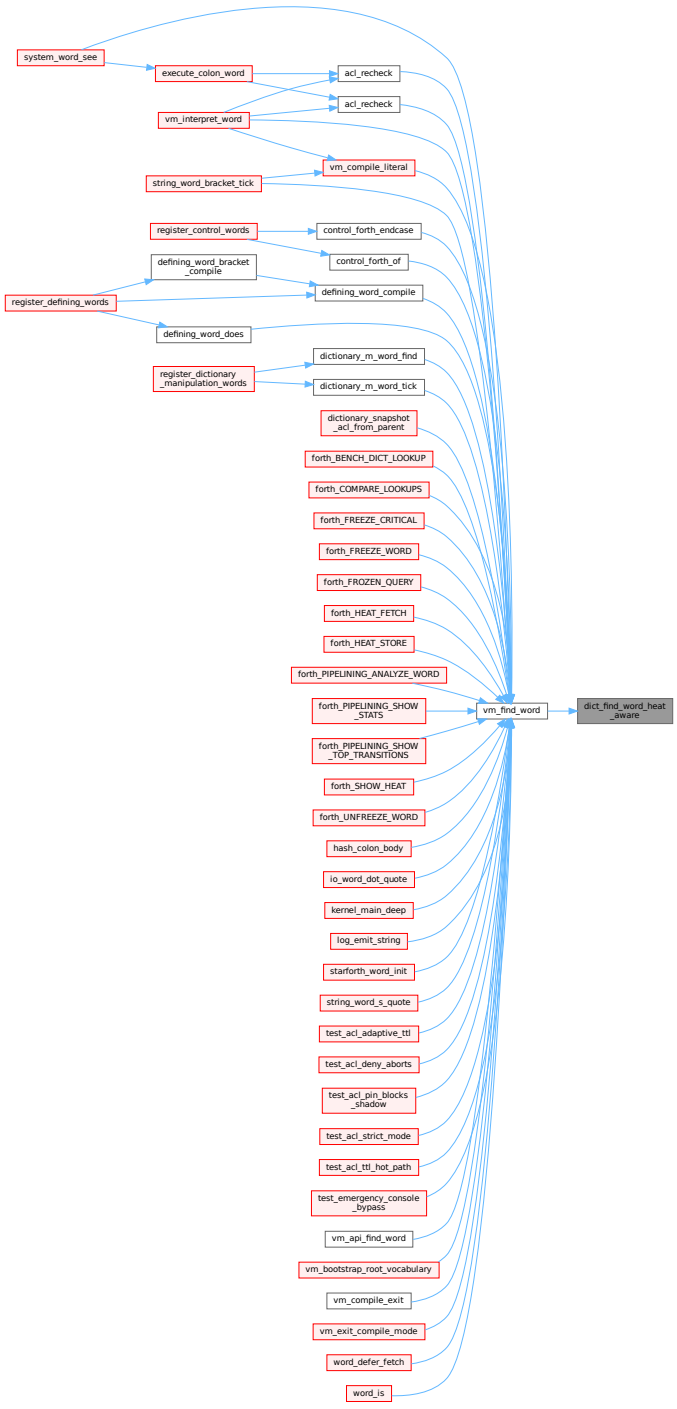
References name, sf_fc_count, sf_fc_list, and vm_dict_resolve_in_bucket().

Referenced by vm_find_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.202.2.3 dict_reorganize_buckets_by_heat()

```
void dict_reorganize_buckets_by_heat (
            VM * vm )
```

Reorganize dictionary buckets by inference-driven priorities.

Uses rolling window co-execution patterns to reorder words within buckets. Hot/frequently co-executing words are placed first for faster lookup.

Reorganize dictionary buckets by inference-driven priorities.

Holds `vm->dict_lock` for the entire operation to prevent concurrent `DictEntry` frees (from word redefinition or GC) from invalidating the pointer array while it is being sorted. For each non-empty bucket in `sf_fc_list`[0..255]:

1. Reads `sf_fc_count`[bucket_id] and `sf_fc_cap`[bucket_id]; skips the bucket if `count > cap` (indicates a TOCTOU race — logs a warning).
2. Allocates a temporary copy of the bucket pointer array.
3. Re-verifies `count` has not changed (TOCTOU double-check); if it has, frees the copy and skips.
4. Sorts the copy with `qsort` + `compare_dict_entry_heat_desc` so that the hottest `DictEntry*` comes first.
5. Copies the sorted array back over the original bucket.
6. Frees the temporary copy.

After returning, `vm->last_bucket_reorg_ns` is updated to the completion timestamp so that `dict_adaptive_optimization_pass()` can enforce its 1 Hz throttle. The elapsed reorganization time is emitted at `LOG_DEBUG`.

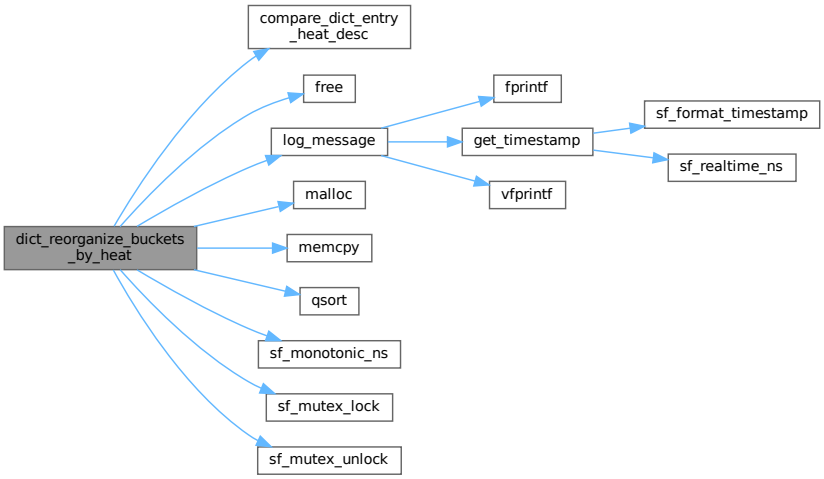
Parameters

vm	The VM whose first-character hash buckets are reorganized. No-op if NULL. `dict_lock` is held during the entire scan.

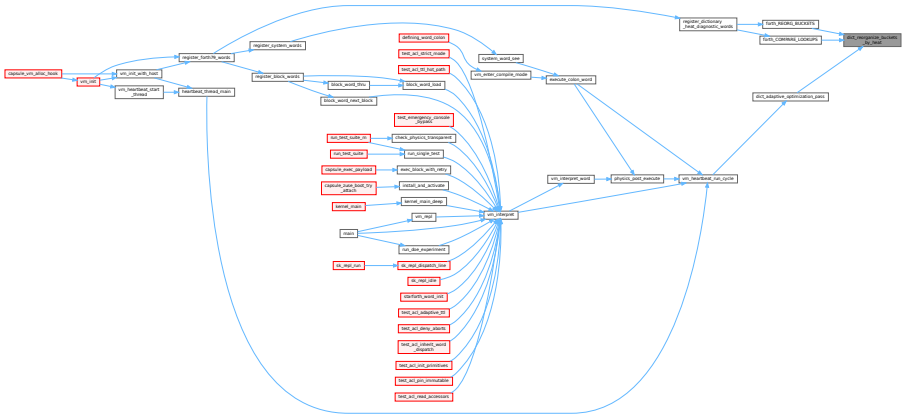
References compare_dict_entry_heat_desc(), count, VM::dict_lock, free(), VM::last_bucket_reorg_ns, LOG_DEBUG, log_message(), LOG_WARN, malloc(), memcpy(), qsort(), sf_fc_cap, sf_fc_count, sf_fc_list, sf_monotonic_ns(), sf_mutex_lock(), and sf_mutex_unlock().

Referenced by dict_adaptive_optimization_pass(), forth_COMPARE_LOOKUPS(), and forth_REORG_BUCKETS().

Here is the call graph for this function:



Here is the caller graph for this function:



7.202.2.4 dict_update_heat_percentiles()

```
void dict_update_heat_percentiles (
            VM * vm )
```

Calculate heat percentile thresholds for all words in dictionary.

Computes 25th, 50th, 75th percentiles of execution_heat across all dictionary entries. Used to partition lookup search order.

Calculate heat percentile thresholds for all words in dictionary.

Traverses the full linked-list dictionary under `vm->dict_lock` to snapshot every word's `execution_heat` into a stack-allocated scratch array (up to `DICTIONARY_SIZE` entries). After releasing the lock it sorts the snapshot with `qsort` + `compare_heat_ascending` and computes percentile indices using integer arithmetic:

```
vm->heat_threshold_25th = heats[(count * 25) / 100];
vm->heat_threshold_50th = heats[(count * 50) / 100];
vm->heat_threshold_75th = heats[(count * 75) / 100];
```

If the dictionary contains no words all three thresholds are set to 0. The lock is held only during traversal; sorting is performed outside the critical section to minimise contention with the concurrent heartbeat decay sweep that also walks the word list.

The resulting thresholds are consumed by `dict_find_word_heat_aware()` (which stratifies its search by heat tier) and by `dict_adaptive_optimization_pass()` (which calls this function as its first step every ∼1 Hz).

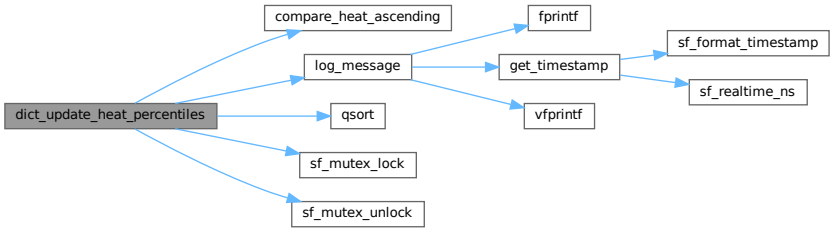
Parameters

vm	The VM whose dictionary is sampled and whose `heat_threshold_25th` / `heat_threshold_50th` / `heat_threshold_75th` fields are updated. No-op if NULL.

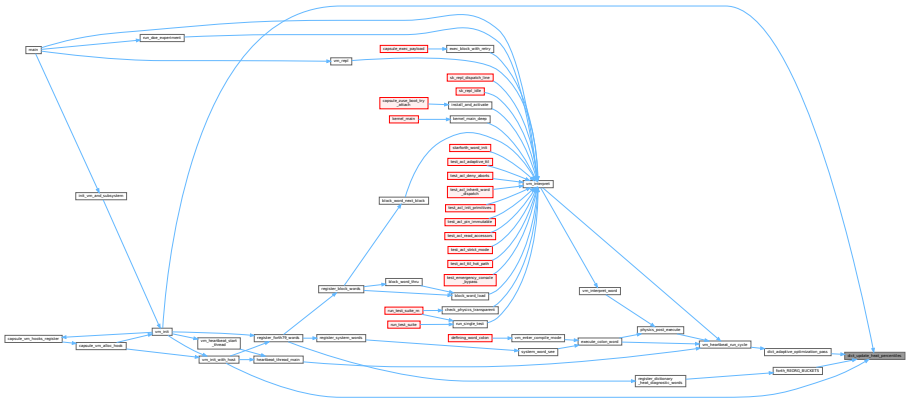
References compare_heat_ascending(), count, VM::dict_lock, DICTIONARY_SIZE, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::latest, LOG_DEBUG, log_message(), qsort(), sf_mutex_lock(), and sf_mutex_unlock().

Referenced by dict_adaptive_optimization_pass(), forth_REORG_BUCKETS(), vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.202.3 Variable Documentation

7.202.3.1 sf_fc_cap

```
size_t sf_fc_cap[256]  [extern]
```

Referenced by dict_reorganize_buckets_by_heat(), sf_fc_clear_all(), and sf_fc_reserve().

7.202.3.2 sf_fc_count

```
size_t sf_fc_count[256]  [extern]
```

Referenced by dict_find_word_heat_aware(), and dict_reorganize_buckets_by_heat().

7.202.3.3 sf_fc_list

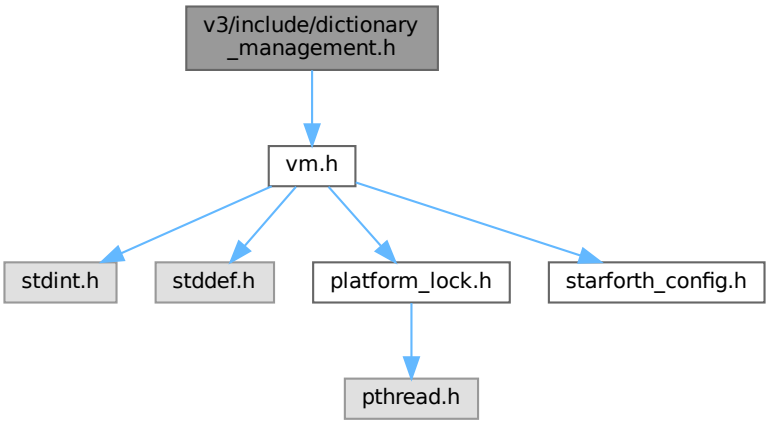
```
DictEntry** sf_fc_list[256]  [extern]
```

Referenced by dict_find_word_heat_aware(), and dict_reorganize_buckets_by_heat().

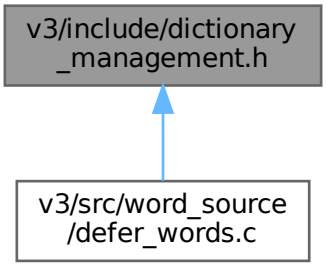
7.203 v3/include/dictionary_management.h File Reference

```
#include "vm.h"
```

Include dependency graph for dictionary_management.h:



This graph shows which files directly or indirectly include this file:



Functions

- void dictionary_init (VM *vm)

  Initialize the Forth dictionary.
- void dictionary_register (VM *vm, const char *name, word_func_t func)

  Register a new word in the dictionary.
- word_func_t dictionary_lookup (VM *vm, const char *name)

  Look up a word in the dictionary by name.
- void dictionary_clear (VM *vm)

  Clear the entire dictionary.

7.203.1 Function Documentation

7.203.1.1 dictionary_clear()

```
void dictionary_clear (
            VM * vm )
```

Clear the entire dictionary.

Parameters

vm	Pointer to the virtual machine instance

Removes all words from the dictionary and frees associated memory. This is an optional operation that may be used during cleanup.

7.203.1.2 dictionary_init()

```
void dictionary_init (
            VM * vm )
```

Initialize the Forth dictionary.

Parameters

vm	Pointer to the virtual machine instance

Initializes all dictionary structures and prepares them for use. Must be called before any other dictionary operations.

7.203.1.3 dictionary_lookup()

```
word_func_t dictionary_lookup (
            VM * vm,
            const char * name )
```

Look up a word in the dictionary by name.

Parameters

vm	Pointer to the virtual machine instance
name	Name of the word to look up

Returns

word_func_t Function pointer to the word's implementation or NULL if not found

7.203.1.4 dictionary_register()

```
void dictionary_register (
            VM * vm,
            const char * name,
            word_func_t func )
```

Register a new word in the dictionary.

Parameters

vm	Pointer to the virtual machine instance
name	Name of the word to register
func	Function pointer to the word's implementation

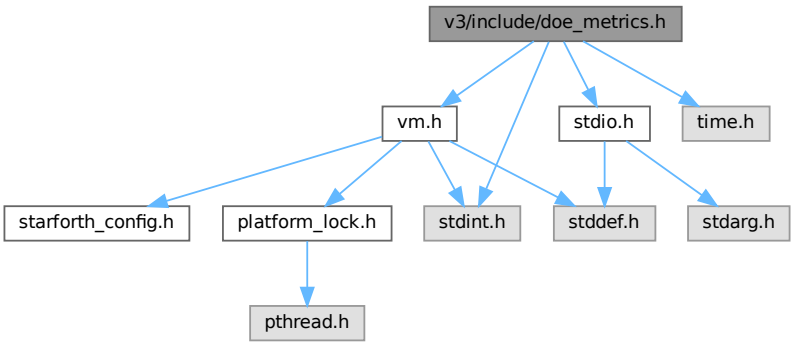
Adds a new word to the dictionary with the specified name and implementation.

7.204 v3/include/doe_metrics.h File Reference

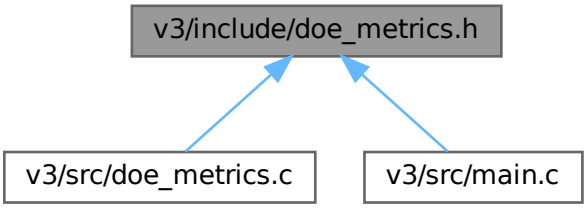
Design of Experiments (DoE) metrics collection API.

```
#include <stdio.h>
#include <stdint.h>
#include <time.h>
#include "vm.h"
```

Include dependency graph for doe_metrics.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct DoeMetrics

  Complete metrics snapshot for a single DoE run.

Functions

- DoeMetrics metrics_from_vm (VM ∗vm, uint64_t workload_duration_ns, int32_t cpu_temp_delta_c, int32_t cpu_freq_delta_mhz)

  Extract complete metrics snapshot from VM.
- void metrics_write_csv_header (FILE ∗out)

  Write CSV header row.
- void metrics_write_csv_row (FILE ∗out, const DoeMetrics ∗metrics)

  Write metrics as CSV row.
- void metrics_write_james_law_csv_row (FILE ∗out, const DoeMetrics ∗metrics, VM ∗vm)

  Write reduced James Law CSV row (20 columns)
- void metrics_print_text (FILE ∗out, const DoeMetrics ∗metrics)

  Print metrics as human-readable text.
- int32_t metrics_get_cpu_temp_c (void)

  Get current CPU temperature (Celsius)
- int32_t metrics_get_cpu_freq_mhz (void)

  Get current CPU frequency (MHz)
- void metrics_get_timestamp (char ∗buf, size_t bufsize)

  Get current timestamp as ISO 8601 string.

7.204.1 Detailed Description

Design of Experiments (DoE) metrics collection API.

Provides C-level programmatic access to VM metrics for physics-based experiments. Enables efficient in-process data collection without string parsing overhead.

Usage: struct DoeMetrics metrics = metrics_from_vm(vm); metrics_write_csv_row(stdout, &metrics, run_number);

7.204.2 Function Documentation

7.204.2.1 metrics_from_vm()

```
DoeMetrics metrics_from_vm (
            VM * vm,
            uint64_t workload_duration_ns,
            int32_t cpu_temp_delta_c,
            int32_t cpu_freq_delta_mhz )
```

Extract complete metrics snapshot from VM.

Collects all metrics from current VM state after a test harness run. Workload duration and CPU deltas must be supplied by caller.

Parameters

vm	Pointer to Forth VM instance
workload_duration_ns	Actual VM workload execution time (nanoseconds)
cpu_temp_delta_c	CPU temperature change during run (°C)
cpu_freq_delta_mhz	CPU frequency change during run (MHz)

Returns

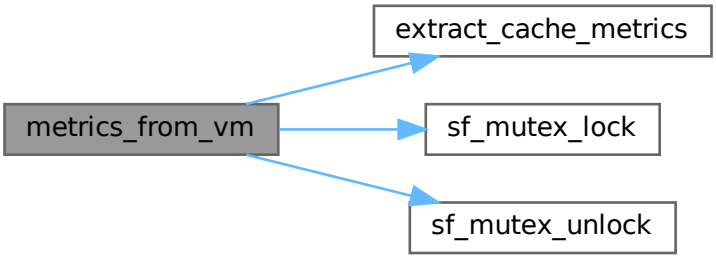
Populated metrics structure (timestamp filled in)

Extract complete metrics snapshot from VM.

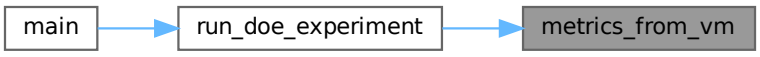
References DoeMetrics::actual_window_size, DoeMetrics::adaptive_shrink_rate, DoeMetrics::avg_word_heat, DoeMetrics::cache_demotions, DoeMetrics::cache_promotions, DoeMetrics::ci_lower_95, DoeMetrics::ci_upper_95, DoeMetrics::context_accuracy_percent, DoeMetrics::context_correct, DoeMetrics::context_predictions_total, DoeMetrics::cpu_freq_delta_mhz_q48, DoeMetrics::cpu_temp_delta_c_q48, DECAY_MIN_INTERVAL, DoeMetrics::decay_min_inter DECAY_RATE_PER_US_Q16, DoeMetrics::decay_rate_q16, DoeMetrics::decay_slope, VM::decay_slope_q48, VM::dict_lock, DoeMetrics::dictionary_lookups, HeartbeatState::dictionary_lookups, DoeMetrics::early_exit_count, HeartbeatState::early_exit_count, RollingWindowOfTruth::effective_window_size, DoeMetrics::enable_hotwords_cache, ENABLE_HOTWORDS_CACHE, DoeMetrics::enable_loop_1_heat_tracking, DoeMetrics::enable_loop_2_rolling_window, DoeMetrics::enable_loop_3_linear_decay, DoeMetrics::enable_loop_4_pipelining, DoeMetrics::enable_loop_5_window_inference, DoeMetrics::enable_loop_6_decay_inference, DoeMetrics::enable_loop_7_adaptive_heartrate, DoeMetrics::enable_pipelining, ENABLE_PIPELINING, HotwordsCache_s::enabled, HotwordsStats::evictions, extract_cache_metrics(), DoeMetrics::final_effective_v VM::heartbeat, DoeMetrics::heat_cache_demotion_threshold, DoeMetrics::hot_word_count, VM::hotwords_cache, HOTWORDS_EXECUTION_HEAT_THRESHOLD, DoeMetrics::inference_run_count, HeartbeatState::inference_run_count, VM::last_inference_outputs, VM::latest, DoeMetrics::memory_allocated_bytes, VM::pipeline_metrics, DoeMetrics::prefetch_accuracy DoeMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_attempts, DoeMetrics::prefetch_hits, PipelineGlobalMetrics::prefetch HotwordsStats::promotions, VM::rolling_window, DoeMetrics::rolling_window_size, ROLLING_WINDOW_SIZE, DoeMetrics::rolling_window_width, sf_mutex_lock(), sf_mutex_unlock(), DoeMetrics::speedup_vs_baseline, DoeMetrics::stale_word_count, DoeMetrics::stale_word_ratio, HotwordsCache_s::stats, DoeMetrics::tick_count, HeartbeatState::tick_count, DoeMetrics::tick_target_ns, HeartbeatState::tick_target_ns, DoeMetrics::total_executions, RollingWindowOfTruth::total_executions, DoeMetrics::total_heat, DoeMetrics::total_lookups, HotwordsStats::total_lookups, DoeMetrics::total_runtime_ms, VM::tuning_lock, DoeMetrics::vm_workload_duration_ns_q48, DoeMetrics::window_diversity_percent DoeMetrics::window_final_size_bytes, DoeMetrics::window_tuning_checks, PipelineGlobalMetrics::window_tuning_checks, DoeMetrics::window_variance_q48, InferenceOutputs::window_variance_q48, word_count, DoeMetrics::words_executed, and HeartbeatState::words_executed.

Referenced by run_doe_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:



7.204.2.2 metrics_get_cpu_freq_mhz()

```
int32_t metrics_get_cpu_freq_mhz (
            void  )
```

Get current CPU frequency (MHz)

Tries multiple sources:

1. /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
2. /proc/cpuinfo Returns 0 if unavailable.

Returns

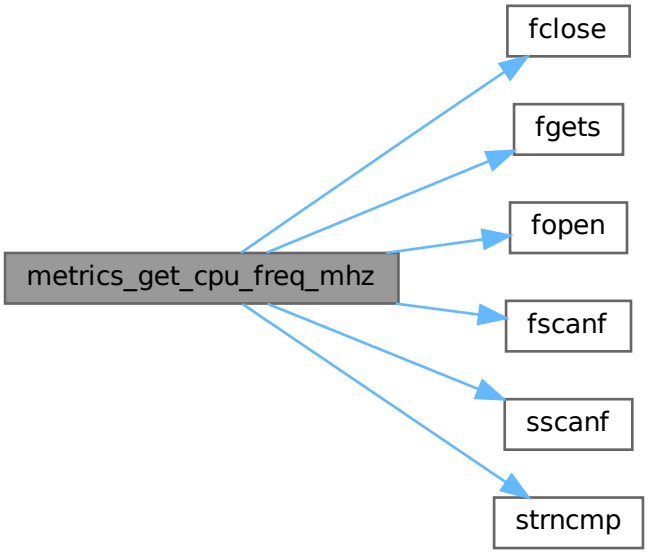
Frequency in MHz, or 0 if unable to read

Get current CPU frequency (MHz)

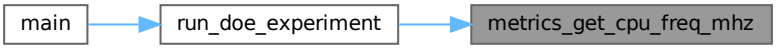
References fclose(), fgets(), fopen(), fscanf(), sscanf(), and strncmp().

Referenced by run_doe_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:



7.204.2.3 metrics_get_cpu_temp_c()

```
int32_t metrics_get_cpu_temp_c (
            void  )
```

Get current CPU temperature (Celsius)

Reads /sys/class/thermal/thermal_zone0/temp on Linux, returns 0 if unavailable.

Returns

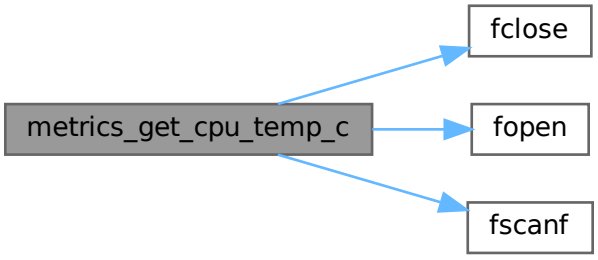
Temperature in °C, or 0 if unable to read

Get current CPU temperature (Celsius)

References fclose(), fopen(), and fscanf().

Referenced by run_doe_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

7.204.2.4 metrics_get_timestamp()

```
void metrics_get_timestamp (
            char * buf,
            size_t bufsize )
```

Get current timestamp as ISO 8601 string.

Format: "YYYY-MM-DDTHH:MM:SS"

Parameters

buf	Buffer to write timestamp (minimum 32 bytes)
bufsize	Size of buffer

7.204.2.5 metrics_print_text()

```
void metrics_print_text (
            FILE * out,
            const DoeMetrics * metrics )
```

Print metrics as human-readable text.

Outputs metrics in formatted text for debugging/verification.

Parameters

out	Output file stream
metrics	Metrics structure to print

References DoeMetrics::bucket_hit_percent, DoeMetrics::bucket_hits, DoeMetrics::bucket_search_latency_ns, DoeMetrics::bucket_search_stddev_ns, DoeMetrics::cache_hit_latency_ns, DoeMetrics::cache_hit_percent, DoeMetrics::cache_hit_stddev_ns, DoeMetrics::cache_hits, DoeMetrics::context_accuracy_percent, DoeMetrics::context_correct, DoeMetrics::context_predictions_total, DoeMetrics::cpu_freq_delta_mhz_q48, DoeMetrics::cpu_temp_delta_c_q48, DoeMetrics::decay_slope, fprintf(), DoeMetrics::rolling_window_width, DoeMetrics::total_lookups, and DoeMetrics::vm_workload_dur

Here is the call graph for this function:

7.204.2.6 metrics_write_csv_header()

```
void metrics_write_csv_header (
            FILE * out )
```

Write CSV header row.

Writes the 35-column CSV header to output stream.

Parameters

out	Output file stream

Write CSV header row.

References fprintf().

Here is the call graph for this function:

7.204.2.7 metrics_write_csv_row()

```
void metrics_write_csv_row (
            FILE * out,
            const DoeMetrics * metrics )
```

Write metrics as CSV row.

Writes a single run's metrics as comma-separated values.

Parameters

out	Output file stream
metrics	Metrics structure to write

Write metrics as CSV row.

References DoeMetrics::actual_window_size, DoeMetrics::adaptive_shrink_rate, DoeMetrics::avg_word_heat, DoeMetrics::bucket_hit_percent, DoeMetrics::bucket_hits, DoeMetrics::bucket_search_latency_ns, DoeMetrics::bucket_search_stdde DoeMetrics::cache_demotions, DoeMetrics::cache_hit_latency_ns, DoeMetrics::cache_hit_percent, DoeMetrics::cache_hit_stddev_n DoeMetrics::cache_hits, DoeMetrics::cache_promotions, DoeMetrics::ci_lower_95, DoeMetrics::ci_upper_95, DoeMetrics::context_accuracy_percent, DoeMetrics::context_correct, DoeMetrics::context_predictions_total, DoeMetrics::cpu_freq_delta_mhz_q48, DoeMetrics::cpu_temp_delta_c_q48, DoeMetrics::decay_min_interval_ns, DoeMetrics::decay_rate_q16, DoeMetrics::decay_slope, DoeMetrics::dictionary_lookups, DoeMetrics::early_exit_count, DoeMetrics::enable_hotwords_cache, DoeMetrics::enable_loop_1_heat_tracking, DoeMetrics::enable_loop_2_rolling_window, DoeMetrics::enable_loop_3_linear_decay, DoeMetrics::enable_loop_4_pipelining, DoeMetrics::enable_loop_5_window_inference, DoeMetrics::enable_loop_6_decay_inference, DoeMetrics::enable_loop_7_adaptive_heartrate, DoeMetrics::enable_pipelining, DoeMetrics::final_effective_window_size, fprintf(), DoeMetrics::heat_cache_demotion_threshold, DoeMetrics::hot_word_count, DoeMetrics::inference_run_count, DoeMetrics::memory_allocated_bytes, DoeMetrics::prefetch_accuracy_percent, DoeMetrics::prefetch_attempts, DoeMetrics::prefetch_hits, DoeMetrics::rolling_window_size, DoeMetrics::rolling_window_width, DoeMetrics::speedup_vs_baseline, DoeMetrics::stale_word_count, DoeMetrics::stale_word_ratio, DoeMetrics::tick_count, DoeMetrics::tick_target_ns, DoeMetrics::total_executions, DoeMetrics::total_heat, DoeMetrics::total_lookups, DoeMetrics::total_runtime_ms, DoeMetrics::vm_workload_duration_ns_q48, DoeMetrics::window_diversity_percent, DoeMetrics::window_final_size_bytes, DoeMetrics::window_tuning_checks, DoeMetrics::window_variance_q48, and DoeMetrics::words_executed.

Here is the call graph for this function:

7.204.2.8 metrics_write_james_law_csv_row()

```
void metrics_write_james_law_csv_row (
            FILE * out,
            const DoeMetrics * metrics,
            VM * vm )
```

Write reduced James Law CSV row (20 columns)

Outputs only the subset of metrics needed for the James Law validation experiment. This matches the header format in run_window_sweep.sh (minus the 5 script-prepended columns).

Parameters

out	Output file stream
metrics	Metrics structure to write
vm	VM instance (for heartbeat bucket aggregates and L8 mode)

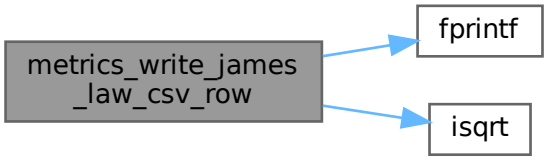
Write reduced James Law CSV row (20 columns)

This outputs only the subset of metrics needed for the James Law validation experiment. Format matches the header in run_window_sweep.sh (minus the 5 script-prepended columns).

Expected columns (20): workload_duration_ns, l8_mode_final, lambda_effective, K_statistic, K_deviation, entropy, cv, temporal_decay, stability_score, ns_per_word, total_runtime_ms, window_final, hot_word_count, avg_heat, bucket_mean_K, bucket_cv_K, bucket_window_var, bucket_heat_var, bucket_mode_transitions, bucket_collapse↩
_flag

References DoeMetrics::avg_word_heat, HeartbeatState::bucket_collapse_flag, HeartbeatState::bucket_mode_transitions, HeartbeatState::bucket_sum_heat_variance, HeartbeatState::bucket_sum_K, HeartbeatState::bucket_sum_K_squared, HeartbeatState::bucket_sum_window_variance, HeartbeatState::bucket_tick_count, DoeMetrics::cache_hit_latency_ns, DoeMetrics::cache_hit_stddev_ns, DoeMetrics::decay_slope, DoeMetrics::final_effective_window_size, fprintf(), VM::heartbeat, HEARTBEAT_TICK_BUFFER_SIZE, DoeMetrics::hot_word_count, isqrt(), HeartbeatTickSnapshot::l8_mode, DoeMetrics::rolling_window_size, DoeMetrics::rolling_window_width, DoeMetrics::stale_word_ratio, HeartbeatState::tick_buffer, HeartbeatState::tick_buffer_write_index, DoeMetrics::total_runtime_ms, DoeMetrics::vm_workload_duration_ns_q48, DoeMetrics::window_diversity_percent, and DoeMetrics::words_executed.

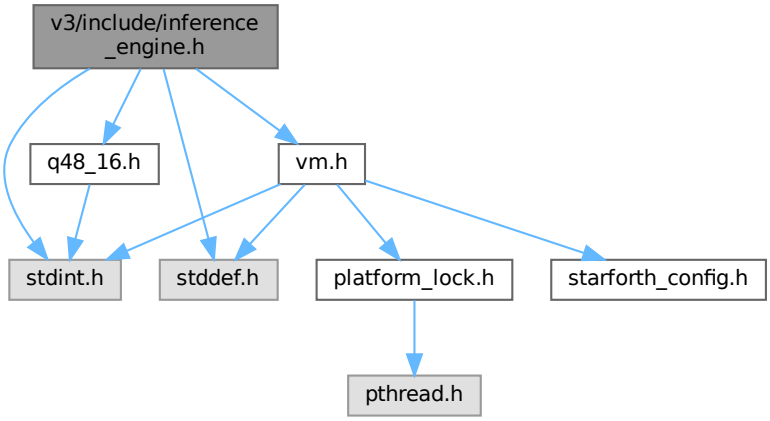
Here is the call graph for this function:



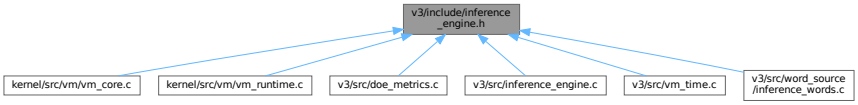
7.205 v3/include/inference_engine.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "vm.h"
#include "q48_16.h"
```

Include dependency graph for inference_engine.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct InferenceInputs
- struct InferenceOutputs

Functions

- void inference_engine_run (InferenceInputs *inputs, InferenceOutputs *outputs)

  Run adaptive inference engine on window data.
- const char * inference_outputs_to_string (const InferenceOutputs *outputs)

  Format inference outputs as a human-readable string.
- int inference_outputs_validate (const InferenceOutputs *outputs)

  Validate inference outputs for sanity.
- q48_16_t compute_variance_q48 (const uint64_t *heat_data, uint64_t length)

  Compute the variance of a heat trajectory in Q48.16 fixed-point.
- uint32_t find_variance_inflection (const uint64_t *heat_data, uint64_t trajectory_length, q48_16_t full_↩ variance)

  Find the minimum window size where heat variance is statistically stable.
- uint64_t infer_decay_slope_q48 (const uint64_t *heat_data, uint64_t length)

  Infer the decay slope from a heat trajectory via log-linear regression.

7.205.1 Function Documentation

7.205.1.1 compute_variance_q48()

```
q48_16_t compute_variance_q48 (
            const uint64_t * heat_data,
            uint64_t length )
```

Compute the variance of a heat trajectory in Q48.16 fixed-point.

Computes the population variance: Σ(heat[i] - mean)^2 / N. All intermediate calculations use Q48.16 arithmetic. Returns 0 for an empty array.

Parameters

heat_data	Array of execution heat values (raw uint64_t counts)
length	Number of elements in `heat_data`

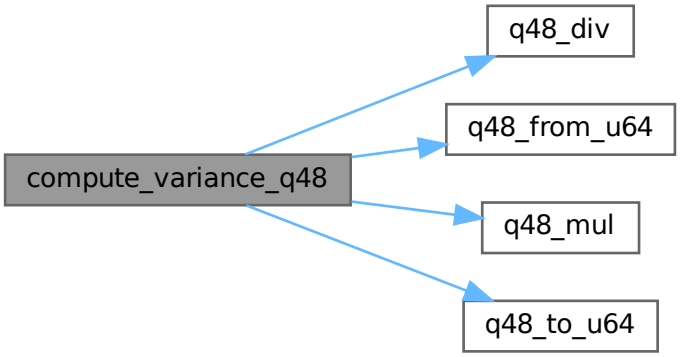
Returns

Population variance in Q48.16 format

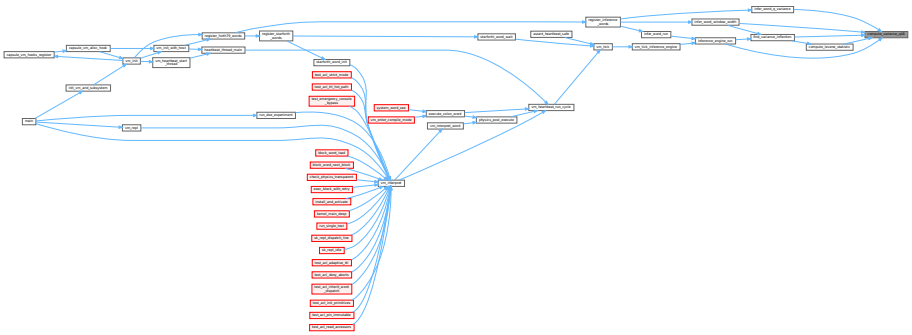
References q48_div(), q48_from_u64(), q48_mul(), and q48_to_u64().

Referenced by compute_levene_statistic(), find_variance_inflection(), infer_word_q_variance(), infer_word_window_width(), and inference_engine_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.205.1.2 find_variance_inflection()

```
uint32_t find_variance_inflection (
            const uint64_t * heat_data,
            uint64_t trajectory_length,
            q48_16_t full_variance )
```

Find the minimum window size where heat variance is statistically stable.

Implements Loop #5 (window width inference). Scans candidate window sizes from ADAPTIVE_MIN_WINDOW↩ _SIZE to the full trajectory in steps of 64, divides the trajectory into disjoint chunks of that size, and applies Levene's test. Returns the first size where W <= 6.5 (α=0.05 critical value). Falls back to the full trajectory length if no size passes. full_variance is unused (retained for API compatibility with callers that pre-compute it).

Parameters

heat_data	Heat trajectory array
trajectory_length	Number of elements in heat_data
full_variance	Unused; retained for API compatibility

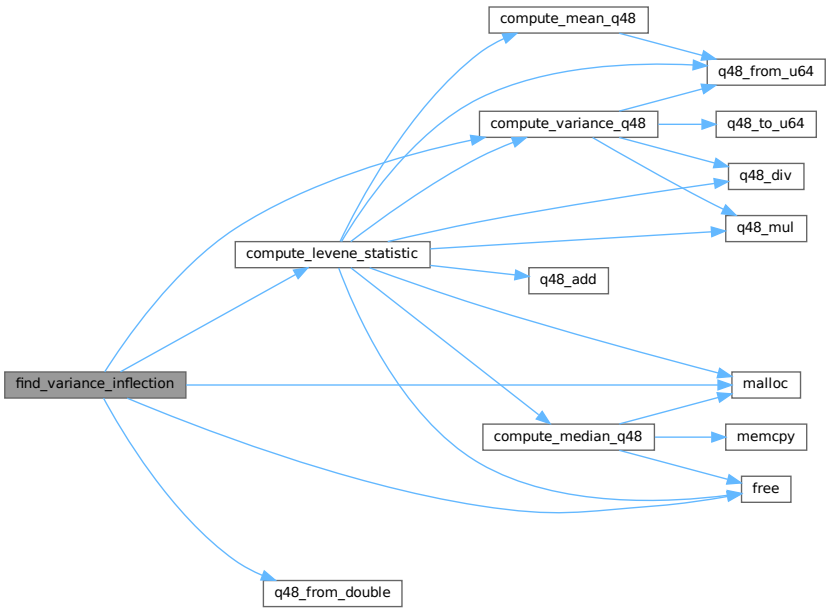
Returns

Minimum sufficient window width in [ADAPTIVE_MIN_WINDOW_SIZE, ROLLING_WINDOW_SIZE]

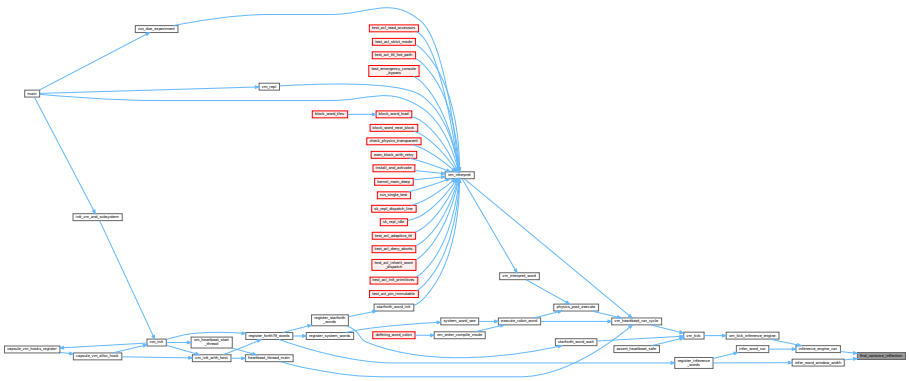
References ADAPTIVE_MIN_WINDOW_SIZE, compute_levene_statistic(), compute_variance_q48(), free(), malloc(), q48_from_double(), and ROLLING_WINDOW_SIZE.

Referenced by infer_word_window_width(), and inference_engine_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.205.1.3 infer_decay_slope_q48()

```
uint64_t infer_decay_slope_q48 (
            const uint64_t * heat_data,
            uint64_t length )
```

Infer the decay slope from a heat trajectory via log-linear regression.

Implements Loop #6 (decay slope inference). Fits the model ln(heat[t]) = ln(h?) - slope∗t using the closed-form OLS solution: slope = (n∗Σ(t·ln(h)) - Σt·Σln(h)) / (n·Σt^2 - $(\Sigma t)^2$). The sign of the numerator is taken as absolute value since decay rate is always positive. Zero heat values are skipped. Returns 0 if fewer than 2 samples are available.

Parameters

heat_data	Array of execution heat values (raw counts)
length	Number of elements in `heat_data`

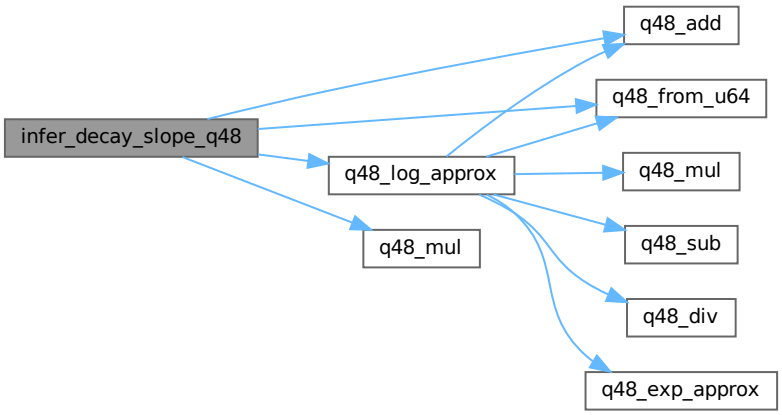
Returns

Inferred decay slope in Q48.16 format (0 = insufficient data)

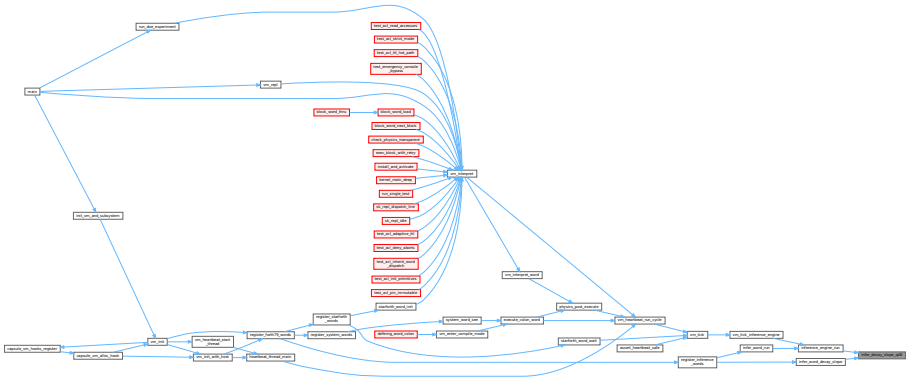
References q48_add(), q48_from_u64(), q48_log_approx(), and q48_mul().

Referenced by infer_word_decay_slope(), and inference_engine_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.205.1.4 inference_engine_run()

```
void inference_engine_run (
            InferenceInputs * inputs,
            InferenceOutputs * outputs )
```

Run adaptive inference engine on window data.

High-level flow:

1. ANOVA Check: Is variance delta $<$ 5%?
   - YES: Return cached outputs immediately (cost: $\sim$100 cycles)
   - NO: Continue to full inference
2. Extract trajectory: Fresh execution_heat snapshot from dictionary
3. Window Width Inference:
   - Compute variance across entire window
   - Detect inflection point (where d(variance)/d(size) $\rightarrow$ 0)
   - Return optimal window size
4. Decay Slope Inference:
   - Transform trajectory to log space (ln(heat[t]))
   - Fit linear regression: ln(heat) = a - slope$*$t
   - Extract slope coefficient in Q48.16
5. Diagnostics:
   - Compute fit quality (R^2 or residual)
   - Store variance metric for next early-exit check

Parameters

inputs	Snapshot of window + metrics (caller-provided)
outputs	Tuning parameters + diagnostics (caller-provided, will be updated)

Returns

void (results_run_01_2025_12_08 in outputs struct)

Preconditions:

- inputs != NULL
- outputs != NULL
- inputs->window != NULL
- inputs->window->is_warm == 1 (window has enough data)
- inputs->trajectory_length > 1

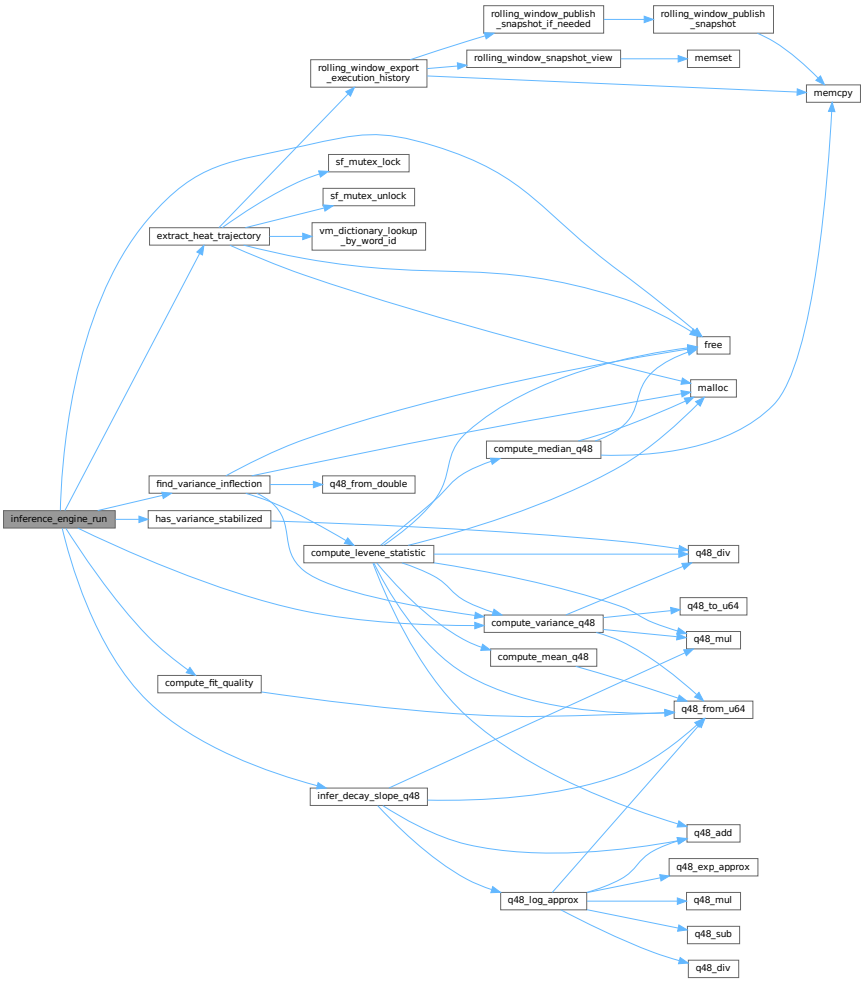
Side Effects:

- Modifies outputs struct (caller must preserve if caching)
- Allocates temporary trajectory buffer (freed on return)

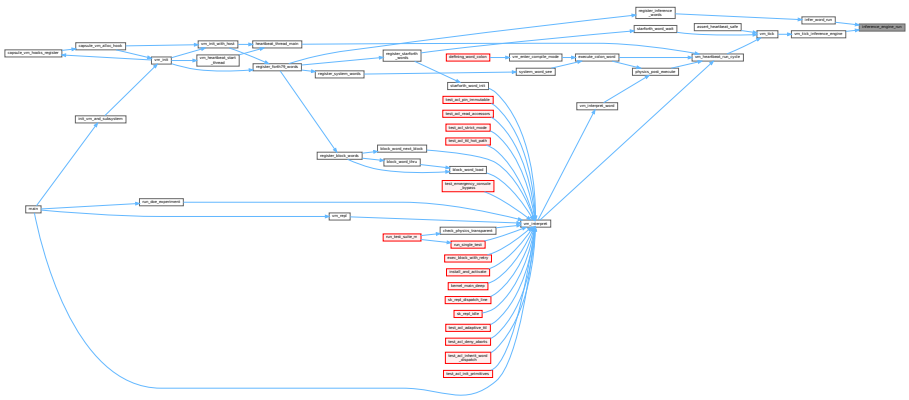
References InferenceOutputs::adaptive_decay_slope, InferenceOutputs::adaptive_window_width, compute_fit_quality(), compute_variance_q48(), InferenceOutputs::early_exited, extract_heat_trajectory(), find_variance_inflection(), free(), has_variance_stabilized(), infer_decay_slope_q48(), ssm_config_t::L5_window_inference, ssm_config_t::L6_decay_inference, InferenceOutputs::slope_fit_quality_q48, VM::ssm_config, InferenceInputs::vm, InferenceInputs::window, and InferenceOutputs::window_variance_q48.

Referenced by infer_word_run(), and vm_tick_inference_engine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.205.1.5 inference_outputs_to_string()

```
const char * inference_outputs_to_string (
```

```
            const InferenceOutputs * outputs )
```

Format inference outputs as a human-readable string.

Writes into a static 256-byte buffer — not thread-safe, for diagnostics only. Format: "window=N var=↩ F slope=F quality=F (cached|full)".

Parameters

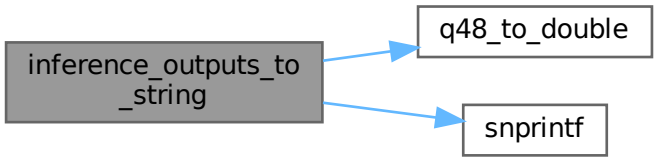
outputs	Inference outputs to format (NULL produces "(null)")

Returns

Pointer to a static string buffer

References InferenceOutputs::adaptive_decay_slope, InferenceOutputs::adaptive_window_width, InferenceOutputs::early_exited, q48_to_double(), InferenceOutputs::slope_fit_quality_q48, snprintf(), and InferenceOutputs::window_variance_q48.

Here is the call graph for this function:



7.205.1.6 inference_outputs_validate()

```
int inference_outputs_validate (
            const InferenceOutputs * outputs )
```

Validate inference outputs for sanity.

Checks that adaptive_window_width is in [ADAPTIVE_MIN_WINDOW_SIZE, ROLLING_WINDOW_↩ SIZE], that adaptive_decay_slope is non-zero and <= 100.0 in Q48.16, and that slope_fit_↩ quality_q48 is in [0.0, 1.0].

Parameters

outputs	Inference outputs to validate (NULL returns 0)

Returns

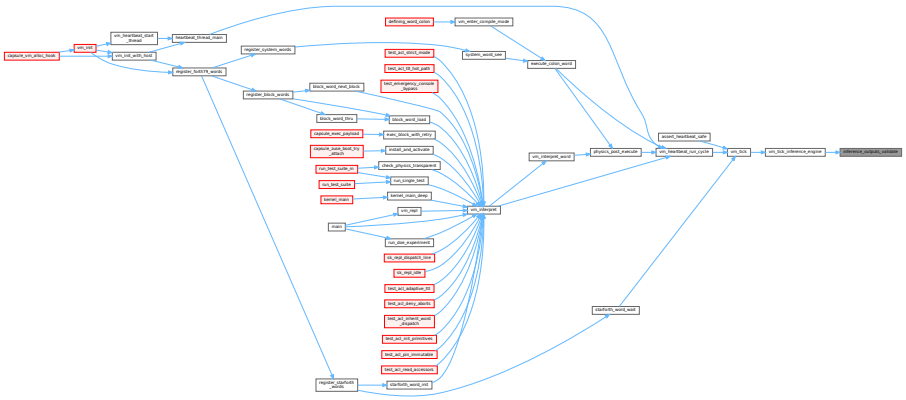
1 if all fields are within acceptable ranges, 0 otherwise

References InferenceOutputs::adaptive_decay_slope, ADAPTIVE_MIN_WINDOW_SIZE, InferenceOutputs::adaptive_window_width, q48_from_u64(), ROLLING_WINDOW_SIZE, and InferenceOutputs::slope_fit_quality_q48.

Referenced by vm_tick_inference_engine().

Here is the call graph for this function:

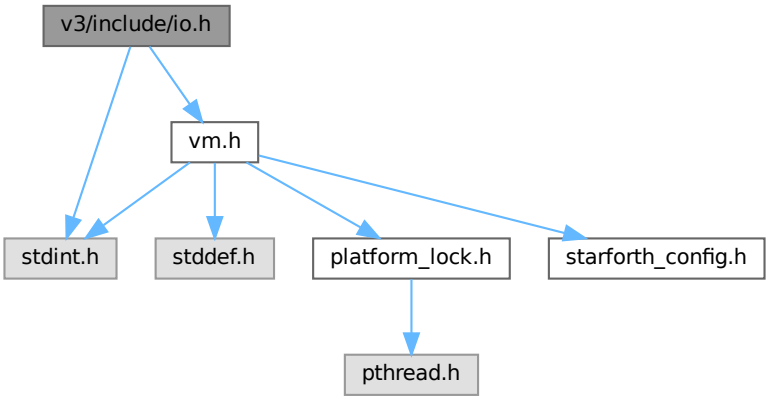
Here is the caller graph for this function:



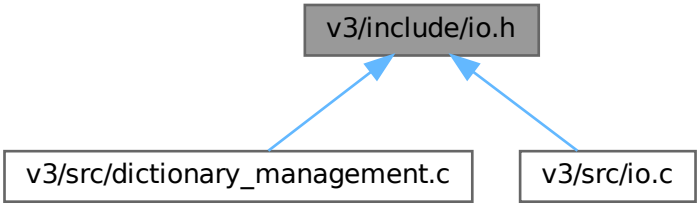
7.206 v3/include/io.h File Reference

```
#include <stdint.h>
#include "vm.h"
```

Include dependency graph for io.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define BLOCK_COUNT 1024

  Total number of blocks available.
- #define MEMORY_SIZE (BLOCK_SIZE ∗ BLOCK_COUNT)

  Total memory size in bytes.

Functions

- void io_init (VM ∗vm)

  Initialize the IO subsystem.
- int io_read_block (VM ∗vm, int block_num, unsigned char ∗buffer)

  Read a block from storage.
- int io_write_block (VM ∗vm, int block_num, const unsigned char ∗buffer)

  Write a block to VM memory.

7.206.1 Macro Definition Documentation

7.206.1.1 BLOCK_COUNT

```
#define BLOCK_COUNT 1024
```

Total number of blocks available.

7.206.1.2 MEMORY_SIZE

```
#define MEMORY_SIZE (BLOCK_SIZE * BLOCK_COUNT)
```

Total memory size in bytes.

7.206.2 Function Documentation

7.206.2.1 io_init()

```
void io_init (
            VM * vm )
```

Initialize the IO subsystem.

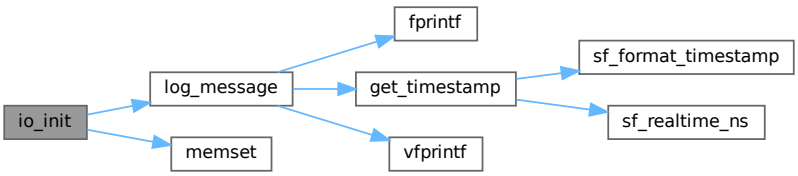
Parameters

vm	Pointer to VM instance

Initializes the block device view over the unified VM memory. Zeros out the entire VM memory space.

References BLOCK_SIZE, LOG_ERROR, LOG_INFO, log_message(), MAX_BLOCKS, VM::memory, memset(), and VM_MEMORY_SIZE.

Here is the call graph for this function:



7.206.2.2 io_read_block()

```
int io_read_block (
            VM * vm,
```

```
            int block_num,
            unsigned char * buffer )
```

Read a block from storage.

Parameters

vm	Pointer to the virtual machine instance
block_num	Block number to read
buffer	Buffer to store the read data

Returns

0 on success, non-zero on failure

7.206.2.3 io_write_block()

```
int io_write_block (
            VM * vm,
            int block_num,
            const unsigned char * buffer )
```

Write a block to VM memory.

Parameters

vm	Pointer to VM instance
block_num	Block number to write (0 to MAX_BLOCKS-1)
buffer	Source buffer containing block data

Returns

0 on success, -1 on error

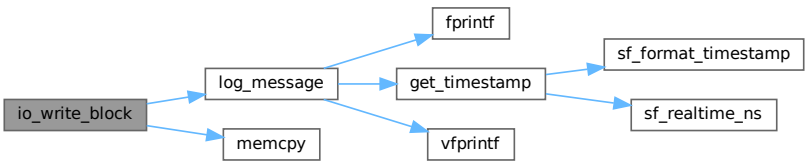
Writes a BLOCK_SIZE chunk of data from the provided buffer into the VM's memory at the specified block number offset. The block number must be within valid range. The entire block is written atomically.

Note

The buffer must contain at least BLOCK_SIZE bytes of valid data

References BLOCK_SIZE, LOG_DEBUG, LOG_ERROR, log_message(), MAX_BLOCKS, memcpy(), and VM::memory.

Here is the call graph for this function:



7.207 kernel/include/starkernel/log.h File Reference

Macros

- #define LOG_MSG_LINE_MAX 256

Enumerations

- enum LogLevel {
  LOG_NONE = -1 , LOG_ERROR = 0 , LOG_WARN , LOG_INFO ,
  LOG_TEST , LOG_DEBUG }

  Log levels — identical to the hosted enumeration so kernel VM code compiled against either header produces compatible call sites.
- enum TestResult { TEST_PASS = 0 , TEST_FAIL , TEST_SKIP , TEST_ERROR }

Functions

- void log_set_level (LogLevel level)

  Set the global logging level.
- LogLevel log_get_level (void)

  Gets the current global logging level.
- void log_message (LogLevel level, const char *fmt,...)

  Log a formatted message at the specified level.
- void log_test_result (const char *word_name, TestResult result)

  Log a test result with colored output.
- void log_set_vm (struct VM *vm)

7.207.1 Macro Definition Documentation

7.207.1.1 LOG_MSG_LINE_MAX

```
#define LOG_MSG_LINE_MAX 256
```

7.207.2 Enumeration Type Documentation

7.207.2.1 LogLevel

```
enum LogLevel
```

Log levels — identical to the hosted enumeration so kernel VM code compiled against either header produces compatible call sites.

Enumerator

LOG_NONE	
LOG_ERROR	
LOG_WARN	
LOG_INFO	
LOG_TEST	
LOG_DEBUG	

7.207.2.2 TestResult

```
enum TestResult
```

Enumerator

TEST_PASS	
TEST_FAIL	
TEST_SKIP	
TEST_ERROR	

7.207.3 Function Documentation

7.207.3.1 log_get_level()

```
LogLevel log_get_level (
            void  )
```

Gets the current global logging level.

Get the current logging level.

Returns

LogLevel Current logging level

7.207.3.2 log_message()

```
void log_message (
            LogLevel level,
            const char * fmt,
             ...  )
```

Log a formatted message at the specified level.

Parameters

level	The logging level for this message
fmt	Printf-style format string
...	Variable arguments for format string

7.207.3.3 log_set_level()

```
void log_set_level (
            LogLevel level )
```

Set the global logging level.

Parameters

level	The new logging level to set

Note

Default level is LOG_INFO

7.207.3.4 log_set_vm()

```
void log_set_vm (
            struct VM * vm )
```

7.207.3.5 log_test_result()

```
void log_test_result (
            const char * word_name,
            TestResult result )
```

Log a test result with colored output.

Parameters

word_name	Name of the word being tested
result	Result of the test execution

7.208 v3/include/log.h File Reference

Macros

- #define LOG_MSG_LINE_MAX 256

Enumerations

- enum LogLevel { LOG_NONE = -1 , LOG_ERROR = 0 , LOG_WARN , LOG_INFO , LOG_TEST , LOG_DEBUG }

  Logging level enumeration.
- enum TestResult { TEST_PASS = 0 , TEST_FAIL , TEST_SKIP , TEST_ERROR }

  Test result type enumeration.

Functions

- void log_set_level (LogLevel level)

  Set the global logging level.
- LogLevel log_get_level (void)

  Get the current logging level.
- void log_message (LogLevel level, const char *fmt,...)

  Log a formatted message at the specified level.
- void log_test_result (const char *word_name, TestResult result)

  Log a test result with colored output.
- void log_set_vm (struct VM *vm)

  Set the VM instance for persistent logging.

7.208.1 Macro Definition Documentation

7.208.1.1 LOG_MSG_LINE_MAX

```
#define LOG_MSG_LINE_MAX 256
```

7.208.2 Enumeration Type Documentation

7.208.2.1 LogLevel

```
enum LogLevel
```

Logging level enumeration.

Defines the available logging levels in order of increasing verbosity.

Enumerator

LOG_NONE	Completely disable all logging.
LOG_ERROR	Error messages only.
LOG_WARN	Warning and error messages.
LOG_INFO	Informational, warning and error messages.
LOG_TEST	Test results_run_01_2025_12_08 and all previous levels.
LOG_DEBUG	Debug messages and all previous levels.

7.208.2.2 TestResult

```
enum TestResult
```

Test result type enumeration.

Defines the possible outcomes of test execution.

Enumerator

TEST_PASS	Test completed successfully.
TEST_FAIL	Test failed.
TEST_SKIP	Test was skipped.
TEST_ERROR	Test encountered an error during execution.

7.208.3 Function Documentation

7.208.3.1 log_get_level()

```
LogLevel log_get_level (
            void  )
```

Get the current logging level.

Returns

The current logging level

Get the current logging level.

Returns

LogLevel Current logging level

References current_level.

Referenced by log_word_get_level().

Here is the caller graph for this function:

7.208.3.2 log_message()

```
void log_message (
            LogLevel level,
            const char * fmt,
             ...  )
```

Log a formatted message at the specified level.

Parameters

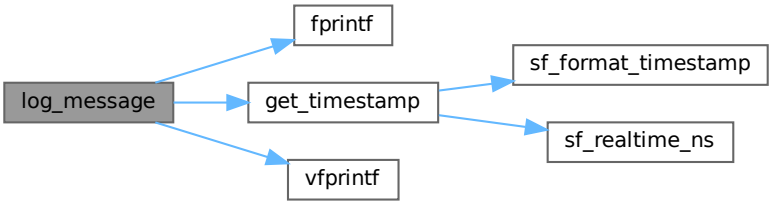
level	The logging level for this message
fmt	Printf-style format string
...	Variable arguments for format string

References color_reset, current_level, fprintf(), get_timestamp(), level, level_colors, level_names, LOG_NONE, stderr, and vfprintf().

Referenced by __attribute__(), acl_recheck(), acl_recheck(), arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), array_ptr(), assert_heartbeat_safe(), assert_stack_depth(), assert_stack_top(), assert_vm_error(), blk_format_or_load_disk(), blk_subsys_add_raw_device(), blk_subsys_attach_device(), blk_subsys_detach_device(), blk_subsys_relocate_block(), capsule_birth_baby(), capsule_birth_mama(), capsule_exec_payload(), capsule_run_experiment(), capsule_vm_kill_all_nonmama(), cf_epoch_sync(), cf_pop_item(), cf_push_item(), check_physics_transparent(), control_forth_0branch(), control_forth_again(), control_forth_begin(), control_forth_branch(), control_forth_case(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_EXIT(), control_forth_I(), control_forth_if(), control_forth_J(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_runtime_do(), control_forth_runtime_leave(), control_forth_runtime_loop(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), control_forth_then(), control_forth_UNLOOP(), control_forth_until(), control_forth_while(), defer_runtime(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_does_rt(), defining_runtime_lit(), defining_runtime_variable(), defining_word_colon(), defining_word_compile(), defining_word_constant(), defining_word_create(), defining_word_does(), defining_word_immediate(), defining_word_left_bracket(), defining_word_literal(), defining_word_right_bracket(), defining_word_variable(), dict_adaptive_optimization_pass(), dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), dictionary_m_word_tick(), dictionary_m_word_to_body(), dictionary_word_fence(), dictionary_word_forget(), dump_and_die(), editor_word_edit(), editor_word_l(), editor_word_s(), editor_word_show(), emit_cell(), enable_benchmark_mode(), execute_colon_word(), fail_and_exit(), fbw_plot(), format_word_hold(), forth_acl_allow_store(), forth_acl_inherit(), forth_acl_mode_store(), forth_acl_ttl_store(), forth_paren(), hosted_lifecycle_birth(), hosted_lifecycle_kill(), hosted_lifecycle_pause(), hosted_lifecycle_resume(), hosted_lifecycle_use(), infer_word_run(), init_blkio(), init_report(), init_vocabulary_system(), io_init(), io_word_type(), io_write_block(), log_do_emit(), log_emit_string(), log_str_emit(), log_word_set_level(), logical_word_and(), logical_word_equals(), logical_word_false(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_true(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), main(), memory_word_cells(), mixed_math_word_mod(), pop_xt(), print_module_summary(), register_arithmetic_words(), register_forth79_words(), register_word(), return_stack_word_r_fetch(), return_stack_word_r_from(), return_stack_word_to_r(), rolling_window_run_adaptive_pass(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), run_acl_words_tests(), run_all_tests(), run_arithmetic_words_tests(), run_benchmark_mode(), run_block_words_tests(), run_break_me_tests(), run_control_words_tests(), run_defining_words_tests(), run_dictionary_manipulation_words_tests(), run_dictionary_words_tests(), run_doe_experiment(), run_double_words_tests(), run_format_words_tests(), run_inference_words_tests(), run_integration_tests(), run_io_words_tests(), run_logical_words_tests(), run_mama_forth_words_tests(), run_memory_words_tests(), run_mixed_arithmetic_words_tests(), run_module_tests(), run_physics_freeze_words_tests(), run_q48_words_tests(), run_return_stack_words_tests(), run_single_test(), run_stack_words_tests(), run_starforth_words_tests(), run_stress_tests(), run_string_words_tests(), run_system_words_tests(), run_test_suite(), run_test_suite_m(), run_vocabulary_words_tests(), run_word_tests(), sk_repl_dispatch_line(), sk_repl_idle(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), stack_word_depth(), stack_word_drop(), stack_word_dup(), stack_word_minus_rot(), stack_word_over(), stack_word_pick(), stack_word_question_dup(), stack_word_roll(), stack_word_rot(), stack_word_swap(), starforth_word_execution_heat_fetch(), starforth_word_execution_heat_store(), starforth_word_init(), starforth_word_paren_dash(), starforth_word_random(), starforth_word_seed(), starforth_word_top_words(), starforth_word_wait(), starforth_word_zuse_pubkey_fetch(), string_word_blank(), string_word_bracket_tick(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_count(), string_word_expect(), string_word_minus_trailing(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), system_word_reboot(), test_acl_adaptive_ttl(), test_acl_strict_mode(), vblk_io(), verify_mint(), vm_align(),

vm_allot(), vm_assert_interpreter_enabled(), vm_bootstrap_root_vocabulary(), vm_bootstrap_scr(), vm_compile_call(), vm_compile_exit(), vm_compile_literal(), vm_compile_word(), vm_create_word(), vm_dictionary_track_entry(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_get_block_addr(), vm_heartbeat_start_thread(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_interpret_word(), vm_load_cell(), vm_load_u8(), vm_make_immediate(), vm_pin_execution_heat(), vm_pop(), vm_push(), vm_repl(), vm_smudge_word(), vm_store_cell(), vm_store_u8(), vm_tick_inference_engine(), vm_tick_slope_validator(), vm_tick_window_tuner(), vm_unpin_execution_heat(), vocabulary_create_vocabulary_direct(), vocabulary_select_runtime(), vocabulary_word_definitions(), vocabulary_word_forth(), vocabulary_word_vocabulary(), vt100_enable_ttf(), vt100_init(), word_defer(), word_defer_fetch(), word_divide(), word_divmod(), word_is(), xhci_bot_read_data_in(), xhci_bot_receive_csw(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_data_out(), xhci_bringup(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), xhci_cmd_configure_hid_endpoint(), xhci_cmd_disable_slot(), xhci_cmd_enable_slot(), xhci_cmd_reset_endpoint(), xhci_cmd_set_tr_dequeue_pointer(), xhci_drain_pending_event(), xhci_ep0_bot_mass_storage_reset(), xhci_ep0_clear_endpoint_ha
xhci_ep0_get_config_descriptor(), xhci_ep0_get_device_descriptor(), xhci_ep0_hid_set_protocol(), xhci_ep0_set_configuration(), xhci_find_and_map(), xhci_handle_bulk_stall(), xhci_handle_port_connected(), xhci_log_hex32(), xhci_poll_one(), xhci_queue_pending_event(), xhci_stall_fail(), and xhci_take_from_bios().

Here is the call graph for this function:



7.208.3.3 log_set_level()

```
void log_set_level (
            LogLevel level )
```

Set the global logging level.

Parameters

level	The new logging level to set

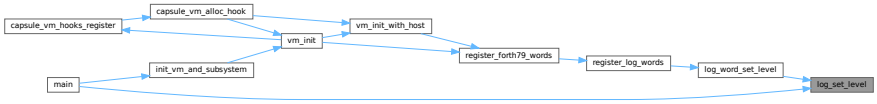
Note

Default level is LOG_INFO

References current_level, and level.

Referenced by log_word_set_level(), and main().

Here is the caller graph for this function:



7.208.3.4 log_set_vm()

```
void log_set_vm (
            struct VM * vm )
```

Set the VM instance for persistent logging.

Parameters

vm	VM instance to use for persistent logging (NULL to disable)

7.208.3.5 log_test_result()

```
void log_test_result (
            const char * word_name,
            TestResult result )
```

Log a test result with colored output.

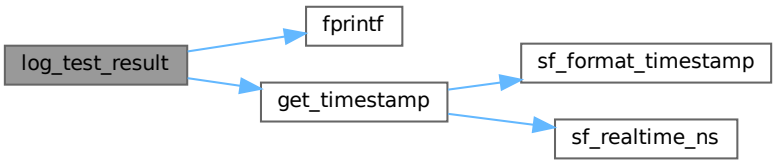
Parameters

word_name	Name of the word being tested
result	Result of the test execution

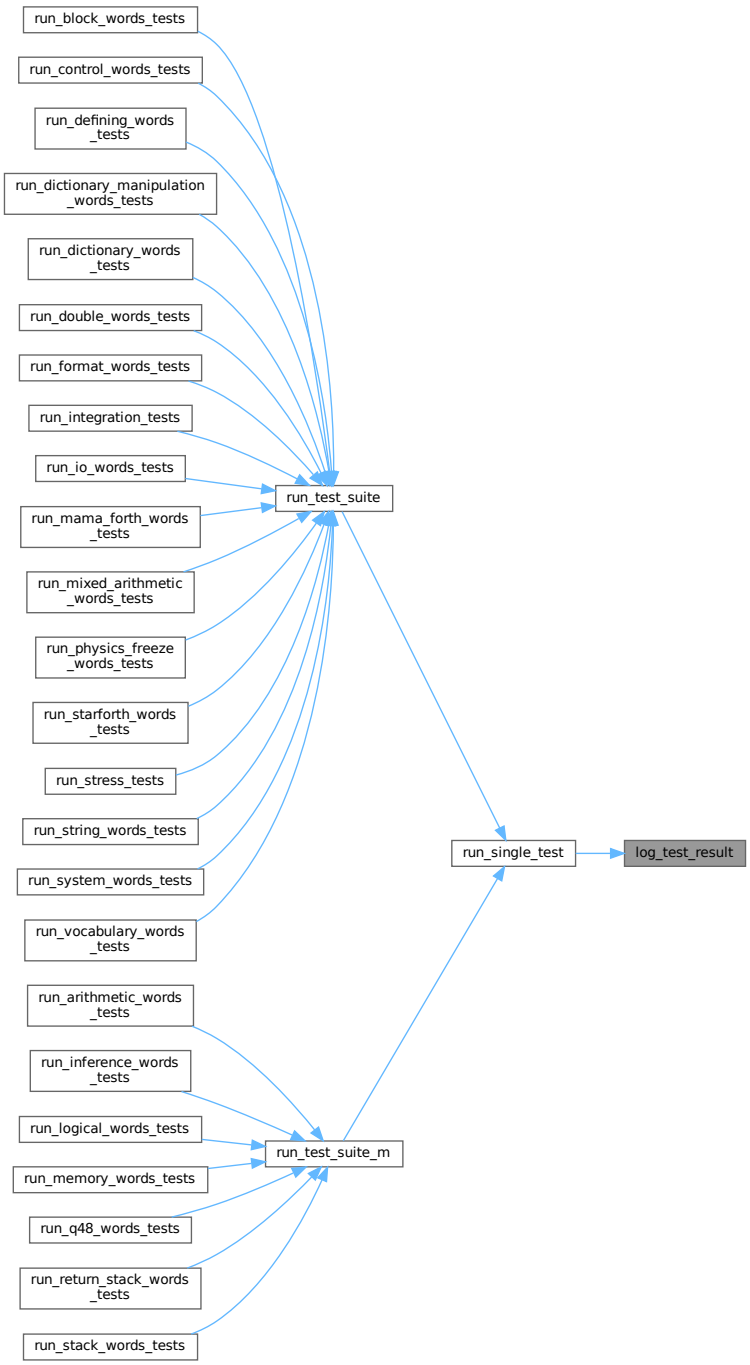
References color_reset, current_level, fprintf(), get_timestamp(), LOG_NONE, LOG_TEST, stderr, TEST_FAIL, test_fail_color, TEST_PASS, test_pass_color, TEST_SKIP, and test_skip_color.

Referenced by run_single_test().

Here is the call graph for this function:



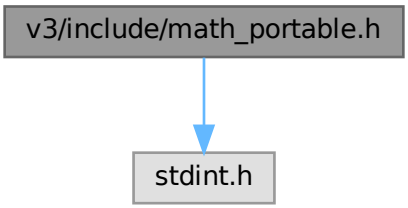
Here is the caller graph for this function:



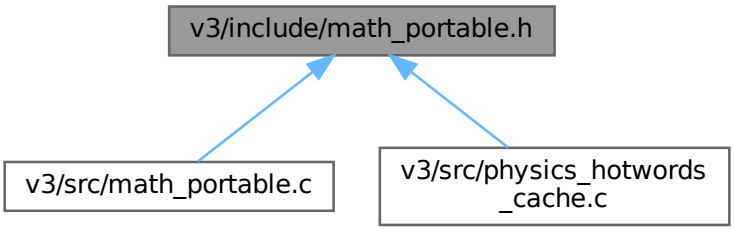
7.209 v3/include/math_portable.h File Reference

```
#include <stdint.h>
```

Include dependency graph for math_portable.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define Q48_SCALE (1LL << 16) /* 65536 */

Functions

- int64_t sqrt_q48 (int64_t x)

  Square root in Q48.16 fixed-point Input: x in Q48.16 format Output: sqrt(x) in Q48.16 format Uses Newton-Raphson: x_{n+1} = (x_n + a/x_n) / 2 Converges in ~10-15 iterations Returns 0 for negative inputs (guard against domain error)
- int64_t erf_q48 (int64_t x)

  Error function in Q48.16 fixed-point Input: x in Q48.16 format (dimensionless Z-score) Output: erf(x) in Q48.16 format (range: -Q48_SCALE to +Q48_SCALE) Uses Abramowitz & Stegun approximation (7.1.26) Maximum error: 1.5e-7 relative to floating-point erf Properties:

7.209.1 Macro Definition Documentation

7.209.1.1 Q48_SCALE

```
#define Q48_SCALE (1LL << 16) /* 65536 */
```

7.209.2 Function Documentation

7.209.2.1 erf_q48()

```
int64_t erf_q48 (
            int64_t x )
```

Error function in Q48.16 fixed-point Input: x in Q48.16 format (dimensionless Z-score) Output: erf(x) in Q48.↩ 16 format (range: -Q48_SCALE to +Q48_SCALE) Uses Abramowitz & Stegun approximation (7.1.26) Maximum error: 1.5e-7 relative to floating-point erf Properties:

- erf(0) = 0
- erf(-x) = -erf(x) (odd function)
- erf(x) approaches ±Q48_SCALE for large |x|

Error function in Q48.16 fixed-point Input: x in Q48.16 format (dimensionless Z-score) Output: erf(x) in Q48.↩ 16 format (range: -Q48_SCALE to +Q48_SCALE) Uses Abramowitz & Stegun approximation (7.1.26) Maximum error: 1.5e-7 relative to floating-point erf Properties:

Implements the rational approximation from Abramowitz & Stegun, formula 7.1.26:
`erf(x) ≈ sign(x) × sqrt(1 ? exp(?x² × (4/π + a·x²) / (1 + a·x²)))`

where a ≈ 0.147 (in Q48.16: 9633). The maximum error of the true mathematical approximation is $|\varepsilon| < 1.5\times10^{-}$?; the fixed-point encoding introduces additional rounding of approximately 1 ULP in Q48.16.

All arithmetic is performed in 64-bit integer Q48.16 format — no floating-point operations. The computation sequence:

1. Extract sign; work with |x|.
2. Compute x^2 with 8-bit prescaling to prevent overflow: x_sq = (x >> 8) * (x >> 8).
3. Evaluate the A&S numerator (4/π + a·x^2) and denominator (1 + a·x^2).
4. Compute ratio = numerator / denominator (shift to preserve Q48.16 scale).
5. Compute exp_arg = ?x^2 × ratio and evaluate exp_q48(exp_arg).
6. Compute base = 1 ? exp_val, clamp to [0, 1].
7. Return sign × sqrt_q48(base).

Q48.16 constants used:

- a_q48 = 9633 (0.147 × 65536)
- four_over_pi_q48 = 83328 (4/π × 65536 ≈ 1.2732 × 65536)

Consumed by hotwords_speedup_estimate() to compute P(speedup > 1.1×) and P(speedup > 2.0×) via the normal CDF identity:
`P(Z ≤ z) = 0.5 × (1 + erf(z / √·2))`

where √·2 in Q48.16 = 92681.

Parameters

x	Input value in Q48.16 format (dimensionless Z-score; 1.0 = 65536). May be negative; sign determines the sign of the result.

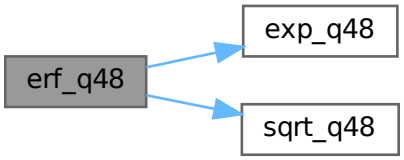
Returns

erf(x) in Q48.16 format, in the range [?65536, +65536] corresponding to (?1.0, +1.0).

References base, exp_q48(), and sqrt_q48().

Referenced by hotwords_speedup_estimate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.209.2.2 sqrt_q48()

```
int64_t sqrt_q48 (
            int64_t x )
```

Square root in Q48.16 fixed-point Input: x in Q48.16 format Output: sqrt(x) in Q48.16 format Uses Newton-↩ Raphson: x_{n+1} = (x_n + a/x_n) / 2 Converges in ∼10-15 iterations Returns 0 for negative inputs (guard against domain error)

Square root in Q48.16 fixed-point Input: x in Q48.16 format Output: sqrt(x) in Q48.16 format Uses Newton-↩ Raphson: x_{n+1} = (x_n + a/x_n) / 2 Converges in ∼10-15 iterations Returns 0 for negative inputs (guard against domain error)

Applies the Newton-Raphson recurrence:
$r_{n+1} = (r_n^2 + x) / (2 \times r_n)$

for up to 20 iterations, with all arithmetic scaled to the Q48.16 representation (1.0 = 65536). Intermediate products use 8-bit right shifts on both operands before squaring to prevent 64-bit overflow:
`guess_squared = (guess » 8) * (guess » 8)`

This sacrifices 16 bits of precision in the squared term, which is acceptable because the division by `2*guess` restores the correct order of magnitude. Convergence is declared when successive iterates differ by at most 1 in Q48.16 units (sub-nanosecond for timing uses).

Special cases handled without iteration:

- x $\leq$ 0 $\rightarrow$ returns 0 (no imaginary results)
- x == 65536 (1.0) $\rightarrow$ returns 65536 exactly

Initial guess: x/2 for x $>$ 1.0, else 1.0. Values above 1.0 converge from above; values in (0, 1.0) converge from the 1.0 initial guess.

Parameters

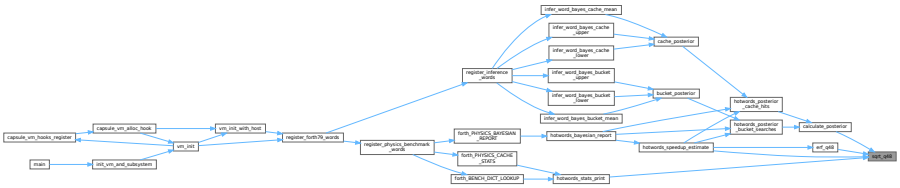
x	Non-negative value in Q48.16 format (1.0 = 65536). Negative values are treated as zero.

Returns

sqrt(x) in Q48.16 format; 0 for x $\leq$ 0.

Referenced by calculate_posterior(), erf_q48(), hotwords_speedup_estimate(), and hotwords_stats_print().

Here is the caller graph for this function:

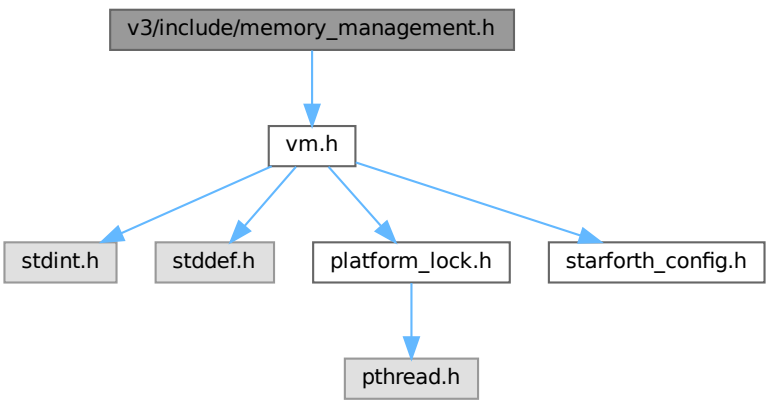


7.210 v3/include/memory_management.h File Reference

Memory management functions for the StarForth virtual machine.

```
#include "vm.h"
```

Include dependency graph for memory_management.h:



Functions

- void * vm_allot (VM *vm, size_t bytes)

  Allocates memory from the VM's dictionary space.
- void vm_align (VM *vm)

  Aligns the VM's dictionary pointer (here) to the cell boundary.

7.210.1 Detailed Description

Memory management functions for the StarForth virtual machine.

This file contains functions for managing memory allocation and alignment in the StarForth virtual machine implementation.

7.210.2 Function Documentation

7.210.2.1 vm_align()

```
void vm_align (
            VM * vm )
```

Aligns the VM's dictionary pointer (here) to the cell boundary.

Parameters

vm	Pointer to the VM instance

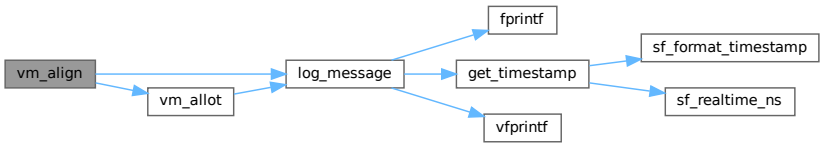
Note

Adds padding bytes if necessary to ensure proper alignment for cell_t type

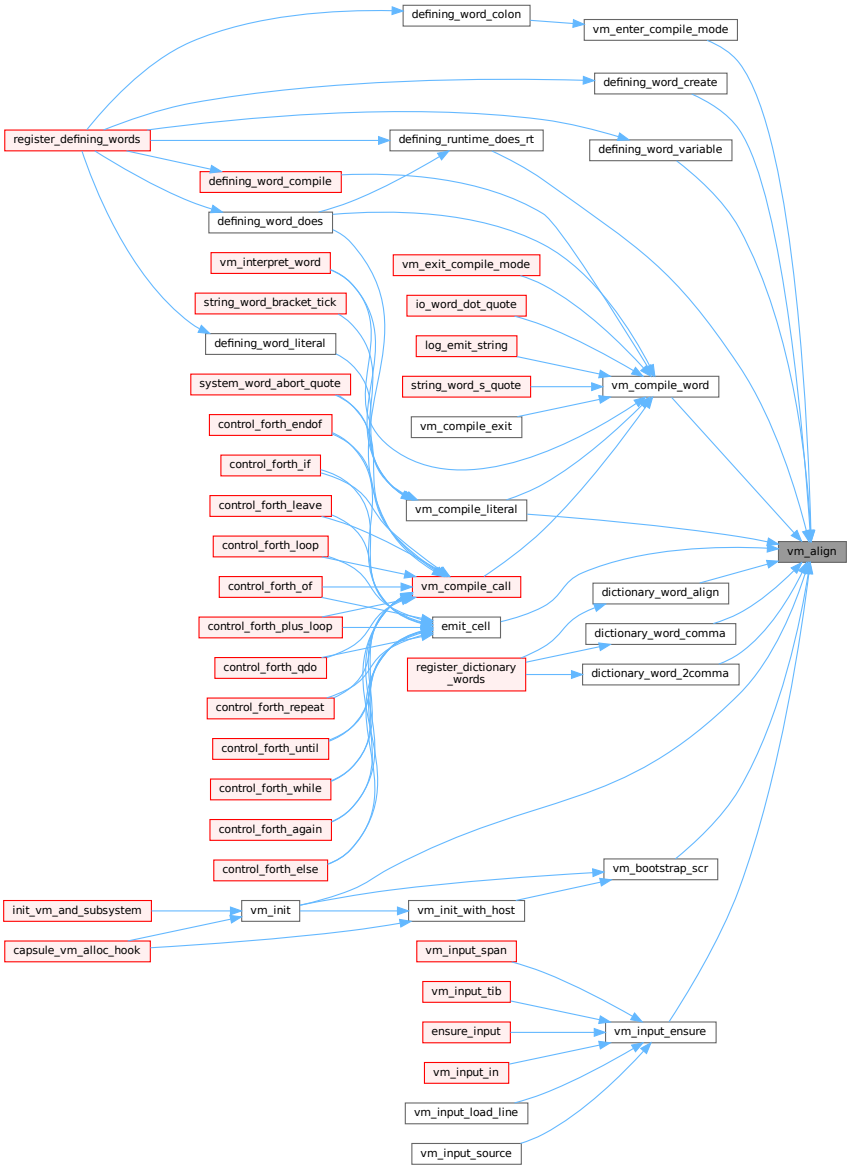
References bytes, VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), and vm_allot().

Referenced by defining_runtime_does_rt(), defining_word_create(), defining_word_variable(), dictionary_word_2comma(), dictionary_word_align(), dictionary_word_comma(), emit_cell(), vm_bootstrap_scr(), vm_compile_literal(), vm_compile_word(), vm_enter_compile_mode(), vm_init(), and vm_input_ensure().

Here is the call graph for this function:



Here is the caller graph for this function:



7.210.2.2 vm_allot()

```
void * vm_allot (
            VM * vm,
            size_t bytes )
```

Allocates memory from the VM's dictionary space.

Parameters

vm	Pointer to the VM instance
bytes	Number of bytes to allocate

Returns

Pointer to the allocated memory or NULL if allocation fails

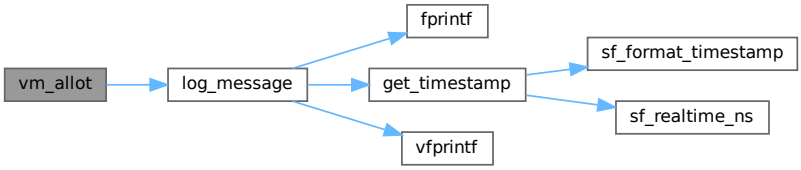
Note

The allocation is done from the VM's dictionary space which is limited to DICTIONARY_MEMORY_SIZE

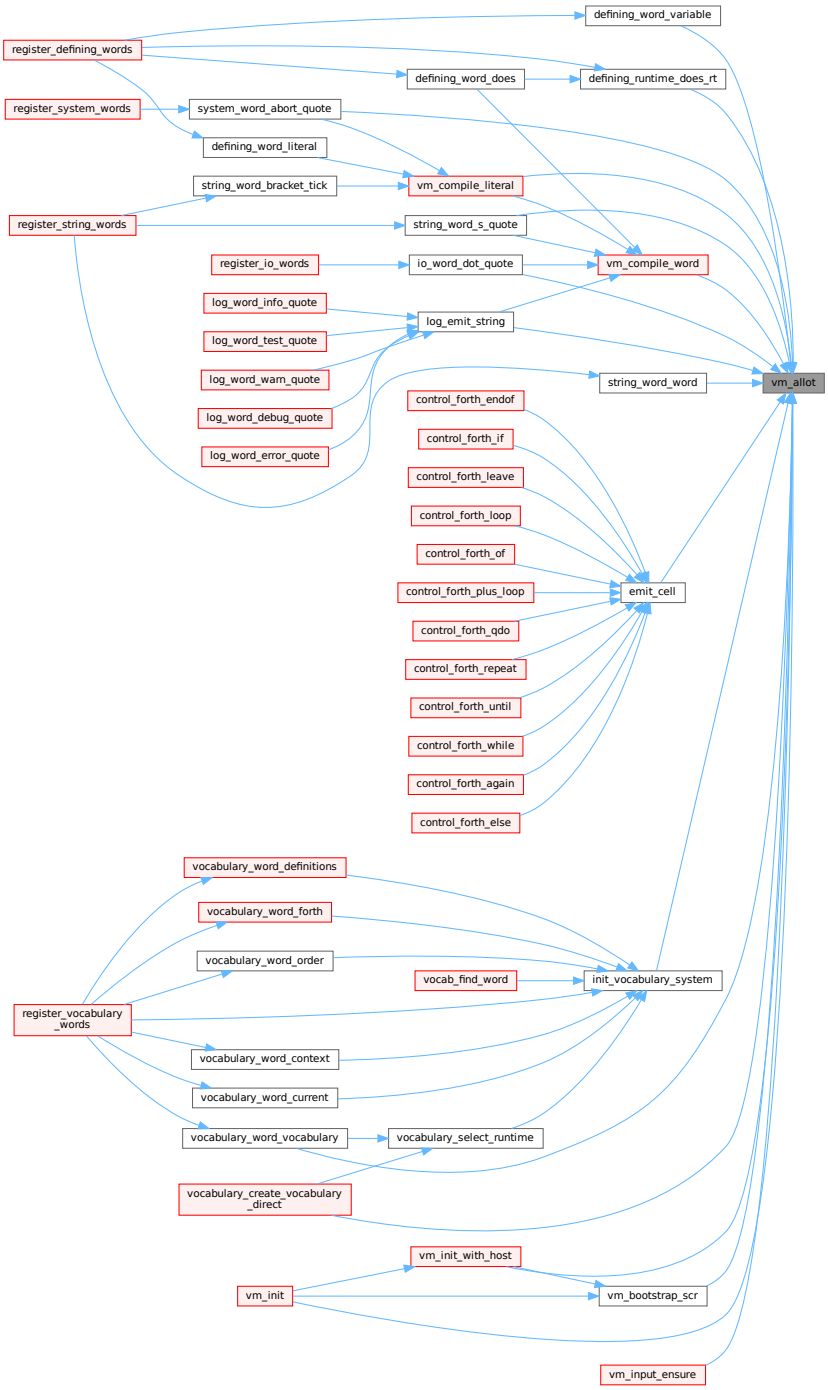
References BLOCK_SIZE, bytes, DICTIONARY_MEMORY_SIZE, VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), and VM::memory.

Referenced by defining_runtime_does_rt(), defining_word_variable(), emit_cell(), init_vocabulary_system(), io_word_dot_quote(), log_emit_string(), string_word_s_quote(), string_word_word(), system_word_abort_quote(), vm_bootstrap_scr(), vm_compile_literal(), vm_compile_word(), vm_init(), vm_init_with_host(), vm_input_ensure(), vocabulary_create_vocabulary_direct(), and vocabulary_word_vocabulary().

Here is the call graph for this function:



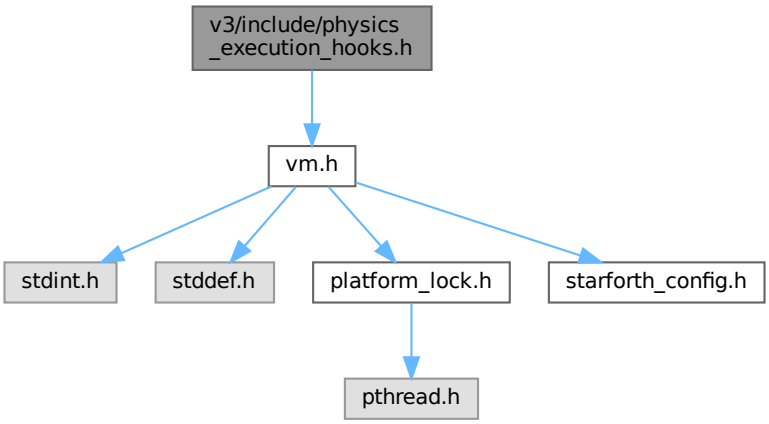
Here is the caller graph for this function:



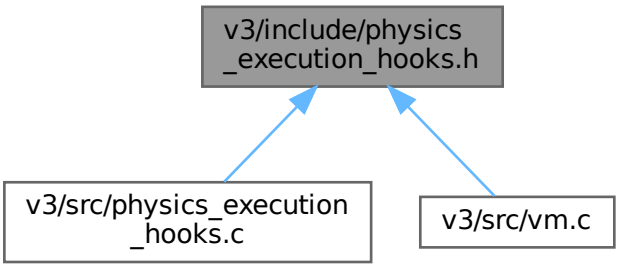
7.211 v3/include/physics_execution_hooks.h File Reference

```
#include "vm.h"
```

Include dependency graph for physics_execution_hooks.h:



This graph shows which files directly or indirectly include this file:



Functions

- void physics_pre_execute (VM *vm, DictEntry *word, DictEntry *prev)

  Pre-execution physics hook — called before every word dispatch.
- void physics_post_execute (VM *vm, DictEntry *word)

  Post-execution physics hook — called after every word dispatch.
- void physics_on_lookup (VM *vm, DictEntry *entry, DictEntry *canon)

  Physics hook for the outer interpreter word-lookup path.

7.211.1 Function Documentation

7.211.1.1 physics_on_lookup()

```
void physics_on_lookup (
            VM * vm,
```

```
        DictEntry * entry,
        DictEntry * canon )
```

Physics hook for the outer interpreter word-lookup path.

Called when the outer interpreter successfully finds a word by name, before executing it. Applies linear decay and increments heat on both the found entry and the canonical entry (if different), and updates `last_active_ns` / `last_decay_ns` on both. Acquires `vm->dict_lock` for the entire operation to ensure thread safety.

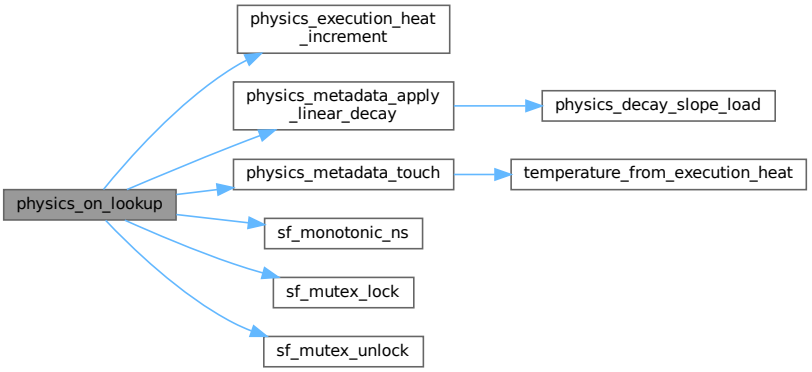
Parameters

vm	Pointer to the VM structure
entry	DictEntry found by name search
canon	Canonical DictEntry (may equal `entry` or be the smudge target)

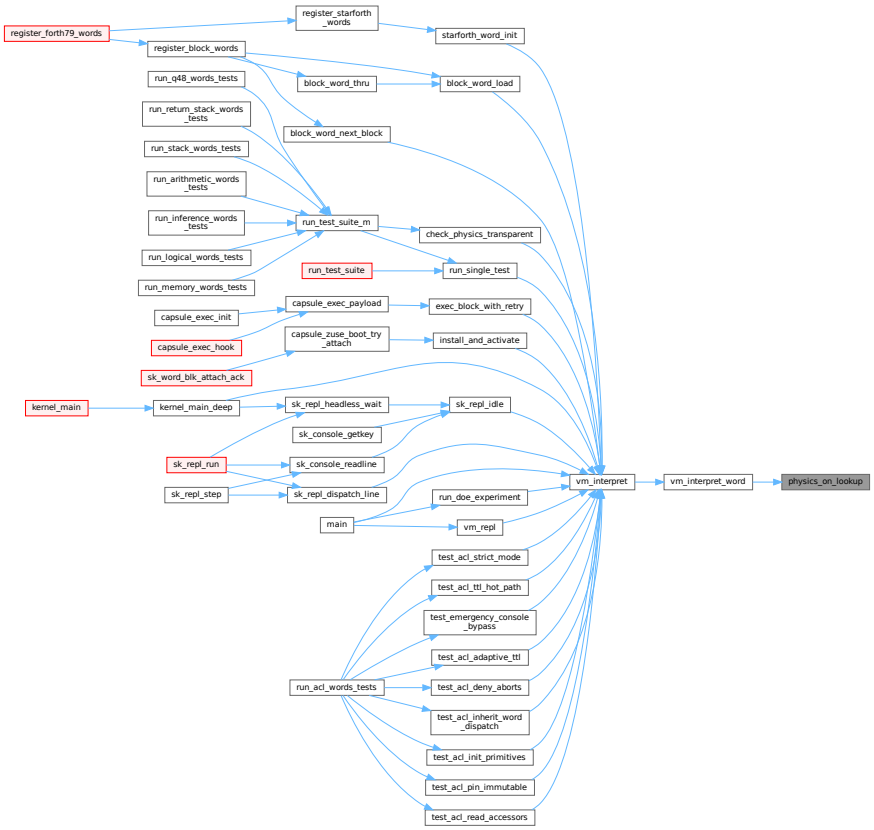
References VM::dict_lock, entry, DictEntry::execution_heat, VM::heartbeat, DictPhysics::last_active_ns, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, DictEntry::physics, physics_execution_heat_increment(), physics_metadata_apply_linear_decay(), physics_metadata_touch(), sf_monotonic_ns(), sf_mutex_lock(), sf_mutex_unlock(), and HeartbeatState::tick_count.

Referenced by vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.211.1.2 physics_post_execute()

```
void physics_post_execute (
            VM * vm,
            DictEntry * word )
```

Post-execution physics hook — called after every word dispatch.

Consolidates all physics work that must happen after a word completes:

- Updates physics metadata (`temperature_q8`, `last_active_ns`) via `physics_metadata_touch()`
- Notifies the profiler (`profiler_word_exit()`)
- Increments `vm->heartbeat.words_executed`
- When running without a background heartbeat thread, checks whether enough words have been executed to trigger a heartbeat cycle (`vm_heartbeat_run_cycle()`)

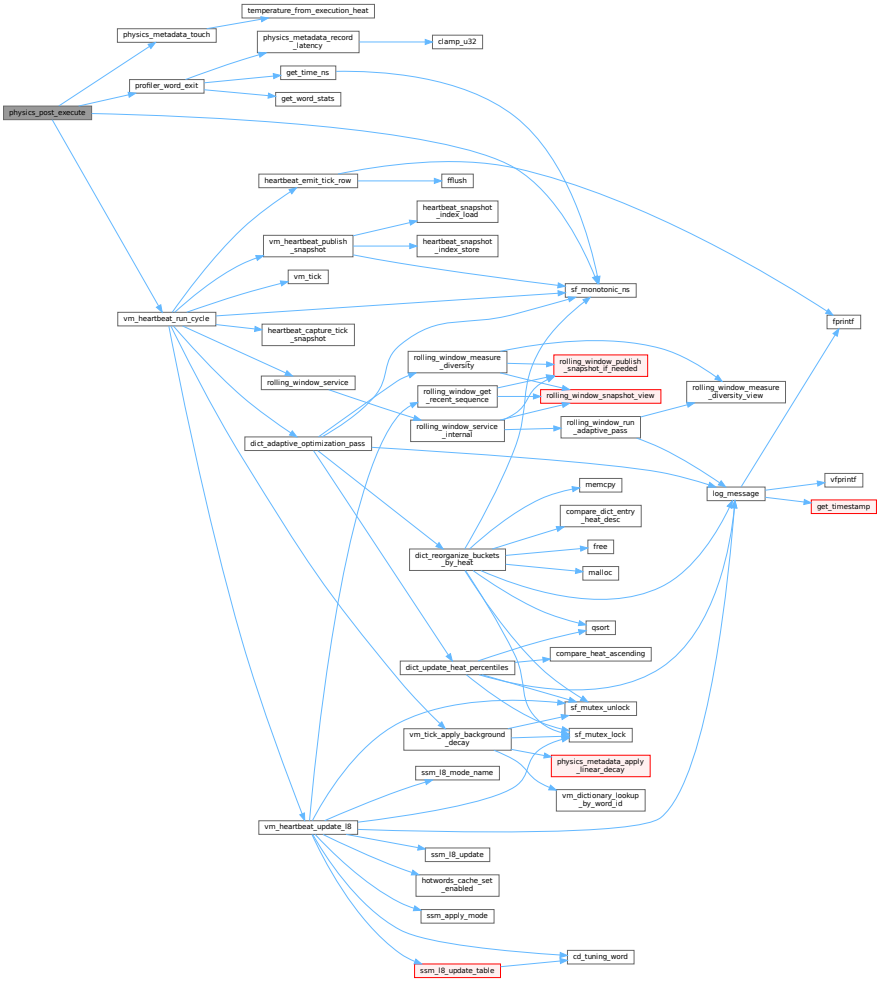
Parameters

vm	Pointer to the VM structure
word	DictEntry that just finished executing

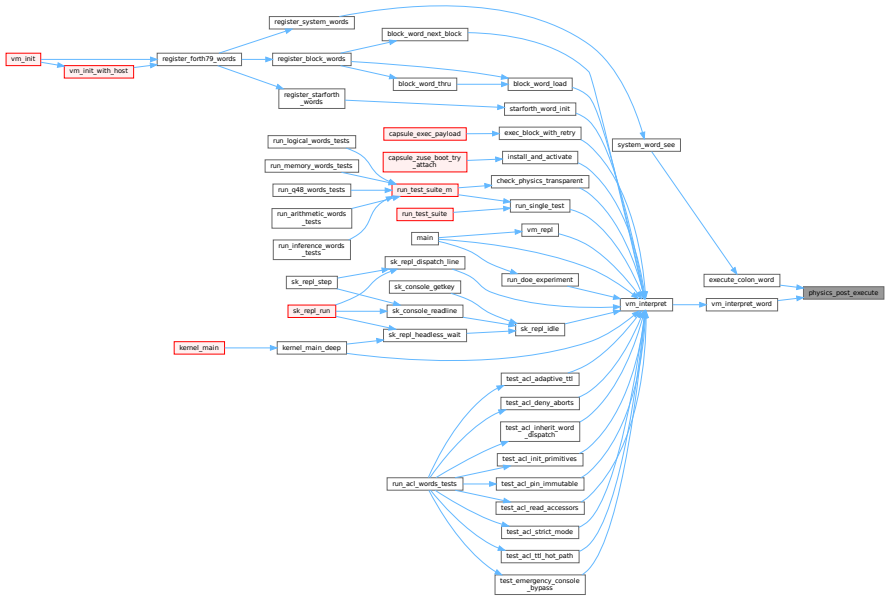
References HeartbeatState::check_counter, DictEntry::execution_heat, VM::heartbeat, HEARTBEAT_CHECK_FREQUENCY, physics_metadata_touch(), profiler_word_exit(), sf_monotonic_ns(), vm_heartbeat_run_cycle(), HeartbeatState::words_executed, and HeartbeatState::worker.

Referenced by execute_colon_word(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.211.1.3 physics_pre_execute()

```
void physics_pre_execute (
            VM * vm,
            DictEntry * word,
            DictEntry * prev )
```

Pre-execution physics hook — called before every word dispatch.

Consolidates all physics work that must happen before a word runs:

- Loop #3: applies linear decay based on elapsed time since last execution
- Loop #1: increments `execution_heat` for the word
- Loop #2: records the word-id into the rolling window of truth
- Loop #4: detects a pipelining prefetch hit (if `prev`'s speculation matched `word`), records the transition from `prev` to `word`, updates the probability cache, and issues a speculative hotwords cache promotion for the most likely next word

Parameters

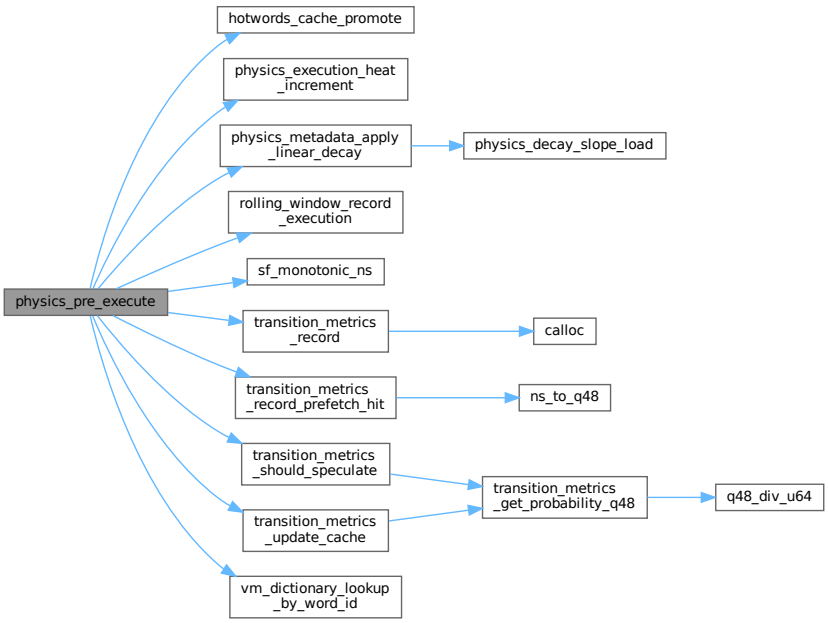
vm	Pointer to the VM structure
word	DictEntry being executed
prev	DictEntry executed immediately before `word` (may be NULL)

References DICTIONARY_SIZE, ENABLE_HOTWORDS_CACHE, ENABLE_PIPELINING, HotwordsCache_s::enabled, DictEntry::execution_heat, VM::heartbeat, VM::hotwords_cache, hotwords_cache_promote(), HOTWORDS_EXECUTION_HEAT_TH

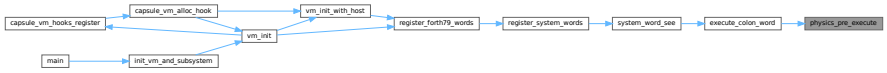
ssm_config_t::L4_pipelining, DictPhysics::last_active_ns, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, WordTransitionMetrics::most_likely_next_word_id, DictEntry::physics, physics_execution_heat_increment(), physics_metadata_apply_linear_decay(), VM::pipeline_metrics, WordTransitionMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::rolling_window, rolling_window_record_executio sf_monotonic_ns(), VM::ssm_config, HeartbeatState::tick_count, DictEntry::transition_metrics, transition_metrics_record(), transition_metrics_record_prefetch_hit(), transition_metrics_should_speculate(), transition_metrics_update_cache(), vm_dictionary_lookup_by_word_id(), word_id, and DictEntry::word_id.

Referenced by execute_colon_word().

Here is the call graph for this function:



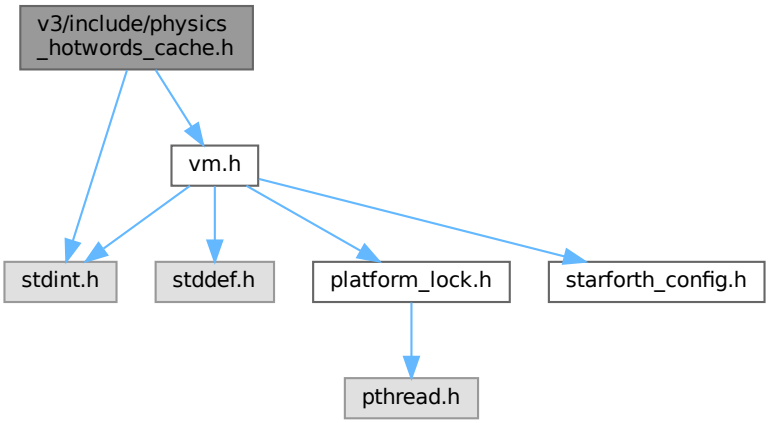
Here is the caller graph for this function:



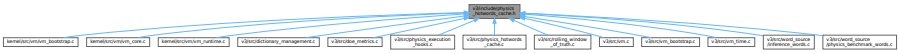
7.212 v3/include/physics_hotwords_cache.h File Reference

```
#include <stdint.h>
#include "vm.h"
```

Include dependency graph for physics_hotwords_cache.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct HotwordsStats
- struct HotwordsCache_s
- struct BayesianLatencyPosterior

  Bayesian statistics structure for inference and credible intervals All values stored in Q48.16 fixed-point (64-bit signed integer) representing nanoseconds with 2^-16 ≈ 0.0000153 ns precision.
- struct SpeedupEstimate

  Calculate speedup estimate with credible interval All values in Q48.16 fixed-point format (64-bit signed integer) Probabilities are in range [0, Q48_SCALE] representing [0%, 100%].

Macros

- #define HOTWORDS_CACHE_SIZE 32

  Maximum number of hot words to cache (typically 16-32)
- #define HOTWORDS_EXECUTION_HEAT_THRESHOLD 50

  Execution heat threshold to consider a word "hot" (execution count)
- #define HOTWORDS_EXECUTION_HEAT_DELTA_THRESHOLD 100

  Reorder bucket when execution heat delta exceeds this (avoid thrashing)

Typedefs

- typedef struct HotwordsCache_s HotwordsCache

Functions

- void hotwords_cache_init (HotwordsCache *cache)

  Initialize hot-words cache Should be called during VM initialization.
- void hotwords_cache_cleanup (HotwordsCache *cache)

  Cleanup hot-words cache.
- DictEntry * hotwords_cache_lookup (VM *vm, HotwordsCache *cache, DictEntry **bucket, size_t bucket_count, const char *name, size_t len)

  Lookup word in cache first, then fall back to bucket search Returns the found word, or NULL if not found.
- void hotwords_cache_promote (HotwordsCache *cache, DictEntry *word)

  Promote a word to the hot-words cache Called when a word's execution heat exceeds threshold.
- void hotwords_cache_evict_entry (HotwordsCache *cache, const DictEntry *word)

  Evict every cache slot holding this exact entry.
- void hotwords_cache_evict_name (HotwordsCache *cache, const char *name, size_t len)

  Evict every cache slot whose entry matches this name.
- void hotwords_bucket_reorder (DictEntry **bucket, size_t count)

  Reorder a bucket by execution heat (move hot words forward) Called when execution heat delta exceeds threshold.
- void hotwords_stats_reset (HotwordsStats *stats)

  Reset statistics (for before/after comparison)
- void hotwords_stats_print (const HotwordsStats *stats)

  Print statistics to stdout.
- void hotwords_cache_set_enabled (HotwordsCache *cache, int enabled)

  Enable/disable cache at runtime (for A/B testing)
- BayesianLatencyPosterior hotwords_posterior_cache_hits (const HotwordsStats *stats)

  Calculate posterior distribution for cache hits Uses accumulated samples to compute credible intervals.
- BayesianLatencyPosterior hotwords_posterior_bucket_searches (const HotwordsStats *stats)

  Calculate posterior distribution for bucket searches Uses accumulated samples to compute credible intervals.
- struct SpeedupEstimate hotwords_speedup_estimate (const HotwordsStats *stats)

  Calculate speedup estimate with uncertainty.
- void hotwords_bayesian_report (const HotwordsStats *stats_with_cache, const HotwordsStats *stats_without_cache)

  Generate detailed Bayesian inference report Compares cache vs bucket performance with confidence measures.

7.212.1 Macro Definition Documentation

7.212.1.1 HOTWORDS_CACHE_SIZE

```
#define HOTWORDS_CACHE_SIZE 32
```

Maximum number of hot words to cache (typically 16-32)

7.212.1.2 HOTWORDS_EXECUTION_HEAT_DELTA_THRESHOLD

```
#define HOTWORDS_EXECUTION_HEAT_DELTA_THRESHOLD 100
```

Reorder bucket when execution heat delta exceeds this (avoid thrashing)

7.212.1.3 HOTWORDS_EXECUTION_HEAT_THRESHOLD

```
#define HOTWORDS_EXECUTION_HEAT_THRESHOLD 50
```

Execution heat threshold to consider a word "hot" (execution count)

7.212.2 Typedef Documentation

7.212.2.1 HotwordsCache

```
typedef struct HotwordsCache_s HotwordsCache
```

7.212.3 Function Documentation

7.212.3.1 hotwords_bayesian_report()

```
void hotwords_bayesian_report (
            const HotwordsStats * stats_with_cache,
            const HotwordsStats * stats_without_cache )
```

Generate detailed Bayesian inference report Compares cache vs bucket performance with confidence measures.

Generate detailed Bayesian inference report Compares cache vs bucket performance with confidence measures.

Emits a structured Unicode report to `stdout` comparing the latency distributions of two measurement sets — one collected with the hot-words cache enabled, the other collected without (baseline). The report is divided into five sections:

1. **WITH CACHE ENABLED** — posterior mean, stddev, and 95%/99% credible intervals for cache-hits and bucket-searches separately, via `hotwords_posterior_cache_hits()` and `hotwords_posterior_bucket_sea`
2. **WITHOUT CACHE (Baseline)** — posterior mean, stddev, and 95% credible interval for all lookups (treated as bucket searches). If no bucket data is available in `stats_without_cache`, the bucket-search mean from the with-cache run is used as an approximation.
3. **SPEEDUP ESTIMATE (Bucket vs Cache)** — `hotwords_speedup_estimate()` point estimate, 95% credible interval, P(>1.1×), and P(>2.0×).
4. **CACHE EFFECTIVENESS** — total lookups, cache-hit rate, bucket-hit rate, and miss rate as percentages of total.
5. **OVERALL PERFORMANCE** — weighted-average latency with cache vs. baseline all-bucket latency, overall speedup, and absolute time saved per lookup. The weighted average is: (N_cache×avg_cache + N_↩ bucket×avg_bucket) / (N_cache + N_bucket).

All latency values are displayed in nanoseconds (floating-point via `q48_to_ns_double()`) for readability; underlying accumulators are Q48.16 integer throughout.

No-op if either `stats_with_cache` or `stats_without_cache` is NULL.

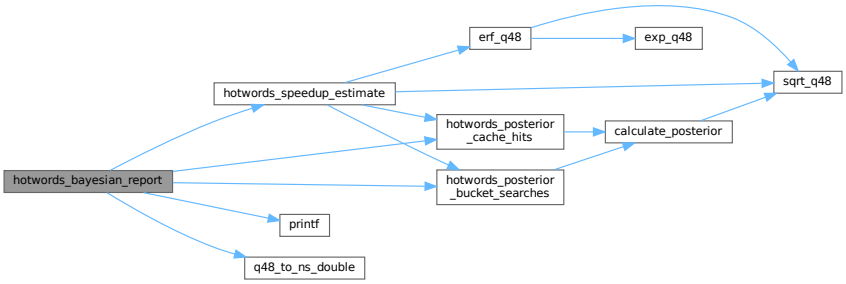
Parameters

stats_with_cache	Statistics from a run with cache enabled.
stats_without_cache	Statistics from a baseline run without cache.

References HotwordsStats::bucket_hits, HotwordsStats::cache_hits, BayesianLatencyPosterior::credible_lower_95, SpeedupEstimate::credible_lower_95_q48, BayesianLatencyPosterior::credible_lower_99, BayesianLatencyPosterior::credible_upper SpeedupEstimate::credible_upper_95_q48, BayesianLatencyPosterior::credible_upper_99, hotwords_posterior_bucket_searches(), hotwords_posterior_cache_hits(), hotwords_speedup_estimate(), BayesianLatencyPosterior::mean_ns_q48, HotwordsStats::misses, printf(), SpeedupEstimate::probability_gt_10pct_q48, SpeedupEstimate::probability_gt_double_q48, q48_to_ns_double(), BayesianLatencyPosterior::sample_count, SpeedupEstimate::speedup_factor_q48, and BayesianLatencyPosterior::stddev_ns_q48.

Referenced by forth_PHYSICS_BAYESIAN_REPORT().

Here is the call graph for this function:



Here is the caller graph for this function:

7.212.3.2 hotwords_bucket_reorder()

```
void hotwords_bucket_reorder (
            DictEntry ** bucket,
            size_t count )
```

Reorder a bucket by execution heat (move hot words forward) Called when execution heat delta exceeds threshold.

Reorder a bucket by execution heat (move hot words forward) Called when execution heat delta exceeds threshold.

Performs a classic in-place bubble sort over the `bucket` array so that words with higher `execution_heat` appear at lower array indices. The hot-words cache and heat-stratified lookup both benefit from having the hottest words at the front of each bucket: the first-character hash buckets are linear arrays, and placing hot words up front reduces the scan depth on Stage 1 cache misses that fall through to Stage 2 bucket search.

Bubble sort is chosen deliberately: buckets are small (typically single digits to low dozens of words per first-character class), already nearly sorted after the previous reorganization pass, and the cost of a O(n^2) inner loop on $n \leq 30$ is negligible compared to the outer `qsort` on all 256 buckets in `dict_reorganize_buckets_by_heat()`. For the individual per-bucket call path (e.g. after a single promotion), bubble sort avoids the `malloc` overhead that `qsort's` temporary storage would require.

No-op for `count` < 2.

Parameters

bucket	Array of `DictEntry*` pointers to sort in-place. NULL-check is the caller's responsibility.
count	Number of valid entries in `bucket`.

References count, and execution_heat.

7.212.3.3 hotwords_cache_cleanup()

```
void hotwords_cache_cleanup (
            HotwordsCache * cache )
```

Cleanup hot-words cache.

Cleanup hot-words cache.

Calls `memset(cache, 0, sizeof(*cache))` to scrub every field including the inline `cache`[] pointer array and all statistics. The cache holds no separately allocated memory (the `DictEntry` pointers inside `cache`[] are owned by the dictionary, not the cache), so no `free()` calls are needed.

Intended for use at VM teardown or when the caller wants to reset the cache to a known-zero state between benchmark runs. After this call the cache is in an unusable state (`enabled` = 0); call `hotwords_cache_init()` to reinitialise it.

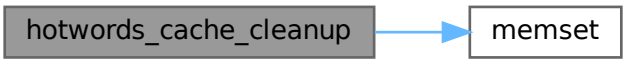
Parameters

cache	Cache to zero; no-op if NULL.

References memset().

Referenced by vm_cleanup().

Here is the call graph for this function:



Here is the caller graph for this function:

7.212.3.4 hotwords_cache_evict_entry()

```
void hotwords_cache_evict_entry (
            HotwordsCache * cache,
            const DictEntry * word )
```

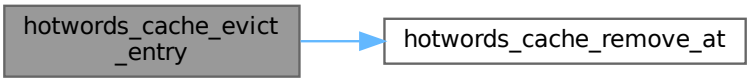
Evict every cache slot holding this exact entry.

Must be called before an entry is freed (FORGET) — a stale pointer in the ring is a use-after-free.

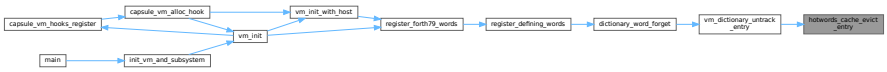
References HotwordsCache_s::cache, HotwordsCache_s::cache_count, and hotwords_cache_remove_at().

Referenced by vm_dictionary_untrack_entry().

Here is the call graph for this function:



Here is the caller graph for this function:



7.212.3.5 hotwords_cache_evict_name()

```
void hotwords_cache_evict_name (
            HotwordsCache * cache,
            const char * name,
            size_t len )
```

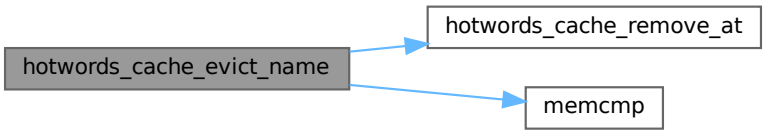
Evict every cache slot whose entry matches this name.

Called whenever name resolution may change: word creation (shadowing), smudge/unsmudge, hide. The next lookup re-resolves through the bucket.

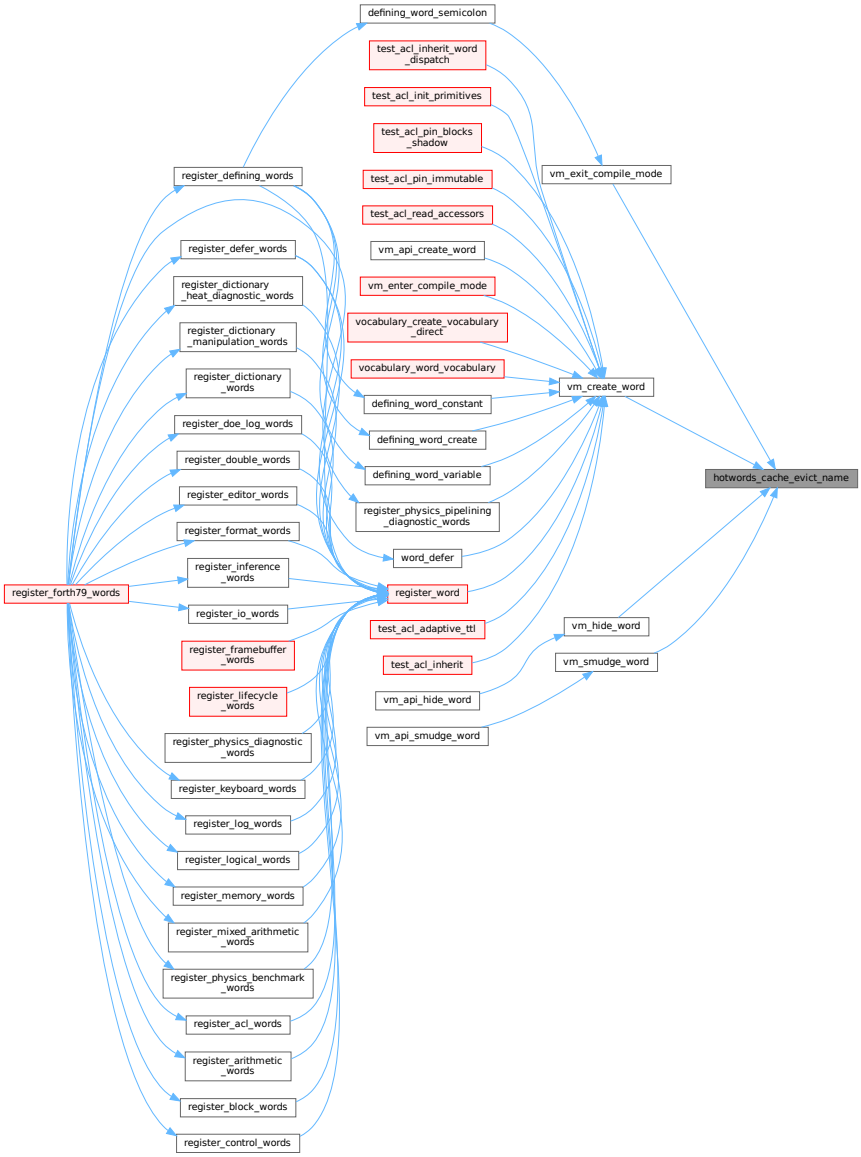
References HotwordsCache_s::cache, HotwordsCache_s::cache_count, hotwords_cache_remove_at(), memcmp(), name, DictEntry::name, and DictEntry::name_len.

Referenced by vm_create_word(), vm_exit_compile_mode(), vm_hide_word(), and vm_smudge_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.212.3.6 hotwords_cache_init()

```
void hotwords_cache_init (
            HotwordsCache * cache )
```

Initialize hot-words cache Should be called during VM initialization.

Initialize hot-words cache Should be called during VM initialization.

Zeroes all fields via `memset` then explicitly sets:

- `cache->enabled` = `ENABLE_HOTWORDS_CACHE` (build-time compile flag; 0 disables the cache globally without a code change).
- `cache->cache_count` = 0, `cache->lru_index` = 0 — empty round-robin eviction ring.
- All counter fields in `cache->stats` to 0.
- `stats.min_cache_hit_ns` = `INT64_MAX`, `stats.max_cache_hit_ns` = `INT64_MIN` (sentinel values; updated to real observations on the first cache hit).
- `stats.min_bucket_search_ns` = `INT64_MAX`, `stats.max_bucket_search_ns` = `INT64_MIN` (same sentinel convention for bucket searches).
- All Q48.16 variance accumulators to 0.

Safe to call multiple times (idempotent via `memset`). Must be called before the first `hotwords_cache_lookup()` invocation; failure to do so leaves the sentinel min/max fields uninitialised, producing nonsense in the Bayesian posterior report.

Parameters

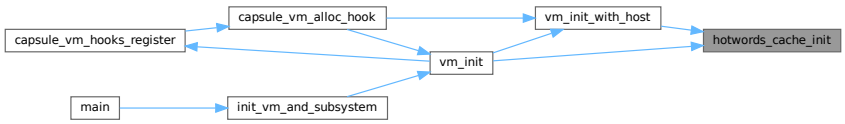
cache	Cache to initialise; no-op if NULL.

References HotwordsStats::bucket_hits, HotwordsStats::bucket_reorders, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsCache_s::cache_count, HotwordsStats::cache_hit_samples, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, HotwordsStats::cache_hits, ENABLE_HOTWORDS_CACHE, HotwordsCache_s::enabled, HotwordsStats::evictions, HotwordsCache_s::lru_index, HotwordsStats::max_bucket_search_ns, HotwordsStats::max_cache_hit_ns, memset(), HotwordsStats::min_bucket_search_ns, HotwordsStats::min_cache_hit_ns, HotwordsStats::misses, HotwordsStats::promotions, HotwordsCache_s::stats, and HotwordsStats::total_lookups.

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:

Here is the caller graph for this function:



7.212.3.7 hotwords_cache_lookup()

```
DictEntry * hotwords_cache_lookup (
            VM * vm,
            HotwordsCache * cache,
            DictEntry ** bucket,
            size_t bucket_count,
            const char * name,
            size_t len )
```

Lookup word in cache first, then fall back to bucket search Returns the found word, or NULL if not found.

This is the main function that replaces vm_find_word() logic. Takes the VM so the bucket fallback can resolve same-named entries by definition age (vm_dict_resolve_in_bucket) — bucket order alone is not authoritative once dict_reorganize_buckets_by_heat() has run.

Lookup word in cache first, then fall back to bucket search Returns the found word, or NULL if not found.

The primary dictionary fast-path. Records a start timestamp via `sf_monotonic_ns()` and then executes a two-stage search:

Stage 1 — Cache check (O(HOTWORDS_CACHE_SIZE)): Scans the `cache->cache[]` ring for an entry whose `name_len` and name match `name/len`, using the cheap-first filter (`name_len` → last-character → `memcmp`). On a hit the elapsed nanoseconds are converted to Q48.16 and accumulated into `stats.cache_hit_total_ns_q48`; the squared value is accumulated into `stats.cache_hit_variance_sum_q48` for Bayesian posterior computation by `calculate_posterior()`. Min/max latency sentinels are also updated.

Stage 2 — Bucket fallback (O(bucket_count)): Scans `bucket[]` in reverse-insertion order (newest definition wins). On a hit the elapsed time is recorded in the `bucket_search_*` accumulators with the same Q48.16 and variance treatment as Stage 1. If the found word's `execution_heat` exceeds `HOTWORDS_EXECUTION_HEAT_THRESHOLD` it is promoted into the cache via `hotwords_cache_promote()`.

Disabled path: If `cache` is NULL or `cache->enabled` is 0, the function skips Stage 1 and performs a bare reverse-linear scan of `bucket[]` returning the result immediately (no statistics are recorded).

A miss increments `stats.misses` and returns NULL.

Parameters

cache	Hot-words cache; may be NULL or disabled (falls back to bucket scan without statistics).
bucket	First-character hash bucket — array of `DictEntry*` pointers, searched in reverse order.
bucket_count	Number of valid pointers in `bucket`.
name	Word name to find (not NUL-terminated).
len	Length of `name` in bytes.

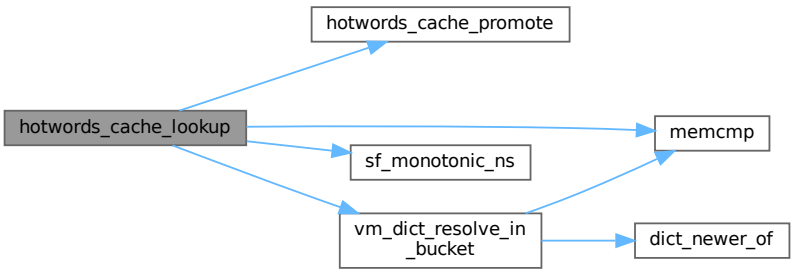
Returns

Matching `DictEntry*`, or NULL if not found.

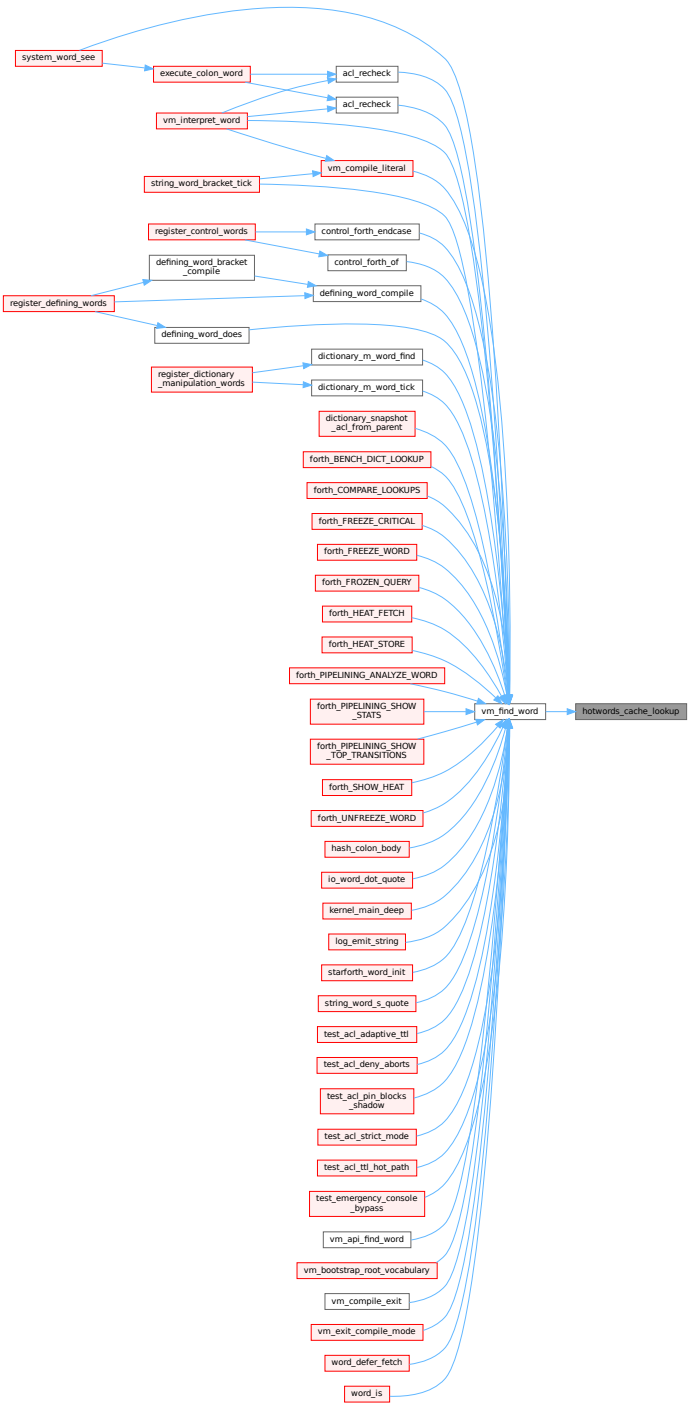
References HotwordsStats::bucket_hits, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsCache_s::cache, HotwordsCache_s::cache_count, HotwordsStats::cache_hit_samples, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, HotwordsStats::cache_hits, HotwordsCache_s::enabled, DictEntry::execution_heat, DictEntry::flags, hotwords_cache_promote(), HOTWORDS_EXECUTION_HEAT_THRESHOLD, HotwordsStats::max_bucket_search_ns, HotwordsStats::max_cache_hit_ns, memcmp(), HotwordsStats::min_bucket_search_ns, HotwordsStats::min_cache_hit_ns, HotwordsStats::misses, name, DictEntry::name, DictEntry::name_len, sf_monotonic_ns(), HotwordsCache_s::stats, HotwordsStats::total_lookups, vm_dict_resolve_in_bucket(), WORD_HIDDEN, and WORD_SMUDGED.

Referenced by vm_find_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.212.3.8 hotwords_cache_promote()

```
void hotwords_cache_promote (
            HotwordsCache * cache,
            DictEntry * word )
```

Promote a word to the hot-words cache Called when a word's execution heat exceeds threshold.

Promote a word to the hot-words cache Called when a word's execution heat exceeds threshold.

Two paths depending on whether the cache has a free slot:

- **Slot available** (`cache->cache_count < HOTWORDS_CACHE_SIZE`): Appends `word` at `cache->cache[cache->cache_count++]` after verifying it is not already present (linear scan). Increments `stats.promotions`.
- **Cache full** (`cache->cache_count >= HOTWORDS_CACHE_SIZE`): Overwrites `cache->cache[cache->lru_index]` with `word` (round-robin least-recently-used eviction) and advances `lru_index` modulo `HOTWORDS_CACHE_SIZE`. Increments `stats.evictions`. The evicted entry is not freed — it remains alive in the dictionary; only its slot in the cache ring is surrendered.

Called from `hotwords_cache_lookup()` whenever a bucket search finds a word hot enough to warrant caching (heat > `HOTWORDS_EXECUTION_HEAT_THRESHOLD`).

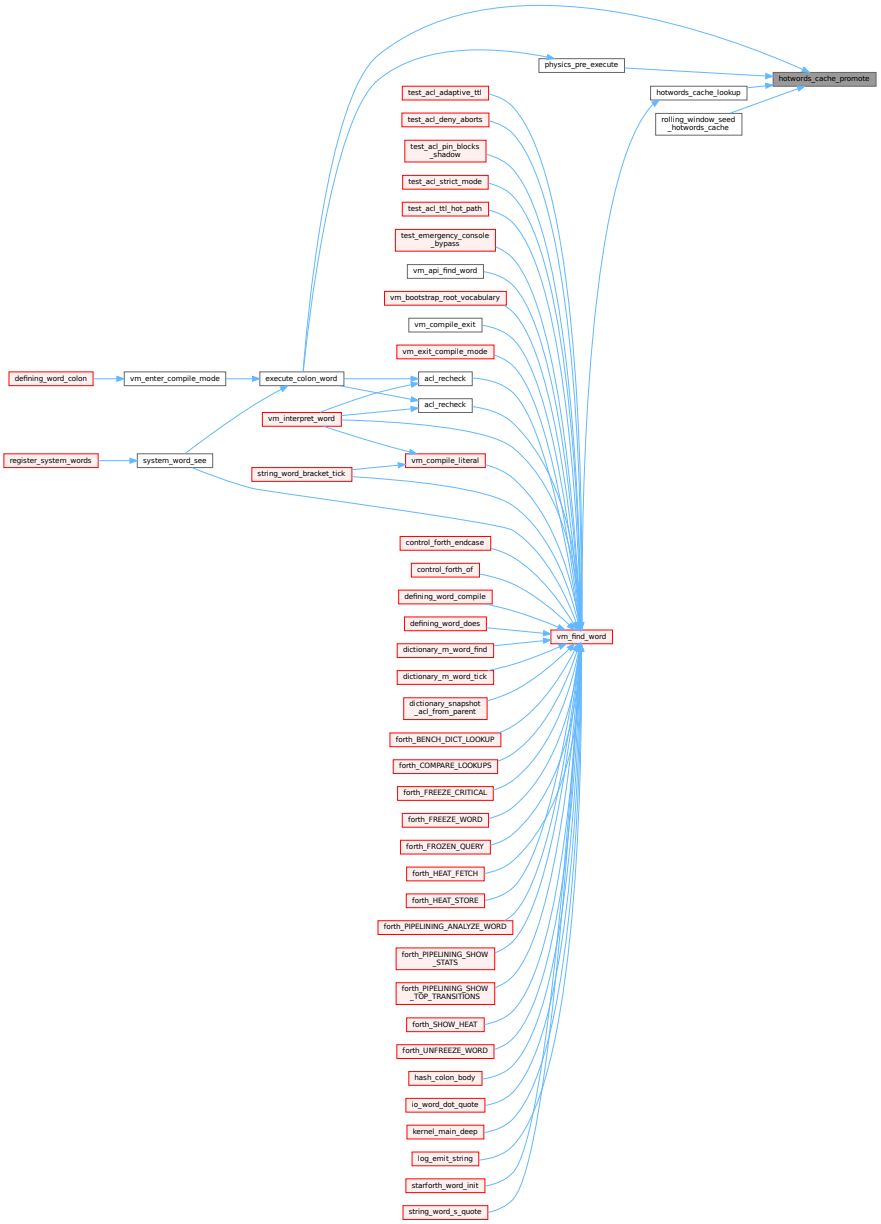
Parameters

cache	The cache to promote into; no-op if NULL.
word	The `DictEntry` to promote; no-op if NULL. Ownership is not transferred — the cache holds a borrowed pointer into the dictionary, which owns the entry.

References HotwordsCache_s::cache, HotwordsCache_s::cache_count, HotwordsStats::evictions, HOTWORDS_CACHE_SIZE, HotwordsCache_s::lru_index, HotwordsStats::promotions, and HotwordsCache_s::stats.

Referenced by execute_colon_word(), hotwords_cache_lookup(), physics_pre_execute(), and rolling_window_seed_hotwords_cache(

Here is the caller graph for this function:



7.212.3.9 hotwords_cache_set_enabled()

```
void hotwords_cache_set_enabled (
            HotwordsCache * cache,
            int enabled )
```

Enable/disable cache at runtime (for A/B testing)

Enable/disable cache at runtime (for A/B testing)

Sets `cache->enabled` to `enabled` (non-zero = active, 0 = bypassed). When disabled, `hotwords_cache_lookup()` skips Stage 1 entirely and falls through directly to a bare bucket scan, recording no statistics. This allows benchmark harnesses to compare cache-on vs. cache-off behaviour without rebuilding the binary.

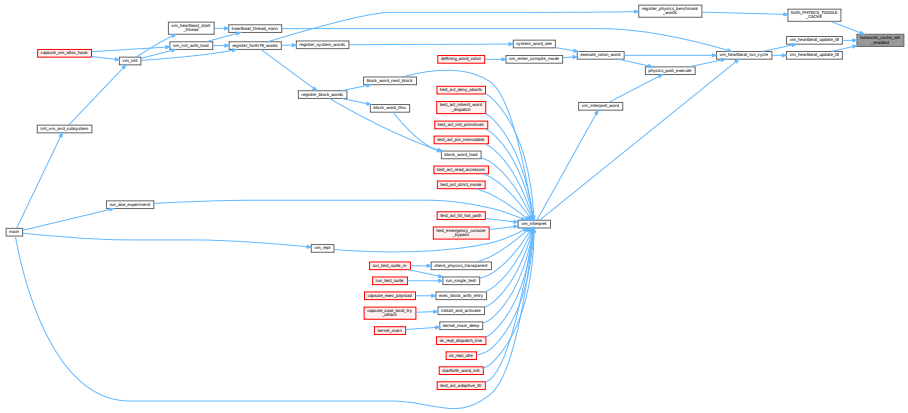
Parameters

cache	Cache to configure; no-op if NULL.
enabled	Non-zero to enable the cache; 0 to disable.

References HotwordsCache_s::enabled, and enabled.

Referenced by forth_PHYSICS_TOGGLE_CACHE(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the caller graph for this function:



7.212.3.10 hotwords_posterior_bucket_searches()

```
BayesianLatencyPosterior hotwords_posterior_bucket_searches (
            const HotwordsStats * stats )
```

Calculate posterior distribution for bucket searches Uses accumulated samples to compute credible intervals.

Calculate posterior distribution for bucket searches Uses accumulated samples to compute credible intervals.

Forwards the bucket-search accumulators from `stats` to `calculate_posterior()`: `bucket_search_↩total_ns_q48`, `bucket_search_variance_sum_q48`, `min_bucket_search_ns`, `max_bucket↩_search_ns`, and `bucket_search_samples`. Returns a zeroed `BayesianLatencyPosterior` if `stats` is NULL or no bucket-search samples have been recorded.

The returned posterior's mean and credible-interval fields feed `hotwords_speedup_estimate()` (as the slower distribution) and the baseline section of `hotwords_bayesian_report()`.

Parameters

stats	`HotwordsStats` containing bucket-search timing accumulators.

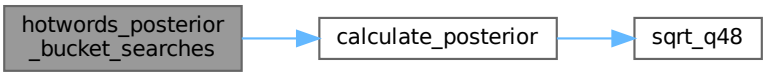
Returns

Posterior in Q48.16; all fields zero if no samples exist.

References HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_v
calculate_posterior(), HotwordsStats::max_bucket_search_ns, and HotwordsStats::min_bucket_search_ns.

Referenced by bucket_posterior(), hotwords_bayesian_report(), and hotwords_speedup_estimate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.212.3.11 hotwords_posterior_cache_hits()

```
BayesianLatencyPosterior hotwords_posterior_cache_hits (
            const HotwordsStats * stats )
```

Calculate posterior distribution for cache hits Uses accumulated samples to compute credible intervals.

Calculate posterior distribution for cache hits Uses accumulated samples to compute credible intervals.

Forwards the cache-hit accumulators from `stats` to `calculate_posterior()`: `cache_hit_total_ns_q48`, `cache_hit_variance_sum_q48`, `min_cache_hit_ns`, `max_cache_hit_ns`, and `cache_hit_samples`. Returns a zeroed `BayesianLatencyPosterior` if `stats` is NULL or no cache-hit samples have been recorded yet.

The returned posterior's `mean_ns_q48`, `stddev_ns_q48`, and 95%/99% credible-interval fields are consumed by `hotwords_speedup_estimate()` and `hotwords_bayesian_report()` to compare cache-hit latency against bucket-search latency.

Parameters

stats	`HotwordsStats` containing cache-hit timing accumulators.

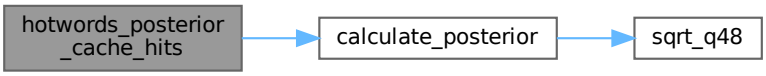
Returns

Posterior in Q48.16; all fields zero if no samples exist.

References HotwordsStats::cache_hit_samples, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q
calculate_posterior(), HotwordsStats::max_cache_hit_ns, and HotwordsStats::min_cache_hit_ns.

Referenced by cache_posterior(), hotwords_bayesian_report(), and hotwords_speedup_estimate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.212.3.12 hotwords_speedup_estimate()

```
struct SpeedupEstimate hotwords_speedup_estimate (
            const HotwordsStats * stats )
```

Calculate speedup estimate with uncertainty.

Calculate speedup estimate with uncertainty.

Derives a dimensionless speedup ratio `S` = bucket_mean / cache_mean in Q48.16 fixed-point and propagates uncertainty via the delta method for ratios:
`SE(S) ≈ S × sqrt( (SE_bucket / bucket_mean)² + (SE_cache / cache_mean)² )`

All arithmetic is Q48.16 integer throughout; the only deviation is the intermediate `sqrt()` call which also runs in Q48.16 via `sqrt_q48()`.

Computed fields in the returned `SpeedupEstimate` (all Q48.16):

- `speedup_factor_q48` — point estimate of `S`.
- `credible_lower_95_q48` / `credible_upper_95_q48` — 95% credible interval: `S` ± 1.96 × SE(S) (Z?? = 128431 in Q48.16). Lower bound clamped at 0.
- `probability_gt_10pct_q48` — P(speedup > 1.1×), computed as `0.5` × (1 + erf(z / √2)) where `z` = (1.1 ? S) / SE(S) is negated to give the right-tail probability and √2 = 92681 in Q48.16.
- `probability_gt_double_q48` — P(speedup > 2.0×), same formula with target = 2.0 × 65536 = 131072.

Returns a default `SpeedupEstimate` with `speedup_factor_q48` = 65536 (1.0×) if either distribution has zero samples or if `stats` is NULL.

Parameters

stats	`HotwordsStats` with both cache-hit and bucket-search samples.

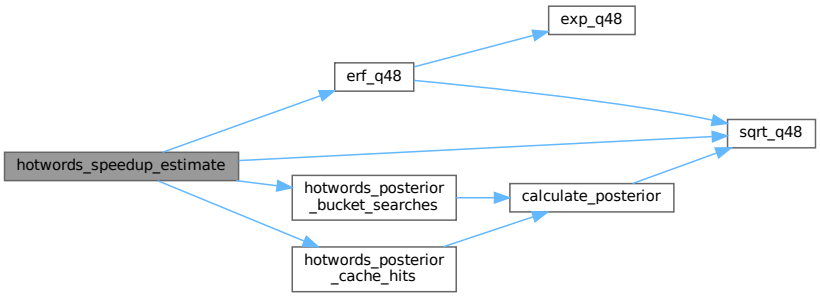
Returns

`SpeedupEstimate` with all fields in Q48.16 fixed-point.

References SpeedupEstimate::credible_lower_95_q48, SpeedupEstimate::credible_upper_95_q48, erf_q48(), hotwords_posterior_bucket_searches(), hotwords_posterior_cache_hits(), BayesianLatencyPosterior::mean_ns_q48, SpeedupEstimate::probability_gt_10pct_q48, SpeedupEstimate::probability_gt_double_q48, SpeedupEstimate::speedup_factor_q48, sqrt_q48(), and BayesianLatencyPosterior::stddev_ns_q48.

Referenced by hotwords_bayesian_report().

Here is the call graph for this function:



Here is the caller graph for this function:

7.212.3.13 hotwords_stats_print()

```
void hotwords_stats_print (
            const HotwordsStats * stats )
```

Print statistics to stdout.

Print statistics to stdout.

Emits a Unicode box-drawing table covering three sections:

LOOKUPS — total, cache hits, bucket hits, misses with percentages.

LATENCY STATISTICS — for each of cache-hits and bucket-searches (when their sample counts are non-zero):

- Min / avg / max in nanoseconds, derived from the Q48.16 accumulators by right-shifting 16 bits: `avg_ns` = `total_ns_q48` / samples >> 16.
- Standard deviation computed from the Welford-style variance accumulator: `var` = E[X^2] ? E[X]^2, then `stddev` = sqrt_q48(var), converted to nanoseconds via `q48_to_ns_double()` for display only.
- Speedup ratio (bucket avg / cache avg) when both sample sets are present.

CACHE MANAGEMENT — promotions, evictions, and bucket reorder counts.

If `stats` is NULL or the total lookup count is zero, prints a brief "No lookups performed." notice instead of the full table. All floating-point usage in this function is display-only (via `q48_to_ns_double()`); the underlying accumulators remain in Q48.16 integer format.

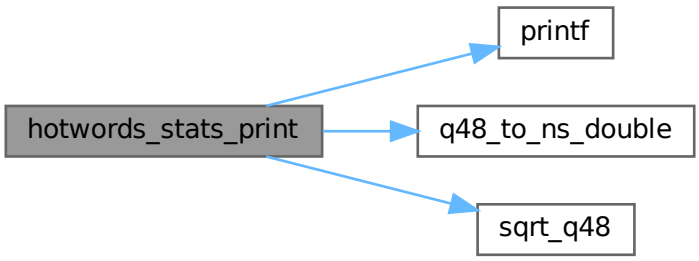
Parameters

stats	Statistics to report; no-op if NULL.

References HotwordsStats::bucket_hits, HotwordsStats::bucket_reorders, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsStats::cache_hit_samples, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, HotwordsStats::cache_hits, HotwordsStats::evictions, HotwordsStats::max_bucket_search_ns, HotwordsStats::max_cache_hit_ns, HotwordsStats::min_bucket_s, HotwordsStats::min_cache_hit_ns, HotwordsStats::misses, printf(), HotwordsStats::promotions, q48_to_ns_double(), and sqrt_q48().

Referenced by forth_BENCH_DICT_LOOKUP(), and forth_PHYSICS_CACHE_STATS().

Here is the call graph for this function:



Here is the caller graph for this function:

7.212.3.14 hotwords_stats_reset()

```
void hotwords_stats_reset (
            HotwordsStats * stats )
```

Reset statistics (for before/after comparison)

Reset statistics (for before/after comparison)

Resets every counter (`total_lookups`, `cache_hits`, `bucket_hits`, `misses`, `evictions`, `promotions`, `bucket_reorders`) to zero, resets all Q48.16 accumulator fields (`cache_hit_total_ns_q48`, `bucket_search_total_ns_q48`, `cache_hit_variance_sum_q48`, `bucket_search_variance_sum_q48`) to zero, resets sample counts to zero, and restores the min/max latency sentinels:

- `min_*_ns` = `INT64_MAX`
- `max_*_ns` = `INT64_MIN`

Intended for DoE benchmark runs that need a clean statistical baseline between treatment cells. The parent `HotwordsCache` also holds the `cache[]` pointer array and LRU state; those are not touched — only the `HotwordsStats` sub-struct is cleared.

Parameters

stats	Statistics structure to reset; no-op if NULL.

References HotwordsStats::bucket_hits, HotwordsStats::bucket_reorders, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsStats::cache_hit_samples HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, HotwordsStats::cache_hits, HotwordsStats::evictions, HotwordsStats::max_bucket_search_ns, HotwordsStats::max_cache_hit_ns, HotwordsStats::min_bucket_s HotwordsStats::min_cache_hit_ns, HotwordsStats::misses, HotwordsStats::promotions, and HotwordsStats::total_lookups.

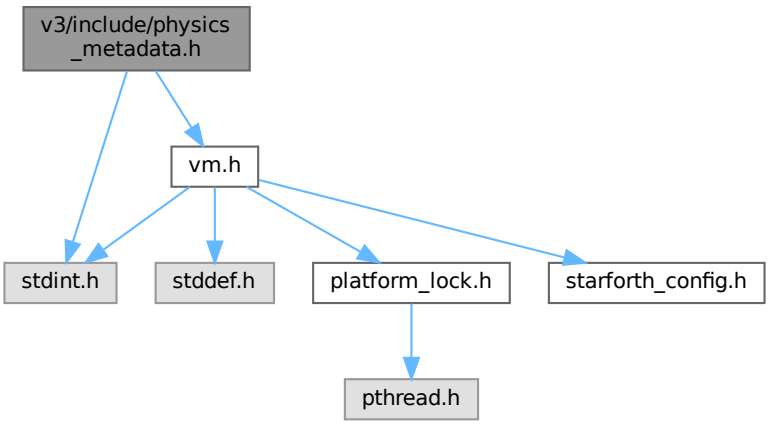
Referenced by forth_BENCH_DICT_LOOKUP(), and forth_PHYSICS_RESET_STATS().

Here is the caller graph for this function:

7.213 v3/include/physics_metadata.h File Reference

```
#include <stdint.h>
#include "vm.h"
```

Include dependency graph for physics_metadata.h:



This graph shows which files directly or indirectly include this file:

Functions

- static void physics_execution_heat_increment (DictEntry *entry)
- static cell_t physics_execution_heat_load (const DictEntry *entry)
- static uint64_t physics_decay_slope_load (const VM *vm)
- void physics_metadata_init (DictEntry *entry, uint32_t header_bytes)

  Initialise physics metadata for a newly registered DictEntry.
- void physics_metadata_set_mass (DictEntry *entry, uint32_t total_bytes)

  Update the mass (byte footprint) of a DictEntry's physics metadata.
- void physics_metadata_touch (DictEntry *entry, cell_t execution_heat, uint64_t now_ns)

  Update physics metadata after a word execution completes.
- void physics_metadata_refresh_state (DictEntry *entry)

  Re-derive and update `state_flags` for a DictEntry.
- void physics_metadata_record_latency (DictEntry *entry, uint64_t sample_ns)

  Record an execution latency sample into the per-word EMA.
- void physics_metadata_apply_seed (DictEntry *entry)

  Apply DoE-derived prior physics values to a named DictEntry.
- void physics_metadata_apply_linear_decay (DictEntry *entry, uint64_t elapsed_ticks, VM *vm)

  Apply Loop #3 linear heat decay to a DictEntry.

7.213.1 Function Documentation

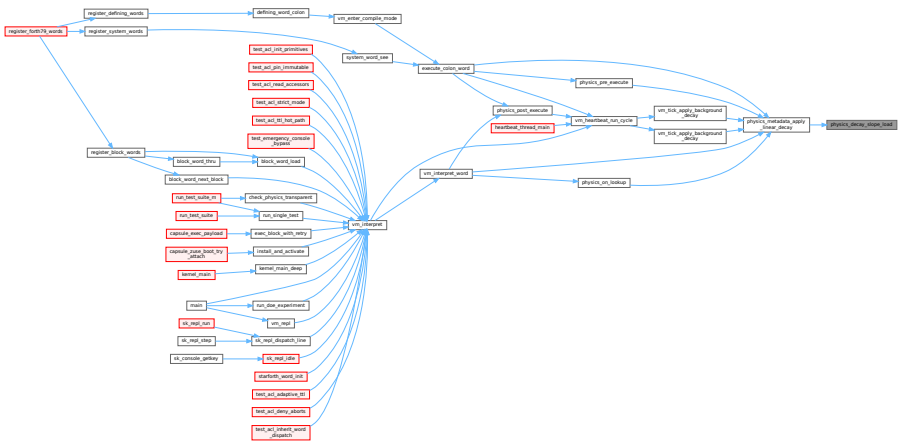
7.213.1.1 physics_decay_slope_load()

```
static uint64_t physics_decay_slope_load (
            const VM * vm )  [inline], [static]
```

References VM::decay_slope_q48.

Referenced by physics_metadata_apply_linear_decay().

Here is the caller graph for this function:



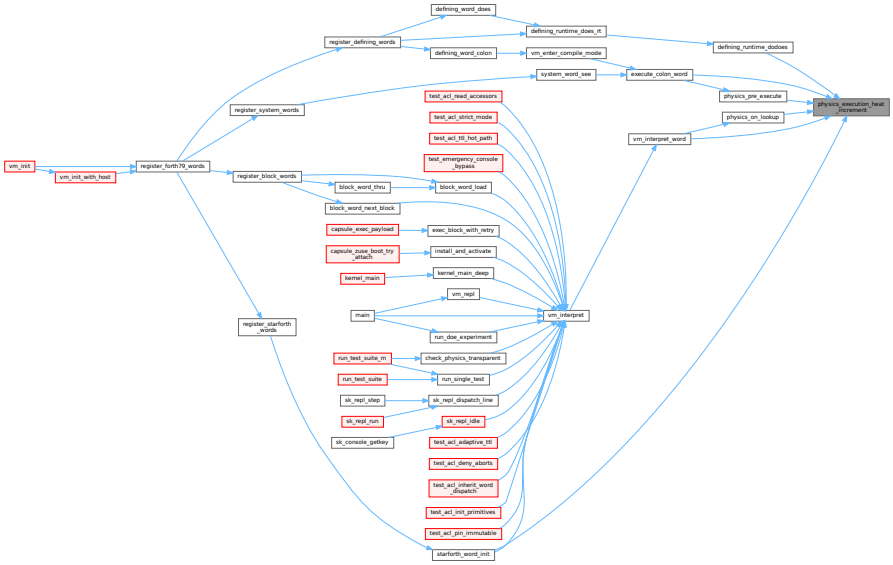
7.213.1.2 physics_execution_heat_increment()

```
static void physics_execution_heat_increment (
            DictEntry * entry )  [inline], [static]
```

References entry.

Referenced by defining_runtime_dodoes(), execute_colon_word(), physics_on_lookup(), physics_pre_execute(), starforth_word_init(), and vm_interpret_word().

Here is the caller graph for this function:



7.213.1.3 physics_execution_heat_load()

```
static cell_t physics_execution_heat_load (
            const DictEntry * entry )  [inline], [static]
```

References entry.

7.213.1.4 physics_metadata_apply_linear_decay()

```
void physics_metadata_apply_linear_decay (
            DictEntry * entry,
            uint64_t elapsed_ticks,
            VM * vm )
```

Apply Loop #3 linear heat decay to a DictEntry.

Implements the thermodynamic dissipation model: H(t) = max(0, H? - d∗t), where t is measured in heartbeat ticks (vm->heartbeat.tick_count), never in wall-clock time. The decay amount is computed as (elapsed_ticks ∗ slope_q48) >> 16 using the Q48.16 adaptive decay slope stored in the VM's heartbeat state. Uses a GCC CAS loop for atomic heat update when __GNUC__ is defined, falling back to a plain subtraction otherwise.

Ticks, not nanoseconds: this entry's `execution_heat` feeds directly into `dict_hash` (see `capsule_dict_hash_hook()`), so any wall-clock dependency here makes dictionary state – and therefore dict_hash – architecture-variable under QEMU/TCG's differing real emulation speed, or whenever real elapsed time between two touches of the same entry varies for reasons unrelated to execution (e.g. a blocking I/O wait between them). `vm->heartbeat.tick_count` only advances from `vm_tick()`, called synchronously from the interpreter's own dispatch path – a purely execution-count-driven clock, immune to how long any intervening real-time work (including blocking I/O) actually took.

No-op when:

- `WORD_FROZEN` flag is set on the entry
- `elapsed_ticks` == 0 (no heartbeat tick has fired since last decay)
- `vm->ssm_config` has L3 disabled by the Jacquard mode selector

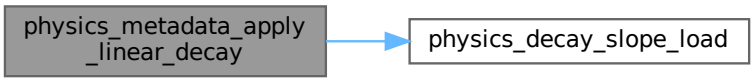
Parameters

entry	DictEntry whose heat is decayed
elapsed_ticks	Heartbeat ticks elapsed since last decay application
vm	VM providing the adaptive decay slope and L3 gate flag

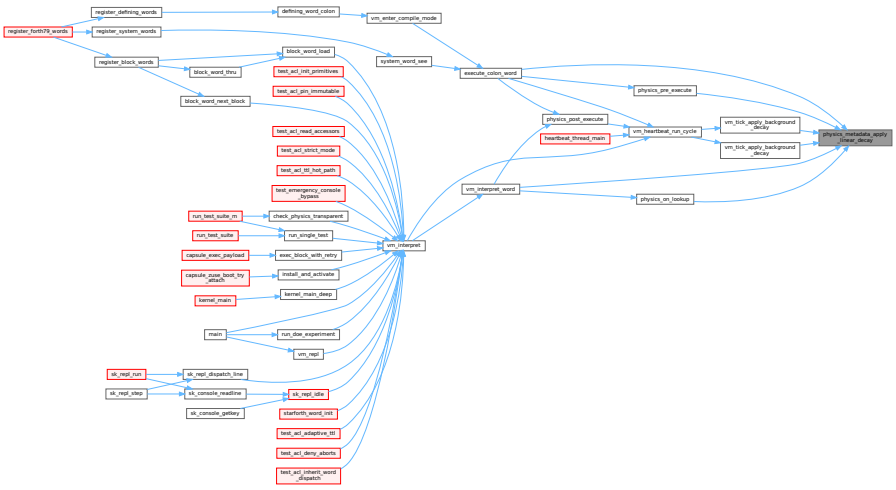
References entry, ssm_config_t::L3_linear_decay, physics_decay_slope_load(), VM::ssm_config, and WORD_FROZEN.

Referenced by execute_colon_word(), physics_on_lookup(), physics_pre_execute(), vm_interpret_word(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

Here is the call graph for this function:



Here is the caller graph for this function:



7.213.1.5 physics_metadata_apply_seed()

```
void physics_metadata_apply_seed (
            DictEntry * entry )
```

Apply DoE-derived prior physics values to a named DictEntry.

Searches the static `physics_seed_table` for an entry matching `entry->name`. If found, seeds `temperature_q8`, `avg_latency_ns`, `acl_hint`, and `pubsub_mask` with values derived from the 2^7 factorial DoE analysis. Only non-zero seed fields overwrite the current value. Called from `register_word()` for built-in primitives.

Parameters

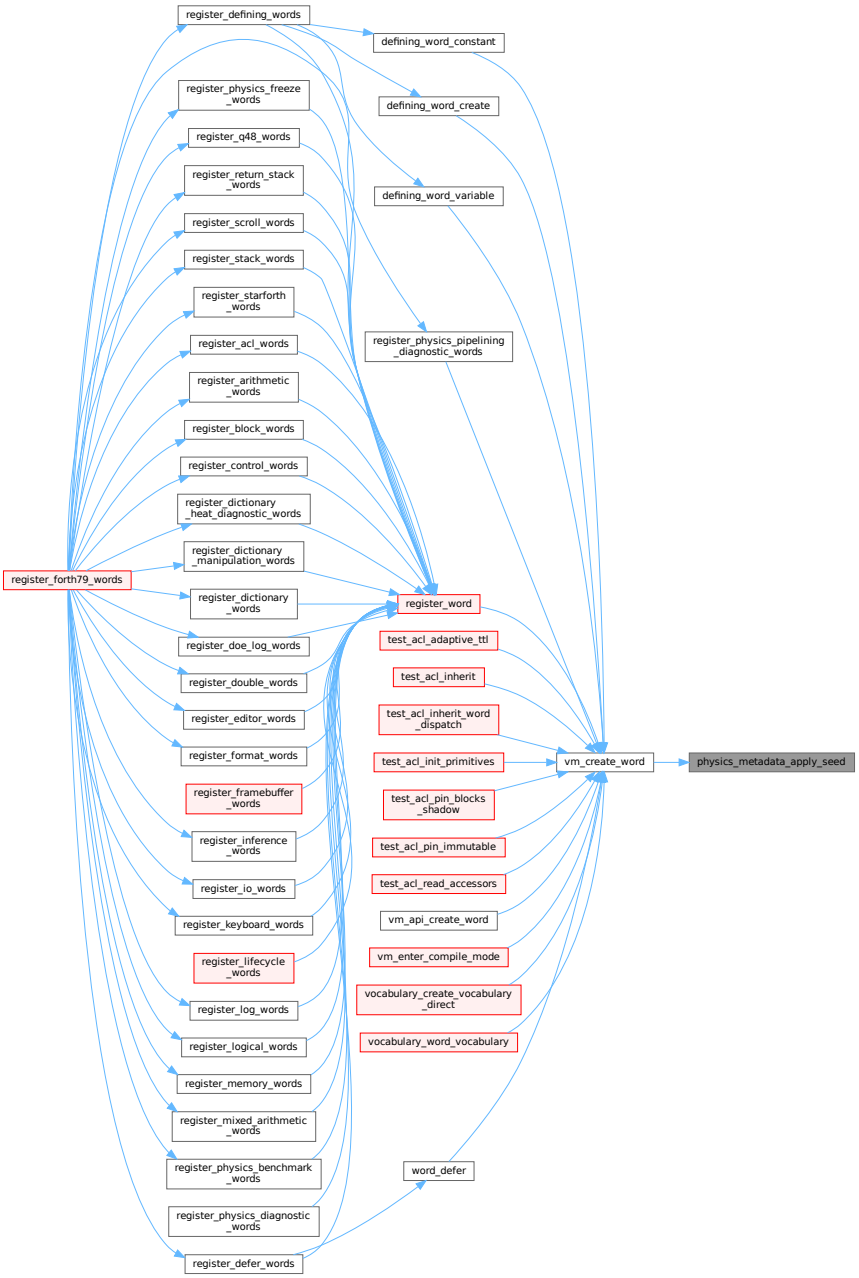
entry	DictEntry to seed (no-op if NULL or no matching table entry)

References entry, physics_seed_definition::name, physics_seed_table, strlen(), strncmp(), and physics_seed_definition::temperature_

Referenced by vm_create_word().

Here is the call graph for this function:

Here is the caller graph for this function:



7.213.1.6 physics_metadata_init()

```
void physics_metadata_init (
            DictEntry * entry,
            uint32_t header_bytes )
```

Initialise physics metadata for a newly registered DictEntry.

Zeroes all `DictPhysics` fields, sets `mass_bytes` to `header_bytes` (the byte size of the entry header), and derives initial `state_flags` from the entry's word flags. Called from `register_word()` immediately after the entry is linked into the dictionary.

Parameters

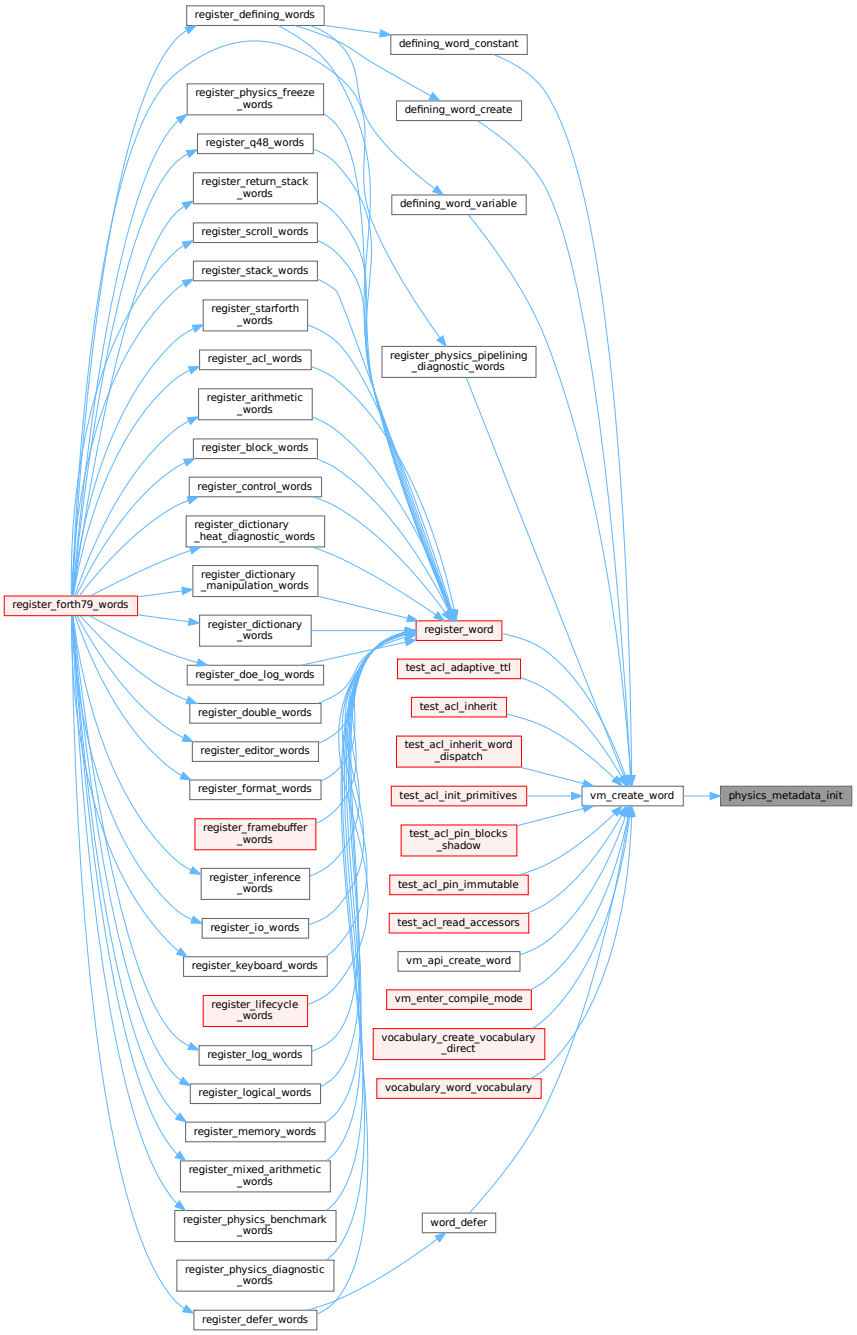
entry	DictEntry to initialise
header_bytes	Byte size of the entry's header region (for mass)

References derive_state_flags(), and entry.

Referenced by vm_create_word().

Here is the call graph for this function:

Here is the caller graph for this function:



7.213.1.7 physics_metadata_record_latency()

```
void physics_metadata_record_latency (
            DictEntry * entry,
            uint64_t sample_ns )
```

Record an execution latency sample into the per-word EMA.

Clamps `sample_ns` to uint32_t, then blends it into `avg_latency_ns` with a 1/4 EMA: new = (3∗prior + sample) / 4. Seeds `avg_latency_ns` directly from the first sample.

Parameters

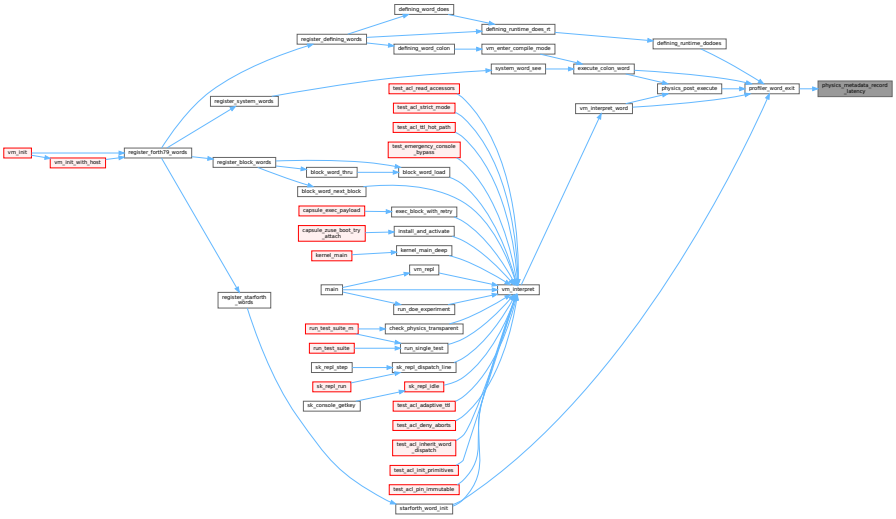
entry	DictEntry whose latency EMA is updated
sample_ns	Measured execution duration in nanoseconds

References clamp_u32(), and entry.

Referenced by profiler_word_exit().

Here is the call graph for this function:

Here is the caller graph for this function:



7.213.1.8 physics_metadata_refresh_state()

```
void physics_metadata_refresh_state (
            DictEntry * entry )
```

Re-derive and update `state_flags` for a DictEntry.

Called after word flags change (e.g. IMMEDIATE is set during compilation) to keep the physics state flags in sync with the word flags.

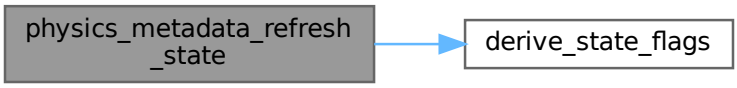
Parameters

entry	DictEntry whose state flags are to be refreshed

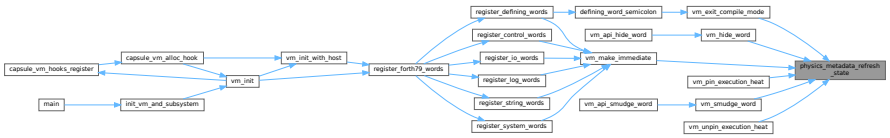
References derive_state_flags(), and entry.

Referenced by vm_exit_compile_mode(), vm_hide_word(), vm_make_immediate(), vm_pin_execution_heat(), vm_smudge_word(), and vm_unpin_execution_heat().

Here is the call graph for this function:



Here is the caller graph for this function:



7.213.1.9 physics_metadata_set_mass()

```
void physics_metadata_set_mass (
            DictEntry * entry,
            uint32_t total_bytes )
```

Update the mass (byte footprint) of a DictEntry's physics metadata.

Called after a colon definition or CREATE body is fully compiled so the mass reflects the total compiled size, not just the header.

Parameters

entry	DictEntry to update
total_bytes	Total compiled byte size (header + body)

References entry.

Referenced by vm_exit_compile_mode().

Here is the caller graph for this function:

7.213.1.10 physics_metadata_touch()

```
void physics_metadata_touch (
            DictEntry * entry,
            cell_t execution_heat,
            uint64_t now_ns )
```

Update physics metadata after a word execution completes.

Derives and stores a new smoothed `temperature_q8` from `execution_heat`, and updates both `last↩_active_ns` and `last_decay_ns` to `now_ns`. Called by `physics_post_execute()` on the post-execution path.

Parameters

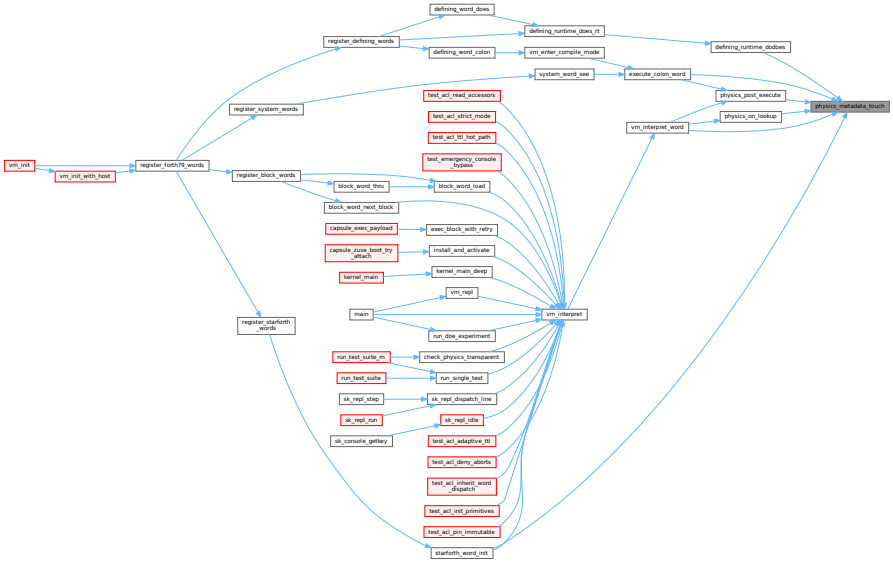
entry	DictEntry that was just executed
execution_heat	Current raw execution heat counter
now_ns	Current monotonic timestamp in nanoseconds

References entry, execution_heat, and temperature_from_execution_heat().

Referenced by defining_runtime_dodoes(), execute_colon_word(), physics_on_lookup(), physics_post_execute(), starforth_word_init(), and vm_interpret_word().

Here is the call graph for this function:

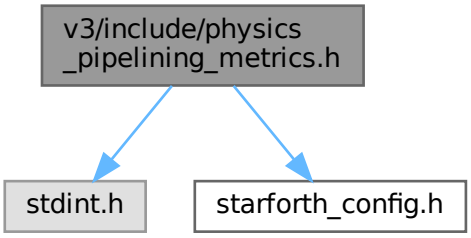
Here is the caller graph for this function:



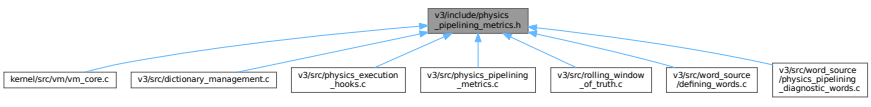
7.214 v3/include/physics_pipelining_metrics.h File Reference

```
#include <stdint.h>
#include "starforth_config.h"
```

Include dependency graph for physics_pipelining_metrics.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct WordTransitionMetrics

  SPECULATION_THRESHOLD_Q48: Minimum probability (Q48.16) to speculate on a word transition.

Typedefs

- typedef struct WordTransitionMetrics WordTransitionMetrics

  SPECULATION_THRESHOLD_Q48: Minimum probability (Q48.16) to speculate on a word transition.

Functions

- int transition_metrics_init (WordTransitionMetrics *metrics)

  Initialise a `WordTransitionMetrics` structure to its empty, ready-to-record state.
- int transition_metrics_record (WordTransitionMetrics *metrics, uint32_t next_word_id, uint32_t dict_size)

  Record one word-to-word transition in the transition heat table.
- int64_t transition_metrics_get_probability_q48 (const WordTransitionMetrics *metrics, uint32_t target_word_id)

  Return the transition probability to `target_word_id` in Q48.16 format.
- void transition_metrics_update_cache (WordTransitionMetrics *metrics, uint32_t dict_size)

  Scan the transition heat table and cache the highest-probability next word.
- void transition_metrics_record_prefetch_hit (WordTransitionMetrics *metrics, int64_t latency_saved_ns)

  Record a successful speculative prefetch and the latency it saved.
- void transition_metrics_record_prefetch_miss (WordTransitionMetrics *metrics, int64_t recovery_cost_ns)

  Record a failed speculative prefetch and its recovery overhead.
- int transition_metrics_should_speculate (const WordTransitionMetrics *metrics, uint32_t target_word_id)

  Decide whether to speculatively prefetch `target_word_id`.
- char * transition_metrics_stats_string (const WordTransitionMetrics *metrics)

  Allocate and return a human-readable transition metrics summary string.
- void transition_metrics_reset (WordTransitionMetrics *metrics)

  Reset a `WordTransitionMetrics` to empty without freeing the context window.
- void transition_metrics_cleanup (WordTransitionMetrics *metrics)

  Release all heap memory held by a `WordTransitionMetrics`.
- int transition_metrics_record_context (WordTransitionMetrics *metrics, const uint32_t *context, uint32_t window_size, uint32_t next_word_id, uint32_t dict_size)

  Record a context-conditioned transition (Phase 1 collection stub).
- int transition_metrics_update_context_window (WordTransitionMetrics *metrics, uint32_t word_id)

  Slide the context window forward by one word, appending `word_id`.
- char * transition_metrics_context_accuracy_string (const WordTransitionMetrics *metrics)

  Allocate and return a context-window status report string (Phase 1 stub).
- uint32_t transition_metrics_binary_chop_suggest_window (const WordTransitionMetrics *metrics, uint32_t current_window, double accuracy_at_current)

  Suggest a new context window size using binary-chop search (Phase 1 stub).

7.214.1 Typedef Documentation

7.214.1.1 WordTransitionMetrics

```
typedef struct WordTransitionMetrics WordTransitionMetrics
```

SPECULATION_THRESHOLD_Q48: Minimum probability (Q48.16) to speculate on a word transition.

Range: 0.10 (very aggressive) to 0.95 (very conservative) Default: see starforth_config.h (STARFORTH↩_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT) Q48.16 encoding: 0.50 = 0x8000LL (1LL << 15) SPECULATION_DEPTH: How many words ahead to prefetch speculatively. Range: 1 (next word only) to 4 (very aggressive) Default: see starforth_config.h (STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT) MIN_↩SAMPLES_FOR_SPECULATION: Minimum number of transitions observed before making speculation decisions. Range: 1 (immediate) to 100 (very conservative) Default: see starforth_config.h (STARFORTH_CONFIG_MIN↩_SAMPLES_FOR_SPECULATION_DEFAULT) MISPREDICTION_COST_Q48: Estimated cost (in nanoseconds, Q48.16 format) of recovering from wrong speculation. This is calibrated to the actual recovery time on target hardware. Range: 0 to 100 ns Default: see starforth_config.h (STARFORTH_CONFIG_MISPREDICTION_COST↩_Q48_DEFAULT) Q48.16 encoding: 25 ns = 25 << 16 = 0x190000LL MINIMUM_PREFETCH_ROI: Minimum expected Return-On-Investment ratio for speculation to be worthwhile. Speculation succeeds only if: (prefetch↩_latency_saved / total_attempts) > MINIMUM_PREFETCH_ROI Range: 1.0 (always speculate) to 2.0 (very selective) Default: see starforth_config.h (STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT) Q48.16 encoding: 1.10 = 1.10 ∗ (1 << 16) = 0x1199AL TRANSITION_WINDOW_SIZE: Knob #6 - Execution context depth for prediction

How many previous words to remember when predicting next word. Larger window captures deeper patterns but uses more memory.

Window=1: Only immediate predecessor (A→B predictions) Window=2: Previous 2 words (A,B→C predictions) - good balance Window=4: Previous 4 words (captures multi-level patterns) Window=8: Very deep, expensive – see starforth_config.h for the actual shipped default

Binary Chop Search Space (Phase 2): Iteration 1: Try window=2 (baseline) Iteration 2: Try window=1 and window=4 Iteration 3: Narrow to optimal (1, 2, or 4) Converge: Pick size with highest prediction accuracy

Range: 1 to 8 Default: see starforth_config.h (STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT) Trade-off: Pattern depth vs. memory usage & complexity

Tuning: make TRANSITION_WINDOW_SIZE=1 (or 2, 4, 8)

7.214.2 Function Documentation

7.214.2.1 transition_metrics_binary_chop_suggest_window()

```
uint32_t transition_metrics_binary_chop_suggest_window (
            const WordTransitionMetrics * metrics,
            uint32_t current_window,
            double accuracy_at_current )
```

Suggest a new context window size using binary-chop search (Phase 1 stub).

In Phase 2 this function will implement a binary-chop search over window sizes to converge on the width that maximises context-prediction accuracy:

- Start at window = 2.

- If accuracy improves when doubling, try 4; if it declines, try 1.
- Continue halving/doubling until the accuracy delta falls below a threshold.

Phase 1 stub behaviour: Doubles the current window each call up to 8, then wraps back to 1:

- `current_window` == 1 → suggests 2
- `current_window` == 2 → suggests 4
- `current_window` == 4 → suggests 8
- `current_window` ≥ 8 → suggests 1 (cycle back)
- Any other value → returns `current_window` unchanged.

The `accuracy_at_current` parameter (prediction accuracy [0.0, 1.0] at the current window size) is accepted for API stability with the Phase 2 caller but is not used in the stub.

Parameters

metrics	The metrics structure (unused in Phase 1 stub; reserved for Phase 2 accuracy history).
current_window	Currently active context window size.
accuracy_at_current	Prediction accuracy at `current_window` (unused in Phase 1 stub; range [0.0, 1.0]).

Returns

Suggested new window size.

7.214.2.2 transition_metrics_cleanup()

```
void transition_metrics_cleanup (
            WordTransitionMetrics * metrics )
```

Release all heap memory held by a `WordTransitionMetrics`.

Frees and NULLs:

- `metrics->transition_heat` — the per-word transition count array.
- `metrics->context_window` — the sliding context-window array.
- `metrics->context_transitions` — the Phase 2 sparse hash table (if allocated).

Does NOT zero the struct fields beyond NULLing the freed pointers. The caller must not use `metrics` after this call without a subsequent `transition_metrics_init()`. Contrast with `transition_metrics_reset()`, which keeps the context window allocated for reuse.

No-op if `metrics` is NULL.

Parameters

metrics	The metrics structure to clean up; no-op if NULL.

References WordTransitionMetrics::context_transitions, WordTransitionMetrics::context_window, free(), and WordTransitionMetrics::transition_heat.

Referenced by dictionary_word_forget().

Here is the call graph for this function:

Here is the caller graph for this function:

7.214.2.3 transition_metrics_context_accuracy_string()

```
char * transition_metrics_context_accuracy_string (
            const WordTransitionMetrics * metrics )
```

Allocate and return a context-window status report string (Phase 1 stub).

Allocates a 1024-byte buffer and fills it with a brief Phase 1 collection status message:

- `actual_window_size` — the active window width (0 if allocation failed).
- `total_context_transitions` — the number of context transitions recorded.
- A "[Phase 1 Collection - Phase 2 Analysis Pending]" status line.

Phase 2 will replace this stub with a real accuracy report showing per-window-size prediction accuracy derived from the sparse context-pattern hash table. The accuracy data will feed `transition_metrics_binary_chop_suggest_windo` to converge on the optimal context window width.

The caller is responsible for calling `free()` on the returned pointer. Returns NULL if `metrics` is NULL or `malloc()` fails.

Parameters

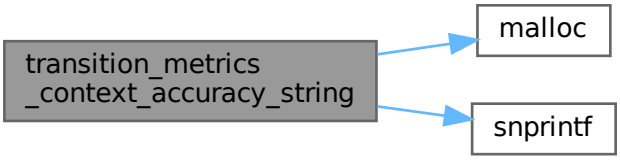
metrics	The metrics structure to report on.

Returns

Heap-allocated NUL-terminated string; caller must `free()`. NULL on allocation failure or NULL `metrics`.

References WordTransitionMetrics::actual_window_size, malloc(), snprintf(), and WordTransitionMetrics::total_context_transitions.

Here is the call graph for this function:



7.214.2.4 transition_metrics_get_probability_q48()

```
int64_t transition_metrics_get_probability_q48 (
            const WordTransitionMetrics * metrics,
            uint32_t target_word_id )
```

Return the transition probability to `target_word_id` in Q48.16 format.

Computes P(next = `target_word_id` | current word) as a Q48.16 fixed-point ratio:
`P = (transition_heat[target_word_id] « 16) / total_transitions`

using `q48_div_u64()` to guard against divide-by-zero.

Returns 0 if:

- `metrics` is NULL.
- `metrics->total_transitions` == 0 (no transitions recorded yet).
- `metrics->transition_heat` is NULL (lazy allocation has not occurred).

The caller is responsible for ensuring `target_word_id` is within the bounds of the allocated `transition_↩ heat` array; this function does not re-check bounds after verifying the NULL guard above (bounds were enforced during `transition_metrics_record()`).

Consumed by `transition_metrics_update_cache()` (scans all words to find the maximum) and `transition_metrics_should_speculate()` (checks whether a specific word's probability exceeds `SPECULATION_THRESHOLD_Q48`).

Parameters

metrics	The metrics structure to query.
target_word↩ _id	Zero-based word ID whose transition probability is requested; must be within the dict_size used at record time.

Returns

P(next = `target_word_id`) in Q48.16; 0 if unavailable.

References count, q48_div_u64(), WordTransitionMetrics::total_transitions, and WordTransitionMetrics::transition_heat.

Referenced by transition_metrics_should_speculate(), and transition_metrics_update_cache().

Here is the call graph for this function:

Here is the caller graph for this function:

7.214.2.5 transition_metrics_init()

```
int transition_metrics_init (
            WordTransitionMetrics * metrics )
```

Initialise a `WordTransitionMetrics` structure to its empty, ready-to-record state.

Sets all counter and accumulator fields to zero and performs the following allocations:

- `metrics->transition_heat` is left NULL — it is lazily allocated by the first call to `transition_metrics_record(` to avoid the `dict_size` parameter being required at construction time.
- If `TRANSITION_WINDOW_SIZE` > 0, allocates `metrics->context_window` as a zero-filled array of `TRANSITION_WINDOW_SIZE` `uint32_t` word IDs and sets `metrics->actual_window_size` accordingly. If the allocation fails, `actual_window_size` is set to 0 and context tracking degrades gracefully (all context functions become no-ops).
- `metrics->context_transitions` is left NULL — it is lazily allocated when Phase 2 context-pattern analysis is implemented.

Returns 0 on success or ?1 if `metrics` is NULL.

Parameters

metrics	The `WordTransitionMetrics` to initialise; must not be NULL.

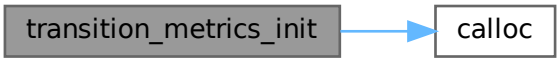
Returns

0 on success; ?1 if `metrics` is NULL.

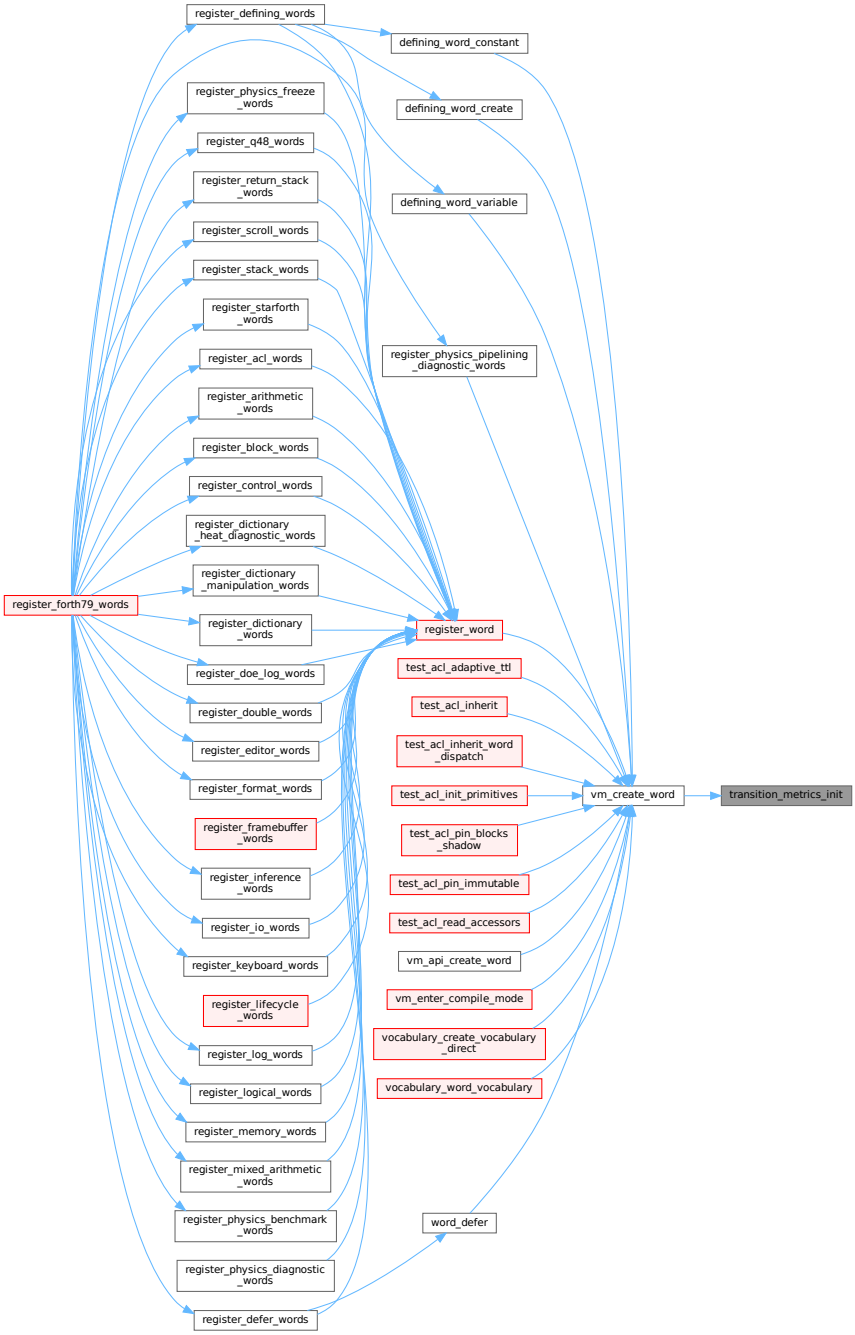
References WordTransitionMetrics::actual_window_size, calloc(), WordTransitionMetrics::context_transitions, WordTransitionMetrics::context_window, WordTransitionMetrics::context_window_pos, WordTransitionMetrics::max_transition_probab WordTransitionMetrics::misprediction_cost_q48, WordTransitionMetrics::most_likely_next_word_id, WordTransitionMetrics::prefetch_a WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_latency_saved_q48, WordTransitionMetrics::prefetch_misses, WordTransitionMetrics::total_context_transitions, WordTransitionMetrics::total_transitions, WordTransitionMetrics::transition_heat, and TRANSITION_WINDOW_SIZE.

Referenced by vm_create_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.214.2.6 transition_metrics_record()

```
int transition_metrics_record (
            WordTransitionMetrics * metrics,
            uint32_t next_word_id,
            uint32_t dict_size )
```

Record one word-to-word transition in the transition heat table.

Called from the interpreter loop each time a word executes and there is a known previous word. Increments `metrics->transition_heat`[next_word_id] by one and `metrics->total_transitions` by one.

Lazy allocation: The `transition_heat` array is not allocated at `transition_metrics_init()` time; on the first call when `metrics->transition_heat` == NULL, it is allocated as a zero-filled array of `dict_size` `uint64_t` counters. If the allocation fails ?1 is returned and the transition is silently dropped.

If `next_word_id` $\geq$ `dict_size` the call is rejected with ?1 to prevent an out-of-bounds write into `transition_heat` (which was allocated with size `dict_size`).

The accumulated `transition_heat`[] table is consumed by `transition_metrics_get_probability_q48()` and `transition_metrics_update_cache()` to determine the most-likely next word for speculative prefetch.

Parameters

metrics	The metrics structure to update.
next_word_id	Zero-based word ID of the word that followed the current word; must be $<$ `dict_size`.
dict_size	Current dictionary size, used both as the bounds check for `next_word_id` and as the allocation size for the lazy-allocated `transition_heat` array.

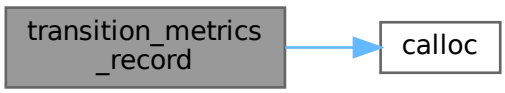
Returns

0 on success; ?1 on NULL `metrics`, out-of-bounds `next_word_id`, or allocation failure.

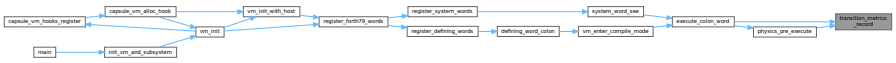
References calloc(), WordTransitionMetrics::total_transitions, and WordTransitionMetrics::transition_heat.

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the call graph for this function:



Here is the caller graph for this function:



7.214.2.7 transition_metrics_record_context()

```
int transition_metrics_record_context (
            WordTransitionMetrics * metrics,
            const uint32_t * context,
            uint32_t window_size,
            uint32_t next_word_id,
            uint32_t dict_size )
```

Record a context-conditioned transition (Phase 1 collection stub).

Accepts a `context` window of `window_size` preceding word IDs and the `next_word_id` that followed. In Phase 1 only the total count is tracked: `metrics->total_context_transitions` is incremented by one, providing a baseline count that Phase 2 analysis can use to size its hash tables.

Phase 2 TODO: Build a sparse hash table keyed on the context pattern (context[0], ..., context[window_size-1]) to record per-context next-word distributions. The Phase 2 analysis will feed `transition_metrics_binary_chop_suggest_w` with accuracy data to tune the optimal window size.

Returns ?1 (no-op) if `metrics` or `context` are NULL, `window_size` is 0, or `next_word_id` $\geq$ `dict_size`.

Parameters

metrics	The metrics structure to update.
context	Array of `window_size` word IDs preceding `next_word_id`.
window_size	Number of entries in `context`; must be > 0.
next_word_id	Word ID that followed the context sequence.
dict_size	Current dictionary size; used for bounds checking `next_word_id`.

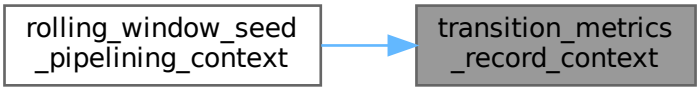
Returns

0 on success; ?1 on invalid arguments.

References WordTransitionMetrics::total_context_transitions.

Referenced by rolling_window_seed_pipelining_context().

Here is the caller graph for this function:



7.214.2.8 transition_metrics_record_prefetch_hit()

```
void transition_metrics_record_prefetch_hit (
            WordTransitionMetrics * metrics,
            int64_t latency_saved_ns )
```

Record a successful speculative prefetch and the latency it saved.

Increments `metrics->prefetch_hits` by one and accumulates `latency_saved_ns` (converted to Q48.16 via `ns_to_q48()`) into `metrics->prefetch_latency_saved_q48`. The accumulated total is reported by `transition_metrics_stats_string()` as the aggregate latency benefit of correct speculation.

Called by the speculative-execution path when a prefetched word matches the word that was actually executed next.

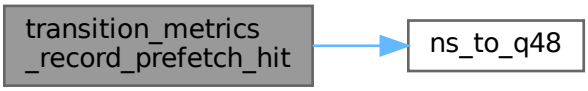
Parameters

metrics	The metrics structure to update; no-op if NULL.
latency_saved_ns	The number of nanoseconds saved by avoiding a full dictionary lookup on this particular hit.

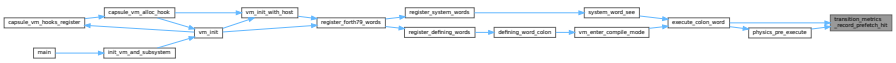
References ns_to_q48(), WordTransitionMetrics::prefetch_hits, and WordTransitionMetrics::prefetch_latency_saved_q48.

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the call graph for this function:



Here is the caller graph for this function:



7.214.2.9 transition_metrics_record_prefetch_miss()

```
void transition_metrics_record_prefetch_miss (
            WordTransitionMetrics * metrics,
            int64_t recovery_cost_ns )
```

Record a failed speculative prefetch and its recovery overhead.

Increments `metrics->prefetch_misses` by one and accumulates `recovery_cost_ns` (converted to Q48.16 via `ns_to_q48()`) into `metrics->misprediction_cost_q48`. The accumulated total is the aggregate penalty paid for incorrect speculation and is reported by `transition_metrics_stats_string()` as "Misprediction Cost".

The net benefit of the pipelining subsystem is:
```
net_ns = prefetch_latency_saved_q48 - misprediction_cost_q48
```

A positive net indicates that speculation is profitable overall.

Called by the speculative-execution path when the prefetched word does not match the word that was actually executed next (branch misprediction analogue).

Parameters

metrics	The metrics structure to update; no-op if NULL.
recovery_cost_ns	The number of nanoseconds spent unwinding the incorrect speculation (cache flush, state restore).

References WordTransitionMetrics::misprediction_cost_q48, ns_to_q48(), and WordTransitionMetrics::prefetch_misses.

Here is the call graph for this function:

7.214.2.10 transition_metrics_reset()

```
void transition_metrics_reset (
            WordTransitionMetrics * metrics )
```

Reset a `WordTransitionMetrics` to empty without freeing the context window.

Frees and NULLs `metrics->transition_heat` (so the next `transition_metrics_record()` will re-allocate it), then zeroes all counter and accumulator fields. Also zeroes the context-window array (if allocated) and resets `context_window_pos` to 0, and frees and NULLs `context_transitions`. `actual_window_size` and the `context_window` pointer itself are preserved — the window stays allocated and ready for reuse.

Intended for DoE benchmark runs that need a clean statistical baseline between treatment cells without the overhead of full teardown and reinitialisation. Cheaper than `transition_metrics_cleanup()` + `transition_metrics_init()` because it avoids reallocating the context window.

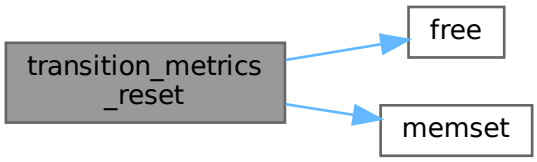
Parameters

metrics	The metrics structure to reset; no-op if NULL.

References WordTransitionMetrics::actual_window_size, WordTransitionMetrics::context_transitions, WordTransitionMetrics::context_
WordTransitionMetrics::context_window_pos, free(), WordTransitionMetrics::max_transition_probability_q48,
memset(), WordTransitionMetrics::misprediction_cost_q48, WordTransitionMetrics::most_likely_next_word_id,
WordTransitionMetrics::prefetch_attempts, WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_latency_saved_q48
WordTransitionMetrics::prefetch_misses, WordTransitionMetrics::total_context_transitions, WordTransitionMetrics::total_transitions,
and WordTransitionMetrics::transition_heat.

Referenced by forth_PIPELINING_RESET_ALL().

Here is the call graph for this function:



Here is the caller graph for this function:

7.214.2.11 transition_metrics_should_speculate()

```
int transition_metrics_should_speculate (
            const WordTransitionMetrics * metrics,
            uint32_t target_word_id )
```

Decide whether to speculatively prefetch `target_word_id`.

Applies a two-gate policy before authorising speculation:

1. **Minimum sample gate:** `metrics->total_transitions` must be $\geq$ `MIN_SAMPLES_FOR_SPECULATION`. Below this threshold the transition probabilities are statistically unreliable — speculation would be indistinguishable from guessing.
2. **Probability threshold gate:** the Q48.16 probability returned by `transition_metrics_get_probability_q48`(metrics, `target_word_id`) must be $\geq$ `SPECULATION_THRESHOLD_Q48`. Words with very low conditional probability of following the current word are not worth prefetching.

Both gates must pass for the function to return 1. A Phase 4 ROI check (comparing expected latency savings against expected misprediction cost) is planned but not yet implemented; a TODO comment marks the extension point.

Returns 0 if `metrics` or `transition_heat` are NULL, or if either gate fails.

Parameters

metrics	The metrics structure to consult.
target_word↩ _id	Zero-based ID of the candidate next word.

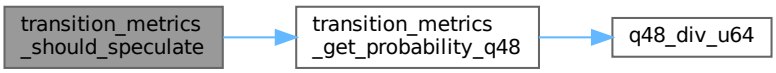
Returns

1 if speculation is recommended; 0 otherwise.

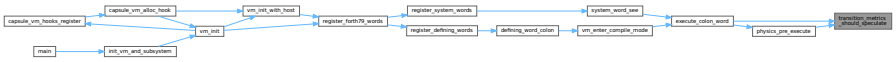
References MIN_SAMPLES_FOR_SPECULATION, SPECULATION_THRESHOLD_Q48, WordTransitionMetrics::total_transitions, WordTransitionMetrics::transition_heat, and transition_metrics_get_probability_q48().

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the call graph for this function:



Here is the caller graph for this function:



7.214.2.12 transition_metrics_stats_string()

```
char * transition_metrics_stats_string (
            const WordTransitionMetrics * metrics )
```

Allocate and return a human-readable transition metrics summary string.

Allocates a 2048-byte buffer and fills it with a multi-line report:

- Total transitions recorded.
- Prefetch attempts, hits (with hit-rate %), and misses.
- Aggregate latency saved by correct speculation (from `prefetch_latency_saved_q48`, converted to integer ns via `q48_to_ns()`).
- Aggregate misprediction cost (from `misprediction_cost_q48`).
- Net benefit (saved ? cost) in nanoseconds.
- The most-likely next word ID and its raw Q48.16 probability.

The caller is responsible for calling `free()` on the returned pointer. Returns NULL if `metrics` is NULL or `malloc()` fails.

Parameters

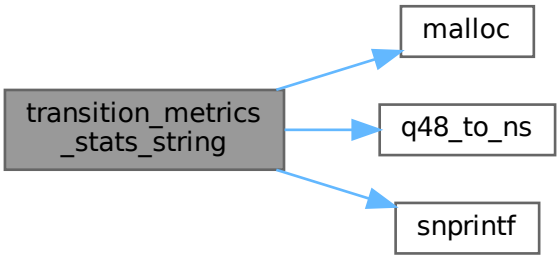
metrics	The metrics structure to summarise.

Returns

Heap-allocated NUL-terminated stats string; caller must `free()`. NULL on allocation failure or NULL `metrics`.

References malloc(), WordTransitionMetrics::max_transition_probability_q48, WordTransitionMetrics::misprediction_cost_q48, WordTransitionMetrics::most_likely_next_word_id, WordTransitionMetrics::prefetch_attempts, WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_latency_saved_q48, WordTransitionMetrics::prefetch_misses, q48_to_ns(), snprintf(), and WordTransitionMetrics::total_transitions.

Here is the call graph for this function:



7.214.2.13 transition_metrics_update_cache()

```
void transition_metrics_update_cache (
            WordTransitionMetrics * metrics,
            uint32_t dict_size )
```

Scan the transition heat table and cache the highest-probability next word.

Iterates over all `dict_size` entries in `metrics->transition_heat[]`, calling `transition_metrics_get_probabil` for each non-zero entry and tracking the maximum. After the scan:

- `metrics->max_transition_probability_q48` holds the peak probability (Q48.16).
- `metrics->most_likely_next_word_id` holds the word ID of that peak.

These two cached fields allow `transition_metrics_should_speculate()` to make a prefetch decision in O(1) without re-scanning the full table on every word dispatch.

If `metrics`, `transition_heat`, or `total_transitions` are not valid, both fields are reset to 0 and the function returns immediately.

Called after each `transition_metrics_record()` in the high-frequency interpreter path, so the implementation must be fast. The full `dict_size` scan runs in O(dict_size) and skips zero-count entries early.

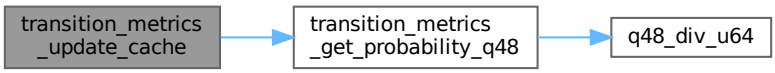
Parameters

metrics	The metrics structure to update.
dict_size	Number of entries in `metrics->transition_heat[]`; must match the size used at allocation time.

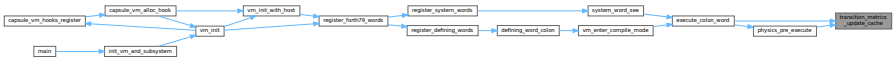
References WordTransitionMetrics::max_transition_probability_q48, WordTransitionMetrics::most_likely_next_word_id, WordTransitionMetrics::total_transitions, WordTransitionMetrics::transition_heat, and transition_metrics_get_probability_q48().

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the call graph for this function:



Here is the caller graph for this function:



7.214.2.14 transition_metrics_update_context_window()

```
int transition_metrics_update_context_window (
            WordTransitionMetrics * metrics,
            uint32_t word_id )
```

Slide the context window forward by one word, appending `word_id`.

Implements a fixed-length sliding window over the recent word-ID history. Each call shifts all existing entries one position to the left (dropping the oldest), then writes `word_id` into the last slot:

```
context_window[0..n-2] = context_window[1..n-1]
context_window[n-1] = word_id
```

This produces a FIFO queue of the last `actual_window_size` word IDs that can be passed to `transition_metrics_record_context()` for context-sensitive transition tracking (Phase 2).

Returns ?1 and is a no-op if:

- `metrics` is NULL.
- `context_window` is NULL (allocation at init failed — degraded mode).
- `actual_window_size` == 0.

Parameters

metrics	The metrics structure whose context window is updated.
word↩ _id	The word ID just executed; appended to the right of the window.

Returns

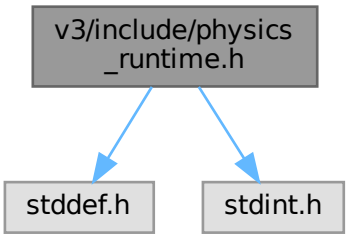
0 on success; ?1 if the context window is unavailable.

References WordTransitionMetrics::actual_window_size, WordTransitionMetrics::context_window, and word_id.

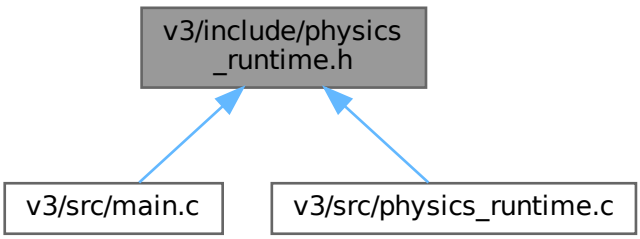
7.215 v3/include/physics_runtime.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for physics_runtime.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct physics_host_snapshot

  Snapshot of scheduler-adjacent host signals.
- struct physics_analytics_event_header

  Event header written into the analytics ring buffer.
- struct physics_analytics_header

  Analytics heap header seen by both producer (VM) and consumer (HOLA).
- struct physics_analytics_heap

  Simple view onto the analytics heap layout.

Macros

- #define PHYSICS_ANALYTICS_DEFAULT_HEAP_BYTES (10u ∗ 1024u ∗ 1024u)

  Default analytics heap size (10 MiB)
- #define HOLA_SHARED_MAGIC 0x484f4c41u

  Magic tag for HOLA analytics heap headers ('HOLA')
- #define HOLA_PROTOCOL_MAJOR 0x0001u

  HOLA protocol version (major.minor)
- #define HOLA_PROTOCOL_MINOR 0x0000u
- #define PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT 0x00000002u

  Analytics channel ids.
- #define PHYSICS_HOST_FLAG_PSI 0x0001u

  Host snapshot flag bits.
- #define PHYSICS_HOST_FLAG_CGROUP 0x0002u
- #define PHYSICS_HOST_FLAG_CPU_STAT 0x0004u

Typedefs

- typedef enum physics_host_backend physics_host_backend_t

  Host backend identifier.
- typedef struct physics_host_snapshot physics_host_snapshot_t

  Snapshot of scheduler-adjacent host signals.
- typedef struct physics_analytics_event_header physics_analytics_event_header_t

  Event header written into the analytics ring buffer.
- typedef struct physics_analytics_header physics_analytics_header_t

  Analytics heap header seen by both producer (VM) and consumer (HOLA).
- typedef struct physics_analytics_heap physics_analytics_heap_t

  Simple view onto the analytics heap layout.

Enumerations

- enum physics_host_backend { PHYSICS_HOST_BACKEND_POSIX = 0 , PHYSICS_HOST_BACKEND_L4RE = 1 , PHYSICS_HOST_BACKEND_FAKE = 2 }

  Host backend identifier.

Functions

- int physics_runtime_init (size_t analytics_heap_bytes)

  Initialise the physics runtime and analytics heap.
- void physics_runtime_shutdown (void)

  Tear down physics runtime services and release heap storage.
- int physics_host_snapshot (physics_host_snapshot_t *out)

  Capture a host environment snapshot and publish it to the analytics ring.
- int physics_analytics_publish_event (uint32_t channel, const void *payload, uint16_t payload_bytes)

  Publish an analytics event into the ring buffer.
- const physics_analytics_heap_t * physics_analytics_heap_info (void)

  Retrieve the analytics heap descriptor.

7.215.1 Macro Definition Documentation

7.215.1.1 HOLA_PROTOCOL_MAJOR

```
#define HOLA_PROTOCOL_MAJOR 0x0001u
```

HOLA protocol version (major.minor)

7.215.1.2 HOLA_PROTOCOL_MINOR

```
#define HOLA_PROTOCOL_MINOR 0x0000u
```

7.215.1.3 HOLA_SHARED_MAGIC

```
#define HOLA_SHARED_MAGIC 0x484f4c41u
```

Magic tag for HOLA analytics heap headers ('HOLA')

7.215.1.4 PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT

```
#define PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT 0x00000002u
```

Analytics channel ids.

7.215.1.5 PHYSICS_ANALYTICS_DEFAULT_HEAP_BYTES

```
#define PHYSICS_ANALYTICS_DEFAULT_HEAP_BYTES (10u * 1024u * 1024u)
```

Default analytics heap size (10 MiB)

7.215.1.6 PHYSICS_HOST_FLAG_CGROUP

```
#define PHYSICS_HOST_FLAG_CGROUP 0x0002u
```

7.215.1.7 PHYSICS_HOST_FLAG_CPU_STAT

```
#define PHYSICS_HOST_FLAG_CPU_STAT 0x0004u
```

7.215.1.8 PHYSICS_HOST_FLAG_PSI

```
#define PHYSICS_HOST_FLAG_PSI 0x0001u
```

Host snapshot flag bits.

7.215.2 Typedef Documentation

7.215.2.1 physics_analytics_event_header_t

```
typedef struct physics_analytics_event_header physics_analytics_event_header_t
```

Event header written into the analytics ring buffer.

7.215.2.2 physics_analytics_header_t

```
typedef struct physics_analytics_header physics_analytics_header_t
```

Analytics heap header seen by both producer (VM) and consumer (HOLA).

7.215.2.3 physics_analytics_heap_t

```
typedef struct physics_analytics_heap physics_analytics_heap_t
```

Simple view onto the analytics heap layout.

7.215.2.4 physics_host_backend_t

```
typedef enum physics_host_backend physics_host_backend_t
```

Host backend identifier.

7.215.2.5 physics_host_snapshot_t

```
typedef struct physics_host_snapshot physics_host_snapshot_t
```

Snapshot of scheduler-adjacent host signals.

Fields are intentionally fixed width to maximise portability and allow the governance repository to consume binary dumps without per-platform quirks.

7.215.3 Enumeration Type Documentation

7.215.3.1 physics_host_backend

```
enum physics_host_backend
```

Host backend identifier.

Enumerator

PHYSICS_HOST_BACKEND_POSIX	
PHYSICS_HOST_BACKEND_L4RE	
PHYSICS_HOST_BACKEND_FAKE	

7.215.4 Function Documentation

7.215.4.1 physics_analytics_heap_info()

```
const physics_analytics_heap_t * physics_analytics_heap_info (
            void  )
```

Retrieve the analytics heap descriptor.

Returns

Pointer to heap descriptor (NULL if runtime not initialised)

Retrieve the analytics heap descriptor.

Returns

Pointer to the heap descriptor, or NULL if not yet initialised

References g_runtime, physics_runtime_state::heap, and physics_runtime_state::heap_ready.

7.215.4.2 physics_analytics_publish_event()

```
int physics_analytics_publish_event (
            uint32_t channel,
            const void * payload,
            uint16_t payload_bytes )
```

Publish an analytics event into the ring buffer.

Writes a `physics_analytics_event_header_t` followed by the payload (8-byte padded) into the ring at the current write offset. If insufficient space is available, increments `dropped_events` and returns -1. The event is timestamped with `sf_monotonic_ns()` at write time.

Parameters

channel	Event channel identifier (e.g. `PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT`)
payload	Event payload bytes (must not be NULL)
payload_bytes	Length of payload in bytes

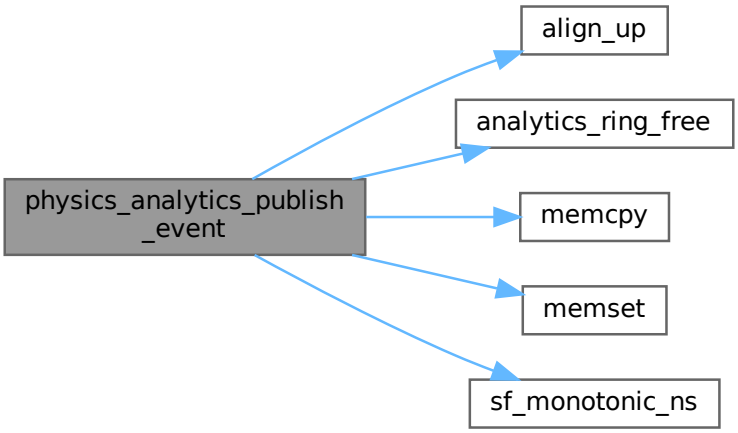
Returns

0 on success, -1 if the heap is not ready, payload is NULL, or ring is full

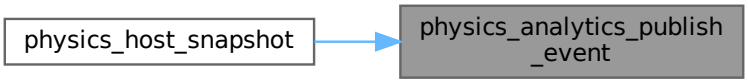
References align_up(), analytics_ring_free(), physics_analytics_event_header::channel, physics_analytics_header::dropped_events, g_runtime, physics_analytics_heap::header, physics_runtime_state::heap, physics_runtime_state::heap_ready, memcpy(), memset(), physics_analytics_event_header::payload_bytes, physics_analytics_header::produce_seq, physics_analytics_event_header::reserved, ring, physics_analytics_heap::ring, physics_analytics_header::ring_bytes, sf_monotonic_ns(), physics_analytics_event_header::timestamp_ns, and physics_analytics_header::write_offset.

Referenced by physics_host_snapshot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.215.4.3 physics_host_snapshot()

```
int physics_host_snapshot (
            physics_host_snapshot_t * out )
```

Capture a host environment snapshot and publish it to the analytics ring.

Uses the POSIX backend. On success also publishes the snapshot as a PHYSICS_ANALYTICS_CHANNEL_↩ HOST_SNAPSHOT event to the analytics ring buffer (if the heap is initialised).

Parameters

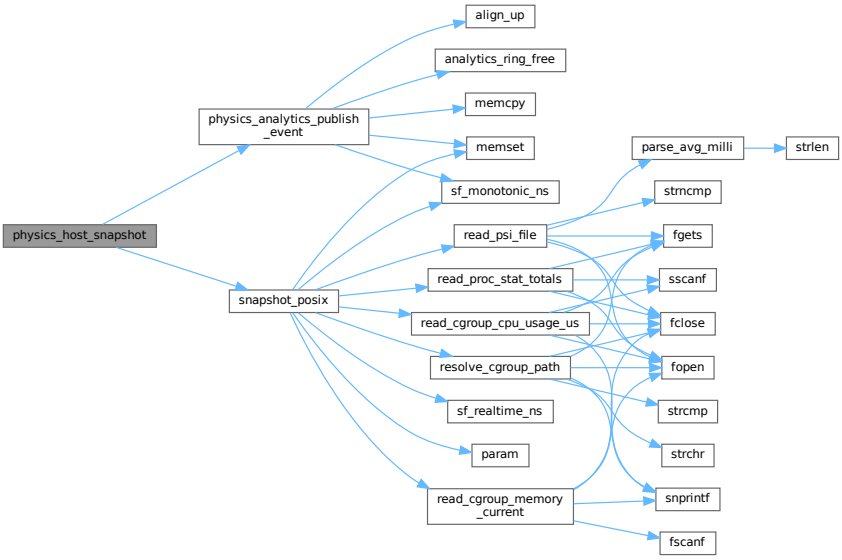
out	Output snapshot struct to populate (must not be NULL)

Returns

0 on success, -1 if `out` is NULL or the backend fails

References g_runtime, physics_runtime_state::heap_ready, PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT, physics_analytics_publish_event(), and snapshot_posix().

Here is the call graph for this function:



7.215.4.4 physics_runtime_init()

```
int physics_runtime_init (
            size_t analytics_heap_bytes )
```

Initialise the physics runtime and analytics heap.

Must be called before any other physics_runtime_∗ function. Idempotent — safe to call multiple times; subsequent calls return 0 immediately without re-allocating.

Parameters

analytics_heap_bytes	Desired analytics heap size (0 = default)

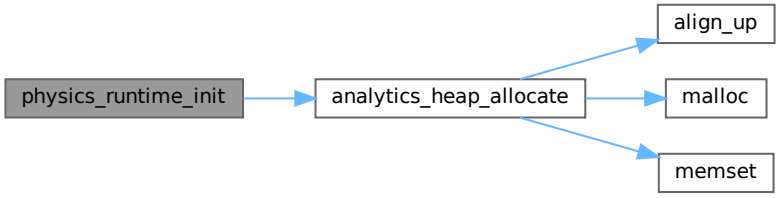
Returns

0 on success, -1 on allocation failure

References analytics_heap_allocate().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

7.215.4.5 physics_runtime_shutdown()

```
void physics_runtime_shutdown (
            void  )
```

Tear down physics runtime services and release heap storage.

Tear down physics runtime services and release heap storage.

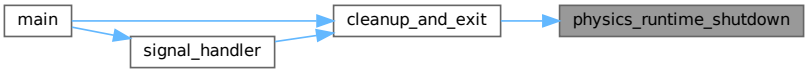
Safe to call when the runtime was never initialised. After this call the analytics heap is gone and no events can be published until `physics_runtime_init()` is called again.

References analytics_heap_release().

Referenced by cleanup_and_exit().

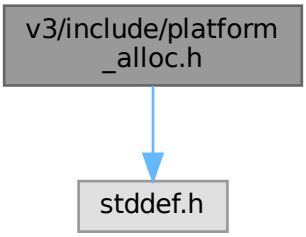
Here is the call graph for this function:

Here is the caller graph for this function:

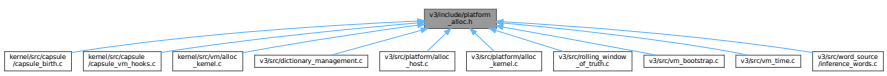


7.216 v3/include/platform_alloc.h File Reference

`#include <stddef.h>`
Include dependency graph for platform_alloc.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct sf_alloc_stats

Typedefs

- typedef struct sf_alloc_stats sf_alloc_stats_t

Functions

- void * sf_malloc (size_t size)

  Allocate `size` bytes from the system heap.
- void * sf_calloc (size_t count, size_t size)

  Allocate a zero-initialised array of `count` elements of `size` bytes each.
- void * sf_realloc (void *ptr, size_t new_size)

  Resize a previously allocated block to `new_size` bytes.
- void sf_free (void *ptr)

  Release a previously allocated block back to the system heap.
- int sf_alloc_init (void)

  Initialise the host allocator statistics.
- void sf_alloc_get_stats (sf_alloc_stats_t *stats)

  Copy the current allocator statistics into `stats`.

7.216.1 Typedef Documentation

7.216.1.1 sf_alloc_stats_t

```
typedef struct sf_alloc_stats sf_alloc_stats_t
```

7.216.2 Function Documentation

7.216.2.1 sf_alloc_get_stats()

```
void sf_alloc_get_stats (
            sf_alloc_stats_t * stats )
```

Copy the current allocator statistics into `stats`.

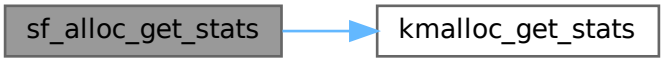
Fills `*stats` with a snapshot of `g_stats`. Note that `used_bytes` overstates the live total when blocks have been freed or reallocated, because freed sizes are not tracked. `free_count` is accurate.

Parameters

stats	Output buffer to populate; no-op if NULL

References sf_alloc_stats::alloc_count, sf_alloc_stats::free_count, g_alloc_count, g_free_count, g_stats, g_stats, kmalloc_get_stats(), kmalloc_stats_t::peak_bytes, sf_alloc_stats::peak_bytes, kmalloc_stats_t::total_bytes, sf_alloc_stats::total_bytes, kmalloc_stats_t::used_bytes, and sf_alloc_stats::used_bytes.

Here is the call graph for this function:



7.216.2.2 sf_alloc_init()

```
int sf_alloc_init (
            void  )
```

Initialise the host allocator statistics.

Resets the `g_stats` struct to zero. Safe to call multiple times; subsequent calls simply reset the counters. No heap memory is allocated by this function — the underlying allocator is the system `malloc`.

Returns

0 always (cannot fail on the hosted path)

Initialise the host allocator statistics.

Resets `g_arena_offset` to zero and clears all `g_stats` fields. Sets `g_initialized` so that `sf_malloc()` skips the lazy-init guard. Safe to call at any point; all prior allocations become invalid after a reset, so this must only be called before any VM allocation takes place.

Returns

0 always (cannot fail)

Initialise the host allocator statistics.

The backing store is now the shared kernel heap (`kmalloc.c`), already initialised at boot (M6) well before any VM allocation can occur – there is nothing left for this function to set up, and "resetting" a heap shared by every other kernel subsystem would be actively wrong. Kept callable (matches its documented "safe to call at any point" contract) so no caller needs to change.

Returns

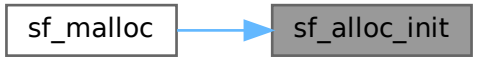
0 always (cannot fail)

References sf_alloc_stats::alloc_count, sf_alloc_stats::free_count, g_arena_offset, g_initialized, g_stats, g_stats, memset(), sf_alloc_stats::peak_bytes, SF_ARENA_SIZE, sf_alloc_stats::total_bytes, and sf_alloc_stats::used_bytes.

Referenced by sf_malloc().

Here is the call graph for this function:

Here is the caller graph for this function:



7.216.2.3 sf_calloc()

```
void * sf_calloc (
            size_t count,
            size_t size )
```

Allocate a zero-initialised array of `count` elements of `size` bytes each.

Delegates to `calloc()`. Updates statistics the same way as `sf_malloc()`: adds `count*size` to `used_bytes`, increments `alloc_count`, and updates `peak_bytes`.

Parameters

count	Number of elements; 0 returns NULL
size	Size of each element in bytes; 0 returns NULL

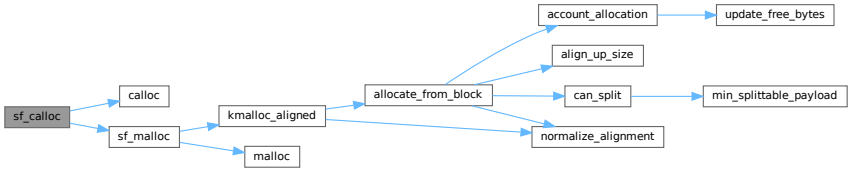
Returns

Pointer to zero-filled memory, or NULL on allocation failure or zero arguments

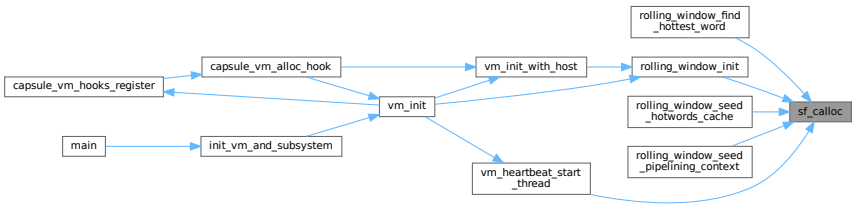
References sf_alloc_stats::alloc_count, calloc(), count, g_stats, sf_alloc_stats::peak_bytes, sf_malloc(), and sf_alloc_stats::used_bytes.

Referenced by rolling_window_find_hottest_word(), rolling_window_init(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), and vm_heartbeat_start_thread().

Here is the call graph for this function:



Here is the caller graph for this function:



7.216.2.4 sf_free()

```
void sf_free (
            void * ptr )
```

Release a previously allocated block back to the system heap.

Delegates to `free()`. Increments `g_stats.free_count` on non-NULL `ptr`. The freed byte count is not subtracted from `used_bytes` because the original size is not tracked — see `sf_realloc()` note.

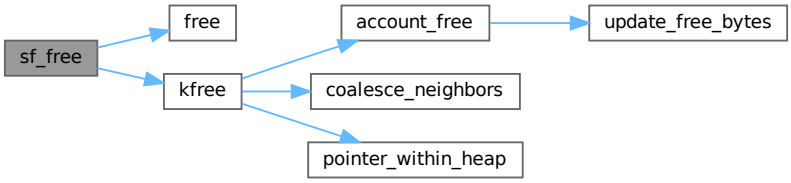
Parameters

ptr	Pointer to free; NULL is a safe no-op

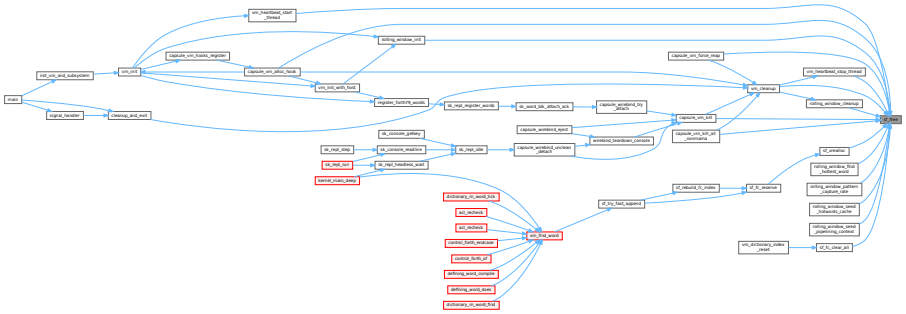
References free(), sf_alloc_stats::free_count, g_free_count, g_stats, g_stats, and kfree().

Referenced by capsule_vm_alloc_hook(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_vm_kill_all_nonmama(), rolling_window_cleanup(), rolling_window_find_hottest_word(), rolling_window_init(), rolling_window_pattern_capture_rate(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), sf_fc_clear_all(), sf_xrealloc(), vm_cleanup(), vm_heartbeat_start_thread(), and vm_heartbeat_stop_thread().

Here is the call graph for this function:



Here is the caller graph for this function:



7.216.2.5 sf_malloc()

```
void * sf_malloc (
            size_t size )
```

Allocate `size` bytes from the system heap.

Delegates to `malloc()`. On success, increments `g_stats.used_bytes` and `g_stats.alloc_count`, and updates `g_stats.peak_bytes` if the new used total exceeds the recorded peak.

Parameters

size	Number of bytes to allocate; 0 returns NULL

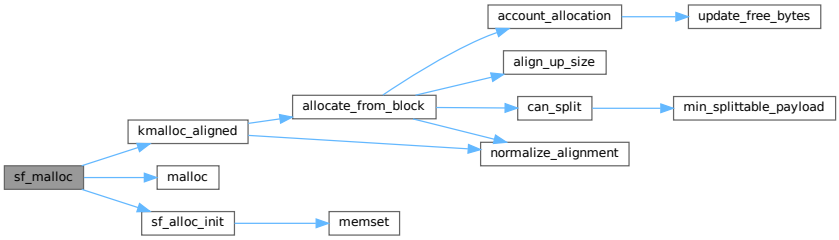
Returns

Pointer to allocated memory, or NULL on allocation failure or zero size

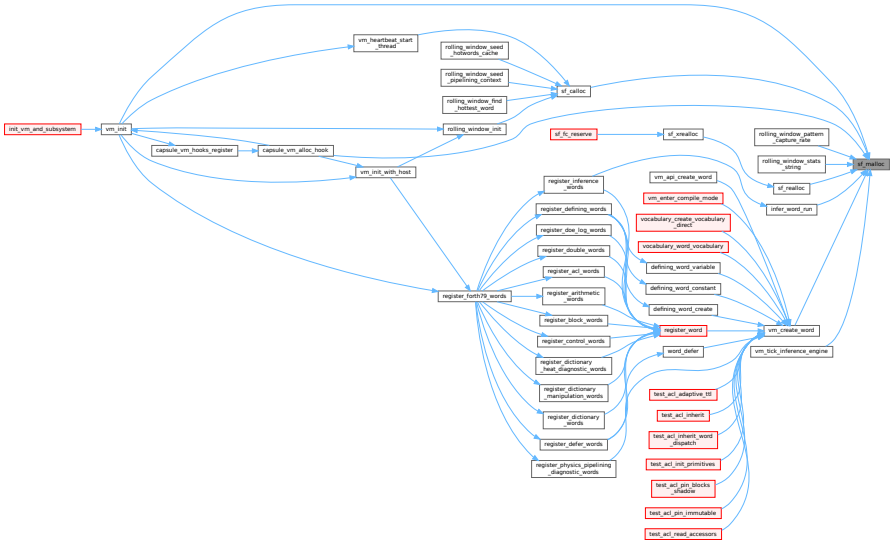
References sf_alloc_stats::alloc_count, g_alloc_count, g_arena_offset, g_initialized, g_stats, g_stats, kmalloc_aligned(), malloc(), sf_alloc_stats::peak_bytes, SF_ALIGN, SF_ALIGN_UP, sf_alloc_init(), SF_ARENA_SIZE, and sf_alloc_stats::used_bytes.

Referenced by capsule_vm_alloc_hook(), infer_word_run(), rolling_window_pattern_capture_rate(), rolling_window_stats_string(), sf_calloc(), sf_realloc(), vm_create_word(), vm_init(), and vm_tick_inference_engine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.216.2.6 sf_realloc()

```
void * sf_realloc (
            void * ptr,
            size_t new_size )
```

Resize a previously allocated block to `new_size` bytes.

Delegates to `realloc()`. On success, increments `g_stats.alloc_count`. The exact byte delta is not tracked because the original allocation size is not stored alongside the pointer — `used_bytes` is therefore approximate after any `sf_realloc()` call. For accurate accounting, wrap allocations with size-tracking headers.

Parameters

ptr	Pointer to an existing allocation, or NULL (behaves like `sf_malloc`)
new_size	Desired new size in bytes; 0 may free the block (platform-defined)

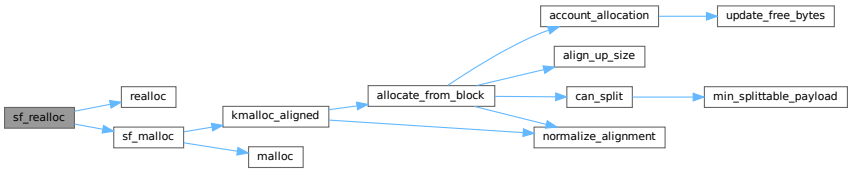
Returns

Pointer to resized memory, or NULL on failure (original block unchanged)

References sf_alloc_stats::alloc_count, g_stats, realloc(), and sf_malloc().

Referenced by sf_xrealloc().

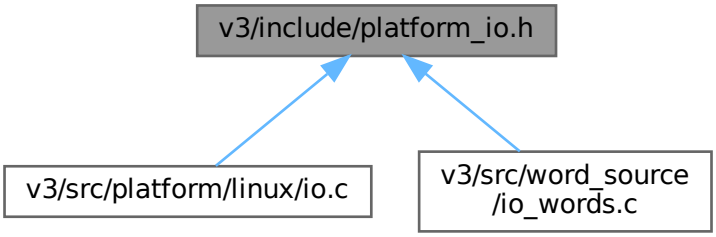
Here is the call graph for this function:



Here is the caller graph for this function:

7.217 v3/include/platform_io.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int sf_terminal_ready (void)

  Non-blocking check for whether a key/character is available to read.

7.217.1 Function Documentation

7.217.1.1 sf_terminal_ready()

```
int sf_terminal_ready (
            void  )
```

Non-blocking check for whether a key/character is available to read.

Must not block and must not consume the byte if one is found (a following KEY/getchar() must still return that same byte).

Returns

1 if input is ready, 0 otherwise

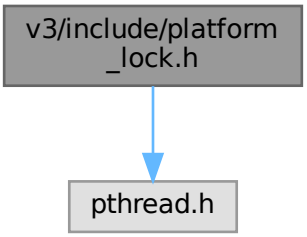
References timeval::tv_sec, and timeval::tv_usec.

Referenced by io_word_question_terminal().

Here is the caller graph for this function:

7.218 v3/include/platform_lock.h File Reference

`#include <pthread.h>`
Include dependency graph for platform_lock.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct sf_mutex

Typedefs

- typedef struct sf_mutex sf_mutex_t

Functions

- int sf_mutex_init (sf_mutex_t *mutex)
  Initialise a POSIX mutex (hosted build).
- void sf_mutex_destroy (sf_mutex_t *mutex)
  Destroy a POSIX mutex (hosted build).
- void sf_mutex_lock (sf_mutex_t *mutex)
  Acquire a POSIX mutex (hosted build).
- void sf_mutex_unlock (sf_mutex_t *mutex)
  Release a POSIX mutex (hosted build).

7.218.1 Typedef Documentation

7.218.1.1 sf_mutex_t

`typedef struct sf_mutex sf_mutex_t`

7.218.2 Function Documentation

7.218.2.1 sf_mutex_destroy()

```
void sf_mutex_destroy (
            sf_mutex_t * mutex )
```

Destroy a POSIX mutex (hosted build).

Delegates to `pthread_mutex_destroy()`. The mutex must not be locked when destroyed. Return value is discarded; callers that need the error code should call `pthread_mutex_destroy()` directly.

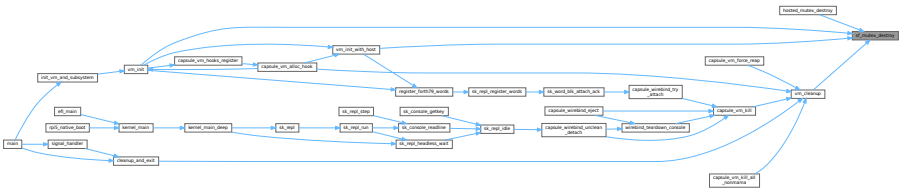
Parameters

mutex	Mutex to destroy; no-op if NULL

References sf_mutex::handle.

Referenced by hosted_mutex_destroy(), vm_cleanup(), vm_init(), and vm_init_with_host().

Here is the caller graph for this function:



7.218.2.2 sf_mutex_init()

```
int sf_mutex_init (
            sf_mutex_t * mutex )
```

Initialise a POSIX mutex (hosted build).

Delegates to `pthread_mutex_init()` with default attributes.

Parameters

mutex	Mutex to initialise; returns -1 if NULL

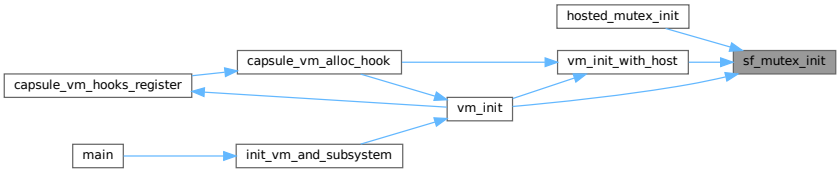
Returns

0 on success, -1 if `mutex` is NULL, or a positive `errno` value propagated from `pthread_mutex_init()`

References sf_mutex::handle.

Referenced by hosted_mutex_init(), vm_init(), and vm_init_with_host().

Here is the caller graph for this function:



7.218.2.3 sf_mutex_lock()

```
void sf_mutex_lock (
            sf_mutex_t * mutex )
```

Acquire a POSIX mutex (hosted build).

Delegates to `pthread_mutex_lock()`. Blocks until the mutex is available. Return value is discarded; a locking error will manifest as a deadlock or data race, not a silent failure.

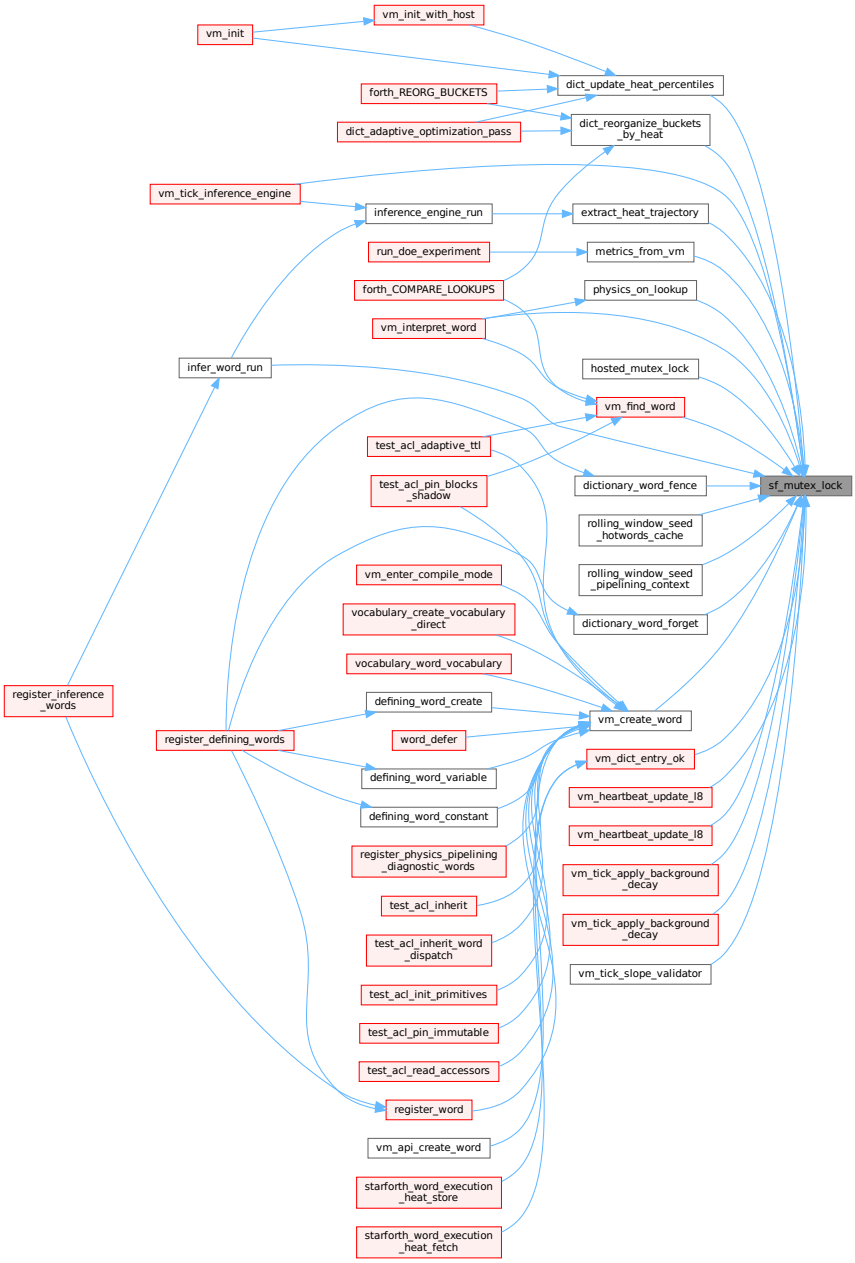
Parameters

mutex	Mutex to lock; no-op if NULL

References sf_mutex::handle.

Referenced by dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), dictionary_word_fence(), dictionary_word_forget(), extract_heat_trajectory(), hosted_mutex_lock(), infer_word_run(), metrics_from_vm(), physics_on_lookup(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), vm_create_word(), vm_dict_entry_ok(), vm_find_word(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_interpret_word(), vm_tick_apply_background_decay(), vm_tick_apply_background_decay(), vm_tick_inference_engine(), and vm_tick_slope_validator().

Here is the caller graph for this function:



7.218.2.4 sf_mutex_unlock()

```
void sf_mutex_unlock (
            sf_mutex_t * mutex )
```

Release a POSIX mutex (hosted build).

Delegates to `pthread_mutex_unlock()`. Must be called by the same thread that acquired the lock. Return value is discarded.

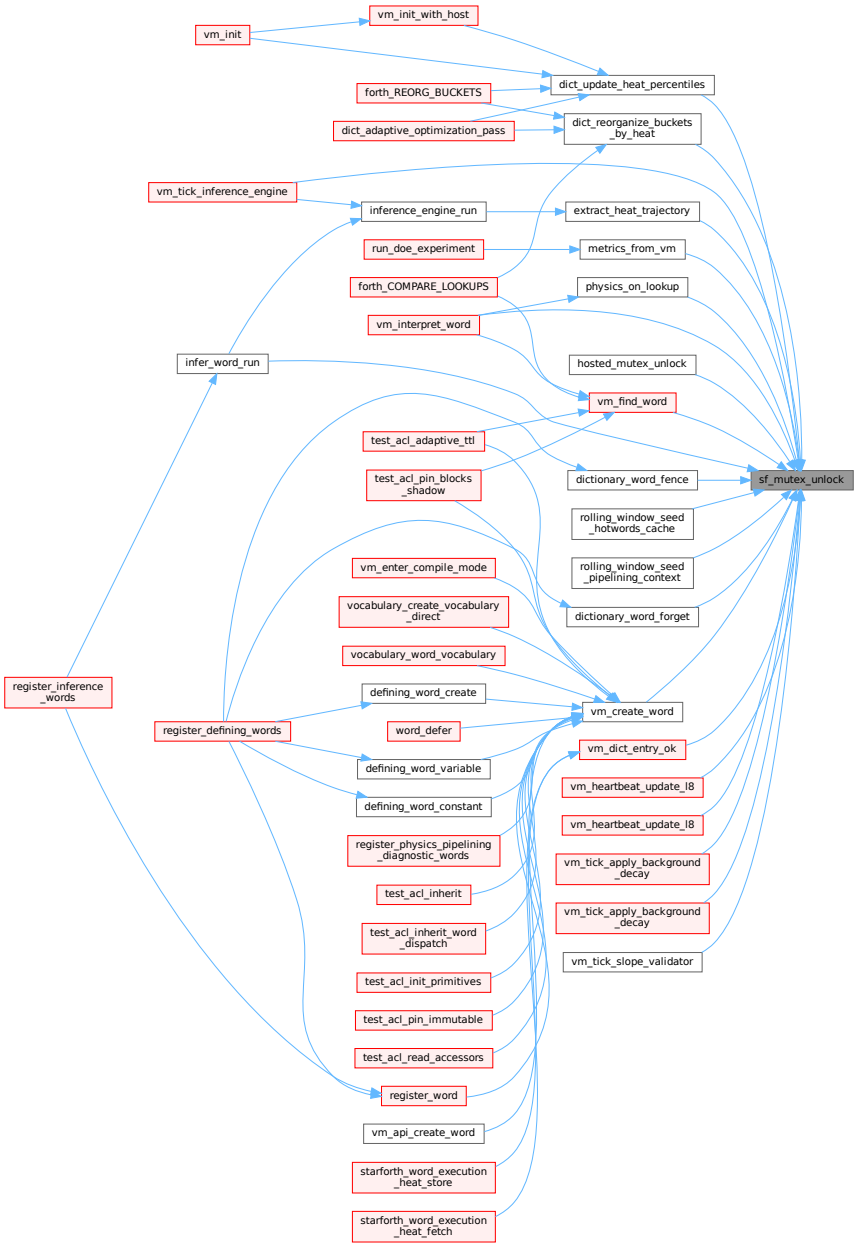
Parameters

mutex	Mutex to unlock; no-op if NULL

References sf_mutex::handle.

Referenced by dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), dictionary_word_fence(), dictionary_word_forget(), extract_heat_trajectory(), hosted_mutex_unlock(), infer_word_run(), metrics_from_vm(), physics_on_lookup(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), vm_create_word(), vm_dict_entry_ok(), vm_find_word(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_interpret_word(), vm_tick_apply_background_decay(), vm_tick_apply_background_decay(), vm_tick_inference_engine(), and vm_tick_slope_validator().

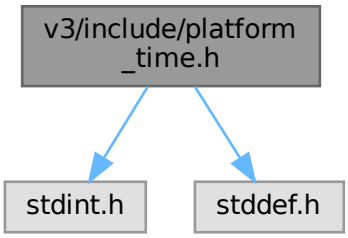
Here is the caller graph for this function:



7.219 v3/include/platform_time.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for platform_time.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct sf_time_backend_t

Macros

- #define SF_TIME_STAMP_SIZE 32

Typedefs

- typedef uint64_t sf_time_ns_t

Functions

- static sf_time_ns_t sf_monotonic_ns (void)
- static sf_time_ns_t sf_realtime_ns (void)
- static int sf_set_realtime_ns (sf_time_ns_t ns)
- static int sf_format_timestamp (sf_time_ns_t ns, char ∗buf, int format_24h)
- static int sf_has_rtc (void)
- static sf_time_ns_t sf_seconds_to_ns (uint64_t seconds)
- static uint64_t sf_ns_to_seconds (sf_time_ns_t ns)
- static uint64_t sf_ns_to_ms (sf_time_ns_t ns)
- static uint64_t sf_ns_to_us (sf_time_ns_t ns)
- void sf_time_init (void)

  Initialize platform time subsystem.

Variables

- const sf_time_backend_t * sf_time_backend
- const sf_time_backend_t sf_time_backend_posix
- const sf_time_backend_t sf_time_backend_l4re

7.219.1 Macro Definition Documentation

7.219.1.1 SF_TIME_STAMP_SIZE

```
#define SF_TIME_STAMP_SIZE 32
```

7.219.2 Typedef Documentation

7.219.2.1 sf_time_ns_t

```
typedef uint64_t sf_time_ns_t
```

7.219.3 Function Documentation

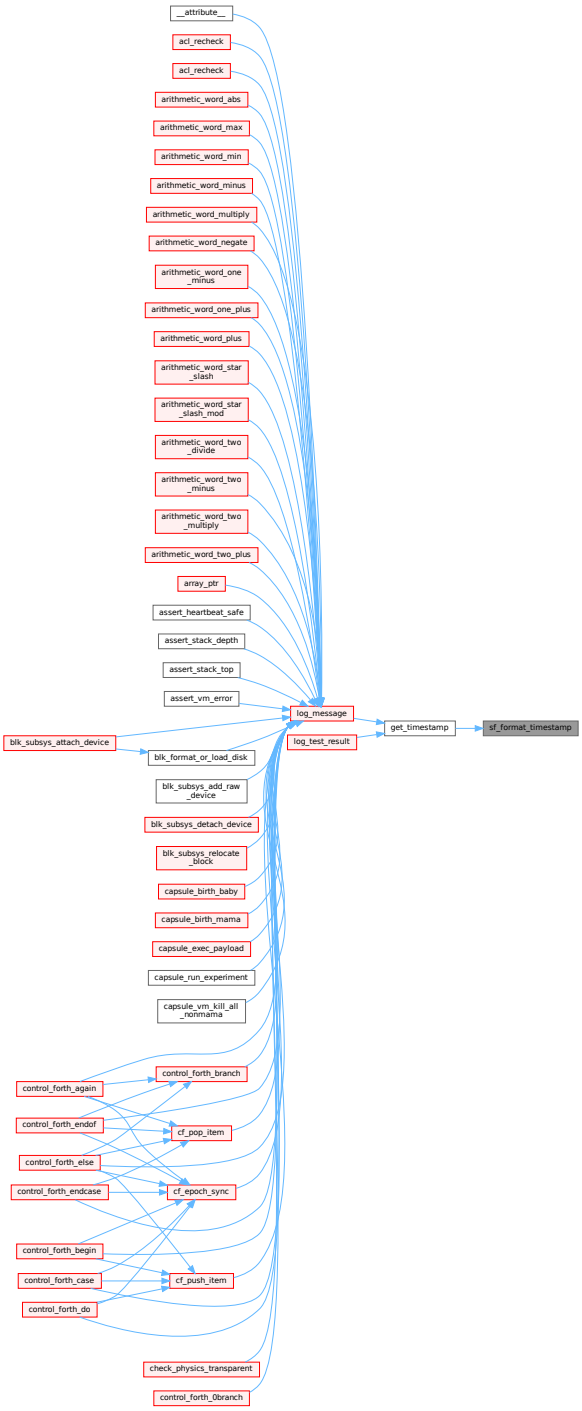
7.219.3.1 sf_format_timestamp()

```
static int sf_format_timestamp (
            sf_time_ns_t ns,
            char * buf,
            int format_24h )  [inline], [static]
```

References sf_time_backend_t::format_timestamp, and sf_time_backend.

Referenced by get_timestamp().

Here is the caller graph for this function:



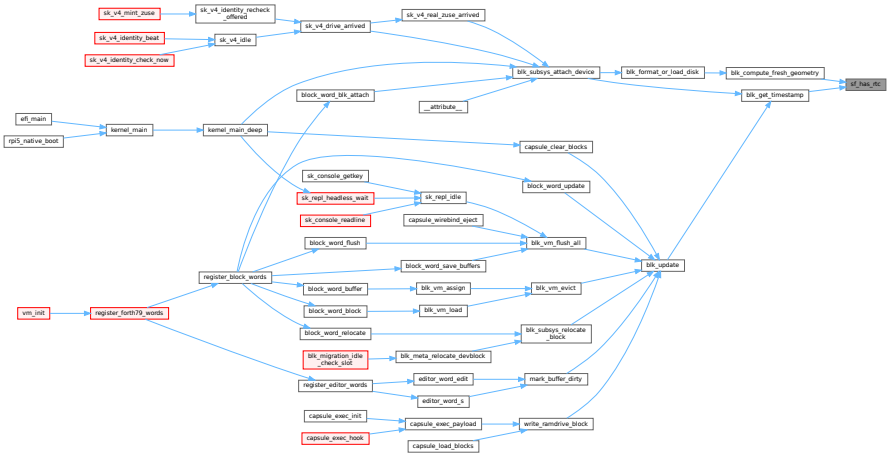
7.219.3.2 sf_has_rtc()

```
static int sf_has_rtc (
            void  )  [inline], [static]
```

References sf_time_backend_t::has_rtc, and sf_time_backend.

Referenced by blk_compute_fresh_geometry(), and blk_get_timestamp().

Here is the caller graph for this function:



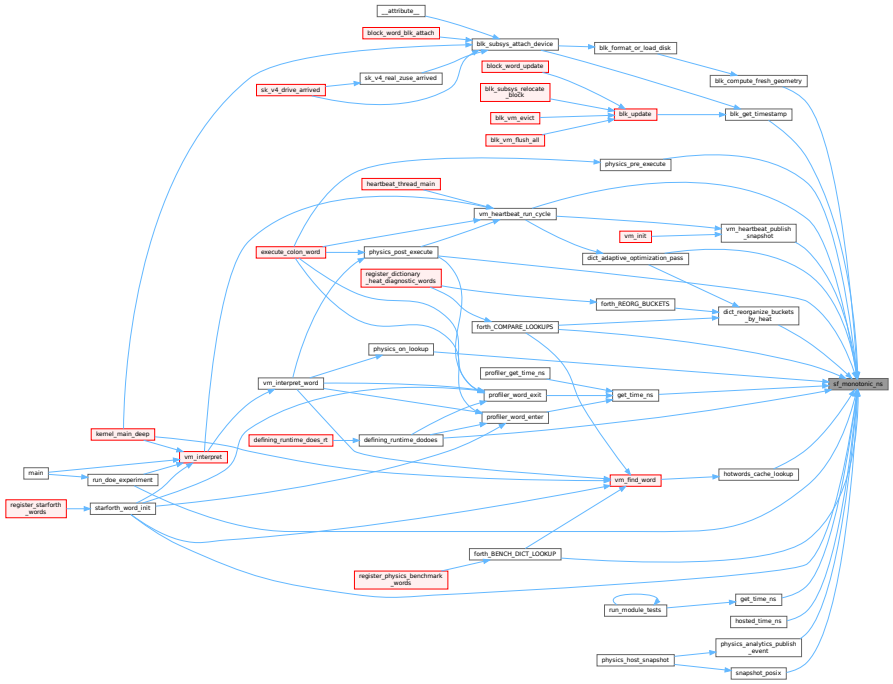
7.219.3.3 sf_monotonic_ns()

```
static sf_time_ns_t sf_monotonic_ns (
            void  )  [inline], [static]
```

References sf_time_backend_t::get_monotonic_ns, and sf_time_backend.

Referenced by blk_compute_fresh_geometry(), blk_get_timestamp(), defining_runtime_dodoes(), dict_adaptive_optimization_pass(), dict_reorganize_buckets_by_heat(), forth_BENCH_DICT_LOOKUP(), forth_COMPARE_LOOKUPS(), get_time_ns(), get_time_ns(), hosted_time_ns(), hotwords_cache_lookup(), physics_analytics_publish_event(), physics_on_lookup(), physics_post_execute(), physics_pre_execute(), run_doe_experiment(), snapshot_posix(), starforth_word_init(), vm_heartbeat_publish_snapshot(), and vm_heartbeat_run_cycle().

Here is the caller graph for this function:



7.219.3.4 sf_ns_to_ms()

```
static uint64_t sf_ns_to_ms (
            sf_time_ns_t ns )  [inline], [static]
```

7.219.3.5 sf_ns_to_seconds()

```
static uint64_t sf_ns_to_seconds (
            sf_time_ns_t ns )  [inline], [static]
```

7.219.3.6 sf_ns_to_us()

```
static uint64_t sf_ns_to_us (
            sf_time_ns_t ns )  [inline], [static]
```

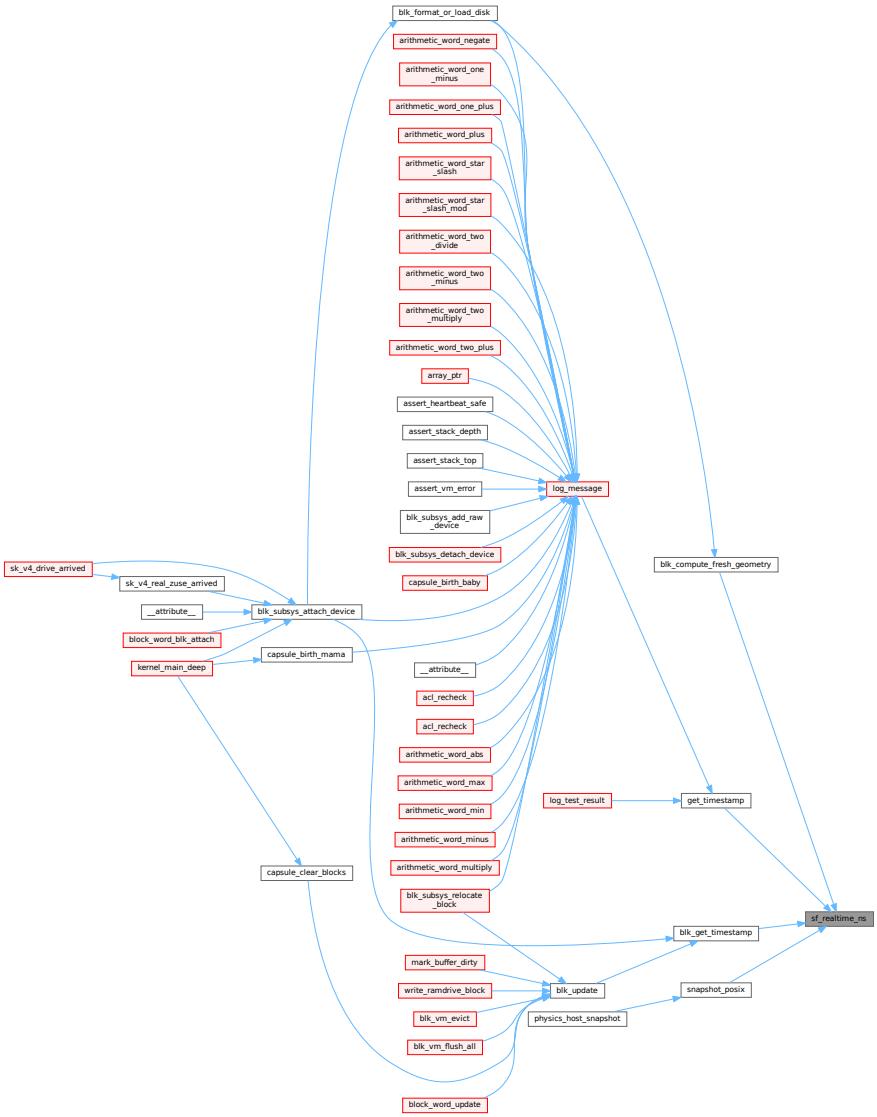
7.219.3.7 sf_realtime_ns()

```
static sf_time_ns_t sf_realtime_ns (
            void  )  [inline], [static]
```

References sf_time_backend_t::get_realtime_ns, and sf_time_backend.

Referenced by blk_compute_fresh_geometry(), blk_get_timestamp(), get_timestamp(), and snapshot_posix().

Here is the caller graph for this function:



7.219.3.8 sf_seconds_to_ns()

```
static sf_time_ns_t sf_seconds_to_ns (
            uint64_t seconds )  [inline], [static]
```

7.219.3.9 sf_set_realtime_ns()

```
static int sf_set_realtime_ns (
            sf_time_ns_t ns )  [inline], [static]
```

References sf_time_backend_t::set_realtime_ns, and sf_time_backend.

7.219.3.10 sf_time_init()

```
void sf_time_init (
            void  )
```

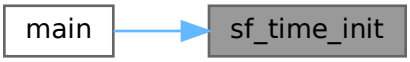
Initialize platform time subsystem.

Selects the POSIX backend (clock_gettime()). Call this once at program startup before using any time functions.

References sf_time_backend, and sf_time_backend_posix.

Referenced by main().

Here is the caller graph for this function:



7.219.4 Variable Documentation

7.219.4.1 sf_time_backend

```
const sf_time_backend_t* sf_time_backend  [extern]
```

Referenced by sf_format_timestamp(), sf_has_rtc(), sf_monotonic_ns(), sf_realtime_ns(), sf_set_realtime_ns(), and sf_time_init().

7.219.4.2 sf_time_backend_l4re

```
const sf_time_backend_t sf_time_backend_l4re  [extern]
```

7.219.4.3 sf_time_backend_posix

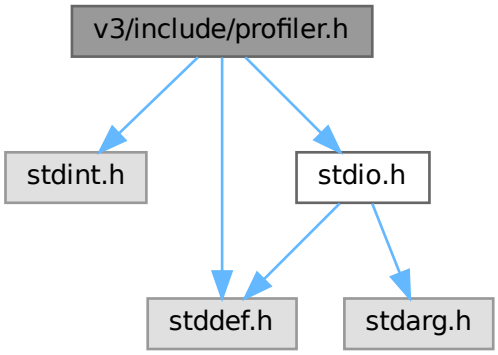
```
const sf_time_backend_t sf_time_backend_posix  [extern]
```

Referenced by sf_time_init().

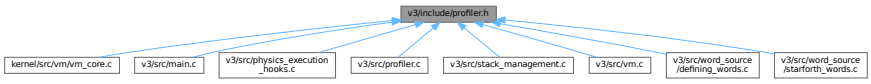
7.220 v3/include/profiler.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <stdio.h>
```

Include dependency graph for profiler.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct ProfileTimer

  Timer for measuring execution duration.
- struct WordProfile

  Execution statistics for a single word.
- struct MemoryProfile
- struct Profiler
- struct ProfileScope

Macros

- #define PROFILE_ENABLED() (0)
- #define PROFILE_START(category) ((void)0)
- #define PROFILE_END(category) ((void)0)
- #define PROFILE_WORD_ENTER(name) ((void)0)
- #define PROFILE_WORD_EXIT(name) ((void)0)
- #define PROFILE_MEMORY_READ(bytes) ((void)0)
- #define PROFILE_MEMORY_WRITE(bytes) ((void)0)
- #define PROFILE_INC_CYCLES(n) ((void)0)
- #define PROFILE_INC_DICT_LOOKUP() ((void)0)
- #define PROFILE_INC_STACK_OP() ((void)0)

Typedefs

- typedef struct VM VM
- typedef struct DictEntry DictEntry

Enumerations

- enum ProfileLevel {
  PROFILE_DISABLED = 0 , PROFILE_BASIC = 1 , PROFILE_DETAILED = 2 , PROFILE_VERBOSE = 3 ,
  PROFILE_FULL = 4 }

  Profiling detail levels.
- enum ProfileCategory {
  PROF_VM_INIT = 0 , PROF_VM_EXECUTE , PROF_VM_COMPILE , PROF_DICT_LOOKUP ,
  PROF_DICT_ADD , PROF_STACK_OPS , PROF_MEMORY_ACCESS , PROF_WORD_EXEC ,
  PROF_IO_OPS , PROF_CONTROL_FLOW , PROF_CATEGORY_COUNT }

  Profiling measurement categories.

Functions

- int profiler_init (ProfileLevel level)

  Initialize the profiling system.
- void profiler_shutdown (void)

  Shutdown and cleanup the profiling system.
- void profiler_reset (void)

  Reset all profiling statistics.
- void profiler_set_level (ProfileLevel level)

  Change the current profiling detail level.
- uint64_t profiler_get_time_ns (void)
- void profiler_start_timer (ProfileCategory category)
- void profiler_end_timer (ProfileCategory category)
- void profiler_word_enter (const DictEntry *entry)

  Start timing a word execution (PROFILE_DETAILED+)
- void profiler_word_exit (const DictEntry *entry)

  End timing a word execution (PROFILE_DETAILED+)
- void profiler_word_count (const DictEntry *entry)

  Track word execution frequency without timing (PROFILE_BASIC+)
- void profiler_memory_read (size_t bytes)
- void profiler_memory_write (size_t bytes)
- void profiler_memory_alloc (size_t bytes)
- void profiler_inc_vm_cycles (uint64_t cycles)
- void profiler_inc_dict_lookup (void)
- void profiler_inc_stack_op (void)
- void profiler_inc_branch (void)
- void profiler_inc_loop_iter (void)
- void profiler_generate_report (void)

  Generate and print detailed profiling report.
- void profiler_print_summary (void)
- void profiler_print_word_analysis (void)
- void profiler_print_memory_analysis (void)
- void profiler_print_hotspots (void)

  Print hot word analysis with optimization recommendations.
- void profiler_analyze_performance (VM *vm)
- void profiler_suggest_optimizations (void)

Variables

- void * g_profiler

7.220.1 Macro Definition Documentation

7.220.1.1 PROFILE_ENABLED

```
#define PROFILE_ENABLED( ) (0)
```

7.220.1.2 PROFILE_END

```
#define PROFILE_END(
              category ) ((void)0)
```

7.220.1.3 PROFILE_INC_CYCLES

```
#define PROFILE_INC_CYCLES(
              n ) ((void)0)
```

7.220.1.4 PROFILE_INC_DICT_LOOKUP

```
#define PROFILE_INC_DICT_LOOKUP( ) ((void)0)
```

7.220.1.5 PROFILE_INC_STACK_OP

```
#define PROFILE_INC_STACK_OP( ) ((void)0)
```

7.220.1.6 PROFILE_MEMORY_READ

```
#define PROFILE_MEMORY_READ(
              bytes ) ((void)0)
```

7.220.1.7 PROFILE_MEMORY_WRITE

```
#define PROFILE_MEMORY_WRITE(
              bytes ) ((void)0)
```

7.220.1.8 PROFILE_START

```
#define PROFILE_START(
              category ) ((void)0)
```

7.220.1.9 PROFILE_WORD_ENTER

```
#define PROFILE_WORD_ENTER(
              name ) ((void)0)
```

7.220.1.10 PROFILE_WORD_EXIT

```
#define PROFILE_WORD_EXIT(
              name ) ((void)0)
```

7.220.2 Typedef Documentation

7.220.2.1 DictEntry

```
typedef struct DictEntry DictEntry
```

7.220.2.2 VM

```
typedef struct VM VM
```

7.220.3 Enumeration Type Documentation

7.220.3.1 ProfileCategory

```
enum ProfileCategory
```

Profiling measurement categories.

Different aspects of VM operation that can be profiled

Enumerator

PROF_VM_INIT	VM initialization.
PROF_VM_EXECUTE	VM execution loop.
PROF_VM_COMPILE	Word compilation.
PROF_DICT_LOOKUP	Dictionary searches.
PROF_DICT_ADD	Dictionary additions.
PROF_STACK_OPS	Stack operations.
PROF_MEMORY_ACCESS	Memory access operations.
PROF_WORD_EXEC	Word execution.
PROF_IO_OPS	I/O operations.
PROF_CONTROL_FLOW	Control flow operations.
PROF_CATEGORY_COUNT	Total number of categories.

7.220.3.2 ProfileLevel

```
enum ProfileLevel
```

Profiling detail levels.

Controls the amount of profiling information collected during execution. Higher levels include all features from lower levels.

Enumerator

PROFILE_DISABLED	Profiling disabled (zero overhead)
PROFILE_BASIC	Word execution frequency tracking (minimal overhead)
PROFILE_DETAILED	Basic + word execution timing (5-10% overhead)
PROFILE_VERBOSE	Detailed + stack/memory access patterns (15-20% overhead)
PROFILE_FULL	Reserved for future use (full instrumentation)

7.220.4 Function Documentation

7.220.4.1 profiler_analyze_performance()

```
void profiler_analyze_performance (
            VM * vm )
```

7.220.4.2 profiler_end_timer()

```
void profiler_end_timer (
            ProfileCategory category )
```

7.220.4.3 profiler_generate_report()

```
void profiler_generate_report (
            void  )
```

Generate and print detailed profiling report.

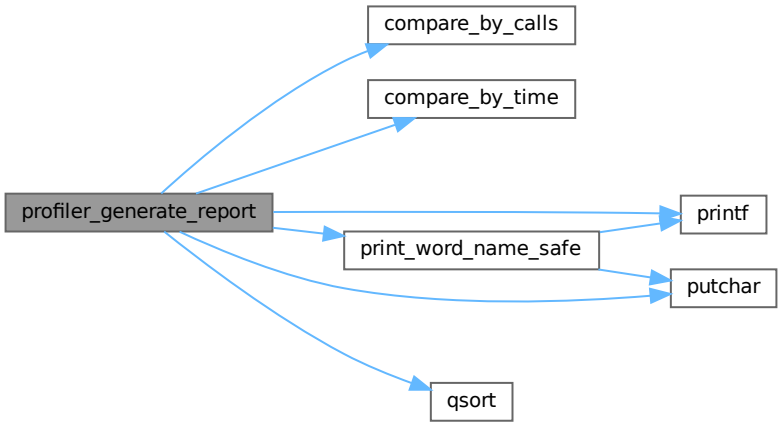
Generates a report including:

- Global statistics (dictionary lookups, stack ops, memory usage)
- Top words by execution time
- Most frequently called words

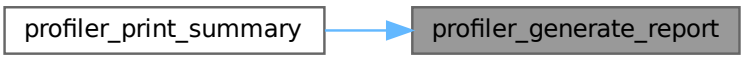
References WordStats::call_count, compare_by_calls(), compare_by_time(), WordStats::entry, limit, WordStats::max_time_ns, name_len, DictEntry::name_len, print_word_name_safe(), printf(), PROFILE_BASIC, PROFILE_DETAILED, PROFILE_VERBOSE, profiler_state, putchar(), qsort(), and WordStats::total_time_ns.

Referenced by profiler_print_summary().

Here is the call graph for this function:



Here is the caller graph for this function:

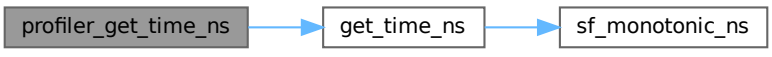


7.220.4.4 profiler_get_time_ns()

```
uint64_t profiler_get_time_ns (
            void  )
```

References get_time_ns().

Here is the call graph for this function:



7.220.4.5 profiler_inc_branch()

```
void profiler_inc_branch (
            void  )
```

7.220.4.6 profiler_inc_dict_lookup()

```
void profiler_inc_dict_lookup (
            void  )
```

References PROFILE_BASIC, and profiler_state.

7.220.4.7 profiler_inc_loop_iter()

```
void profiler_inc_loop_iter (
            void  )
```

7.220.4.8 profiler_inc_stack_op()

```
void profiler_inc_stack_op (
            void  )
```

References PROFILE_VERBOSE, and profiler_state.

7.220.4.9 profiler_inc_vm_cycles()

```
void profiler_inc_vm_cycles (
            uint64_t cycles )
```

7.220.4.10 profiler_init()

```
int profiler_init (
            ProfileLevel level )
```

Initialize the profiling system.

Parameters

level	Initial profiling detail level

Returns

0 on success, non-zero on failure

References calloc(), level, PROFILE_DISABLED, and profiler_state.

Here is the call graph for this function:

7.220.4.11 profiler_memory_alloc()

```
void profiler_memory_alloc (
            size_t bytes )
```

References bytes.

7.220.4.12 profiler_memory_read()

```
void profiler_memory_read (
            size_t bytes )
```

References bytes, PROFILE_VERBOSE, and profiler_state.

7.220.4.13 profiler_memory_write()

```
void profiler_memory_write (
            size_t bytes )
```

References bytes, PROFILE_VERBOSE, and profiler_state.

7.220.4.14 profiler_print_hotspots()

```
void profiler_print_hotspots (
            void  )
```

Print hot word analysis with optimization recommendations.

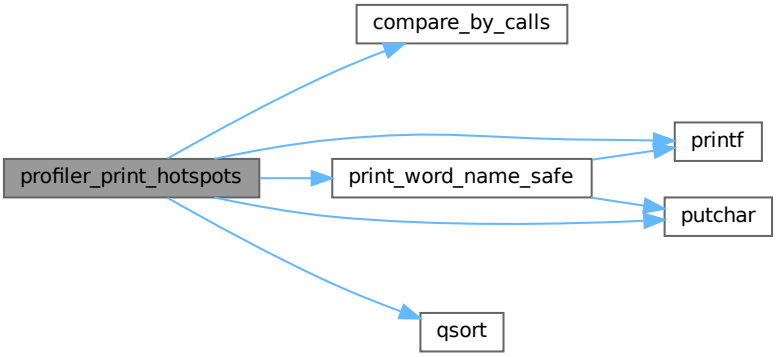
Analyzes word execution frequency and provides actionable recommendations for performance optimization including:

- Assembly optimization candidates
- Inlining opportunities
- Cache-friendly ordering

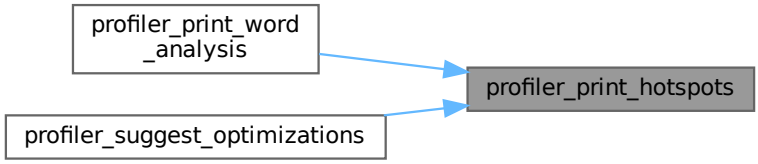
References WordStats::call_count, compare_by_calls(), WordStats::entry, limit, name_len, DictEntry::name_len, print_word_name_safe(), printf(), profiler_state, putchar(), and qsort().

Referenced by profiler_print_word_analysis(), and profiler_suggest_optimizations().

Here is the call graph for this function:



Here is the caller graph for this function:



7.220.4.15 profiler_print_memory_analysis()

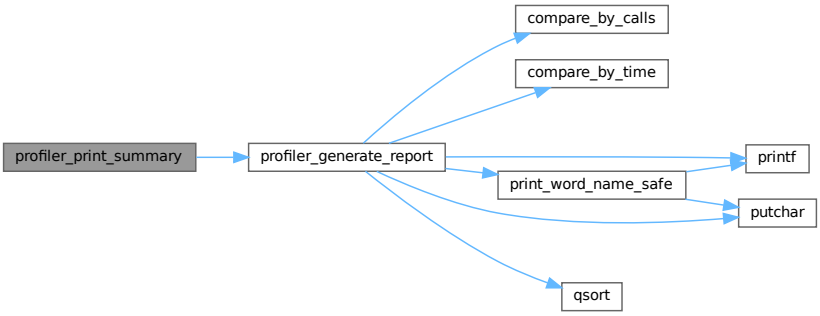
```
void profiler_print_memory_analysis (
            void  )
```

7.220.4.16 profiler_print_summary()

```
void profiler_print_summary (
            void  )
```

References profiler_generate_report().

Here is the call graph for this function:

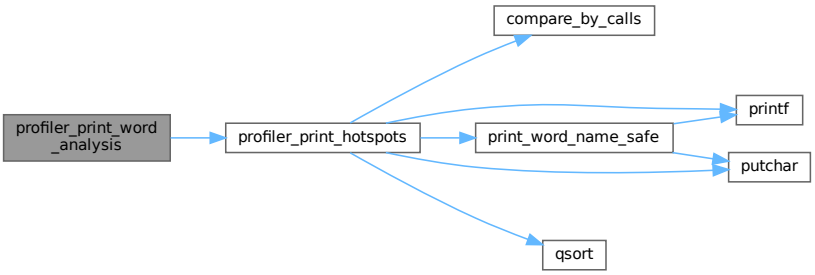


7.220.4.17 profiler_print_word_analysis()

```
void profiler_print_word_analysis (
            void  )
```

References profiler_print_hotspots().

Here is the call graph for this function:



7.220.4.18 profiler_reset()

```
void profiler_reset (
            void  )
```

Reset all profiling statistics.

References profiler_state.

7.220.4.19 profiler_set_level()

```
void profiler_set_level (
            ProfileLevel level )
```

Change the current profiling detail level.

Parameters

level	New profiling level to set

References level, and profiler_state.

7.220.4.20 profiler_shutdown()

```
void profiler_shutdown (
            void  )
```

Shutdown and cleanup the profiling system.

Shutdown and cleanup the profiling system.

References free(), and profiler_state.

Here is the call graph for this function:

7.220.4.21 profiler_start_timer()

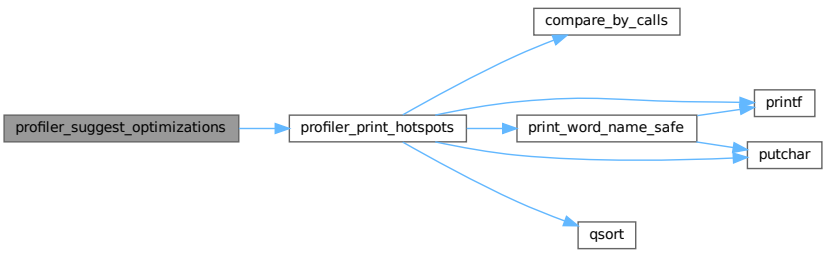
```
void profiler_start_timer (
            ProfileCategory category )
```

7.220.4.22 profiler_suggest_optimizations()

```
void profiler_suggest_optimizations (
            void  )
```

References profiler_print_hotspots().

Here is the call graph for this function:



7.220.4.23 profiler_word_count()

```
void profiler_word_count (
            const DictEntry * entry )
```

Track word execution frequency without timing (PROFILE_BASIC+)

Parameters

entry	Dictionary entry of the word being executed

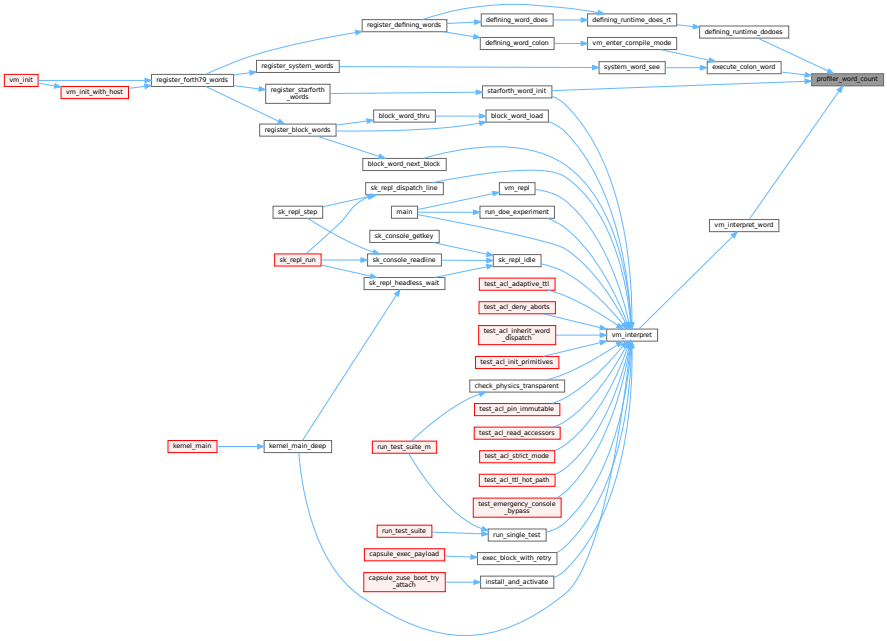
This is a lightweight alternative to profiler_word_enter/exit that only increments call counters without timing overhead. Ideal for identifying hot words (frequently executed) for optimization.

References WordStats::call_count, entry, get_word_stats(), PROFILE_DETAILED, and profiler_state.

Referenced by defining_runtime_dodoes(), execute_colon_word(), starforth_word_init(), and vm_interpret_word().

Here is the call graph for this function:

Here is the caller graph for this function:



7.220.4.24 profiler_word_enter()

```
void profiler_word_enter (
            const DictEntry * entry )
```

Start timing a word execution (PROFILE_DETAILED+)

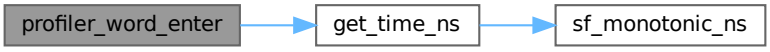
Parameters

entry	Dictionary entry of the word being executed

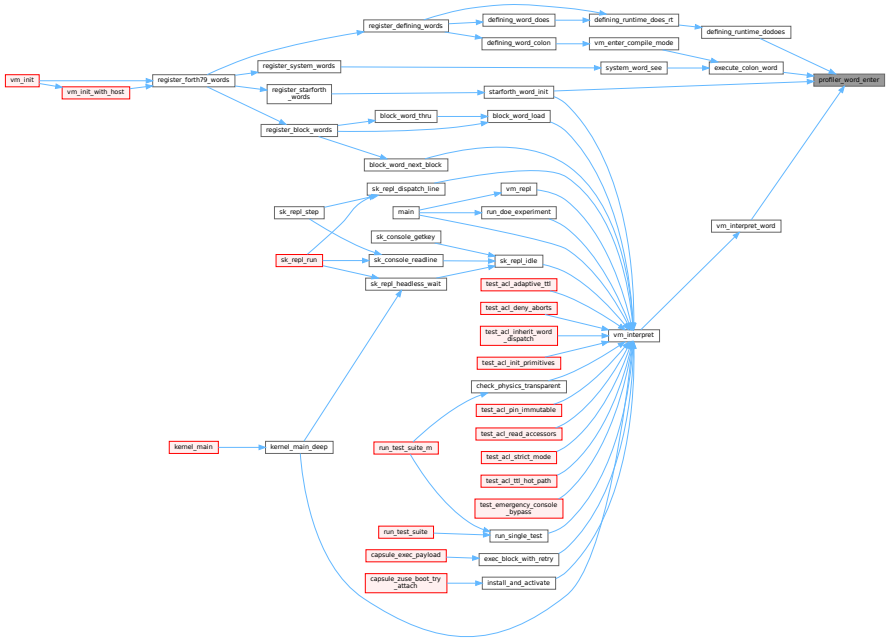
References entry, get_time_ns(), PROFILE_DETAILED, and profiler_state.

Referenced by defining_runtime_dodoes(), execute_colon_word(), starforth_word_init(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.220.4.25 profiler_word_exit()

```
void profiler_word_exit (
            const DictEntry * entry )
```

End timing a word execution (PROFILE_DETAILED+)

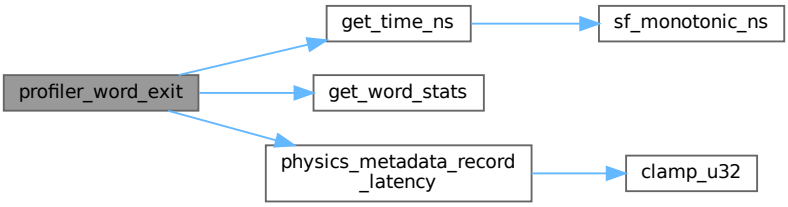
Parameters

entry	Dictionary entry of the word being executed

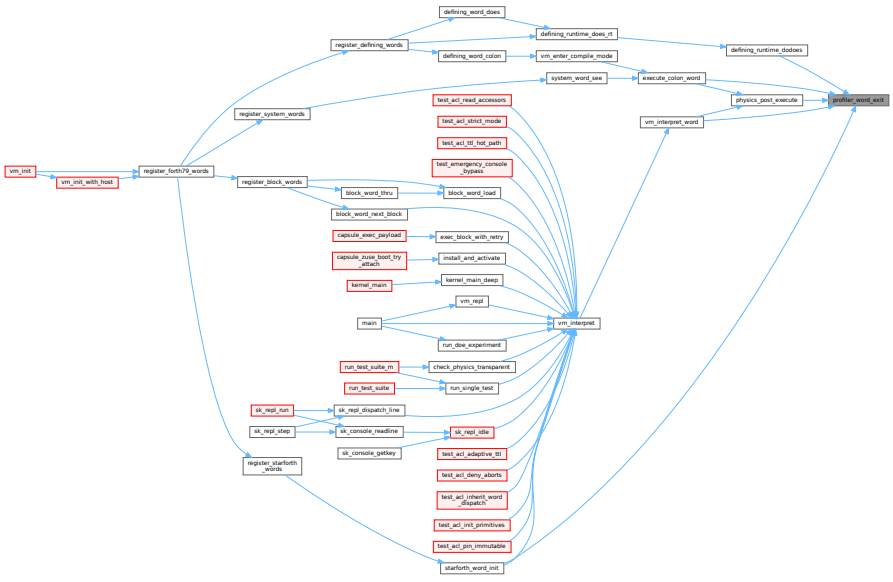
References WordStats::call_count, entry, get_time_ns(), get_word_stats(), WordStats::max_time_ns, WordStats::min_time_ns, physics_metadata_record_latency(), PROFILE_DETAILED, profiler_state, and WordStats::total_time_ns.

Referenced by defining_runtime_dodoes(), execute_colon_word(), physics_post_execute(), starforth_word_init(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.220.5 Variable Documentation

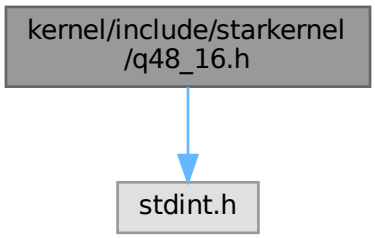
7.220.5.1 g_profiler

```
void* g_profiler  [extern]
```

7.221 kernel/include/starkernel/q48_16.h File Reference

```
#include <stdint.h>
```

Include dependency graph for q48_16.h:



This graph shows which files directly or indirectly include this file:

Macros

- #define Q48_ONE ((q48_16_t)0x10000ULL)

Typedefs

- typedef uint64_t q48_16_t

  q48_16.h - Q48.16 Fixed-Point Arithmetic for StarKernel

Functions

- q48_16_t q48_mul (q48_16_t a, q48_16_t b)

  Multiply two Q48.16 fixed-point values.
- q48_16_t q48_div (q48_16_t a, q48_16_t b)

  Divide two Q48.16 fixed-point values.
- static q48_16_t q48_add (q48_16_t a, q48_16_t b)

  Add two Q48.16 values.
- static q48_16_t q48_sub (q48_16_t a, q48_16_t b)

  Subtract two Q48.16 values.
- static q48_16_t q48_abs (q48_16_t a)

  Absolute value of Q48.16 (treats as unsigned, so just returns a)
- static q48_16_t q48_from_u64 (uint64_t u)

  Convert unsigned 64-bit integer to Q48.16: u << 16.
- static uint64_t q48_to_u64 (q48_16_t q)

Convert Q48.16 to a 64-bit integer (truncate fractional): q >> 16, arithmetic (signed) shift so negative q sign-extends correctly instead of producing garbage from an unsigned logical shift.

- q48_16_t q48_log_approx (q48_16_t x)

  Compute an approximation of `ln(x)` in Q48.16 format (integer-only).
- q48_16_t q48_exp_approx (q48_16_t q)

  Compute an approximation of `e^q` in Q48.16 format (Taylor series).
- q48_16_t q48_sqrt_approx (q48_16_t q)

  Compute an approximation of `sqrt(q)` in Q48.16 format (Newton-Raphson).
- q48_16_t q48_sin_approx (q48_16_t q)

  Compute an approximation of `sin(q)` in Q48.16 format (radians).
- q48_16_t q48_cos_approx (q48_16_t q)

  Compute an approximation of `cos(q)` in Q48.16 format (radians).

7.221.1 Macro Definition Documentation

7.221.1.1 Q48_ONE

```
#define Q48_ONE ((q48_16_t)0x10000ULL)
```

7.221.2 Typedef Documentation

7.221.2.1 q48_16_t

```
typedef uint64_t q48_16_t
```

q48_16.h - Q48.16 Fixed-Point Arithmetic for StarKernel

Format: uint64_t with fixed decimal point after bit 15

- Bits 0-15: Fractional part (1/65536 resolution)
- Bits 16-63: Integer part (up to 2^48-1)

Example: 0x00010000 = 1.0, 0x00018000 = 1.5

All operations are integer-only. NO FLOATING-POINT.

7.221.3 Function Documentation

7.221.3.1 q48_abs()

```
static q48_16_t q48_abs (
            q48_16_t a )  [inline], [static]
```

Absolute value of Q48.16 (treats as unsigned, so just returns a)

7.221.3.2 q48_add()

```
static q48_16_t q48_add (
            q48_16_t a,
            q48_16_t b )  [inline], [static]
```

Add two Q48.16 values.

7.221.3.3 q48_cos_approx()

```
q48_16_t q48_cos_approx (
            q48_16_t q )
```

Compute an approximation of `cos(q)` in Q48.16 format (radians).

Range-reduces into [-pi, pi] via `q48_reduce_angle()`, then sums the Taylor series `1` - x^2/2! + x^4/4! - ... for terms n = 2,4,6,8,10, with an early exit when a term falls below 10. Cosine is even (`cos`(-x) = cos(x)), so no sign correction is needed – `x^2` already absorbs the sign of `x`.

Parameters

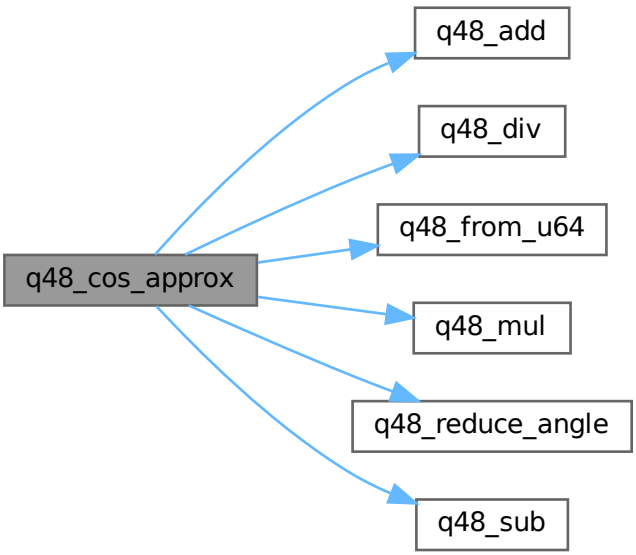
q	Angle in Q48.16 format (radians, any magnitude)

Returns

`cos(q)` in Q48.16 format

References q48_add(), q48_div(), q48_from_u64(), q48_mul(), Q48_ONE, q48_reduce_angle(), and q48_sub().

Here is the call graph for this function:



7.221.3.4 q48_div()

```
q48_16_t q48_div (
            q48_16_t a,
            q48_16_t b )
```

Divide two Q48.16 fixed-point values.

Computes (a << 16) / b. Returns 0 if b == 0 (safe, but callers should guard). Returns UINT64_MAX as a saturation sentinel when a exceeds 2^48, which would overflow the 16-bit left-shift.

Parameters

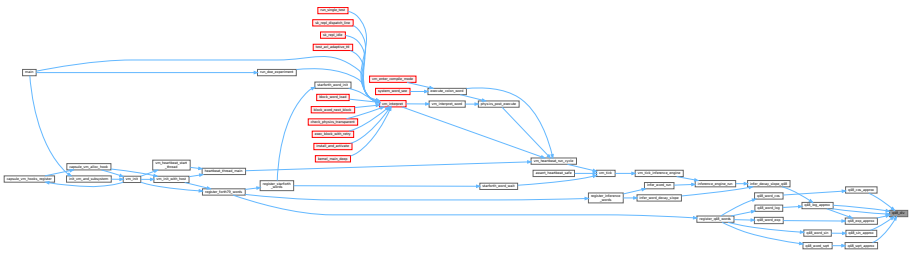
a	Dividend in Q48.16 format
b	Divisor in Q48.16 format; 0 returns 0

Returns

Quotient in Q48.16 format, or UINT64_MAX on saturation

Referenced by q48_cos_approx(), q48_exp_approx(), q48_log_approx(), q48_log_approx(), q48_sin_approx(), and q48_sqrt_approx().

Here is the caller graph for this function:



7.221.3.5 q48_exp_approx()

```
q48_16_t q48_exp_approx (
            q48_16_t q )
```

Compute an approximation of e^q in Q48.16 format (Taylor series).

Supporting function for q48_log_approx() Newton iterations. Uses the Taylor series 1 + x + x^2/2! + ... up to 10 terms, with an early-exit when a term falls below 50 (sub-LSB in Q48.16). Handles negative q by computing e^|q| then returning 1/result. Returns 1.0 (65536) for q == 0, UINT64_MAX for q >= 16.0, and 0 for q <= -16.0.

Parameters

q	Exponent in Q48.16 format (signed interpretation)

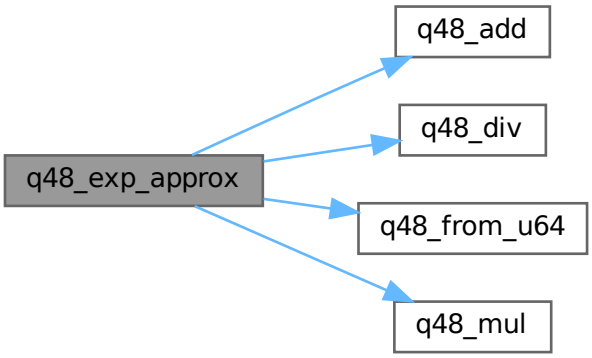
Returns

e^q in Q48.16 format

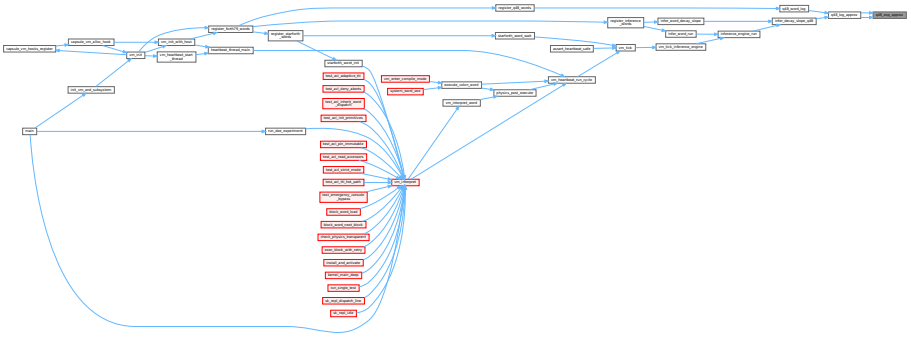
References q48_add(), q48_div(), q48_from_u64(), q48_mul(), and Q48_ONE.

Referenced by q48_log_approx(), and q48_log_approx().

Here is the call graph for this function:



Here is the caller graph for this function:



7.221.3.6 q48_from_u64()

```
static q48_16_t q48_from_u64 (
            uint64_t u )  [inline], [static]
```

Convert unsigned 64-bit integer to Q48.16: u << 16.

7.221.3.7 q48_log_approx()

```
q48_16_t q48_log_approx (
            q48_16_t x )
```

Compute an approximation of `ln(x)` in Q48.16 format (integer-only).

Used by `infer_decay_slope_q48()` (Loop #6) for log-linear OLS regression. Algorithm: range-reduces `x` to the interval [1, 2) by tracking the exponent `k` (so `x` = 2^k * m), applies 6 Newton-Raphson iterations on `e^y` = m starting from the first-order approximation `y_0` = m - 1, then assembles the result as `ln(x)` = k*ln(2) + ln(m). Returns 0 for `x` == 0 (ln undefined) and for `x` == 65536 (ln(1.0) == 0.0 exactly). `LN2_Q48` = 45426 (0.693147... × 65536).

Parameters

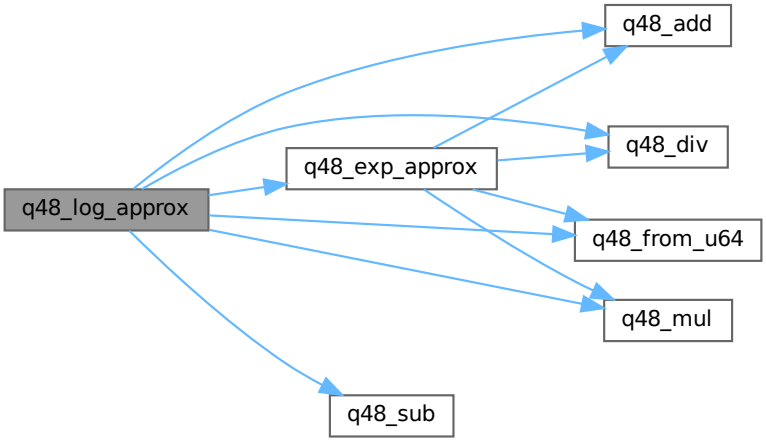
x	Input value in Q48.16 format (uint64_t; value of 65536 = 1.0)

Returns

`ln(x)` in Q48.16 format, or 0 for x==0 or x==65536

References q48_add(), q48_div(), q48_exp_approx(), q48_from_u64(), q48_mul(), Q48_ONE, and q48_sub().

Here is the call graph for this function:



7.221.3.8 q48_mul()

```
q48_16_t q48_mul (
            q48_16_t a,
            q48_16_t b )
```

Multiply two Q48.16 fixed-point values.

Computes (a * b) >> 16 to maintain the Q48.16 scale. Uses `__uint128_t` on GCC/Clang (128-bit intermediate, no precision loss) and falls back to a manual 64-bit four-partial-product path with carry tracking on platforms that lack `__SIZEOF_INT128__`.

Parameters

a	First operand in Q48.16 format
b	Second operand in Q48.16 format

Returns

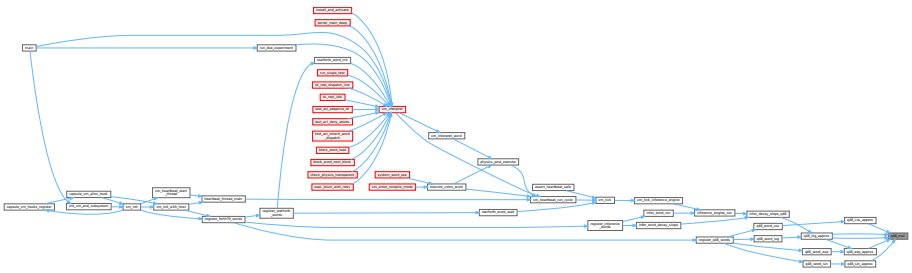
Product in Q48.16 format (truncated, not rounded)

Multiply two Q48.16 fixed-point values.

Freestanding implementation: no libc, no floating-point.

Referenced by q48_cos_approx(), q48_exp_approx(), q48_log_approx(), q48_log_approx(), and q48_sin_approx().

Here is the caller graph for this function:



7.221.3.9 q48_sin_approx()

```
q48_16_t q48_sin_approx (
            q48_16_t q )
```

Compute an approximation of `sin(q)` in Q48.16 format (radians).

Range-reduces into [-pi, pi] via `q48_reduce_angle()`, then sums the Taylor series `x` - x^3/3! + x^5/5! - ... for terms n = 3,5,7,9,11, with an early exit when a term falls below 10 (sub-LSB in Q48.16). Sine is odd (`sin`(-x) = -sin(x)), so the sign is factored out before the series and reapplied to the result.

Parameters

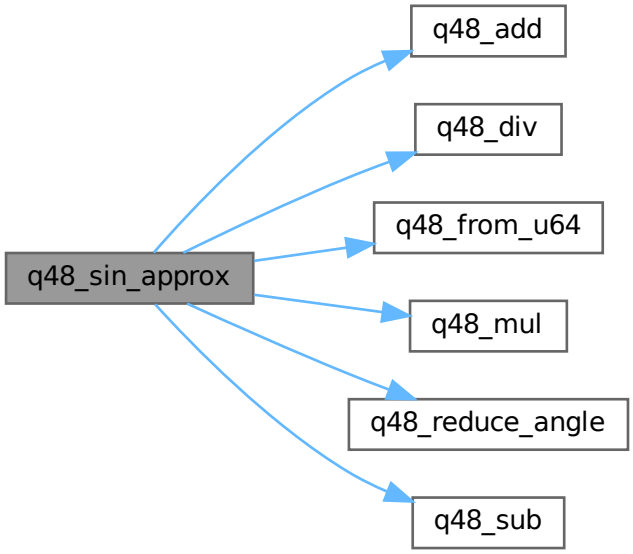
q	Angle in Q48.16 format (radians, any magnitude)

Returns

`sin(q)` in Q48.16 format

References q48_add(), q48_div(), q48_from_u64(), q48_mul(), q48_reduce_angle(), and q48_sub().

Here is the call graph for this function:



7.221.3.10 q48_sqrt_approx()

```
q48_16_t q48_sqrt_approx (
            q48_16_t q )
```

Compute an approximation of `sqrt(q)` in Q48.16 format (Newton-Raphson).

Iterates `x_{n+1}` = (x_n + q/x_n) / 2 starting from (q >> 1) + 16384, for up to 8 iterations. Terminates early when the delta between successive estimates falls below 10 (approximately 0.00015 in Q48.16). Used for R^2 computation in the inference engine. Returns 0 for `q` == 0, and 1.0 (65536) for `q` == 65536.

Parameters

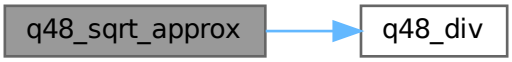
q	Input value in Q48.16 format

Returns

`sqrt(q)` in Q48.16 format

References q48_div(), and Q48_ONE.

Here is the call graph for this function:



7.221.3.11 q48_sub()

```
static q48_16_t q48_sub (
            q48_16_t a,
            q48_16_t b )  [inline], [static]
```

Subtract two Q48.16 values.

7.221.3.12 q48_to_u64()

```
static uint64_t q48_to_u64 (
            q48_16_t q )  [inline], [static]
```

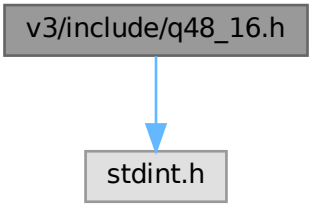
Convert Q48.16 to a 64-bit integer (truncate fractional): q >> 16, arithmetic (signed) shift so negative q sign-extends correctly instead of producing garbage from an unsigned logical shift.

Bit-identical to the old behavior for non-negative q.

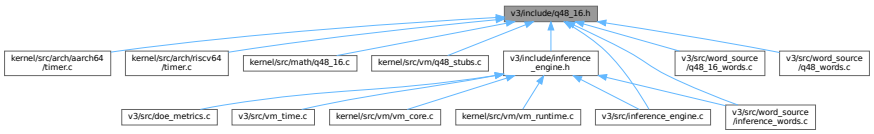
7.222 v3/include/q48_16.h File Reference

```
#include <stdint.h>
```

Include dependency graph for q48_16.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef uint64_t q48_16_t

Functions

- q48_16_t q48_mul (q48_16_t a, q48_16_t b)

  Multiply two Q48.16 fixed-point values.
- q48_16_t q48_div (q48_16_t a, q48_16_t b)

  Divide two Q48.16 fixed-point values.
- static q48_16_t q48_add (q48_16_t a, q48_16_t b)

  Add two Q48.16 values.
- static q48_16_t q48_sub (q48_16_t a, q48_16_t b)

  Subtract two Q48.16 values.
- static q48_16_t q48_abs (q48_16_t a)

  Absolute value of Q48.16.
- static q48_16_t q48_from_u64 (uint64_t u)

  Convert unsigned 64-bit integer to Q48.16.
- static uint64_t q48_to_u64 (q48_16_t q)

  Convert Q48.16 to a 64-bit integer (truncate fractional)
- q48_16_t q48_from_double (double d)

  Convert a `double` to Q48.16 format.
- double q48_to_double (q48_16_t q)

  Convert a Q48.16 value to `double`.
- q48_16_t q48_log_approx (uint64_t x)

  Compute an approximation of `ln(x)` in Q48.16 format (integer-only).
- q48_16_t q48_exp_approx (q48_16_t q)

  Compute an approximation of `e^q` in Q48.16 format (Taylor series).
- q48_16_t q48_sqrt_approx (q48_16_t q)

  Compute an approximation of `sqrt(q)` in Q48.16 format (Newton-Raphson).
- q48_16_t q48_sin_approx (q48_16_t q)

  Compute an approximation of `sin(q)` in Q48.16 format (radians).
- q48_16_t q48_cos_approx (q48_16_t q)

  Compute an approximation of `cos(q)` in Q48.16 format (radians).
- const char * q48_to_string (q48_16_t q)

  Format a Q48.16 value as a decimal string `"INTEGER.FRAC"`.
- int q48_is_valid (q48_16_t q)

  Test whether a Q48.16 value is valid.

7.222.1 Typedef Documentation

7.222.1.1 q48_16_t

```
typedef uint64_t q48_16_t
```

7.222.2 Function Documentation

7.222.2.1 q48_abs()

```
static q48_16_t q48_abs (
            q48_16_t a )  [inline], [static]
```

Absolute value of Q48.16.

Parameters

a	Q48.16 value

Returns

|a| in Q48.16 format

Referenced by q48_word_abs().

Here is the caller graph for this function:

7.222.2.2 q48_add()

```
static q48_16_t q48_add (
            q48_16_t a,
            q48_16_t b )  [inline], [static]
```

Add two Q48.16 values.

Math: (a / 2^16) + (b / 2^16) = (a + b) / 2^16 Returns: a + b (no shift needed, both have same scaling)

Parameters

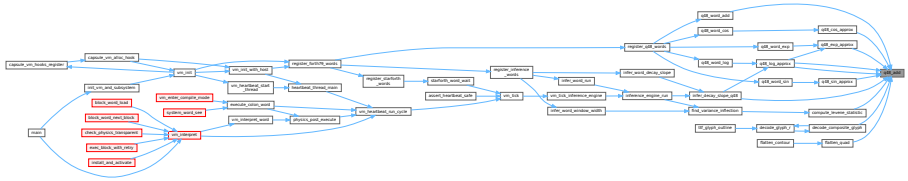
a	First Q48.16 operand
b	Second Q48.16 operand

Returns

a + b in Q48.16 format

Referenced by compute_levene_statistic(), decode_composite_glyph(), flatten_quad(), infer_decay_slope_q48(), q48_cos_approx(), q48_exp_approx(), q48_log_approx(), q48_log_approx(), q48_sin_approx(), and q48_word_add().

Here is the caller graph for this function:



7.222.2.3 q48_cos_approx()

```
q48_16_t q48_cos_approx (
            q48_16_t q )
```

Compute an approximation of `cos(q)` in Q48.16 format (radians).

Range-reduces into [-pi, pi] via `q48_reduce_angle()`, then sums the Taylor series `1 - x^2/2! + x^4/4! - ...` for terms n = 2,4,6,8,10, with an early exit when a term falls below 10. Cosine is even (`cos(-x) = cos(x)`), so no sign correction is needed – `x^2` already absorbs the sign of `x`.

Parameters

q	Angle in Q48.16 format (radians, any magnitude)

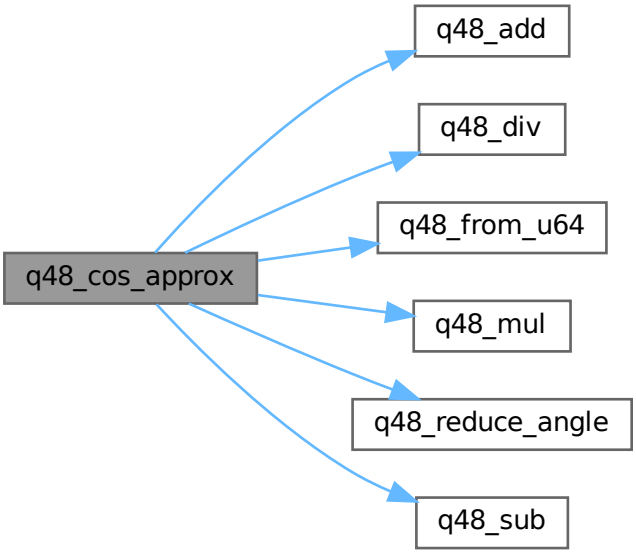
Returns

`cos(q)` in Q48.16 format

References q48_add(), q48_div(), q48_from_u64(), q48_mul(), q48_reduce_angle(), and q48_sub().

Referenced by q48_word_cos().

Here is the call graph for this function:



Here is the caller graph for this function:

7.222.2.4 q48_div()

```
q48_16_t q48_div (
            q48_16_t a,
            q48_16_t b )
```

Divide two Q48.16 fixed-point values.

Computes (a << 16) / b. Returns 0 if b == 0 (safe, but callers should guard). Returns UINT64_MAX as a saturation sentinel when a exceeds 2^48, which would overflow the 16-bit left-shift.

Parameters

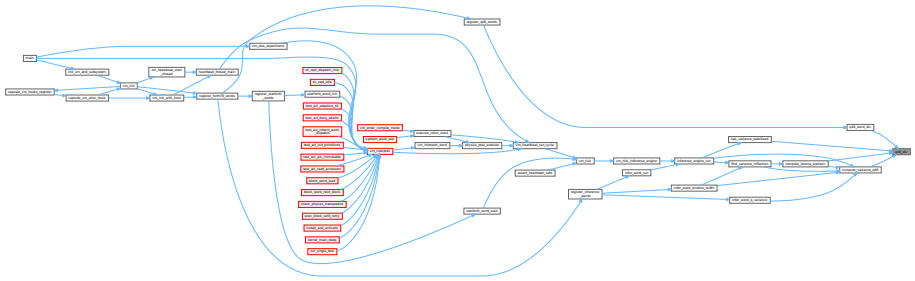
a	Dividend in Q48.16 format
b	Divisor in Q48.16 format; 0 returns 0

Returns

Quotient in Q48.16 format, or UINT64_MAX on saturation

Referenced by compute_levene_statistic(), compute_variance_q48(), has_variance_stabilized(), and q48_word_div().

Here is the caller graph for this function:



7.222.2.5 q48_exp_approx()

```
q48_16_t q48_exp_approx (
            q48_16_t q )
```

Compute an approximation of e^q in Q48.16 format (Taylor series).

Supporting function for `q48_log_approx()` Newton iterations. Uses the Taylor series 1 + x + x^2/2! + ... up to 10 terms, with an early-exit when a term falls below 50 (sub-LSB in Q48.16). Handles negative q by computing e^|q| then returning `1/result`. Returns 1.0 (65536) for q == 0, `UINT64_MAX` for q >= 16.0, and 0 for q <= -16.0.

Parameters

q	Exponent in Q48.16 format (signed interpretation)

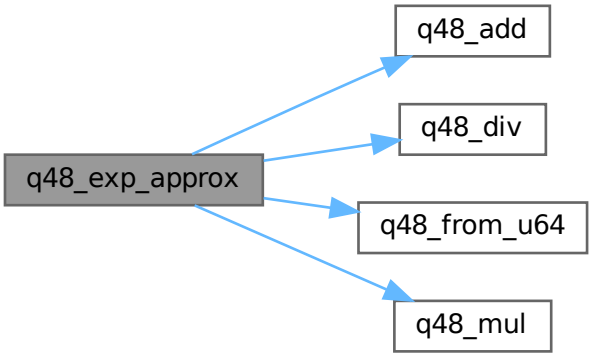
Returns

e^q in Q48.16 format

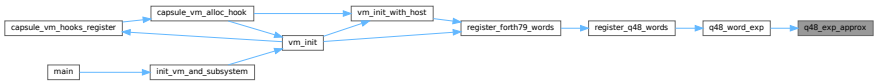
References q48_add(), q48_div(), q48_from_u64(), and q48_mul().

Referenced by q48_word_exp().

Here is the call graph for this function:



Here is the caller graph for this function:



7.222.2.6 q48_from_double()

```
q48_16_t q48_from_double (
            double d )
```

Convert a `double` to Q48.16 format.

Multiplies `d` by 65536.0 and rounds to nearest. Intended for testing and diagnostics only — not called in the inference engine hot path to avoid floating-point dependency.

Parameters

d	Value to convert

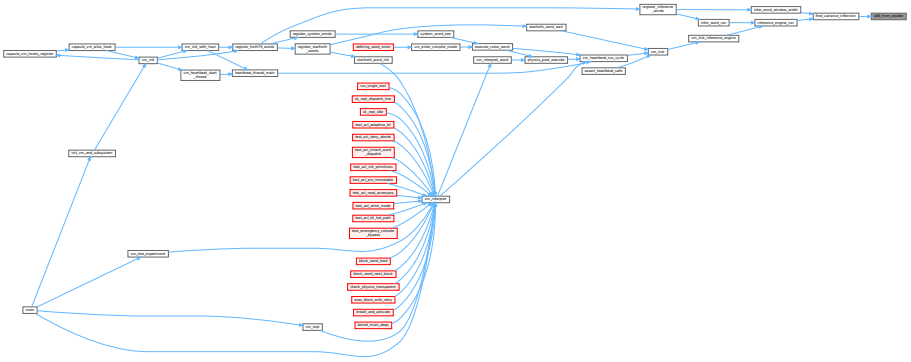
Returns

Nearest Q48.16 representation of `d`

References zero.

Referenced by find_variance_inflection().

Here is the caller graph for this function:



7.222.2.7 q48_from_u64()

```
static q48_16_t q48_from_u64 (
            uint64_t u )  [inline], [static]
```

Convert unsigned 64-bit integer to Q48.16.

Math: u ∗ 2^16 (shift left by 16 bits) Example: q48_from_u64(1) = 0x10000 (1.0 in Q48.16)

Parameters

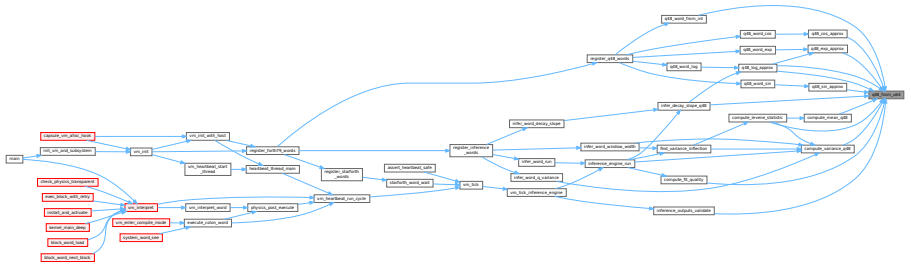
u	Unsigned 64-bit integer (max 2^48-1 to avoid overflow)

Returns

u << 16 in Q48.16 format

Referenced by compute_fit_quality(), compute_levene_statistic(), compute_mean_q48(), compute_variance_q48(), infer_decay_slope_q48(), inference_outputs_validate(), q48_cos_approx(), q48_exp_approx(), q48_log_approx(), q48_log_approx(), q48_sin_approx(), and q48_word_from_int().

Here is the caller graph for this function:



7.222.2.8 q48_is_valid()

```
int q48_is_valid (
            q48_16_t q )
```

Test whether a Q48.16 value is valid.

Currently accepts any `uint64_t` value as valid — all bit patterns are representable in Q48.16. A more sophisticated implementation could check for saturation sentinels or range limits. Returns 1 always.

Parameters

q	Value to validate

Returns

1 always (placeholder for future range validation)

7.222.2.9 q48_log_approx()

```
q48_16_t q48_log_approx (
            uint64_t x )
```

Compute an approximation of `ln(x)` in Q48.16 format (integer-only).

Used by `infer_decay_slope_q48()` (Loop #6) for log-linear OLS regression. Algorithm: range-reduces `x` to the interval [1, 2) by tracking the exponent `k` (so `x` = 2^k ∗ m), applies 6 Newton-Raphson iterations on `e^y` = m starting from the first-order approximation `y_0` = m - 1, then assembles the result as `ln(x)` = k∗ln(2) + ln(m). Returns 0 for `x` == 0 (ln undefined) and for `x` == 65536 (ln(1.0) == 0.0 exactly). `LN2_Q48` = 45426 (0.693147... × 65536).

Parameters

x	Input value in Q48.16 format (uint64_t; value of 65536 = 1.0)

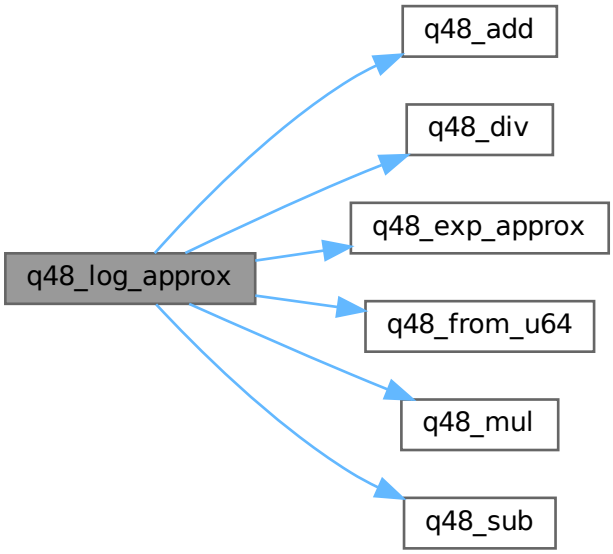
Returns

`ln(x)` in Q48.16 format, or 0 for x==0 or x==65536

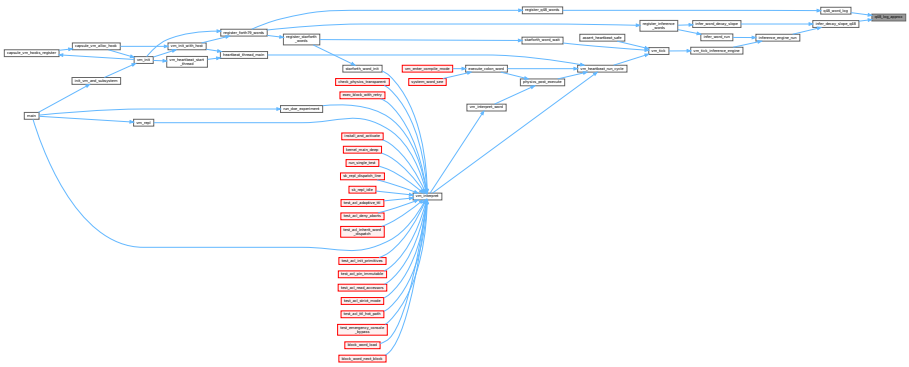
References q48_add(), q48_div(), q48_exp_approx(), q48_from_u64(), q48_mul(), and q48_sub().

Referenced by infer_decay_slope_q48(), and q48_word_log().

Here is the call graph for this function:



Here is the caller graph for this function:



7.222.2.10 q48_mul()

```
q48_16_t q48_mul (
            q48_16_t a,
            q48_16_t b )
```

Multiply two Q48.16 fixed-point values.

Computes (a ∗ b) >> 16 to maintain the Q48.16 scale. Uses __uint128_t on GCC/Clang (128-bit intermediate, no precision loss) and falls back to a manual 64-bit four-partial-product path with carry tracking on platforms that lack __SIZEOF_INT128__.

Parameters

a	First operand in Q48.16 format
b	Second operand in Q48.16 format

Returns

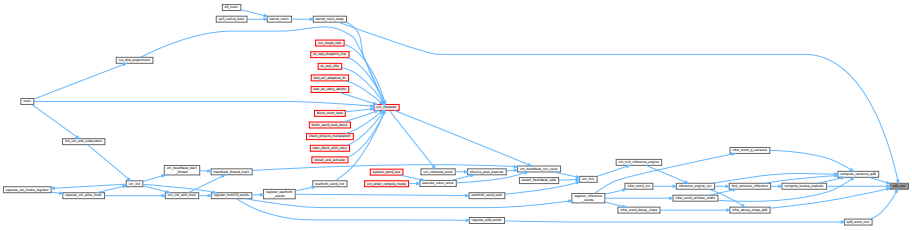
Product in Q48.16 format (truncated, not rounded)

Multiply two Q48.16 fixed-point values.

Freestanding implementation: no libc, no floating-point.

Referenced by compute_levene_statistic(), compute_variance_q48(), infer_decay_slope_q48(), kernel_main_deep(), and q48_word_mul().

Here is the caller graph for this function:



7.222.2.11 q48_sin_approx()

```
q48_16_t q48_sin_approx (
            q48_16_t q )
```

Compute an approximation of `sin(q)` in Q48.16 format (radians).

Range-reduces into [-pi, pi] via `q48_reduce_angle()`, then sums the Taylor series `x` - x^3/3! + x^5/5! - ... for terms n = 3,5,7,9,11, with an early exit when a term falls below 10 (sub-LSB in Q48.16). Sine is odd (`sin`(-x) = -sin(x)), so the sign is factored out before the series and reapplied to the result.

Parameters

q	Angle in Q48.16 format (radians, any magnitude)

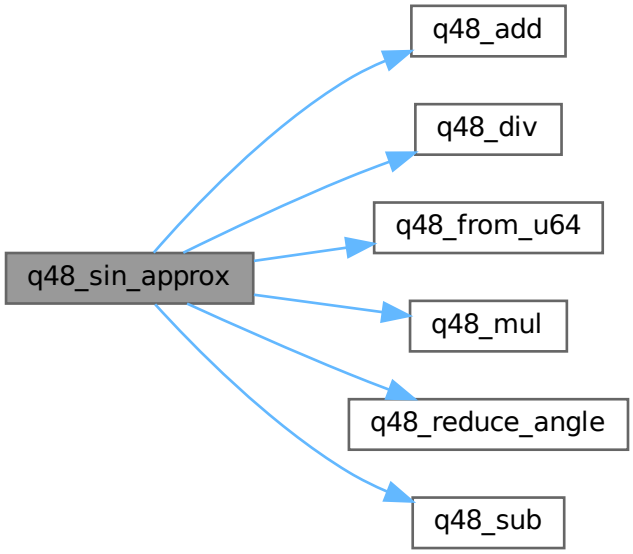
Returns

`sin(q)` in Q48.16 format

References q48_add(), q48_div(), q48_from_u64(), q48_mul(), q48_reduce_angle(), and q48_sub().

Referenced by q48_word_sin().

Here is the call graph for this function:



Here is the caller graph for this function:

7.222.2.12 q48_sqrt_approx()

```
q48_16_t q48_sqrt_approx (
            q48_16_t q )
```

Compute an approximation of `sqrt(q)` in Q48.16 format (Newton-Raphson).

Iterates `x_{n+1}` = (x_n + q/x_n) / 2 starting from (q >> 1) + 16384, for up to 8 iterations. Terminates early when the delta between successive estimates falls below 10 (approximately 0.00015 in Q48.16). Used for R^2 computation in the inference engine. Returns 0 for `q` == 0, and 1.0 (65536) for `q` == 65536.

Parameters

q	Input value in Q48.16 format

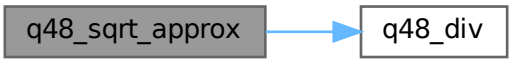
Returns

`sqrt(q)` in Q48.16 format

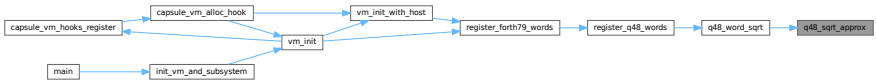
References q48_div().

Referenced by q48_word_sqrt().

Here is the call graph for this function:



Here is the caller graph for this function:



7.222.2.13 q48_sub()

```
static q48_16_t q48_sub (
            q48_16_t a,
            q48_16_t b )  [inline], [static]
```

Subtract two Q48.16 values.

Math: (a / 2^16) - (b / 2^16) = (a - b) / 2^16 Returns: a - b (no shift needed)

Parameters

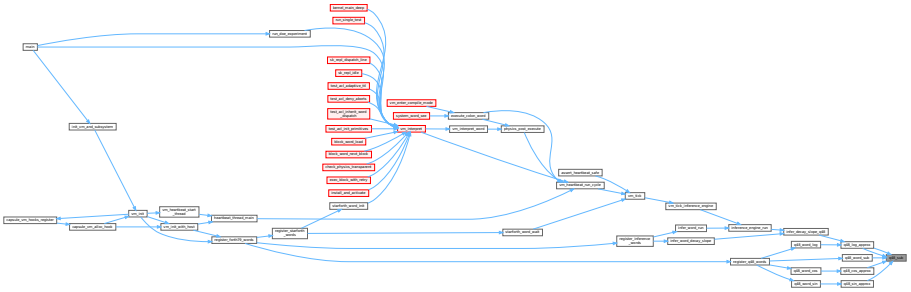
a	Minuend in Q48.16
b	Subtrahend in Q48.16

Returns

a - b in Q48.16 format

Referenced by q48_cos_approx(), q48_log_approx(), q48_log_approx(), q48_sin_approx(), and q48_word_sub().

Here is the caller graph for this function:



7.222.2.14 q48_to_double()

```
double q48_to_double (
            q48_16_t q )
```

Convert a Q48.16 value to `double`.

Divides `q` by 65536.0. Used for logging and RStudio dashboard output; not called in the inference engine hot path.

Parameters

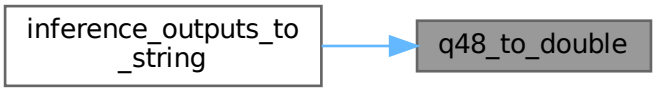
q	Value in Q48.16 format

Returns

Equivalent `double` value

Referenced by inference_outputs_to_string().

Here is the caller graph for this function:



7.222.2.15 q48_to_string()

```
const char * q48_to_string (
            q48_16_t q )
```

Format a Q48.16 value as a decimal string `"INTEGER.FRAC"`.

Writes into a static buffer — not thread-safe and not re-entrant. The fractional part is expressed as 5 decimal digits (`frac_part` * 100000 / 65536). Intended for diagnostic logging.

Parameters

q	Value in Q48.16 format

Returns

Pointer to a static buffer containing the formatted string

References snprintf().

Here is the call graph for this function:

7.222.2.16 q48_to_u64()

```
static uint64_t q48_to_u64 (
            q48_16_t q )  [inline], [static]
```

Convert Q48.16 to a 64-bit integer (truncate fractional)

Math: q / 2^16 (arithmetic shift right by 16 bits) Example: q48_to_u64(0x10000) = 1, q48_to_u64(-0x10000) = -1

q48_16_t values are two's-complement signed under the hood (q48_neg/ q48_abs already treat them that way); a plain unsigned (logical) shift would corrupt negative inputs instead of sign-extending them, so this shifts via a signed intermediate. Bit-identical to the old behavior for non-negative q (the only case this function's other caller, the inference engine's sum-of-squares accumulation, ever produces).

Parameters

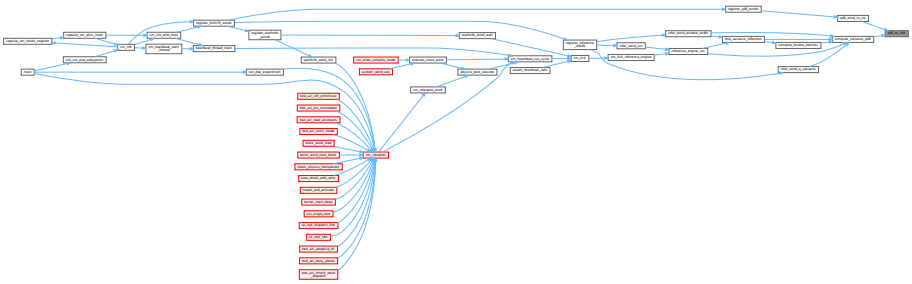
q	Q48.16 value

Returns

q >> 16, sign-extended, reinterpreted as uint64_t

Referenced by compute_variance_q48(), and q48_word_to_int().

Here is the caller graph for this function:



7.223 kernel/include/starkernel/freestanding/README.md File Reference

7.224 kernel/include/starkernel/freestanding/sys/README.md File Reference

7.225 kernel/include/starkernel/hal/README.md File Reference

7.226 kernel/include/starkernel/README.md File Reference

7.227 kernel/include/starkernel/vm/bootstrap/README.md File Reference

7.228 kernel/include/starkernel/vm/README.md File Reference

7.229 kernel/src/arch/aarch64/README.md File Reference

7.230 kernel/src/arch/amd64/README.md File Reference

7.231 kernel/src/arch/riscv64/README.md File Reference

7.232 kernel/src/boot/README.md File Reference

7.233 kernel/src/capsule/README.md File Reference

7.234 kernel/src/hal/README.md File Reference

7.235 kernel/src/hash/README.md File Reference

7.236 kernel/src/math/README.md File Reference

7.237 kernel/src/memory/README.md File Reference

7.238 kernel/src/pci/README.md File Reference

7.239 kernel/src/README.md File Reference

7.240 kernel/src/virtio/README.md File Reference

7.241 kernel/src/vm/bootstrap/README.md File Reference

7.242 kernel/src/vm/host/README.md File Reference

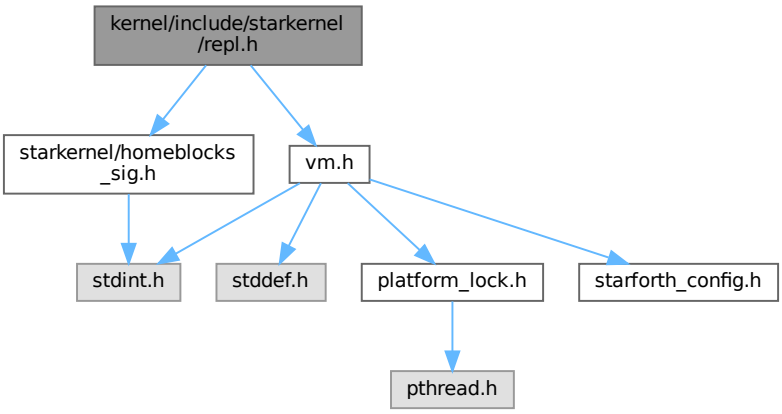
7.243 kernel/src/vm/README.md File Reference

7.244 README.md File Reference

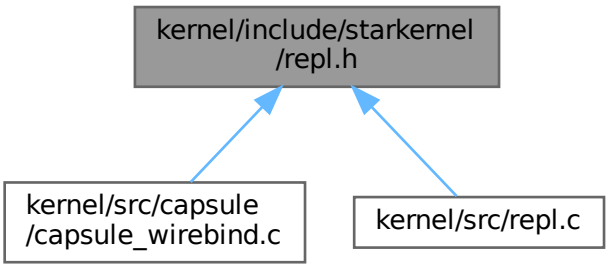
7.245 v3/include/README.md File Reference

```
#include "starkernel/homeblocks_sig.h"
```

Include dependency graph for repl.h:



This graph shows which files directly or indirectly include this file:



Functions

- void sk_repl (VM *vm)

  sk_repl - Run the emergency FORTH REPL on the serial console.
- void sk_repl_run (VM *vm)

  sk_repl_run - Bare REPL loop (no banner).
- int sk_repl_step (VM *vm)

  sk_repl_step - Execute one REPL turn on a VM and return.
- void sk_repl_set_active_vm (VM *vm)

  sk_repl_set_active_vm - Redirect REPL input to a different VM (USE word).
- VM * sk_repl_get_active_vm (void)

  sk_repl_get_active_vm - Return the current USE-redirected VM, or NULL.
- struct blkio_dev * sk_repl_get_homeblocks_dev (void)

sk_repl_get_homeblocks_dev / sk_repl_get_homeblocks_sig - The currently attached home-blocks USB drive, or NULL if none is attached / the attached drive didn't check out as HOMEBLOCKS_SIG_OK (FABRIC-2.md ?F.6/?F.9/?F.18).

- const homeblocks_sig_t * sk_repl_get_homeblocks_sig (void)
- struct blkio_dev * sk_repl_get_attached_blk_dev (void)

  sk_repl_get_attached_blk_dev - The currently attached USB block device, regardless of home-blocks recognition (FABRIC-2.md ?F.8/?F.19) – MINT's own target, since a blank/unminted drive never sets sk_repl_get_homeblocks_dev() above.
- void sk_repl_register_words (VM *vm)

  sk_repl_register_words - Registers repl.c's own FORTH-visible words (currently just BLK-ATTACH-ACK, Artemis's storage-attach reply target – see sk_word_blk_attach_ack()'s doc comment in repl.c).
- int sk_console_getkey (VM *active_vm)

  sk_console_getkey - Real body of the standard dictionary's KEY word (called from shim.c's getchar()).
- int sk_console_key_available (void)

  sk_console_key_available - Real body of the standard dictionary's ?TERMINAL word (called from sf_terminal_ready()).
- int sk_console_readline (char *buf, int size, VM *active_vm, int reanchor_prompt)

  sk_console_readline - Real body of the standard dictionary's QUERY/EXPECT words (called from shim.c's fgets()).
- void sk_repl_headless_wait (VM *mama)

  sk_repl_headless_wait - Idle-service loop with no interactive surface at all: no banner, no prompt, no console_getc()/readline.

7.255.1 Function Documentation

7.255.1.1 sk_console_getkey()

```
int sk_console_getkey (
            VM * active_vm )
```

sk_console_getkey - Real body of the standard dictionary's KEY word (called from shim.c's getchar()).

Blocks until a key is available from either input source (serial console or the PS2/virtio keyboard-event bridge), servicing the heartbeat/idle loop while waiting so a KEY call from inside any word never stalls the heartbeat. No echo – that's the caller's responsibility, same as any standard KEY.

Parameters

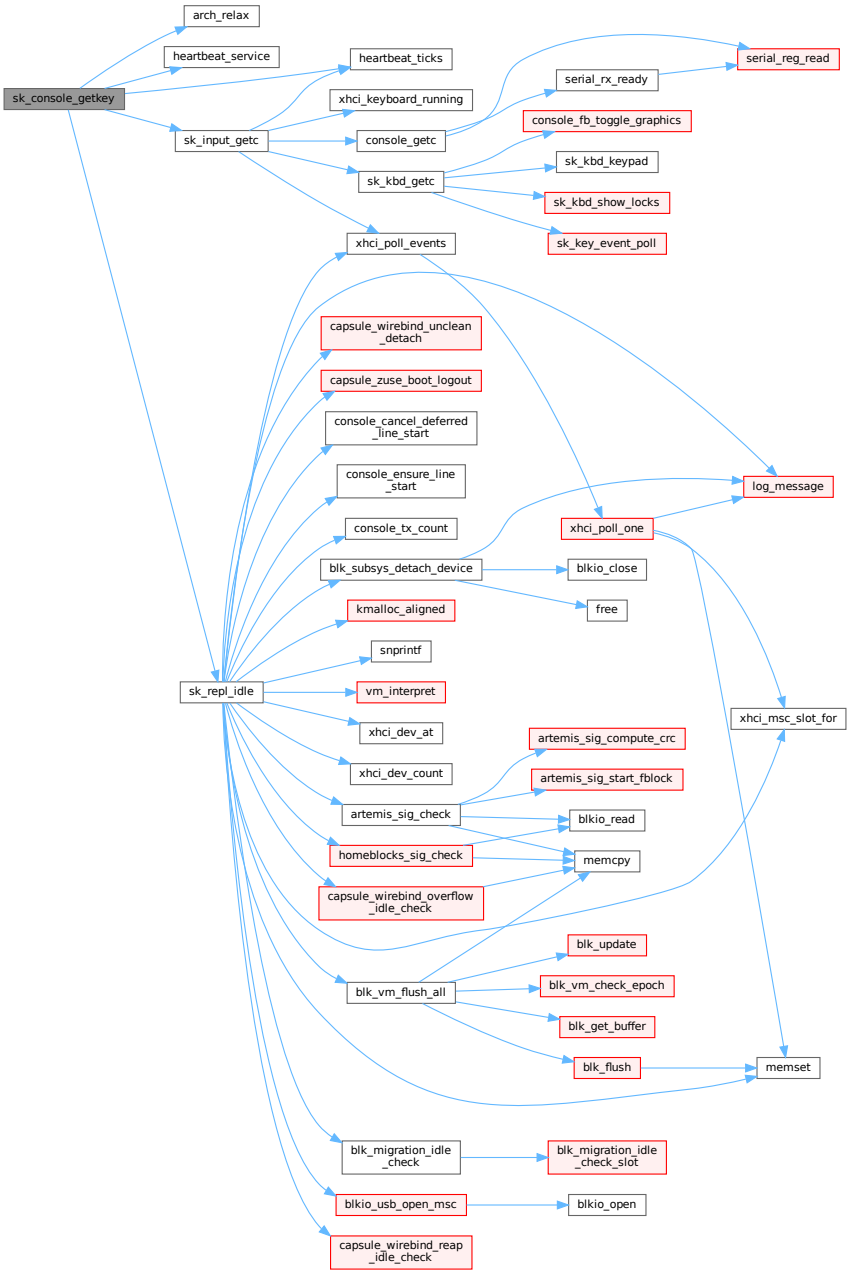
active_vm	VM whose idle dispatch runs while waiting (see sk_repl_idle()'s own doc comment on why this is a parameter rather than read via sk_repl_get_active_vm())

Returns

the key read, as an unsigned byte value

References arch_relax(), g_console_pending_key, g_last_beat_tick, g_repl_active_vm, heartbeat_service(), heartbeat_ticks(), SK_IDLE_BEAT_INTERVAL, sk_input_getc(), and sk_repl_idle().

Here is the call graph for this function:



7.255.1.2 sk_console_key_available()

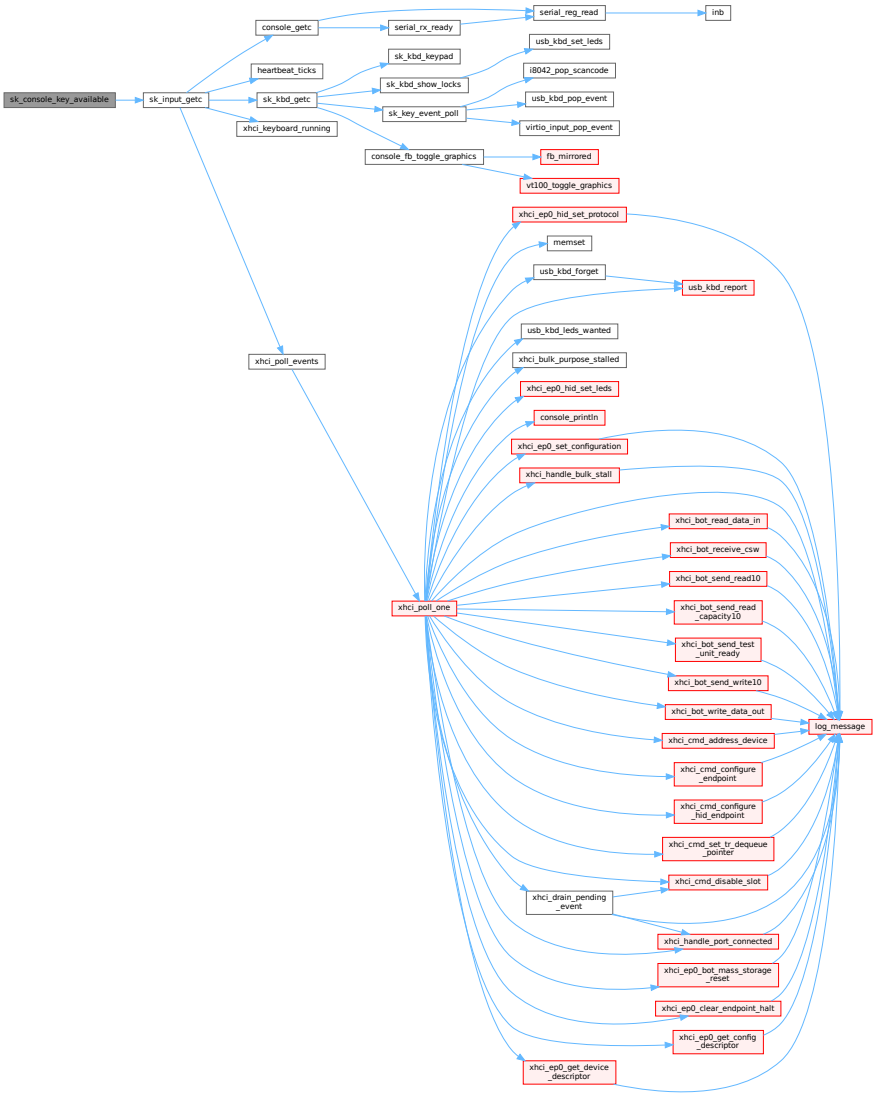
```
int sk_console_key_available (
            void  )
```

sk_console_key_available - Real body of the standard dictionary's ?TERMINAL word (called from sf_terminal_ready()).

Non-blocking peek: returns 1 if a key is ready without consuming it (a following sk_console_getkey() returns that exact key), 0 otherwise.

References g_console_pending_key, and sk_input_getc().

Here is the call graph for this function:



7.255.1.3 sk_console_readline()

```
int sk_console_readline (
            char * buf,
            int size,
            VM * active_vm,
            int reanchor_prompt )
```

sk_console_readline - Real body of the standard dictionary's QUERY/EXPECT words (called from shim.c's fgets()).

Reads one line from the console with echo and backspace support, servicing the heartbeat/ idle loop while waiting – the same line editor the REPL's own prompt uses internally, so a mid-word EXPECT behaves identically to typing at "ok>" itself.

Parameters

buf	Destination buffer
size	Buffer capacity, including the NUL terminator
active_vm	VM whose idle dispatch runs while waiting
reanchor_prompt	Nonzero to re-print the "ok> " prompt whenever an idle bottom half (heartbeat, USB attach/detach) writes to the console while this readline blocks at a bare, untyped prompt – keeps the top-level prompt as the last thing shown once the chatter dies down. Callers whose prompt line is their own (shim.c's fgets(), i.e. QUERY/EXPECT/ACCEPT) pass 0 so "ok> " never gets stamped onto their mid-word input context.

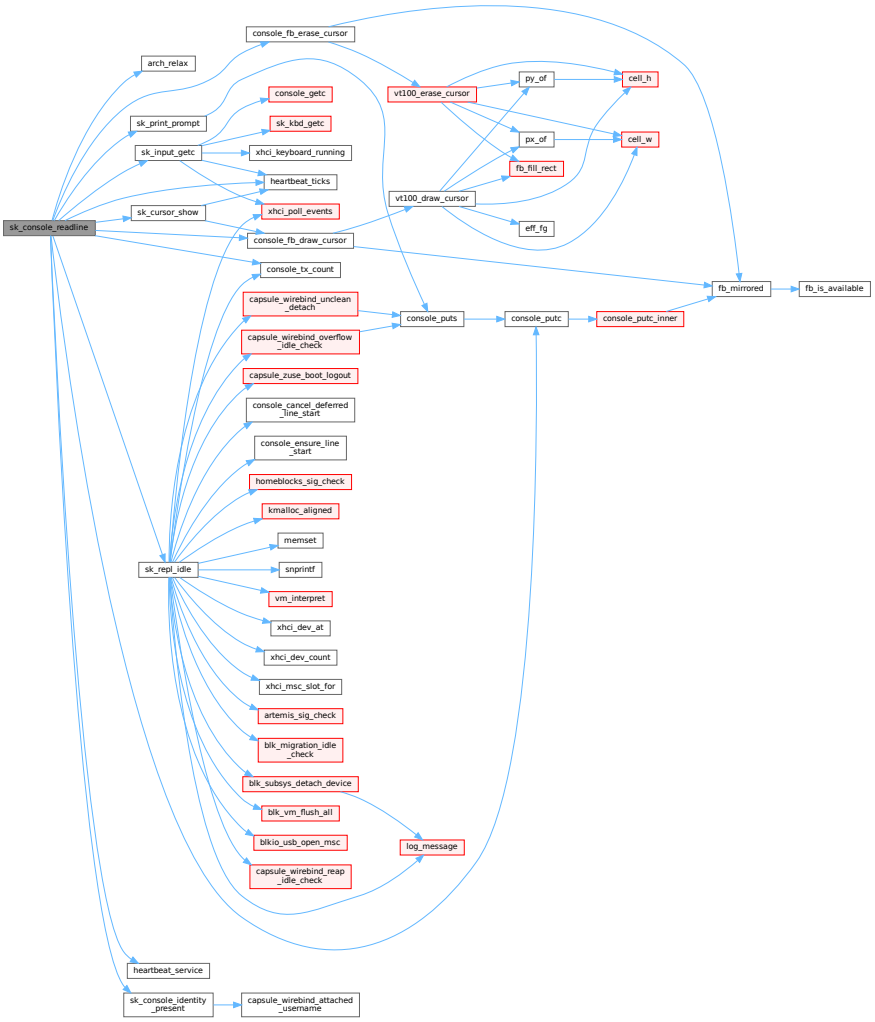
Returns

number of characters placed in buf, not counting the NUL, or -1 (2026-09-06) when reanchor_prompt is nonzero and the attached identity logged out while this call was blocked waiting for input with nothing typed yet – see repl.c's own doc comment on this function for what a caller must do with -1.

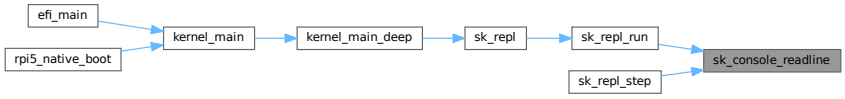
References arch_relax(), console_fb_draw_cursor(), console_fb_erase_cursor(), console_putc(), console_tx_count(), g_console_pending_key, g_cursor_blink_tick, g_cursor_visible, g_last_beat_tick, g_repl_active_vm, heartbeat_service(), heartbeat_ticks(), sk_console_identity_present(), SK_CURSOR_BLINK_INTERVAL, sk_cursor_show(), SK_IDLE_BEAT_INTERVAL, sk_input_getc(), sk_print_prompt(), and sk_repl_idle().

Referenced by sk_repl_run(), and sk_repl_step().

Here is the call graph for this function:



Here is the caller graph for this function:



7.255.1.4 sk_repl()

```
void sk_repl (
            VM * vm )
```

sk_repl - Run the emergency FORTH REPL on the serial console.

Blocks until vm->halted is set (BYE word) or the VM encounters a halt. Runs with interrupts enabled; the APIC heartbeat continues to fire.

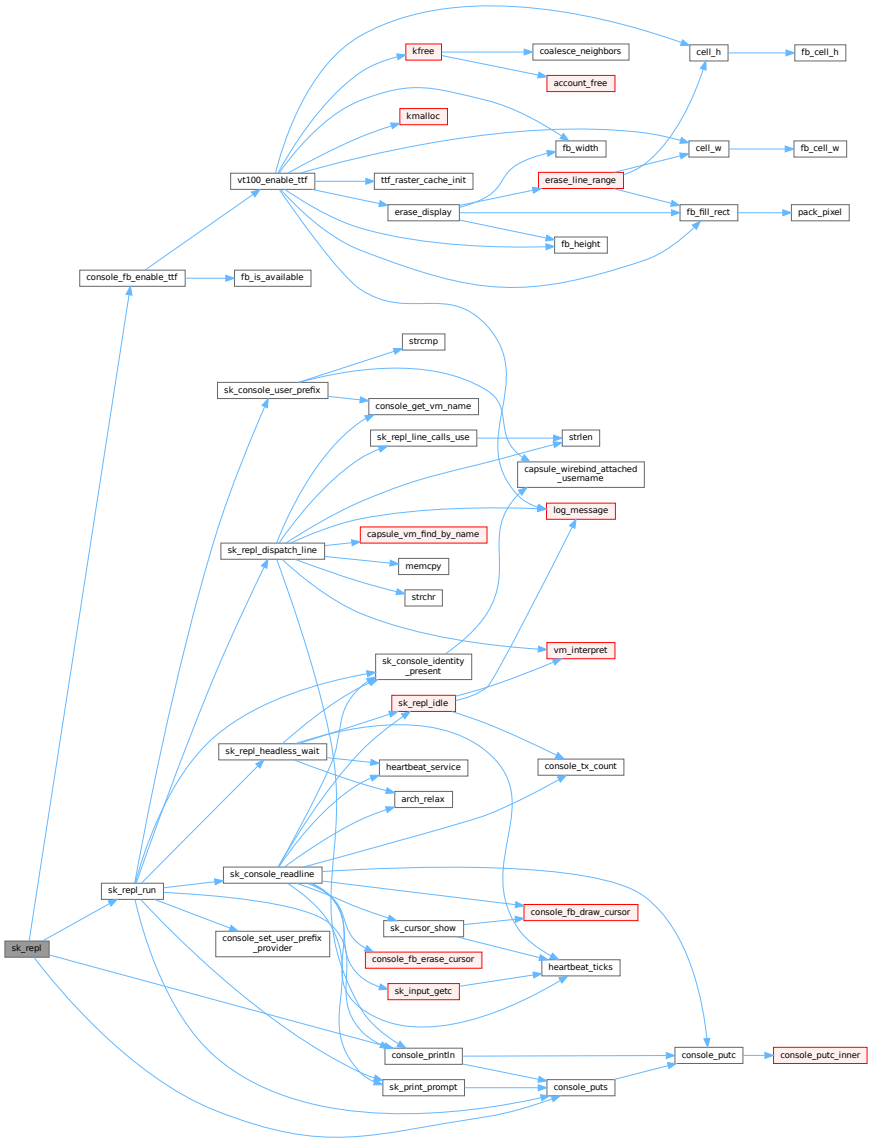
Parameters

vm	Mama VM instance (must be fully initialised)

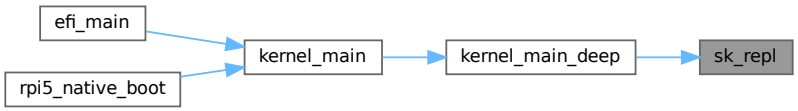
References console_fb_enable_ttf(), console_println(), console_puts(), lithos_version, sk_repl_run(), and STARFORTH_VERSION.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



7.255.1.5 sk_repl_get_active_vm()

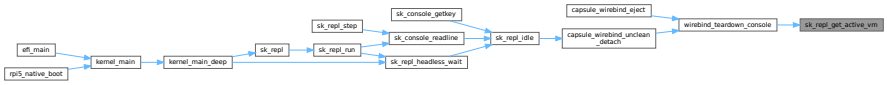
```
VM * sk_repl_get_active_vm (
            void  )
```

sk_repl_get_active_vm - Return the current USE-redirected VM, or NULL.

References g_repl_active_vm.

Referenced by wirebind_teardown_console().

Here is the caller graph for this function:



7.255.1.6 sk_repl_get_attached_blk_dev()

```
struct blkio_dev * sk_repl_get_attached_blk_dev (
            void  )
```

sk_repl_get_attached_blk_dev - The currently attached USB block device, regardless of home-blocks recognition (FABRIC-2.md ?F.8/?F.19) – MINT's own target, since a blank/unminted drive never sets sk_repl_get_homeblocks_dev() above.

NULL if nothing is attached.

References g_attached_blk_dev.

7.255.1.7 sk_repl_get_homeblocks_dev()

```
struct blkio_dev * sk_repl_get_homeblocks_dev (
            void  )
```

sk_repl_get_homeblocks_dev / sk_repl_get_homeblocks_sig - The currently attached home-blocks USB drive, or NULL if none is attached / the attached drive didn't check out as HOMEBLOCKS_SIG_OK (FABRIC-2.md ?F.6/?F.9/?F.18).

Both return NULL together; never one without the other.

References g_homeblocks_dev, and g_homeblocks_sig_valid.

7.255.1.8 sk_repl_get_homeblocks_sig()

```
const homeblocks_sig_t * sk_repl_get_homeblocks_sig (
            void  )
```

References g_homeblocks_sig, and g_homeblocks_sig_valid.

7.255.1.9 sk_repl_headless_wait()

```
void sk_repl_headless_wait (
            VM * mama )
```

sk_repl_headless_wait - Idle-service loop with no interactive surface at all: no banner, no prompt, no console_getc()/readline.

Runs heartbeat_service() and the same SK_IDLE_BEAT_INTERVAL-gated sk_repl_idle(mama) cadence sk_console_readline()'s own idle branch uses – so USB/WIREBIND/Zuse-attach detection, the heartbeat, and all other idle-tick subsystems keep running – until a real identity is currently attached (Zuse's own session, or a WIREBIND user), at which point it returns.

Revised 2026-09-06: originally exited on a one-way sticky "has anyone ever logged in this boot" flag (sk↩ _console_mark_login()/sk_console_ login_occurred(), both retired) – that let a real gap through, found live↩ : once the flag tripped once, it never reset, so a later full logout (nobody attached at all) fell through to a bare, unauthenticated prompt instead of going silent again. This now checks live attach state instead (repl.c's own sk_console_identity_present()), and is called from two places: once from kernel_main.c in place of an immediate sk_repl(mama) call when EMERGENCY_CONSOLE_ENABLED is off (the default, 2026-09-05) – no thumbdrive, no prompt, at boot – and again from inside sk_repl_run()'s own main loop, every time nobody is currently attached, so the same silence re-engages after any later logout mid-boot too. When EMERGENCY_CONSOLE_ENABLED is on (the debug/ recovery escape hatch), neither call site applies – the console shows immediately and stays visible regardless of attach state, exactly as before this change.

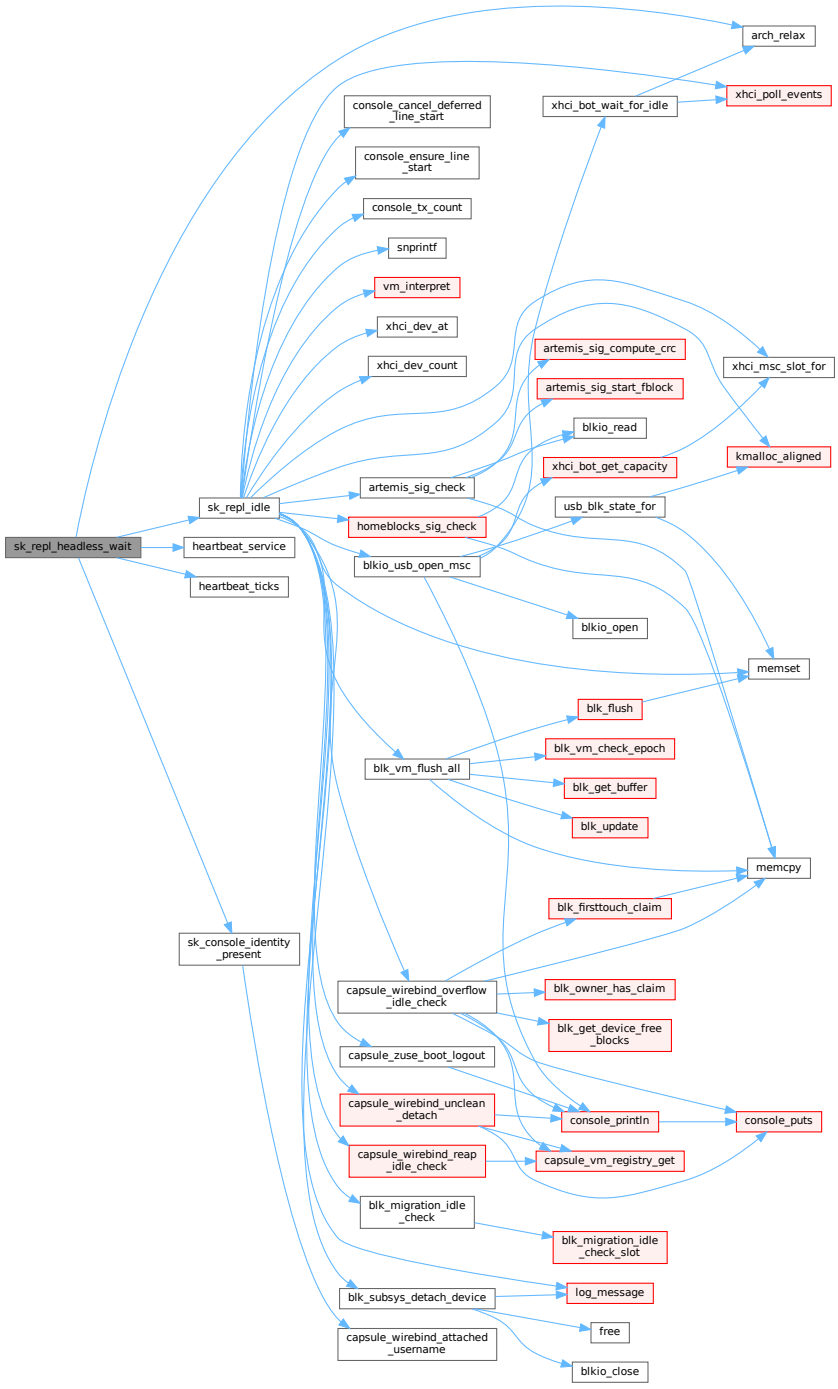
Parameters

mama	Hera's own VM instance – the idle-dispatch target, same as every other sk_repl_idle() caller uses.

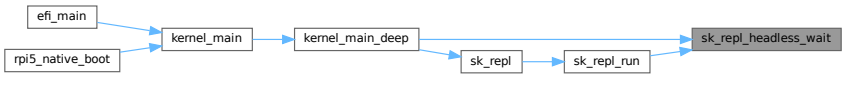
References arch_relax(), g_last_beat_tick, heartbeat_service(), heartbeat_ticks(), sk_console_identity_present(), SK_IDLE_BEAT_INTERVAL, and sk_repl_idle().

Referenced by kernel_main_deep(), and sk_repl_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.255.1.10 sk_repl_register_words()

```
void sk_repl_register_words (
            VM * vm )
```

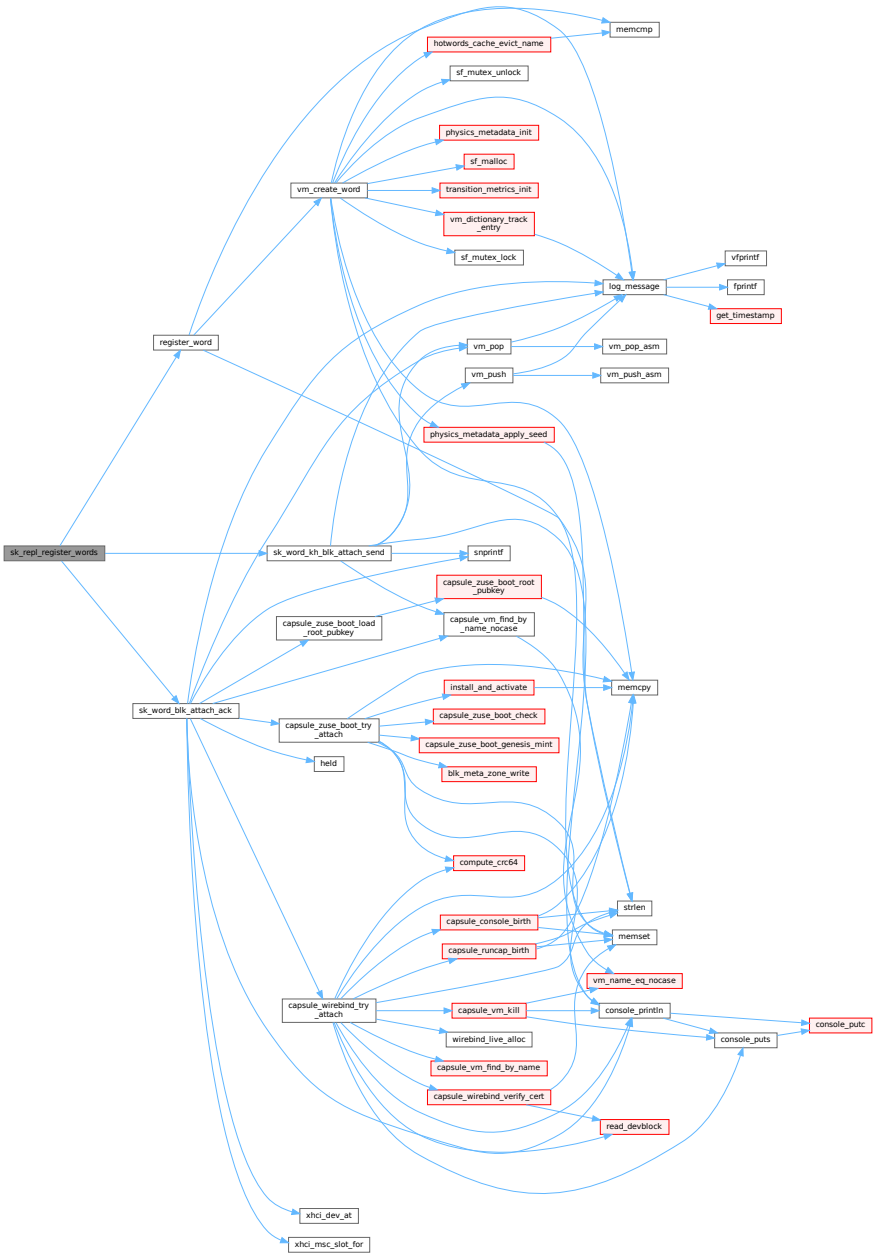
sk_repl_register_words - Registers repl.c's own FORTH-visible words (currently just BLK-ATTACH-ACK, Artemis's storage-attach reply target – see sk_word_blk_attach_ack()'s doc comment in repl.c).

Call once from register_forth79_words() alongside the other **STARKERNEL**-only registration calls.

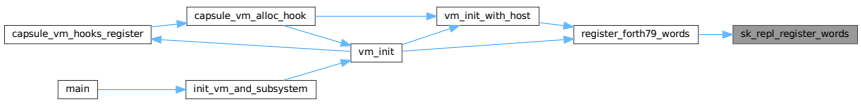
References register_word(), sk_word_blk_attach_ack(), and sk_word_kh_blk_attach_send().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.255.1.11 sk_repl_run()

```
void sk_repl_run (
            VM * vm )
```

sk_repl_run - Bare REPL loop (no banner).

Same as sk_repl but skips the version/welcome banner. Used by START to enter a child VM's interpreter loop without reprinting the header.

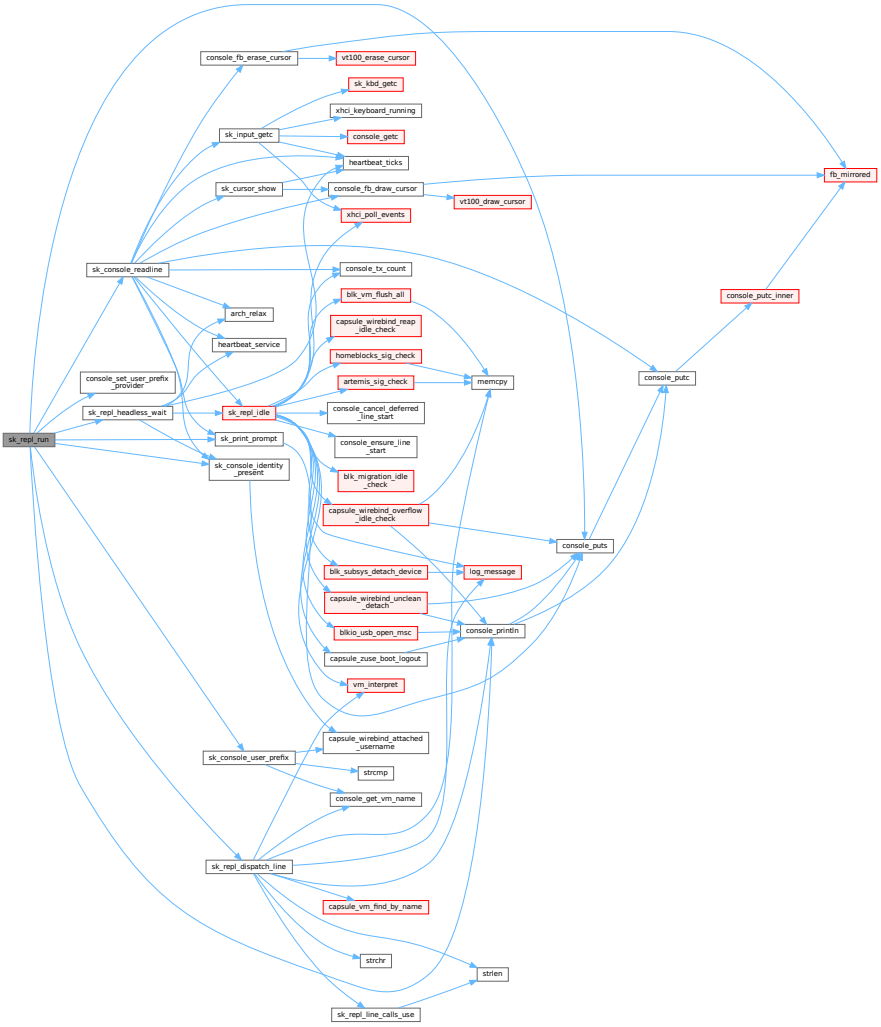
Parameters

vm	Fully initialised VM instance

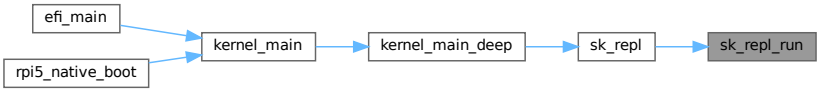
References VM::abort_requested, console_println(), console_puts(), console_set_user_prefix_provider(), VM::error, g_repl_active_vm, VM::halted, INPUT_BUFFER_SIZE, sk_console_identity_present(), sk_console_readline(), sk_console_user_prefix(), sk_print_prompt(), sk_repl_dispatch_line(), and sk_repl_headless_wait().

Referenced by sk_repl().

Here is the call graph for this function:



Here is the caller graph for this function:



7.255.1.12 sk_repl_set_active_vm()

```
void sk_repl_set_active_vm (
            VM * vm )
```

sk_repl_set_active_vm - Redirect REPL input to a different VM (USE word).

Pass NULL to restore default dispatch (Mama's VM). The change takes effect on the next REPL iteration.

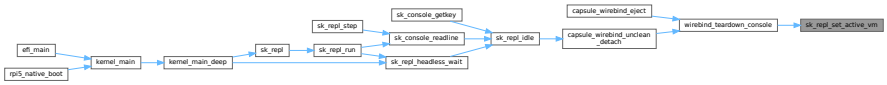
Parameters

vm	Target VM, or NULL for default

References g_repl_active_vm.

Referenced by wirebind_teardown_console().

Here is the caller graph for this function:



7.255.1.13 sk_repl_step()

```
int sk_repl_step (
            VM * vm )
```

sk_repl_step - Execute one REPL turn on a VM and return.

Prints the VM's prompt, reads one line, interprets it, prints ok/ERROR, then returns. Used by the Compudynamics VM-STEP primitive so Hera can give a single REPL quantum to a child VM without surrendering control for the full sk_repl_run() loop.

Parameters

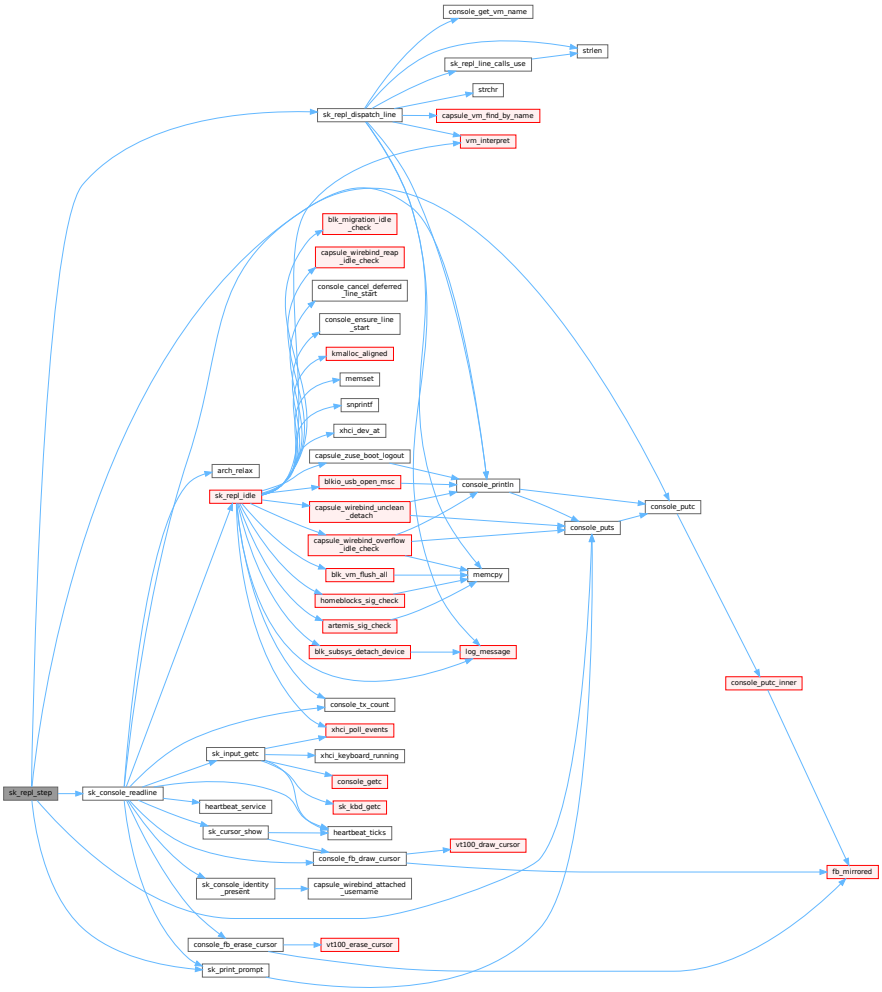
vm	Fully initialised VM instance

Returns

1 if the VM is still running, 0 if it halted during this turn

References VM::abort_requested, console_println(), console_puts(), VM::error, VM::halted, INPUT_BUFFER_SIZE, sk_console_readline(), sk_print_prompt(), and sk_repl_dispatch_line().

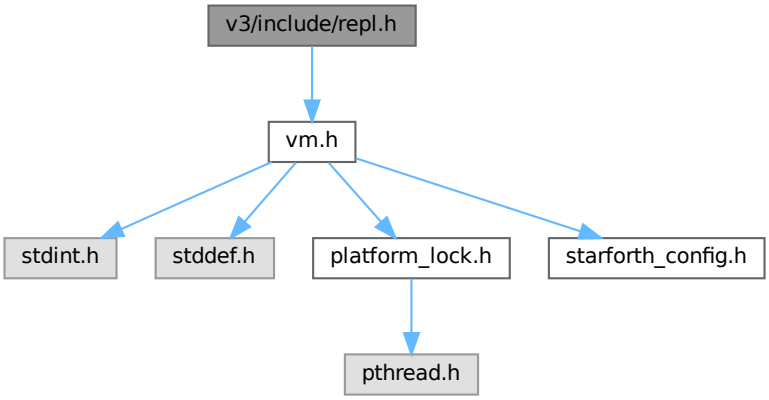
Here is the call graph for this function:



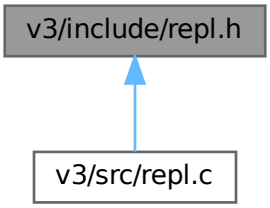
7.256 v3/include/repl.h File Reference

```
#include "vm.h"
```

Include dependency graph for repl.h:



This graph shows which files directly or indirectly include this file:



Functions

- void vm_repl (VM *vm, int script_mode)
  Starts the Forth REPL (Read-Eval-Print Loop)

7.256.1 Function Documentation

7.256.1.1 vm_repl()

```
void vm_repl (
            VM * vm,
            int script_mode )
```

Starts the Forth REPL (Read-Eval-Print Loop)

Parameters

vm	Pointer to the VM structure
script_mode	If 1, suppress prompts and "ok" output (for piped input)

This function implements an interactive REPL for the Forth interpreter. It continuously reads input from the user, interprets it, and prints the result until the VM is halted or an error occurs.

In interactive mode (script_mode=0):

- Prints colored prompt "ok> "
- Acknowledges successful commands with " ok"
- Reports errors with " ERROR"

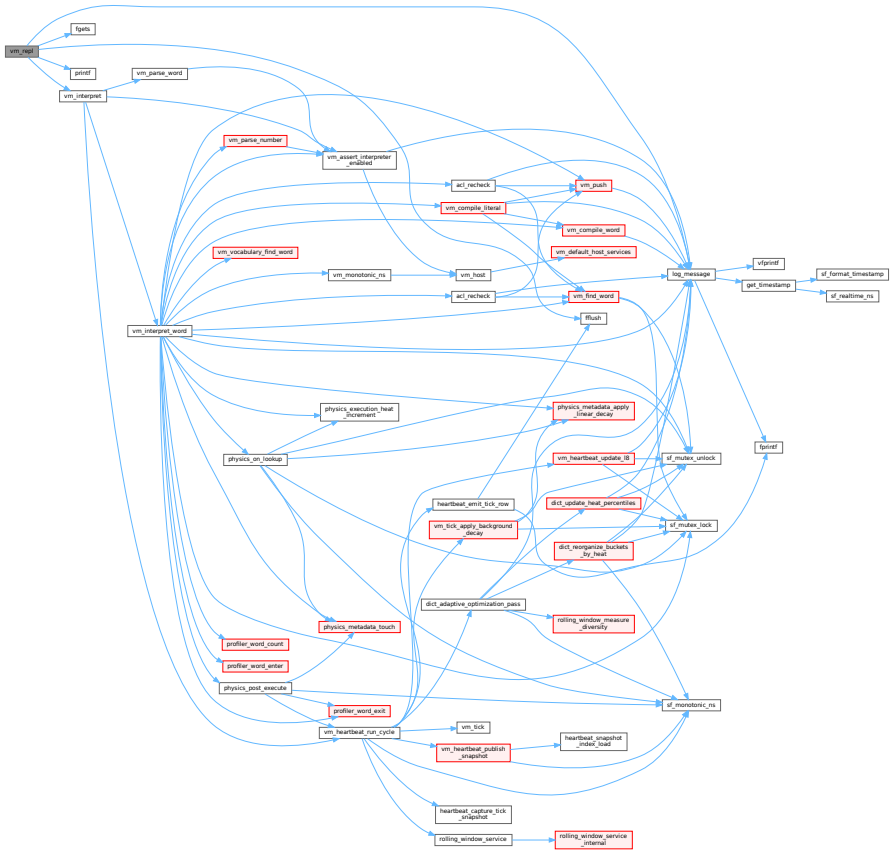
In script mode (script_mode=1):

- Suppresses all prompts and status messages
- Suitable for piped input (heredoc, pipes, redirects)

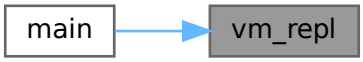
References VM::abort_requested, VM::error, fflush(), fgets(), VM::halted, LOG_INFO, log_message(), printf(), stdin, stdout, vm_interpret(), and VM::zuse_session.

Referenced by main().

Here is the call graph for this function:



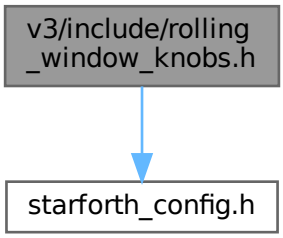
Here is the caller graph for this function:



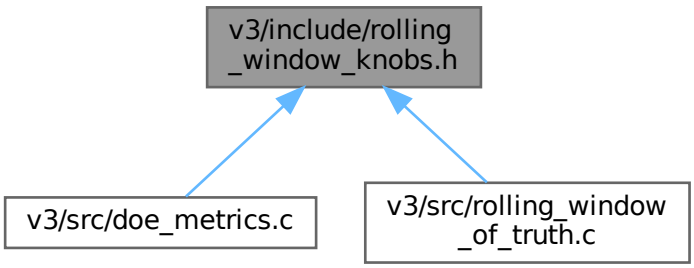
7.257 v3/include/rolling_window_knobs.h File Reference

```
#include "starforth_config.h"
```

Include dependency graph for rolling_window_knobs.h:



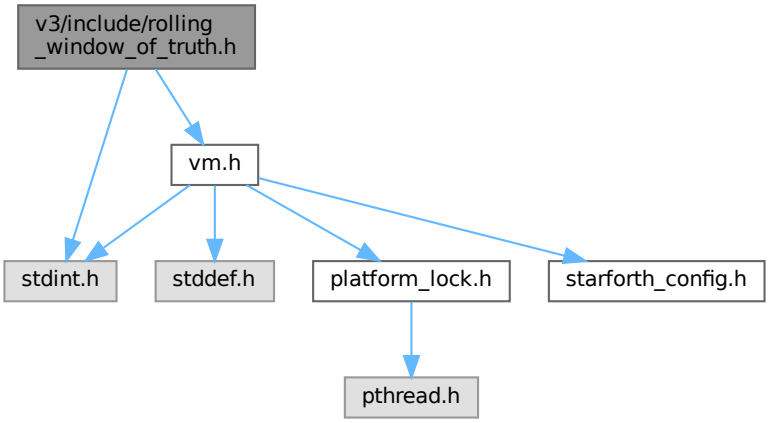
This graph shows which files directly or indirectly include this file:



7.258 v3/include/rolling_window_of_truth.h File Reference

```
#include <stdint.h>
#include "vm.h"
```

Include dependency graph for rolling_window_of_truth.h:



This graph shows which files directly or indirectly include this file:

Functions

- int rolling_window_init (RollingWindowOfTruth *window)

  Initialise a `RollingWindowOfTruth`, allocating its three heap buffers.
- int rolling_window_record_execution (RollingWindowOfTruth *window, uint32_t word_id)

  Record one word execution in the circular history buffer.
- uint32_t rolling_window_get_recent_sequence (const RollingWindowOfTruth *window, uint32_t depth, uint32_t *out_sequence)

  Copy the most recent `depth` word IDs from the rolling window into `out_sequence`.
- uint32_t rolling_window_find_hottest_word (const RollingWindowOfTruth *window, uint32_t dict_size)

  Return the word ID that appears most frequently in the rolling window.
- uint64_t rolling_window_count_transition (const RollingWindowOfTruth *window, uint32_t word_a, uint32_t word_b)

  Count how many times the transition (word_a → word_b) occurs in the history.
- int rolling_window_is_warm (const RollingWindowOfTruth *window)

  Return whether the rolling window has accumulated enough history to be useful.
- char * rolling_window_stats_string (const RollingWindowOfTruth *window)

  Allocate and return a one-page status summary for a `RollingWindowOfTruth`.
- void rolling_window_reset (RollingWindowOfTruth *window)

  Reset the rolling window to an empty cold state without freeing heap memory.

- void rolling_window_cleanup (RollingWindowOfTruth *window)

  Free all heap memory held by a `RollingWindowOfTruth`.
- uint64_t rolling_window_export_execution_history (const RollingWindowOfTruth *window, uint32_t *out_sequence, uint64_t max_count)

  Export up to `max_count` entries from the execution history in chronological order.
- double rolling_window_pattern_capture_rate (const RollingWindowOfTruth *window, uint32_t test_window_size)

  Estimate the fraction of transitions that carry unique contextual information.
- uint64_t rolling_window_measure_diversity (const RollingWindowOfTruth *window)

  Return the current pattern diversity score for the rolling window.
- void rolling_window_check_adaptive_shrink (RollingWindowOfTruth *window)

  Force an immediate adaptive window check, bypassing the accumulator gate.
- void rolling_window_service (RollingWindowOfTruth *window)

  Service the rolling window — publish pending snapshots and run adaptive sizing.

7.258.1 Function Documentation

7.258.1.1 rolling_window_check_adaptive_shrink()

```
void rolling_window_check_adaptive_shrink (
            RollingWindowOfTruth * window )
```

Force an immediate adaptive window check, bypassing the accumulator gate.

Sets `window->adaptive_pending` = 1 unconditionally and then calls `rolling_window_service_internal()` to drain it immediately. This allows external callers (e.g. the heartbeat timer or a FORTH word) to trigger a window-size evaluation outside the normal `ADAPTIVE_CHECK_FREQUENCY` cadence — for example, after a major workload transition or at benchmark checkpoints.

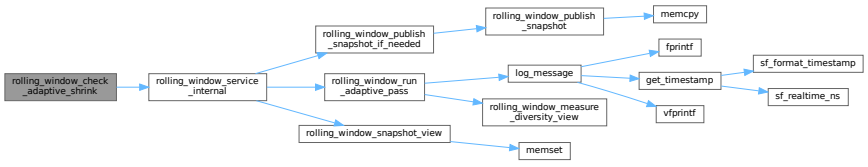
If the window is not warm the adaptive pass is a no-op internally.

Parameters

window	The `RollingWindowOfTruth` to check; no-op if NULL.

References RollingWindowOfTruth::adaptive_pending, and rolling_window_service_internal().

Here is the call graph for this function:



7.258.1.2 rolling_window_cleanup()

```
void rolling_window_cleanup (
            RollingWindowOfTruth * window )
```

Free all heap memory held by a RollingWindowOfTruth.

Calls sf_free() on execution_history and both snapshot_buffers[], setting each to NULL after freeing. Does not zero the scalar fields; the caller must not use window after this call without a subsequent rolling_window_init().

No-op if window is NULL. Each buffer pointer is individually NULL-checked before freeing (defensive against partial-init failures from rolling_window_init()).

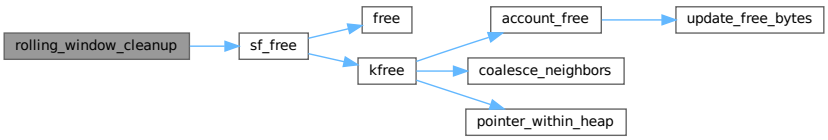
Parameters

window	The RollingWindowOfTruth to clean up; no-op if NULL.

References RollingWindowOfTruth::execution_history, sf_free(), and RollingWindowOfTruth::snapshot_buffers.

Referenced by vm_cleanup().

Here is the call graph for this function:



Here is the caller graph for this function:

7.258.1.3 rolling_window_count_transition()

```
uint64_t rolling_window_count_transition (
            const RollingWindowOfTruth * window,
            uint32_t word_a,
            uint32_t word_b )
```

Count how many times the transition (word_a → word_b) occurs in the history.

Scans all ROLLING_WINDOW_SIZE consecutive adjacent pairs in the current snapshot (wrapping at the buffer end: pair i is (history[i], history[(i+1) % ROLLING_WINDOW_SIZE])) and counts how many pairs match word_a followed by word_b.

Returns 0 if:

- `window` is NULL.
- `window->is_warm` is 0 (the history is too short to be meaningful).
- The snapshot's `is_warm` flag is 0.

Note: because the history is circular and the scan is linear across all slots regardless of `effective_window_size`, the count includes transitions that span the buffer wrap-around boundary. This is intentional: the full `ROLLING_WINDOW_SIZE` is used for transition counting.

Parameters

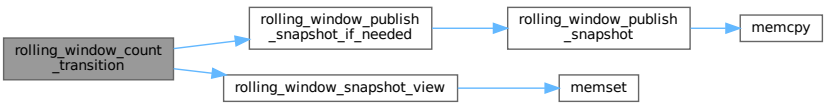
window	The `RollingWindowOfTruth` to query.
word_a	ID of the predecessor word.
word_b	ID of the successor word.

Returns

Count of (word_a → word_b) transitions; 0 if not warm.

References count, rolling_window_view_t::history, RollingWindowOfTruth::is_warm, rolling_window_view_t::is_warm, rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, and rolling_window_snapshot_view().

Here is the call graph for this function:



7.258.1.4 rolling_window_export_execution_history()

```
uint64_t rolling_window_export_execution_history (
            const RollingWindowOfTruth * window,
            uint32_t * out_sequence,
            uint64_t max_count )
```

Export up to `max_count` entries from the execution history in chronological order.

Linearises the internal circular buffer so callers get entries from oldest to newest without knowing the internal `window_pos` cursor. Two cases:

- **Pre-wrap** (`total_executions` < `ROLLING_WINDOW_SIZE`): the buffer has not yet wrapped; entries 0..`total_executions-1` are valid and are copied contiguously from `history`[0].
- **Post-wrap** (`total_executions` ≥ `ROLLING_WINDOW_SIZE`): the oldest entry is at `history`[window_pos]. Two `memcpy` calls stitch the two halves together:
  1. `history`[window_pos .. ROLLING_WINDOW_SIZE-1] (first part)
  2. `history`[0 .. window_pos-1] (second part, if `export_count` > `first_part`)

`export_count` is `min(total_executions, max_count)`. Returns 0 if `window`, `out_sequence` are NULL, `max_count` is 0, the snapshot has no history, or `export_count` is 0.

Parameters

window	The `RollingWindowOfTruth` to export.
out_sequence	Caller-allocated output buffer of at least `max_count` `uint32_t` entries.
max_count	Maximum number of entries to export.

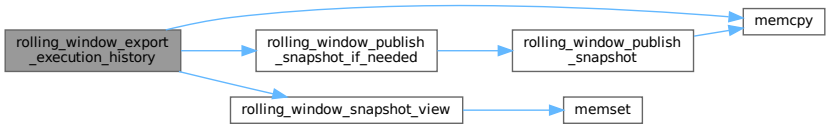
Returns

Actual number of entries written to `out_sequence`.

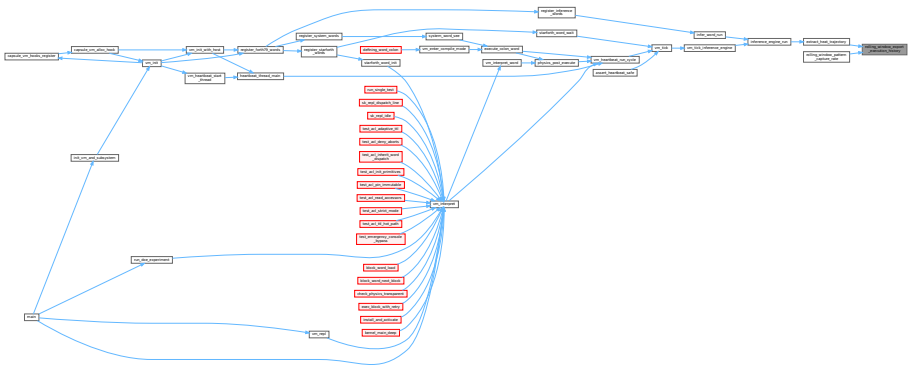
References rolling_window_view_t::history, memcpy(), rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), rolling_window_view_t::total_executions, and rolling_window_view_t::window_pos.

Referenced by extract_heat_trajectory(), and rolling_window_pattern_capture_rate().

Here is the call graph for this function:



Here is the caller graph for this function:



7.258.1.5 rolling_window_find_hottest_word()

```
uint32_t rolling_window_find_hottest_word (
            const RollingWindowOfTruth * window,
            uint32_t dict_size )
```

Return the word ID that appears most frequently in the rolling window.

Obtains the current snapshot, allocates a zero-filled frequency array of `dict_size` `uint32_t` counters via `sf_calloc`, and scans all `ROLLING_WINDOW_SIZE` history slots to tally occurrences. Then performs a linear scan for the maximum and frees the frequency array before returning.

Returns 0 immediately (without allocating) if:

- `window` is NULL.
- `window->is_warm` is 0 (not enough history to be meaningful).
- The snapshot's `is_warm` flag is 0.
- `sf_calloc` fails.

The returned word ID is the zero-based index into the dictionary's word-ID space. Word ID 0 may be returned both as a legitimate hottest word and as the not-warm sentinel; callers should check `rolling_window_is_warm()` before relying on the result.

Parameters

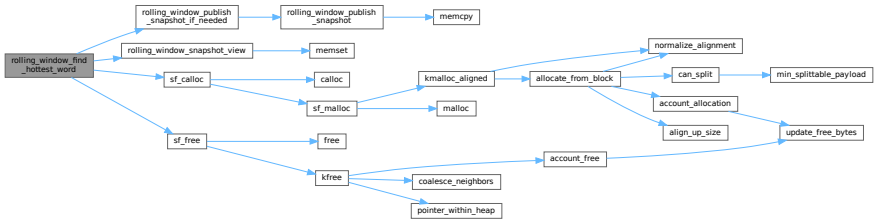
window	The `RollingWindowOfTruth` to analyse.
dict_size	Size of the dictionary's word-ID space; bounds the frequency array and the scan of the history.

Returns

Word ID of the most-frequently-executed word; 0 if the window is not warm or allocation fails.

References rolling_window_view_t::history, RollingWindowOfTruth::is_warm, rolling_window_view_t::is_warm, rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), sf_calloc(), sf_free(), and word_id.

Here is the call graph for this function:



7.258.1.6 rolling_window_get_recent_sequence()

```
uint32_t rolling_window_get_recent_sequence (
            const RollingWindowOfTruth * window,
            uint32_t depth,
            uint32_t * out_sequence )
```

Copy the most recent `depth` word IDs from the rolling window into `out_sequence`.

Calls `rolling_window_publish_snapshot_if_needed()` then reads the current double-buffer snapshot via `rolling_window_snapshot_view()`. Copies at most `min(depth, available)` entries from the most recent portion of the circular buffer into `out_sequence` in chronological order (oldest entry at index 0, most recent at index `actual_depth - 1`).

The available count is capped at `min(total_executions, ROLLING_WINDOW_SIZE)` so callers requesting more history than has been recorded get what is available rather than stale zeros.

Entries are addressed via:
`idx = (window_pos - actual_depth + i + ROLLING_WINDOW_SIZE) % ROLLING_WINDOW_SIZE`

Returns 0 immediately if `window`, `out_sequence` are NULL or `depth` is 0.

Parameters

window	The `RollingWindowOfTruth` to sample.
depth	Maximum number of recent word IDs to copy.
out_sequence	Caller-allocated output buffer of at least `depth` `uint32_t` entries.

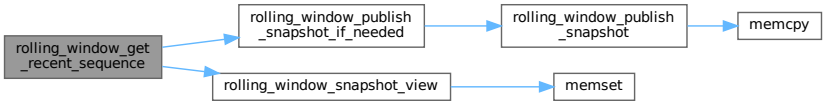
Returns

Actual number of entries written to `out_sequence`; $\leq$ `depth`.

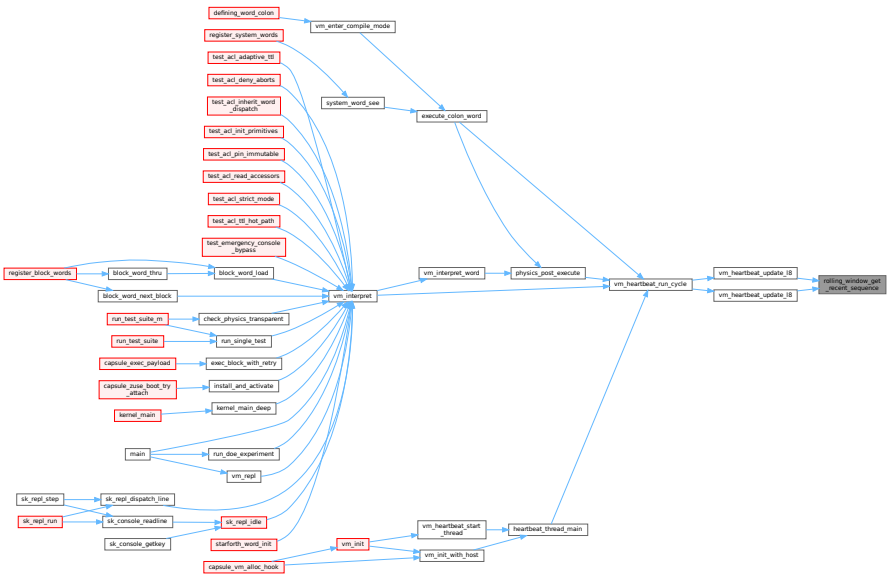
References rolling_window_view_t::history, rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), rolling_window_view_t::total_executions, and rolling_window_view_t::window_pos.

Referenced by vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the call graph for this function:



Here is the caller graph for this function:



7.258.1.7 rolling_window_init()

```
int rolling_window_init (
            RollingWindowOfTruth * window )
```

Initialise a RollingWindowOfTruth, allocating its three heap buffers.

Allocates and zero-initialises three uint32_t arrays, each of ROLLING_WINDOW_SIZE elements, via sf_calloc:

- window->execution_history — the live circular write buffer.
- window->snapshot_buffers[0] — first double-buffer read slot.
- window->snapshot_buffers[1] — second double-buffer read slot.

If any allocation fails, all previously allocated buffers are freed and nulled before returning ?1. Initialises scalar state:

- window_pos = 0, total_executions = 0, is_warm = 0 (cold start).
- effective_window_size = ROLLING_WINDOW_SIZE (start at maximum).
- snapshot_index = 0, snapshot_pending = 1.
- All snapshot_window_pos[] / snapshot_total_executions[] / etc. = 0.
- adaptive_check_accumulator = 0, adaptive_pending = 0.

Immediately calls rolling_window_publish_snapshot() to push the zeroed state into both snapshot slots so that any early read-side call before the first rolling_window_record_execution() gets a coherent (empty) view.

Parameters

window	The RollingWindowOfTruth to initialise; must not be NULL.

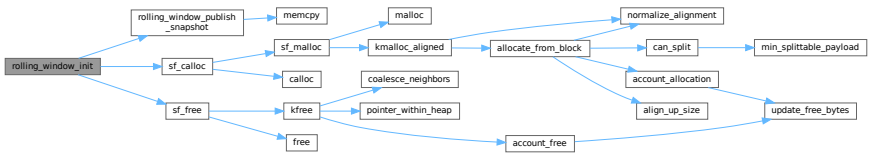
Returns

0 on success; ?1 on NULL window or allocation failure.

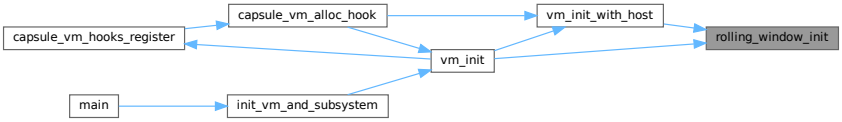
References RollingWindowOfTruth::adaptive_check_accumulator, RollingWindowOfTruth::adaptive_pending, RollingWindowOfTruth::effective_window_size, RollingWindowOfTruth::execution_history, RollingWindowOfTruth::is_warm, RollingWindowOfTruth::last_pattern_diversity, RollingWindowOfTruth::pattern_diversity_check_count, rolling_window_publish_snapsh ROLLING_WINDOW_SIZE, sf_calloc(), sf_free(), RollingWindowOfTruth::snapshot_buffers, RollingWindowOfTruth::snapshot_effectiv RollingWindowOfTruth::snapshot_index, RollingWindowOfTruth::snapshot_is_warm, RollingWindowOfTruth::snapshot_pending, RollingWindowOfTruth::snapshot_total_executions, RollingWindowOfTruth::snapshot_window_pos, RollingWindowOfTruth::total_exec and RollingWindowOfTruth::window_pos.

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.258.1.8 rolling_window_is_warm()

```
int rolling_window_is_warm (
            const RollingWindowOfTruth * window )
```

Return whether the rolling window has accumulated enough history to be useful.

Returns `window->is_warm`, which is set to 1 by `rolling_window_record_execution()` once `total_executions` $\geq$ 1024. Before that threshold the window is considered "cold" — its contents do not yet represent a stable distribution of the program's word usage, and analysis functions (`rolling_window_find_hottest_word()`, `rolling_window_count_transition()`, adaptive sizing) return early with conservative defaults when `is_warm` is 0.

Parameters

window	The `RollingWindowOfTruth` to query; NULL $\rightarrow$ returns 0.

Returns

1 if the window is warm ($\geq$ 1024 executions); 0 otherwise.

References RollingWindowOfTruth::is_warm.

7.258.1.9 rolling_window_measure_diversity()

```
uint64_t rolling_window_measure_diversity (
            const RollingWindowOfTruth * window )
```

Return the current pattern diversity score for the rolling window.

Triggers a lazy snapshot publish via `rolling_window_publish_snapshot_if_needed()`, acquires a read-side view via `rolling_window_snapshot_view()`, and delegates to the internal `rolling_window_measure_diversity_view()` to count the number of unique adjacent-pair transitions in the active window.

Higher diversity means the window is observing a wide variety of word sequences, indicating that the execution pattern is volatile and heat-stratified lookup is more beneficial than stable-pattern nearest-first lookup. Lower diversity means the program is executing a narrow, repetitive sequence — pattern knowledge is concentrated and reliable.

Used by `dict_adaptive_optimization_pass()` to select lookup strategy (threshold = 70 unique transitions) and by `rolling_window_run_adaptive_pass()` to calibrate `effective_window_size`.

Parameters

window	The `RollingWindowOfTruth` to measure.

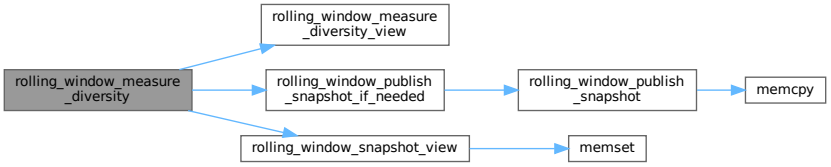
Returns

Count of unique adjacent-pair transitions in the active window; 0 if the window has fewer than 2 executions or `window` is NULL.

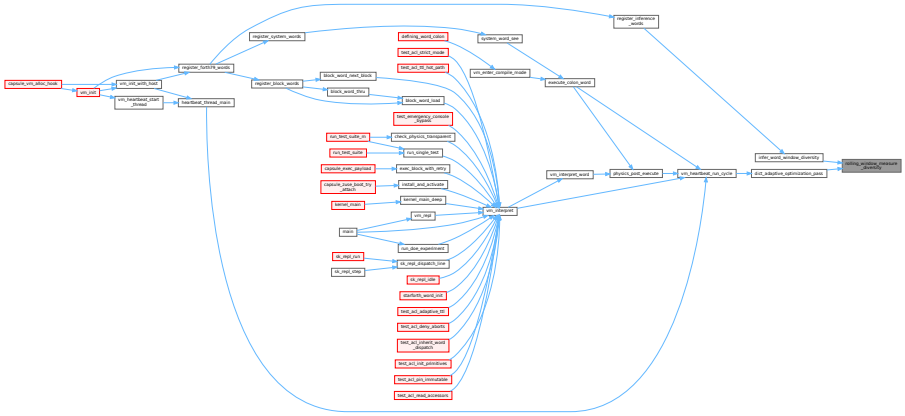
References rolling_window_measure_diversity_view(), rolling_window_publish_snapshot_if_needed(), and rolling_window_snapshot_view().

Referenced by dict_adaptive_optimization_pass(), and infer_word_window_diversity().

Here is the call graph for this function:



Here is the caller graph for this function:



7.258.1.10 rolling_window_pattern_capture_rate()

```
double rolling_window_pattern_capture_rate (
            const RollingWindowOfTruth * window,
            uint32_t test_window_size )
```

Estimate the fraction of transitions that carry unique contextual information.

Measures how many of the recorded adjacent-pair transitions produce a non-zero hash when the preceding `test_window_size` word IDs are incorporated:

```
pattern_hash = (... ((word[i-ws] × 31 + word[i-ws+1]) × 31 + ...) × 31 + word[i+1])
               mod 0x7FFFFFFF
```

A non-zero hash is counted as a "captured" pattern. The capture rate is:

```
rate = patterns_seen / total_transitions × 100  (clamped to [0, 100])
```

In practice, hash collisions with the zero sentinel are extraordinarily rare (probability ~1/2^31), so nearly all transitions contribute to `patterns_seen`. The function thus measures whether `test_window_size` is wide enough to produce a non-trivial hash — a proxy for whether the window introduces meaningful context beyond a single predecessor.

Linearises the circular buffer via `rolling_window_export_execution_history()` before scanning. Returns 0.0 if `window` or `test_window_size` < 1, if the linearised history has fewer than 2 entries, or if `sf_malloc` fails.

Parameters

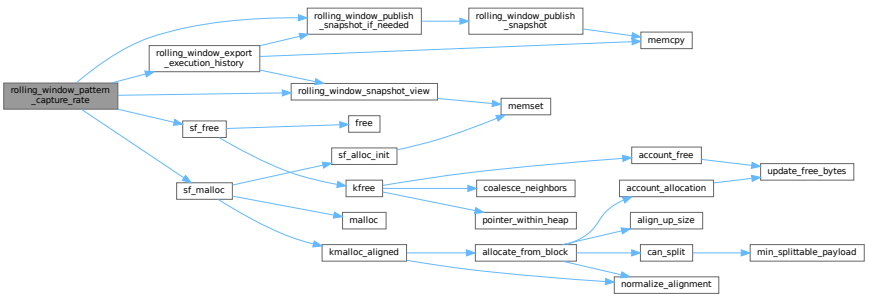
window	The `RollingWindowOfTruth` to analyse.
test_window_size	Number of preceding word IDs to fold into each pattern hash.

Returns

Capture rate as a percentage in [0.0, 100.0].

References rolling_window_view_t::history, rolling_window_export_execution_history(), rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), sf_free(), sf_malloc(), and rolling_window_view_t::total_executions.

Here is the call graph for this function:



7.258.1.11 rolling_window_record_execution()

```
int rolling_window_record_execution (
            RollingWindowOfTruth * window,
            uint32_t word_id )
```

Record one word execution in the circular history buffer.

Writes `word_id` into `window->execution_history`[window_pos], advances `window_pos` modulo `ROLLING_WINDOW_SIZE` (circular wrap), increments `total_executions`, and checks the warm threshold:

- Once `total_executions` $\geq$ 1024 and `is_warm` is still 0, sets `is_warm` = 1. The 1024-execution threshold is calibrated to the POST harness which generates approximately 1450 executions; reaching 1024 guarantees a statistically meaningful execution distribution before diversity or percentile analysis begins.

Sets `snapshot_pending` = 1 to signal that a new snapshot should be published on the next read-side access. When the window is warm and `adaptive_check_accumulator` reaches `ADAPTIVE_CHECK_FREQUENCY`, also sets `adaptive_pending` = 1 to request an `effective_window_size` update.

Called from the interpreter hot path on every word dispatch. All operations are O(1) with no locks and no allocations.

Parameters

window	The `RollingWindowOfTruth` to record into.
word_id	Stable numeric ID of the word just executed.

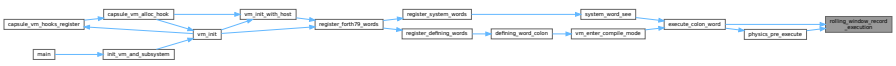
Returns

0 on success; ?1 if `window` or `execution_history` are NULL.

References RollingWindowOfTruth::adaptive_check_accumulator, ADAPTIVE_CHECK_FREQUENCY, RollingWindowOfTruth::adapti RollingWindowOfTruth::execution_history, RollingWindowOfTruth::is_warm, ROLLING_WINDOW_SIZE, RollingWindowOfTruth::snap RollingWindowOfTruth::total_executions, RollingWindowOfTruth::window_pos, and word_id.

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the caller graph for this function:



7.258.1.12 rolling_window_reset()

```
void rolling_window_reset (
            RollingWindowOfTruth * window )
```

Reset the rolling window to an empty cold state without freeing heap memory.

Zeros execution_history via memset and resets the scalar fields: window_pos = 0, total_executions = 0, is_warm = 0. Sets snapshot_pending = 1 so that the next read will publish an empty snapshot.

The three heap buffers (execution_history, snapshot_buffers[0/1]) are preserved; only their contents are zeroed. Intended for benchmark harnesses that need a clean baseline between test runs without the overhead of rolling_window_cleanup() + rolling_window_init().

Parameters

window	The `RollingWindowOfTruth` to reset; no-op if NULL or `execution_history` is NULL.

References RollingWindowOfTruth::execution_history, RollingWindowOfTruth::is_warm, memset(), ROLLING_WINDOW_SIZE, RollingWindowOfTruth::snapshot_pending, RollingWindowOfTruth::total_executions, and RollingWindowOfTruth::window_pos.

Here is the call graph for this function:

7.258.1.13 rolling_window_service()

```
void rolling_window_service (
            RollingWindowOfTruth * window )
```

Service the rolling window — publish pending snapshots and run adaptive sizing.

The public entry point for the rolling window's periodic maintenance. Intended to be called from the heartbeat tick handler or the REPL idle loop at a rate determined by the caller (typically 100 Hz from the APIC timer ISR via `heartbeat_tick()`). Delegates entirely to `rolling_window_service_internal()`:

- Flushes the pending snapshot if `snapshot_pending` is set.
- Runs `rolling_window_run_adaptive_pass()` if `adaptive_pending` is set.

Unlike `rolling_window_check_adaptive_shrink()`, this function does NOT force `adaptive_↩ pending` = 1; it only services flags that the record path has already set. The normal adaptive-pass cadence is therefore governed entirely by `ADAPTIVE_CHECK_FREQUENCY` executions between checks.

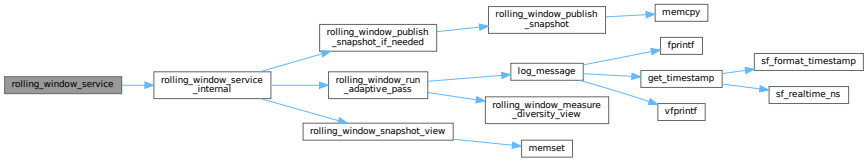
Parameters

window	The `RollingWindowOfTruth` to service; no-op if NULL.

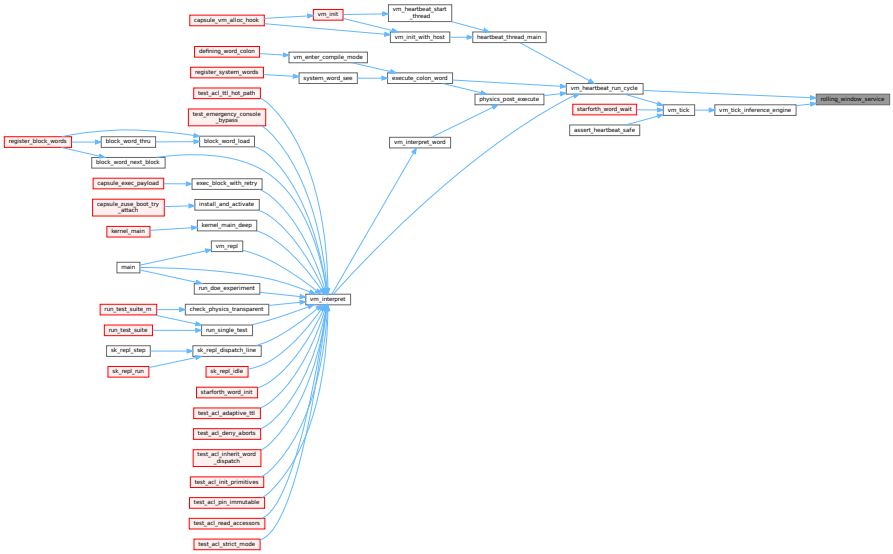
References rolling_window_service_internal().

Referenced by vm_heartbeat_run_cycle(), and vm_tick_inference_engine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.258.1.14 rolling_window_stats_string()

```
char * rolling_window_stats_string (
            const RollingWindowOfTruth * window )
```

Allocate and return a one-page status summary for a `RollingWindowOfTruth`.

Allocates a 1024-byte buffer via `sf_malloc` and fills it with a multi-line report covering:

- `total_executions` — lifetime word-execution count.
- `ROLLING_WINDOW_SIZE` — compile-time capacity constant.
- Warm/cold status: "WARM (representative)" or "COLD (warming up)".
- `window_pos` / `ROLLING_WINDOW_SIZE` — current write-cursor position.

Values are read from the current double-buffer snapshot obtained via `rolling_window_snapshot_view()`; a lazy publish is triggered first by `rolling_window_publish_snapshot_if_needed()`.

The caller is responsible for calling `sf_free()` on the returned pointer. Returns NULL if `window` is NULL or `sf_malloc` fails.

Parameters

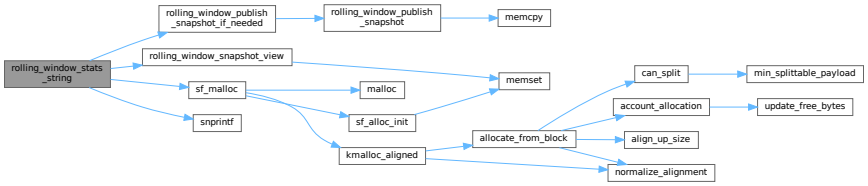
window	The `RollingWindowOfTruth` to describe.

Returns

Heap-allocated NUL-terminated summary string; caller must `sf_free()` it. NULL on failure.

References rolling_window_view_t::is_warm, rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), sf_malloc(), snprintf(), rolling_window_view_t::total_executions, and rolling_window_view_t::window

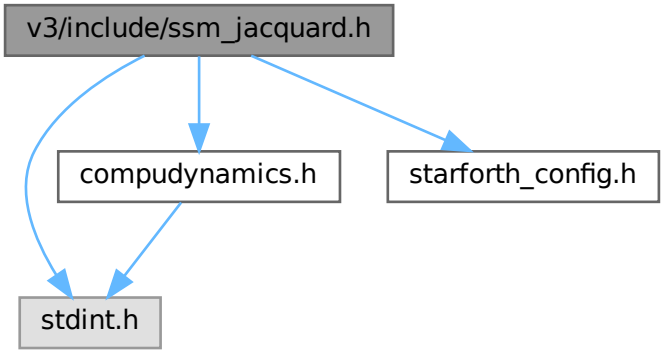
Here is the call graph for this function:



7.259 v3/include/ssm_jacquard.h File Reference

```
#include <stdint.h>
#include "compudynamics.h"
#include "starforth_config.h"
```

Include dependency graph for ssm_jacquard.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct ssm_l8_metrics_t
- struct ssm_config_t
- struct ssm_l8_state_t

Macros

- #define SSM_CFG_L1 0x40u
- #define SSM_CFG_L2 0x20u
- #define SSM_CFG_L3 0x10u
- #define SSM_CFG_L4 0x08u
- #define SSM_CFG_L5 0x04u
- #define SSM_CFG_L6 0x02u
- #define SSM_CFG_L7 0x01u

Enumerations

- enum ssm_l8_mode_t {
  SSM_MODE_C0 = 0x0 , SSM_MODE_C1 = 0x1 , SSM_MODE_C2 = 0x2 , SSM_MODE_C3 = 0x3 ,
  SSM_MODE_C4 = 0x4 , SSM_MODE_C5 = 0x5 , SSM_MODE_C6 = 0x6 , SSM_MODE_C7 = 0x7 ,
  SSM_MODE_C8 = 0x8 , SSM_MODE_C9 = 0x9 , SSM_MODE_C10 = 0xA , SSM_MODE_C11 = 0xB ,
  SSM_MODE_C12 = 0xC , SSM_MODE_C13 = 0xD , SSM_MODE_C14 = 0xE , SSM_MODE_C15 = 0xF }

Functions

- void ssm_l8_init (ssm_l8_state_t *state, ssm_l8_mode_t initial_mode)

  Initialize L8 state (legacy fields only; call ssm_l8_init_table separately)
- void ssm_l8_init_table (ssm_l8_state_t *state)

  Allocate and seed the adaptive config table Sets state->table; must be called after ssm_l8_init().
- void ssm_l8_free_table (ssm_l8_state_t *state)

  Free the adaptive config table.
- void ssm_l8_update_table (ssm_l8_state_t *state, const ssm_l8_metrics_t *metrics, ssm_config_t *config, uint32_t current_window, const uint32_t *recent_word_ids, uint32_t recent_word_ids_count)

  Tick the adaptive UCB table: classify, score, and reselect.
- void ssm_apply_mode_from_table (const ssm_l8_state_t *state, ssm_config_t *config)

  Apply the current table config to ssm_config_t Used on first tick to set the initial DoE-seeded config immediately.
- void ssm_l8_force_config (ssm_l8_state_t *state, ssm_config_t *config, uint8_t config_idx)

  Force the adaptive table onto an externally-chosen config (e.g.
- void ssm_l8_update (const ssm_l8_metrics_t *metrics, ssm_l8_state_t *state)

  Update L8 state based on current metrics (legacy threshold path)
- void ssm_apply_mode (const ssm_l8_state_t *state, ssm_config_t *config)

  Apply L8 mode to SSM configuration (legacy threshold path)
- const char * ssm_l8_mode_name (ssm_l8_mode_t mode)

  Get human-readable mode name.

7.259.1 Macro Definition Documentation

7.259.1.1 SSM_CFG_L1

```
#define SSM_CFG_L1 0x40u
```

7.259.1.2 SSM_CFG_L2

```
#define SSM_CFG_L2 0x20u
```

7.259.1.3 SSM_CFG_L3

```
#define SSM_CFG_L3 0x10u
```

7.259.1.4 SSM_CFG_L4

```
#define SSM_CFG_L4 0x08u
```

7.259.1.5 SSM_CFG_L5

```
#define SSM_CFG_L5 0x04u
```

7.259.1.6 SSM_CFG_L6

```
#define SSM_CFG_L6 0x02u
```

7.259.1.7 SSM_CFG_L7

```
#define SSM_CFG_L7 0x01u
```

7.259.2 Enumeration Type Documentation

7.259.2.1 ssm_l8_mode_t

```
enum ssm_l8_mode_t
```

Enumerator

SSM_MODE_C0	
SSM_MODE_C1	
SSM_MODE_C2	
SSM_MODE_C3	
SSM_MODE_C4	
SSM_MODE_C5	
SSM_MODE_C6	
SSM_MODE_C7	
SSM_MODE_C8	
SSM_MODE_C9	
SSM_MODE_C10	
SSM_MODE_C11	
SSM_MODE_C12	
SSM_MODE_C13	
SSM_MODE_C14	
SSM_MODE_C15	

7.259.3 Function Documentation

7.259.3.1 ssm_apply_mode()

```
void ssm_apply_mode (
            const ssm_l8_state_t * state,
            ssm_config_t * config )
```

Apply L8 mode to SSM configuration (legacy threshold path)

Apply L8 mode to SSM configuration (legacy threshold path)

Extracts the four loop-enable bits (L2/L3/L5/L6) from `state->current_mode` and writes them into `config`. Used by the legacy (non-table) L8 path when the adaptive UCB table is not present.

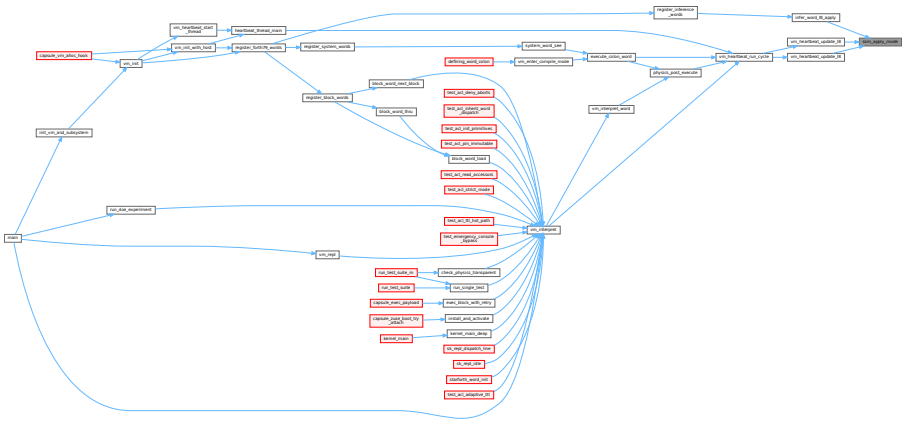
Parameters

state	L8 state containing the committed current mode
config	Runtime configuration to update

References ssm_l8_state_t::current_mode, ssm_config_t::L2_rolling_window, ssm_config_t::L3_linear_decay, ssm_config_t::L5_window_inference, and ssm_config_t::L6_decay_inference.

Referenced by infer_word_l8_apply(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the caller graph for this function:



7.259.3.2 ssm_apply_mode_from_table()

```
void ssm_apply_mode_from_table (
            const ssm_l8_state_t * state,
            ssm_config_t * config )
```

Apply the current table config to ssm_config_t Used on first tick to set the initial DoE-seeded config immediately.

Apply the current table config to ssm_config_t Used on first tick to set the initial DoE-seeded config immediately.

Reads the current winning config entry from the adaptive table and writes all 7 of its bits into `config`. Use this instead of `ssm_apply_mode()` when the adaptive table is present (`state->table` != NULL).

Parameters

state	L8 state with an attached, initialised adaptive table
config	Runtime configuration to update

References CDConfigEntry::config_bits, CDConfigTable::current_config, CDConfigTable::entries, ssm_config_t::L1_heat_tracking, ssm_config_t::L2_rolling_window, ssm_config_t::L3_linear_decay, ssm_config_t::L4_pipelining, ssm_config_t::L5_window_inference, ssm_config_t::L6_decay_inference, ssm_config_t::L7_adaptive_heartrate, SSM_CFG_L1, SSM_CFG_L2, SSM_CFG_L3, SSM_CFG_L4, SSM_CFG_L5, SSM_CFG_L6, SSM_CFG_L7, and ssm_l8_state_t::table.

Referenced by ssm_l8_force_config().

Here is the caller graph for this function:

7.259.3.3 ssm_l8_force_config()

```
void ssm_l8_force_config (
            ssm_l8_state_t * state,
            ssm_config_t * config,
            uint8_t config_idx )
```

Force the adaptive table onto an externally-chosen config (e.g.

a DoE campaign driving a specific tick's config rather than letting the bandit's own UCB selection choose it). See definition for details on why this stays coherent with the bandit's own reward loop.

Force the adaptive table onto an externally-chosen config (e.g.

Sets `current_config` directly (as if the bandit's own UCB selection had picked it) and applies its bits immediately. Since scoring and UCB reselection now run on every heartbeat tick (rev r: one clock, no batching of any kind), this is a one-tick nudge, not a hold: the forced config earns exactly one tick's worth of reward signal into its own regime_score, then the very next tick's ssm_l8_update_table() call scores it and reselects on its own, same as any other tick. Calling this repeatedly (e.g. once per DoE draw, faster than the VM's own tick cadence) keeps a config in effect in practice, not because it's pinned – because it's what's currently forced, most recently.

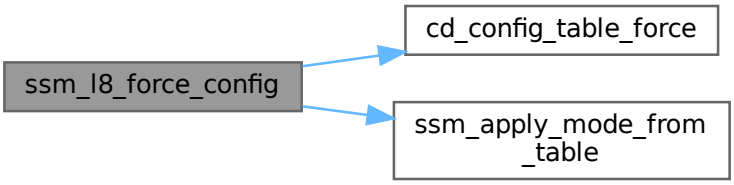
Parameters

state	L8 state with an attached, initialised adaptive table
config	Runtime configuration to update immediately
config_idx	Desired config, masked to the 128-entry (7-bit) space

References cd_config_table_force(), ssm_apply_mode_from_table(), and ssm_l8_state_t::table.

Referenced by infer_word_l8_table_force().

Here is the call graph for this function:



Here is the caller graph for this function:

7.259.3.4 ssm_l8_free_table()

```
void ssm_l8_free_table (
            ssm_l8_state_t * state )
```

Free the adaptive config table.

Free the adaptive config table.

Safe to call with `state->table` == NULL. Sets `state->table` to NULL after freeing so the state falls back to legacy mode.

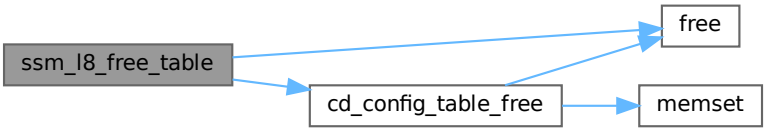
Parameters

state	L8 state whose table is freed

References cd_config_table_free(), free(), and ssm_l8_state_t::table.

Referenced by vm_cleanup().

Here is the call graph for this function:



Here is the caller graph for this function:

7.259.3.5 ssm_l8_init()

```
void ssm_l8_init (
            ssm_l8_state_t * state,
            ssm_l8_mode_t initial_mode )
```

Initialize L8 state (legacy fields only; call ssm_l8_init_table separately)

Initialize L8 state (legacy fields only; call ssm_l8_init_table separately)

Sets `current_mode` and `pending_mode` to `initial_mode`, zeroes `hysteresis_counter`, and sets `table` to NULL. Callers must invoke `ssm_l8_init_table()` separately if the adaptive UCB table is required.

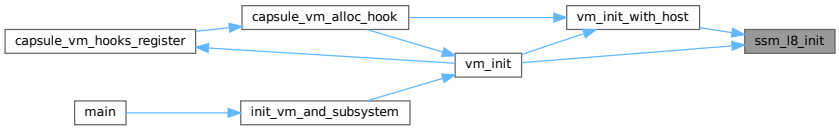
Parameters

state	Pointer to the L8 state to initialise
initial_mode	Starting mode (typically `SSM_MODE_C0`)

References ssm_l8_state_t::current_mode, ssm_l8_state_t::hysteresis_counter, ssm_l8_state_t::pending_mode, and ssm_l8_state_t::table.

Referenced by vm_init(), and vm_init_with_host().

Here is the caller graph for this function:



7.259.3.6 ssm_l8_init_table()

```
void ssm_l8_init_table (
            ssm_l8_state_t * state )
```

Allocate and seed the adaptive config table Sets state->table; must be called after ssm_l8_init().

On allocation failure, state->table remains NULL (legacy mode is used).

Allocate and seed the adaptive config table Sets state->table; must be called after ssm_l8_init().

Mallocs and initialises a `CDConfigTable` sized by `cd_tuning_word()`, seeds it with `ssm_l8_seed_from_doe()`, and attaches it to `state->table`. If either allocation fails, `state->table` is set to NULL and the system falls back to legacy mode (`ssm_apply_mode()`).

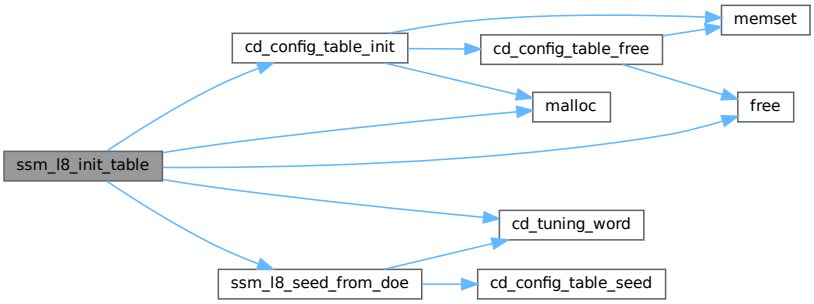
Parameters

state	L8 state that will own the table

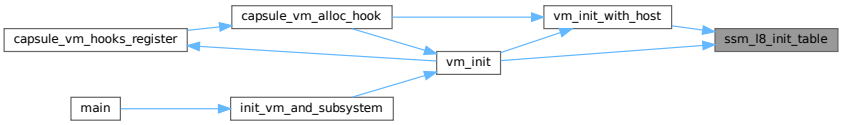
References cd_config_table_init(), cd_tuning_word(), free(), malloc(), ssm_l8_seed_from_doe(), and ssm_l8_state_t::table.

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.259.3.7 ssm_l8_mode_name()

```
const char * ssm_l8_mode_name (
            ssm_l8_mode_t mode )
```

Get human-readable mode name.

Get human-readable mode name.

Returns a static string such as "C4_TEMPORAL" or "C15_FULL_ADAPTIVE". Modes marked ? TOP 5% are the configurations favoured by the DoE study. Returns "UNKNOWN" for out-of-range values.

Parameters

mode	L8 mode enumeration value

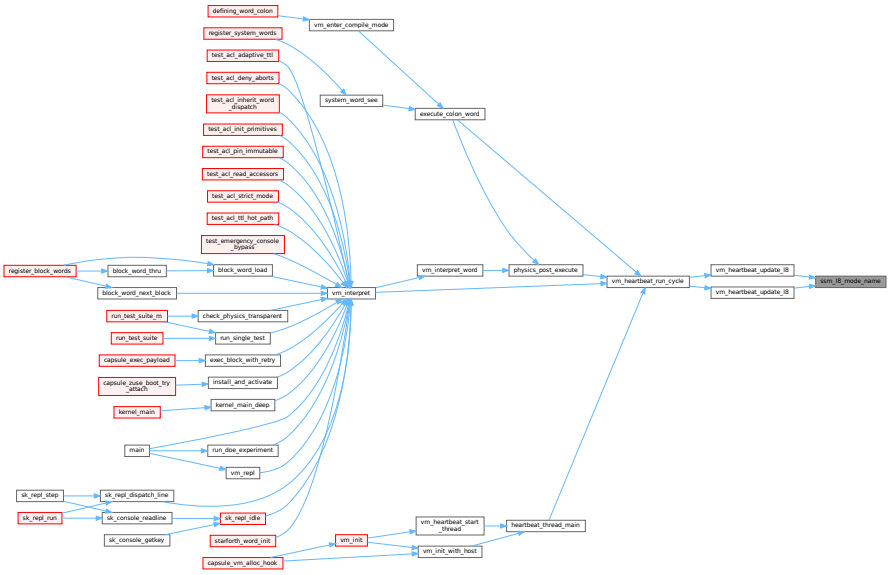
Returns

Pointer to a static null-terminated string (never NULL)

References SSM_MODE_C0, SSM_MODE_C1, SSM_MODE_C10, SSM_MODE_C11, SSM_MODE_C12, SSM_MODE_C13, SSM_MODE_C14, SSM_MODE_C15, SSM_MODE_C2, SSM_MODE_C3, SSM_MODE_C4, SSM_MODE_C5, SSM_MODE_C6, SSM_MODE_C7, SSM_MODE_C8, and SSM_MODE_C9.

Referenced by vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the caller graph for this function:



7.259.3.8 ssm_l8_update()

```
void ssm_l8_update (
            const ssm_l8_metrics_t * metrics,
            ssm_l8_state_t * state )
```

Update L8 state based on current metrics (legacy threshold path)

Update L8 state based on current metrics (legacy threshold path)

Classifies `metrics` into four binary feature bits (L2/L3/L5/L6) derived from DoE top-5% analysis and computes a target mode. Applies hysteresis: the mode only commits after `SSM_HYSTERESIS_TICKS` consecutive ticks with the same target. Called by the heartbeat thread on every tick.

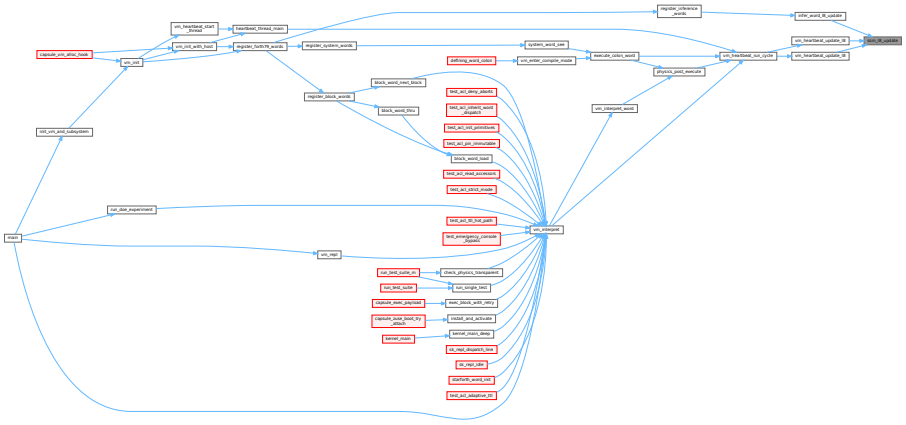
Parameters

metrics	Runtime metrics snapshot (entropy, CV, temporal decay)
state	L8 state to update; `current_mode` is written on commit

References ssm_l8_state_t::current_mode, ssm_l8_metrics_t::cv, ssm_l8_metrics_t::entropy, ssm_l8_state_t::hysteresis_counter, ssm_l8_state_t::pending_mode, SSM_CV_HIGH_THRESHOLD, SSM_ENTROPY_HIGH_THRESHOLD, SSM_HYSTERESIS_TICK SSM_MODE_C0, SSM_TEMPORAL_DECAY_LOW_THRESHOLD, SSM_TEMPORAL_DECAY_THRESHOLD, and ssm_l8_metrics_t::temporal_decay.

Referenced by infer_word_l8_update(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the caller graph for this function:



7.259.3.9 ssm_l8_update_table()

```
void ssm_l8_update_table (
            ssm_l8_state_t * state,
            const ssm_l8_metrics_t * metrics,
            ssm_config_t * config,
            uint32_t current_window,
            const uint32_t * recent_word_ids,
            uint32_t recent_word_ids_count )
```

Tick the adaptive UCB table: classify, score, and reselect.

Classifies the current tick into one of 8 regimes (entropy x CV x locality) and stores it unconditionally, then scores the current config from this tick's outcome and UCB-reselects, every single call – no batching, no derived counter of any kind (VM-FLEET-ATTRACTOR-DESIGN- 20260705.md rev r: one clock, the heartbeat tick, applied uniformly). The locality component of both the regime and the joint-convergence reward comes from cd_classify_ids() over recent_word_ids – purely execution-count-derived, no wall-clock input (rev s; see header comment on this function's declaration for why that matters). Also mirrors the winning L2/L3/L5/L6 bits back into `state->current_mode` for diagnostic display. No-op if `state->table` is NULL.

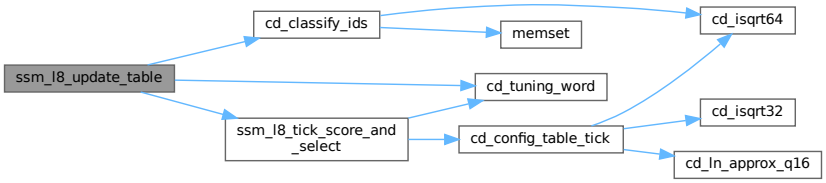
Parameters

state	L8 state with attached adaptive table
metrics	Current runtime metrics snapshot
config	Runtime config updated this tick
current_window	Current rolling window width
recent_word_ids	Recent word-execution-ID history, oldest first
recent_word_ids_count	Number of valid entries in recent_word_ids

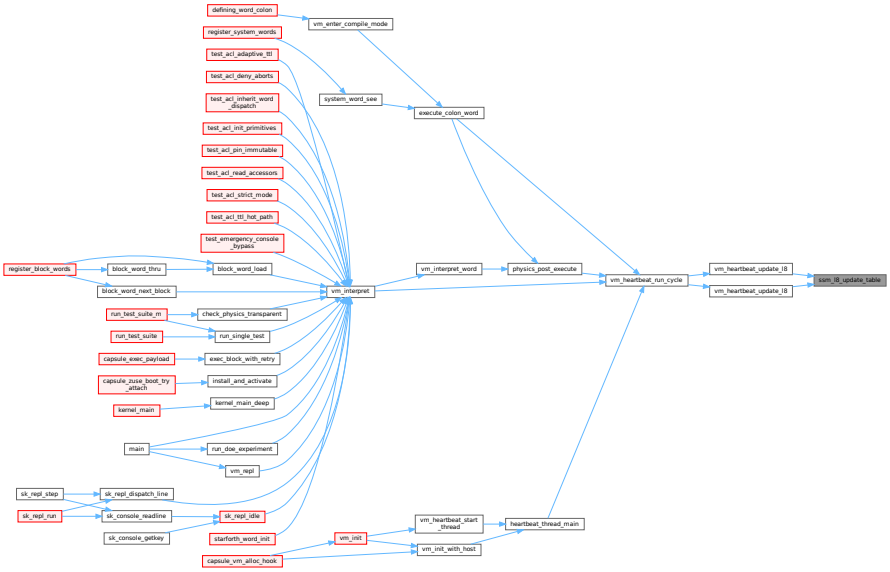
References cd_classify_ids(), cd_tuning_word(), CDConfigEntry::config_bits, CDConfigTable::current_config, ssm_l8_state_t::current_mode, CDConfigTable::current_regime, ssm_l8_metrics_t::cv, CDConfigTable::entries, ssm_l8_metrics_t::entropy, CDTuning::locality_high_threshold_q16, CDTuning::locality_lookback, CDRegimeMetrics::locality_q16, SSM_CFG_L2, SSM_CFG_L3, SSM_CFG_L5, SSM_CFG_L6, SSM_CV_HIGH_THRESHOLD, SSM_ENTROPY_HIGH_THRESHO ssm_l8_tick_score_and_select(), and ssm_l8_state_t::table.

Referenced by vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the call graph for this function:



Here is the caller graph for this function:



7.260 v3/include/starforth_config.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define STARFORTH_CONFIG_STRICT_PTR_DEFAULT 1
- #define STARFORTH_CONFIG_ENABLE_HOTWORDS_CACHE_DEFAULT 0
- #define STARFORTH_CONFIG_ENABLE_PIPELINING_DEFAULT 0
- #define STARFORTH_CONFIG_ROLLING_WINDOW_SIZE_DEFAULT 4096
- #define STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT 8
- #define STARFORTH_CONFIG_ADAPTIVE_SHRINK_RATE_DEFAULT 50
- #define STARFORTH_CONFIG_ADAPTIVE_MIN_WINDOW_SIZE_DEFAULT 256

- #define STARFORTH_CONFIG_ADAPTIVE_CHECK_FREQUENCY_DEFAULT 512
- #define STARFORTH_CONFIG_ADAPTIVE_GROWTH_THRESHOLD_DEFAULT 5
- #define STARFORTH_CONFIG_INITIAL_DECAY_SLOPE_Q48_DEFAULT 21845
- #define STARFORTH_CONFIG_DECAY_MIN_INTERVAL_DEFAULT 1000ULL
- #define STARFORTH_CONFIG_DECAY_RATE_PER_US_Q16_DEFAULT 1
- #define STARFORTH_CONFIG_HEARTBEAT_THREAD_ENABLED_DEFAULT 1
- #define STARFORTH_CONFIG_HEARTBEAT_TICK_NS_DEFAULT 10000ULL
- #define STARFORTH_CONFIG_HEARTBEAT_INFERENCE_FREQUENCY_DEFAULT 1000
- #define STARFORTH_CONFIG_STADIUM_VM_MEMORY_PERCENT_DEFAULT 50
- #define STARFORTH_CONFIG_STADIUM_CONTAINS_DEPTH_MAX_DEFAULT 5
- #define STARFORTH_CONFIG_STADIUM_CAPACITY_TICK_DEFAULT 4000
- #define STARFORTH_CONFIG_STADIUM_MEMORY_PERCENT_DEFAULT 1
- #define STARFORTH_CONFIG_STADIUM_WORD_HEAT_QUANTUM_DEFAULT 2048
- #define STARFORTH_CONFIG_STADIUM_WORD_COOL_RATE_Q48_DEFAULT 21845
- #define STARFORTH_CONFIG_STADIUM_BLOCK_HEAT_QUANTUM_DEFAULT 2048
- #define STARFORTH_CONFIG_STADIUM_BLOCK_COOL_RATE_Q48_DEFAULT 21845
- #define STARFORTH_CONFIG_STADIUM_BLOCK_TRACK_CAP_MULT_DEFAULT 2
- #define STARFORTH_CONFIG_HEARTBEAT_CHECK_FREQUENCY_DEFAULT 256
- #define STARFORTH_CONFIG_HEARTBEAT_WINDOW_TUNING_FREQUENCY_DEFAULT 1000
- #define STARFORTH_CONFIG_HEARTBEAT_SLOPE_VALIDATION_FREQUENCY_DEFAULT 5000
- #define STARFORTH_CONFIG_SSM_ENTROPY_HIGH_THRESHOLD_DEFAULT 0.75
- #define STARFORTH_CONFIG_SSM_CV_HIGH_THRESHOLD_DEFAULT 0.15
- #define STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_THRESHOLD_DEFAULT 0.5
- #define STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_LOW_THRESHOLD_DEFAULT 0.3
- #define STARFORTH_CONFIG_SSM_HYSTERESIS_TICKS_DEFAULT 5
- #define STARFORTH_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT (0x8000LL)
- #define STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT 1
- #define STARFORTH_CONFIG_MIN_SAMPLES_FOR_SPECULATION_DEFAULT 10
- #define STARFORTH_CONFIG_MISPREDICTION_COST_Q48_DEFAULT (25LL << 16)
- #define STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT (0x1199AL)
- #define STRICT_PTR STARFORTH_CONFIG_STRICT_PTR_DEFAULT
- #define VM_STRICT_PTR STRICT_PTR
- #define ENABLE_HOTWORDS_CACHE STARFORTH_CONFIG_ENABLE_HOTWORDS_CACHE_DEFAULT
- #define ENABLE_PIPELINING STARFORTH_CONFIG_ENABLE_PIPELINING_DEFAULT
- #define ROLLING_WINDOW_SIZE STARFORTH_CONFIG_ROLLING_WINDOW_SIZE_DEFAULT
- #define TRANSITION_WINDOW_SIZE STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT
- #define ADAPTIVE_SHRINK_RATE STARFORTH_CONFIG_ADAPTIVE_SHRINK_RATE_DEFAULT
- #define ADAPTIVE_MIN_WINDOW_SIZE STARFORTH_CONFIG_ADAPTIVE_MIN_WINDOW_SIZE_DEFAULT
- #define ADAPTIVE_CHECK_FREQUENCY STARFORTH_CONFIG_ADAPTIVE_CHECK_FREQUENCY_DEFAULT
- #define ADAPTIVE_GROWTH_THRESHOLD STARFORTH_CONFIG_ADAPTIVE_GROWTH_THRESHOLD_DEFAULT
- #define INITIAL_DECAY_SLOPE_Q48 STARFORTH_CONFIG_INITIAL_DECAY_SLOPE_Q48_DEFAULT
- #define DECAY_MIN_INTERVAL STARFORTH_CONFIG_DECAY_MIN_INTERVAL_DEFAULT
- #define DECAY_RATE_PER_US_Q16 STARFORTH_CONFIG_DECAY_RATE_PER_US_Q16_DEFAULT
- #define HEARTBEAT_THREAD_ENABLED STARFORTH_CONFIG_HEARTBEAT_THREAD_ENABLED_DEFAULT
- #define HEARTBEAT_TICK_NS STARFORTH_CONFIG_HEARTBEAT_TICK_NS_DEFAULT
- #define HEARTBEAT_INFERENCE_FREQUENCY STARFORTH_CONFIG_HEARTBEAT_INFERENCE_FREQUENCY_DEFA
- #define STADIUM_VM_MEMORY_PERCENT STARFORTH_CONFIG_STADIUM_VM_MEMORY_PERCENT_DEFAULT
- #define STADIUM_CONTAINS_DEPTH_MAX STARFORTH_CONFIG_STADIUM_CONTAINS_DEPTH_MAX_DEFAULT
- #define STADIUM_CAPACITY_TICK STARFORTH_CONFIG_STADIUM_CAPACITY_TICK_DEFAULT
- #define STADIUM_MEMORY_PERCENT STARFORTH_CONFIG_STADIUM_MEMORY_PERCENT_DEFAULT
- #define STADIUM_WORD_HEAT_QUANTUM STARFORTH_CONFIG_STADIUM_WORD_HEAT_QUANTUM_DEFAULT
- #define STADIUM_WORD_COOL_RATE_Q48 STARFORTH_CONFIG_STADIUM_WORD_COOL_RATE_Q48_DEFAULT
- #define STADIUM_BLOCK_HEAT_QUANTUM STARFORTH_CONFIG_STADIUM_BLOCK_HEAT_QUANTUM_DEFAULT
- #define STADIUM_BLOCK_COOL_RATE_Q48 STARFORTH_CONFIG_STADIUM_BLOCK_COOL_RATE_Q48_DEFAULT
- #define STADIUM_BLOCK_TRACK_CAP_MULT STARFORTH_CONFIG_STADIUM_BLOCK_TRACK_CAP_MULT_DEFAULT

- #define HEARTBEAT_CHECK_FREQUENCY STARFORTH_CONFIG_HEARTBEAT_CHECK_FREQUENCY_DEFAULT
- #define HEARTBEAT_WINDOW_TUNING_FREQUENCY STARFORTH_CONFIG_HEARTBEAT_WINDOW_TUNING_FREQU
- #define HEARTBEAT_SLOPE_VALIDATION_FREQUENCY STARFORTH_CONFIG_HEARTBEAT_SLOPE_VALIDATION_FRE
- #define SSM_ENTROPY_HIGH_THRESHOLD STARFORTH_CONFIG_SSM_ENTROPY_HIGH_THRESHOLD_DEFAULT
- #define SSM_CV_HIGH_THRESHOLD STARFORTH_CONFIG_SSM_CV_HIGH_THRESHOLD_DEFAULT
- #define SSM_TEMPORAL_DECAY_THRESHOLD STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_THRESHOLD_DEFAU
- #define SSM_TEMPORAL_DECAY_LOW_THRESHOLD STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_LOW_THRESH
- #define SSM_HYSTERESIS_TICKS STARFORTH_CONFIG_SSM_HYSTERESIS_TICKS_DEFAULT
- #define SPECULATION_THRESHOLD_Q48 STARFORTH_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT
- #define SPECULATION_DEPTH STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT
- #define MIN_SAMPLES_FOR_SPECULATION STARFORTH_CONFIG_MIN_SAMPLES_FOR_SPECULATION_DEFAULT
- #define MISPREDICTION_COST_Q48 STARFORTH_CONFIG_MISPREDICTION_COST_Q48_DEFAULT
- #define MINIMUM_PREFETCH_ROI STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT

7.260.1 Macro Definition Documentation

7.260.1.1 ADAPTIVE_CHECK_FREQUENCY

```
#define ADAPTIVE_CHECK_FREQUENCY STARFORTH_CONFIG_ADAPTIVE_CHECK_FREQUENCY_DEFAULT
```

7.260.1.2 ADAPTIVE_GROWTH_THRESHOLD

```
#define ADAPTIVE_GROWTH_THRESHOLD STARFORTH_CONFIG_ADAPTIVE_GROWTH_THRESHOLD_DEFAULT
```

7.260.1.3 ADAPTIVE_MIN_WINDOW_SIZE

```
#define ADAPTIVE_MIN_WINDOW_SIZE STARFORTH_CONFIG_ADAPTIVE_MIN_WINDOW_SIZE_DEFAULT
```

7.260.1.4 ADAPTIVE_SHRINK_RATE

```
#define ADAPTIVE_SHRINK_RATE STARFORTH_CONFIG_ADAPTIVE_SHRINK_RATE_DEFAULT
```

7.260.1.5 DECAY_MIN_INTERVAL

```
#define DECAY_MIN_INTERVAL STARFORTH_CONFIG_DECAY_MIN_INTERVAL_DEFAULT
```

7.260.1.6 DECAY_RATE_PER_US_Q16

```
#define DECAY_RATE_PER_US_Q16 STARFORTH_CONFIG_DECAY_RATE_PER_US_Q16_DEFAULT
```

7.260.1.7 ENABLE_HOTWORDS_CACHE

```
#define ENABLE_HOTWORDS_CACHE STARFORTH_CONFIG_ENABLE_HOTWORDS_CACHE_DEFAULT
```

7.260.1.8 ENABLE_PIPELINING

```
#define ENABLE_PIPELINING STARFORTH_CONFIG_ENABLE_PIPELINING_DEFAULT
```

7.260.1.9 HEARTBEAT_CHECK_FREQUENCY

```
#define HEARTBEAT_CHECK_FREQUENCY STARFORTH_CONFIG_HEARTBEAT_CHECK_FREQUENCY_DEFAULT
```

7.260.1.10 HEARTBEAT_INFERENCE_FREQUENCY

```
#define HEARTBEAT_INFERENCE_FREQUENCY STARFORTH_CONFIG_HEARTBEAT_INFERENCE_FREQUENCY_DEFAULT
```

7.260.1.11 HEARTBEAT_SLOPE_VALIDATION_FREQUENCY

```
#define HEARTBEAT_SLOPE_VALIDATION_FREQUENCY STARFORTH_CONFIG_HEARTBEAT_SLOPE_VALIDATION_FREQUENCY_DEFAULT
```

7.260.1.12 HEARTBEAT_THREAD_ENABLED

```
#define HEARTBEAT_THREAD_ENABLED STARFORTH_CONFIG_HEARTBEAT_THREAD_ENABLED_DEFAULT
```

7.260.1.13 HEARTBEAT_TICK_NS

```
#define HEARTBEAT_TICK_NS STARFORTH_CONFIG_HEARTBEAT_TICK_NS_DEFAULT
```

7.260.1.14 HEARTBEAT_WINDOW_TUNING_FREQUENCY

```
#define HEARTBEAT_WINDOW_TUNING_FREQUENCY STARFORTH_CONFIG_HEARTBEAT_WINDOW_TUNING_FREQUENCY_DEFAULT
```

7.260.1.15 INITIAL_DECAY_SLOPE_Q48

```
#define INITIAL_DECAY_SLOPE_Q48 STARFORTH_CONFIG_INITIAL_DECAY_SLOPE_Q48_DEFAULT
```

7.260.1.16 MIN_SAMPLES_FOR_SPECULATION

```
#define MIN_SAMPLES_FOR_SPECULATION STARFORTH_CONFIG_MIN_SAMPLES_FOR_SPECULATION_DEFAULT
```

7.260.1.17 MINIMUM_PREFETCH_ROI

```
#define MINIMUM_PREFETCH_ROI STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT
```

7.260.1.18 MISPREDICTION_COST_Q48

```
#define MISPREDICTION_COST_Q48 STARFORTH_CONFIG_MISPREDICTION_COST_Q48_DEFAULT
```

7.260.1.19 ROLLING_WINDOW_SIZE

```
#define ROLLING_WINDOW_SIZE STARFORTH_CONFIG_ROLLING_WINDOW_SIZE_DEFAULT
```

7.260.1.20 SPECULATION_DEPTH

```
#define SPECULATION_DEPTH STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT
```

7.260.1.21 SPECULATION_THRESHOLD_Q48

```
#define SPECULATION_THRESHOLD_Q48 STARFORTH_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT
```

7.260.1.22 SSM_CV_HIGH_THRESHOLD

```
#define SSM_CV_HIGH_THRESHOLD STARFORTH_CONFIG_SSM_CV_HIGH_THRESHOLD_DEFAULT
```

7.260.1.23 SSM_ENTROPY_HIGH_THRESHOLD

```
#define SSM_ENTROPY_HIGH_THRESHOLD STARFORTH_CONFIG_SSM_ENTROPY_HIGH_THRESHOLD_DEFAULT
```

7.260.1.24 SSM_HYSTERESIS_TICKS

```
#define SSM_HYSTERESIS_TICKS STARFORTH_CONFIG_SSM_HYSTERESIS_TICKS_DEFAULT
```

7.260.1.25 SSM_TEMPORAL_DECAY_LOW_THRESHOLD

```
#define SSM_TEMPORAL_DECAY_LOW_THRESHOLD STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_LOW_THRESHOLD_DEFAULT
```

7.260.1.26 SSM_TEMPORAL_DECAY_THRESHOLD

```
#define SSM_TEMPORAL_DECAY_THRESHOLD STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_THRESHOLD_DEFAULT
```

7.260.1.27 STADIUM_BLOCK_COOL_RATE_Q48

```
#define STADIUM_BLOCK_COOL_RATE_Q48 STARFORTH_CONFIG_STADIUM_BLOCK_COOL_RATE_Q48_DEFAULT
```

7.260.1.28 STADIUM_BLOCK_HEAT_QUANTUM

```
#define STADIUM_BLOCK_HEAT_QUANTUM STARFORTH_CONFIG_STADIUM_BLOCK_HEAT_QUANTUM_DEFAULT
```

7.260.1.29 STADIUM_BLOCK_TRACK_CAP_MULT

```
#define STADIUM_BLOCK_TRACK_CAP_MULT STARFORTH_CONFIG_STADIUM_BLOCK_TRACK_CAP_MULT_DEFAULT
```

7.260.1.30 STADIUM_CAPACITY_TICK

```
#define STADIUM_CAPACITY_TICK STARFORTH_CONFIG_STADIUM_CAPACITY_TICK_DEFAULT
```

7.260.1.31 STADIUM_CONTAINS_DEPTH_MAX

```
#define STADIUM_CONTAINS_DEPTH_MAX STARFORTH_CONFIG_STADIUM_CONTAINS_DEPTH_MAX_DEFAULT
```

7.260.1.32 STADIUM_MEMORY_PERCENT

```
#define STADIUM_MEMORY_PERCENT STARFORTH_CONFIG_STADIUM_MEMORY_PERCENT_DEFAULT
```

7.260.1.33 STADIUM_VM_MEMORY_PERCENT

```
#define STADIUM_VM_MEMORY_PERCENT STARFORTH_CONFIG_STADIUM_VM_MEMORY_PERCENT_DEFAULT
```

7.260.1.34 STADIUM_WORD_COOL_RATE_Q48

```
#define STADIUM_WORD_COOL_RATE_Q48 STARFORTH_CONFIG_STADIUM_WORD_COOL_RATE_Q48_DEFAULT
```

7.260.1.35 STADIUM_WORD_HEAT_QUANTUM

```
#define STADIUM_WORD_HEAT_QUANTUM STARFORTH_CONFIG_STADIUM_WORD_HEAT_QUANTUM_DEFAULT
```

7.260.1.36 STARFORTH_CONFIG_ADAPTIVE_CHECK_FREQUENCY_DEFAULT

```
#define STARFORTH_CONFIG_ADAPTIVE_CHECK_FREQUENCY_DEFAULT 512
```

7.260.1.37 STARFORTH_CONFIG_ADAPTIVE_GROWTH_THRESHOLD_DEFAULT

```
#define STARFORTH_CONFIG_ADAPTIVE_GROWTH_THRESHOLD_DEFAULT 5
```

7.260.1.38 STARFORTH_CONFIG_ADAPTIVE_MIN_WINDOW_SIZE_DEFAULT

```
#define STARFORTH_CONFIG_ADAPTIVE_MIN_WINDOW_SIZE_DEFAULT 256
```

7.260.1.39 STARFORTH_CONFIG_ADAPTIVE_SHRINK_RATE_DEFAULT

```
#define STARFORTH_CONFIG_ADAPTIVE_SHRINK_RATE_DEFAULT 50
```

7.260.1.40 STARFORTH_CONFIG_DECAY_MIN_INTERVAL_DEFAULT

```
#define STARFORTH_CONFIG_DECAY_MIN_INTERVAL_DEFAULT 1000ULL
```

7.260.1.41 STARFORTH_CONFIG_DECAY_RATE_PER_US_Q16_DEFAULT

```
#define STARFORTH_CONFIG_DECAY_RATE_PER_US_Q16_DEFAULT 1
```

7.260.1.42 STARFORTH_CONFIG_ENABLE_HOTWORDS_CACHE_DEFAULT

```
#define STARFORTH_CONFIG_ENABLE_HOTWORDS_CACHE_DEFAULT 0
```

7.260.1.43 STARFORTH_CONFIG_ENABLE_PIPELINING_DEFAULT

```
#define STARFORTH_CONFIG_ENABLE_PIPELINING_DEFAULT 0
```

7.260.1.44 STARFORTH_CONFIG_HEARTBEAT_CHECK_FREQUENCY_DEFAULT

```
#define STARFORTH_CONFIG_HEARTBEAT_CHECK_FREQUENCY_DEFAULT 256
```

7.260.1.45 STARFORTH_CONFIG_HEARTBEAT_INFERENCE_FREQUENCY_DEFAULT

```
#define STARFORTH_CONFIG_HEARTBEAT_INFERENCE_FREQUENCY_DEFAULT 1000
```

7.260.1.46 STARFORTH_CONFIG_HEARTBEAT_SLOPE_VALIDATION_FREQUENCY_DEFAULT

```
#define STARFORTH_CONFIG_HEARTBEAT_SLOPE_VALIDATION_FREQUENCY_DEFAULT 5000
```

7.260.1.47 STARFORTH_CONFIG_HEARTBEAT_THREAD_ENABLED_DEFAULT

```
#define STARFORTH_CONFIG_HEARTBEAT_THREAD_ENABLED_DEFAULT 1
```

7.260.1.48 STARFORTH_CONFIG_HEARTBEAT_TICK_NS_DEFAULT

```
#define STARFORTH_CONFIG_HEARTBEAT_TICK_NS_DEFAULT 10000ULL
```

7.260.1.49 STARFORTH_CONFIG_HEARTBEAT_WINDOW_TUNING_FREQUENCY_DEFAULT

```
#define STARFORTH_CONFIG_HEARTBEAT_WINDOW_TUNING_FREQUENCY_DEFAULT 1000
```

7.260.1.50 STARFORTH_CONFIG_INITIAL_DECAY_SLOPE_Q48_DEFAULT

```
#define STARFORTH_CONFIG_INITIAL_DECAY_SLOPE_Q48_DEFAULT 21845
```

7.260.1.51 STARFORTH_CONFIG_MIN_SAMPLES_FOR_SPECULATION_DEFAULT

```
#define STARFORTH_CONFIG_MIN_SAMPLES_FOR_SPECULATION_DEFAULT 10
```

7.260.1.52 STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT

```
#define STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT (0x1199AL)
```

7.260.1.53 STARFORTH_CONFIG_MISPREDICTION_COST_Q48_DEFAULT

```
#define STARFORTH_CONFIG_MISPREDICTION_COST_Q48_DEFAULT (25LL << 16)
```

7.260.1.54 STARFORTH_CONFIG_ROLLING_WINDOW_SIZE_DEFAULT

```
#define STARFORTH_CONFIG_ROLLING_WINDOW_SIZE_DEFAULT 4096
```

7.260.1.55 STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT

```
#define STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT 1
```

7.260.1.56 STARFORTH_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT

```
#define STARFORTH_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT (0x8000LL)
```

7.260.1.57 STARFORTH_CONFIG_SSM_CV_HIGH_THRESHOLD_DEFAULT

```
#define STARFORTH_CONFIG_SSM_CV_HIGH_THRESHOLD_DEFAULT 0.15
```

7.260.1.58 STARFORTH_CONFIG_SSM_ENTROPY_HIGH_THRESHOLD_DEFAULT

```
#define STARFORTH_CONFIG_SSM_ENTROPY_HIGH_THRESHOLD_DEFAULT 0.75
```

7.260.1.59 STARFORTH_CONFIG_SSM_HYSTERESIS_TICKS_DEFAULT

```
#define STARFORTH_CONFIG_SSM_HYSTERESIS_TICKS_DEFAULT 5
```

7.260.1.60 STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_LOW_THRESHOLD_DEFAULT

```
#define STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_LOW_THRESHOLD_DEFAULT 0.3
```

7.260.1.61 STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_THRESHOLD_DEFAULT

```
#define STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_THRESHOLD_DEFAULT 0.5
```

7.260.1.62 STARFORTH_CONFIG_STADIUM_BLOCK_COOL_RATE_Q48_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_BLOCK_COOL_RATE_Q48_DEFAULT 21845
```

7.260.1.63 STARFORTH_CONFIG_STADIUM_BLOCK_HEAT_QUANTUM_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_BLOCK_HEAT_QUANTUM_DEFAULT 2048
```

7.260.1.64 STARFORTH_CONFIG_STADIUM_BLOCK_TRACK_CAP_MULT_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_BLOCK_TRACK_CAP_MULT_DEFAULT 2
```

7.260.1.65 STARFORTH_CONFIG_STADIUM_CAPACITY_TICK_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_CAPACITY_TICK_DEFAULT 4000
```

7.260.1.66 STARFORTH_CONFIG_STADIUM_CONTAINS_DEPTH_MAX_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_CONTAINS_DEPTH_MAX_DEFAULT 5
```

7.260.1.67 STARFORTH_CONFIG_STADIUM_MEMORY_PERCENT_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_MEMORY_PERCENT_DEFAULT 1
```

7.260.1.68 STARFORTH_CONFIG_STADIUM_VM_MEMORY_PERCENT_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_VM_MEMORY_PERCENT_DEFAULT 50
```

7.260.1.69 STARFORTH_CONFIG_STADIUM_WORD_COOL_RATE_Q48_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_WORD_COOL_RATE_Q48_DEFAULT 21845
```

7.260.1.70 STARFORTH_CONFIG_STADIUM_WORD_HEAT_QUANTUM_DEFAULT

```
#define STARFORTH_CONFIG_STADIUM_WORD_HEAT_QUANTUM_DEFAULT 2048
```

7.260.1.71 STARFORTH_CONFIG_STRICT_PTR_DEFAULT

```
#define STARFORTH_CONFIG_STRICT_PTR_DEFAULT 1
```

7.260.1.72 STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT

```
#define STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT 8
```

7.260.1.73 STRICT_PTR

```
#define STRICT_PTR STARFORTH_CONFIG_STRICT_PTR_DEFAULT
```

7.260.1.74 TRANSITION_WINDOW_SIZE

```
#define TRANSITION_WINDOW_SIZE STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT
```

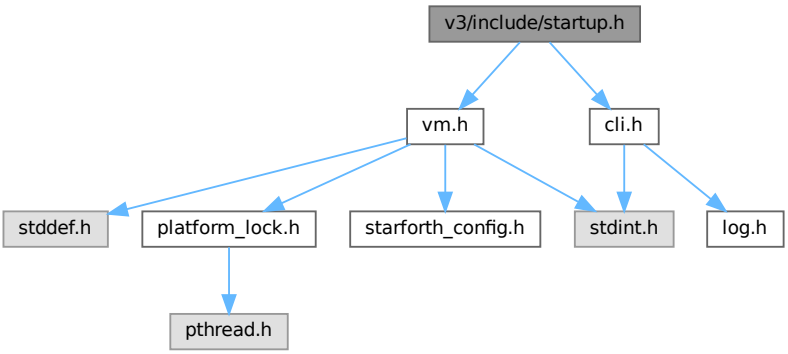
7.260.1.75 VM_STRICT_PTR

```
#define VM_STRICT_PTR STRICT_PTR
```

7.261 v3/include/startup.h File Reference

```
#include "vm.h"
#include "cli.h"
```

Include dependency graph for startup.h:



Functions

- int startup_blkio_init (const CLIConfig *config)
  Initialize block I/O device from CLI configuration.
- void startup_blkio_cleanup (void)
  Clean up block I/O device.
- int startup_vm_init (VM *vm)
  Initialize VM with defaults.
- int startup_block_subsys_init (VM *vm)
  Initialize block subsystem.

7.261.1 Function Documentation

7.261.1.1 startup_blkio_cleanup()

```
void startup_blkio_cleanup (
            void  )
```

Clean up block I/O device.

Flushes and closes block device, frees RAM disk if allocated. Called automatically via atexit().

7.261.1.2 startup_blkio_init()

```
int startup_blkio_init (
            const CLIConfig * config )
```

Initialize block I/O device from CLI configuration.

Opens disk image or allocates RAM disk based on CLIConfig. Must be called before vm_init().

Parameters

config	CLI configuration

Returns

0 on success, 1 on error (logs messages internally)

7.261.1.3 startup_block_subsys_init()

```
int startup_block_subsys_init (
            VM * vm )
```

Initialize block subsystem.

Sets up block buffering and attaches blkio device.

Parameters

vm	Pointer to initialized VM

Returns

0 on success, 1 on error

7.261.1.4 startup_vm_init()

```
int startup_vm_init (
            VM * vm )
```

Initialize VM with defaults.

Initializes the VM state after block device is ready.

Parameters

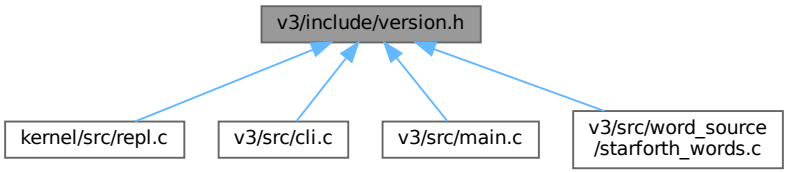
vm	Pointer to VM to initialize

Returns

0 on success, 1 on error

7.262 v3/include/version.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define STARFORTH_VERSION "3.2.0"
- #define STARFORTH_ARCH "amd64"
- #define STARFORTH_TARGET "kernel"
- #define STARFORTH_TIMESTAMP "2026-10-16T22:58:44-04:00"
- #define STARFORTH_VERSION_FULL "StarForth v3.2.0 amd64 kernel 2026-10-16T22:58:44-04:00"
- #define LITHOS_VERSION "2.1.0"
- #define LITHOS_VERSION_STR "LithosAnanke v2.1.0 amd64"

7.262.1 Macro Definition Documentation

7.262.1.1 LITHOS_VERSION

```
#define LITHOS_VERSION "2.1.0"
```

7.262.1.2 LITHOS_VERSION_STR

```
#define LITHOS_VERSION_STR "LithosAnanke v2.1.0 amd64"
```

7.262.1.3 STARFORTH_ARCH

```
#define STARFORTH_ARCH "amd64"
```

7.262.1.4 STARFORTH_TARGET

```
#define STARFORTH_TARGET "kernel"
```

7.262.1.5 STARFORTH_TIMESTAMP

```
#define STARFORTH_TIMESTAMP "2026-10-16T22:58:44-04:00"
```

7.262.1.6 STARFORTH_VERSION

```
#define STARFORTH_VERSION "3.2.0"
```

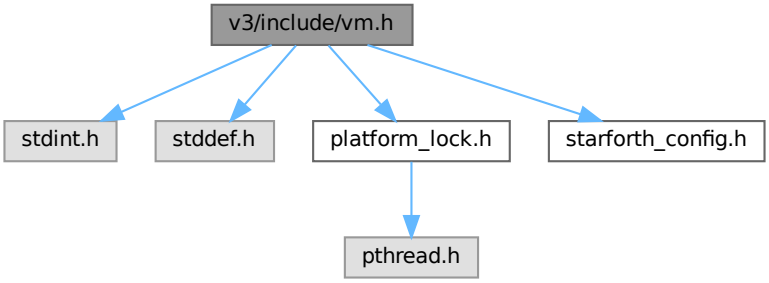
7.262.1.7 STARFORTH_VERSION_FULL

```
#define STARFORTH_VERSION_FULL "StarForth v3.2.0 amd64 kernel 2026-10-16T22:58:44-04:00"
```

7.263 v3/include/vm.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "platform_lock.h"
#include "starforth_config.h"
```

Include dependency graph for vm.h:



Data Structures

- struct RollingWindowOfTruth
- struct HeartbeatSnapshot
- struct HeartbeatTickSnapshot
- struct DictPhysics
- struct HeartbeatState
- struct PipelineGlobalMetrics
- struct DictEntry
- struct VocabularyState
- struct cf_item_t
- struct ControlFlowState
- struct VM

  Main virtual machine state container.

Macros

- #define STACK_SIZE 1024
- #define DICTIONARY_SIZE 1024
- #define WORD_ID_INVALID UINT32_MAX
- #define VM_MEMORY_SIZE (5 * 1024 * 1024) /* 5 MB total VM memory */
- #define INPUT_BUFFER_SIZE 1025
- #define WORD_NAME_MAX 31
- #define COMPILE_BUFFER_SIZE 1024
- #define BLOCK_SIZE 1024 /* 1KB per block */
- #define MAX_BLOCKS (VM_MEMORY_SIZE / BLOCK_SIZE) /* 5120 blocks from 5MB */
- #define BLK_VM_SLOTS 4
- #define BLK_VM_WINDOW_SIZE (BLK_VM_SLOTS * BLOCK_SIZE)
- #define BLK_VM_WINDOW_BASE (VM_MEMORY_SIZE - BLK_VM_WINDOW_SIZE)
- #define DICTIONARY_BLOCKS 2048 /* First 2048 blocks (2MB) for dictionary */
- #define DICTIONARY_MEMORY_SIZE (DICTIONARY_BLOCKS * BLOCK_SIZE)
- #define USER_BLOCKS_START DICTIONARY_BLOCKS /* User blocks start at block 2048 */
- #define LOG_LINE_MAX 64 /* Each log line is 64 bytes */
- #define LOG_LINES_PER_BLOCK (BLOCK_SIZE / LOG_LINE_MAX) /* 16 lines per 1KB block */
- #define LOG_BLOCKS_START 3072 /* Log starts at block 3072 */
- #define LOG_BLOCKS_END 5120 /* Log ends at block 5120 (exclusive) */
- #define LOG_BLOCKS (LOG_BLOCKS_END - LOG_BLOCKS_START) /* 2048 blocks (2MB) for logs */
- #define LOG_LAYER1_MAX_LINES (LOG_BLOCKS * LOG_LINES_PER_BLOCK) /* 32768 max lines */
- #define WORD_IMMEDIATE 0x80 /* Word executes immediately even in compile mode */
- #define WORD_HIDDEN 0x40 /* Word is hidden from dictionary searches */
- #define WORD_SMUDGED 0x20 /* Word is smudged (being defined) - FORTH-79 */
- #define WORD_COMPILED 0x10 /* Word is user-compiled (not built-in) */
- #define WORD_PINNED 0x08 /* Word's execution heat is pinned (cannot decay to zero) */
- #define WORD_FROZEN 0x04 /* Word is frozen: execution heat does not decay (Phase 2) */
- #define ACL_USER_DEFAULT 0x01 /* Default access: users can execute and compile */
- #define ACL_MODE_TTL 0 /* Adaptive TTL countdown; 0 → calls ACL-RECHECK */
- #define ACL_MODE_STRICT 1 /* Re-check every execution (STRICT mode) */
- #define ACL_TTL_OPEN UINT32_MAX /* Permissive TTL before ACL.4th loads */
- #define SPIN_IDLE 0 /* Particle spin state: idle */
- #define CHARGE_NEUTRAL 0 /* Particle charge state: neutral (execution heat acts as charge) */
- #define PHYSICS_STATE_IMMEDIATE 0x01
- #define PHYSICS_STATE_PINNED 0x02
- #define PHYSICS_STATE_HIDDEN 0x04
- #define PHYSICS_STATE_COMPILED 0x08
- #define HEAT_CACHE_DEMOTION_THRESHOLD 10 /* Demotion from hotwords cache threshold */
- #define HEARTBEAT_TICK_BUFFER_SIZE 100000 /* ~100K ticks = ~100 seconds at 1kHz */
- #define VOCAB_FC_BUCKETS 256
- #define CF_STACK_MAX 64
- #define VM_PUSH(vm, val) vm_push_fast(vm, val)
- #define VM_POP(vm) vm_pop_fast(vm)
- #define VM_RPUSH(vm, val) vm_rpush_fast(vm, val)
- #define VM_RPOP(vm) vm_rpop_fast(vm)
- #define LIKELY(x) __builtin_expect(!!(x), 1)
- #define UNLIKELY(x) __builtin_expect(!!(x), 0)

Typedefs

- typedef struct HotwordsCache_s HotwordsCache
- typedef struct WordTransitionMetrics WordTransitionMetrics
- typedef struct VMHostServices VMHostServices
- typedef struct InferenceOutputs InferenceOutputs
- typedef struct HeartbeatWorker HeartbeatWorker
- typedef signed long cell_t
- typedef unsigned long ucell_t
- typedef void(* word_func_t) (struct VM *vm)
- typedef uint64_t vaddr_t
- typedef struct VM VM

Enumerations

- enum vm_mode_t { MODE_INTERPRET = 0 , MODE_COMPILE = 1 }
- enum cf_tag_t {
  CF_BEGIN , CF_IF , CF_ELSE , CF_WHILE ,
  CF_DO , CF_CASE , CF_OF }

Functions

- int vm_addr_ok (struct VM *vm, vaddr_t addr, size_t len)

  Validate that a VM address range lies within `vm->memory`.
- uint8_t * vm_ptr (struct VM *vm, vaddr_t addr)

  Return a host C pointer into VM memory at `addr`.
- uint8_t vm_load_u8 (struct VM *vm, vaddr_t addr)

  Load an unsigned byte from VM memory.
- void vm_store_u8 (struct VM *vm, vaddr_t addr, uint8_t v)

  Store an unsigned byte to VM memory.
- cell_t vm_load_cell (struct VM *vm, vaddr_t addr)

  Load an aligned cell from VM memory.
- void vm_store_cell (struct VM *vm, vaddr_t addr, cell_t v)

  Store an aligned cell to VM memory.
- unsigned vm_get_base (const struct VM *vm)
- void vm_set_base (struct VM *vm, unsigned b)

  Set the VM number base and update both the cell and the host mirror.
- static vaddr_t VM_ADDR (cell_t c)
- static cell_t CELL (vaddr_t a)
- struct DictEntry __attribute__ ((aligned(8))) DictEntry
- void vm_push (VM *vm, cell_t value)

  Push a value onto the data stack.
- cell_t vm_pop (VM *vm)

  Pop a value from the data stack.
- void vm_rpush (VM *vm, cell_t value)

  Push a value onto the return stack.
- cell_t vm_rpop (VM *vm)

  Pop a value from the return stack.
- DictEntry * vm_find_word (VM *vm, const char *name, size_t len)

  Finds a word in the dictionary by name.
- DictEntry * vm_dict_resolve_in_bucket (VM *vm, DictEntry **bucket, size_t n, const char *name, size_t len)

- DictEntry * vm_create_word (VM *vm, const char *name, size_t len, word_func_t func)

  Creates a new word in the dictionary.
- void vm_make_immediate (VM *vm)

  Mark the most recently created word as IMMEDIATE (FORTH-79).
- void vm_bootstrap_root_vocabulary (VM *vm, const char *name)

  Bootstrap a root vocabulary for the VM.
- void vm_hide_word (VM *vm)

  Marks the latest word as hidden.
- void vm_smudge_word (VM *vm)
- void vm_pin_execution_heat (VM *vm)
- void vm_unpin_execution_heat (VM *vm)
- DictEntry * vm_dictionary_find_by_func (VM *vm, word_func_t func)

  Finds a word by its function pointer.
- DictEntry * vm_dictionary_find_latest_by_func (VM *vm, word_func_t func)
- DictEntry * vm_dictionary_lookup_by_word_id (VM *vm, uint32_t word_id)
- int vm_dict_entry_ok (VM *vm, DictEntry *candidate)
- void vm_dictionary_track_entry (VM *vm, DictEntry *entry)
- void vm_dictionary_untrack_entry (VM *vm, DictEntry *entry)
- cell_t * vm_dictionary_get_data_field (DictEntry *entry)

  Gets pointer to a word's data field.
- void vm_compile_word (VM *vm, DictEntry *entry)

  Append a word reference (DictEntry) to the current threaded body.*
- int vm_zuse_cert_install (VM *vm, const uint8_t seed[32], const uint8_t pubkey[32])

  One-time write of the Zuse cert (seed + derived pubkey) – blows the fuse.
- void * vm_allot (VM *vm, size_t bytes)

  Allocates memory from the VM's dictionary space.
- void vm_align (VM *vm)

  Aligns the VM's dictionary pointer (here) to the cell boundary.
- int vm_parse_word (VM *vm, char *word, size_t max_len)

  Parse the next whitespace-delimited token from the input buffer.
- int vm_parse_number (VM *vm, const char *str, cell_t *value)

  Attempt to parse a string as an integer in the current number base.
- void vm_enter_compile_mode (VM *vm, const char *name, size_t len)

  Begin a colon definition and switch the VM to compile mode.
- void vm_exit_compile_mode (VM *vm)

  Finalise a colon definition and return to interpret mode.
- void execute_colon_word (VM *vm)

  Inner interpreter for threaded colon definitions.
- void vm_compile_call (VM *vm, word_func_t func)

  Compile a call to a word identified by its C function pointer.
- void vm_compile_literal (VM *vm, cell_t value)

  Compile a literal value into the current threaded body (or push it).
- void vm_compile_exit (VM *vm)

  Compile an `EXIT` reference into the current threaded body.
- void vm_interpret_word (VM *vm, const char *word_str, size_t len)

  Interpret or compile a single word token.
- void * vm_get_block_addr (VM *vm, int block_num)

  Gets the memory address for a specified block number.
- int vm_addr_to_block (VM *vm, void *addr)

  Converts a memory address to its corresponding block number.
- void vm_run_smoke_tests (VM *vm)

- void vm_cleanup (VM *vm)

  Clean up and free VM resources.
- void vm_tick (VM *vm)

  Central heartbeat dispatcher for time-driven tuning operations.
- void vm_tick_inference_engine (VM *vm)

  Phase 2: Unified Inference Engine - Adaptive Window & Decay Slope Tuning.
- void vm_tick_window_tuner (VM *vm)

  Loop #5: Context-aware window tuning via binary chop search.
- void vm_tick_slope_validator (VM *vm)

  Loop #3: Heat decay slope validation via periodic measurement.
- void vm_snapshot_read (const VM *vm, HeartbeatSnapshot *out_snapshot)
- void heartbeat_capture_tick_snapshot (VM *vm, HeartbeatTickSnapshot *snapshot)

  Capture current VM metrics into a heartbeat tick snapshot.
- void heartbeat_emit_tick_row (VM *vm, HeartbeatTickSnapshot *snapshot)

  Emit a heartbeat tick snapshot as CSV row to stderr.
- void vm_heartbeat_start (VM *vm)
- static void vm_push_fast (VM *vm, cell_t value)
- static cell_t vm_pop_fast (VM *vm)
- static void vm_rpush_fast (VM *vm, cell_t value)
- static cell_t vm_rpop_fast (VM *vm)

Core VM Functions

- void vm_init (VM *vm)

  Initialize a new virtual machine instance.
- void vm_init_with_host (VM *vm, const VMHostServices *host)

  Initialize VM with explicit host services.
- void vm_interpret (VM *vm, const char *input)
- void vm_enable_interpreter (VM *vm)
- void vm_repl (VM *vm, int script_mode)

  Starts the Forth REPL (Read-Eval-Print Loop)

Variables

- typedef __attribute__
- struct DictEntry * link
- word_func_t func
- cell_t flags
- uint8_t name_len
- cell_t execution_heat
- uint8_t acl_default
- uint32_t acl_ttl
- uint8_t acl_allow
- uint8_t acl_mode
- uint8_t acl_pinned
- uint32_t word_id
- DictPhysics physics
- WordTransitionMetrics * transition_metrics
- char name [ ]

7.263.1 Macro Definition Documentation

7.263.1.1 ACL_MODE_STRICT

```
#define ACL_MODE_STRICT 1 /* Re-check every execution (STRICT mode) */
```

7.263.1.2 ACL_MODE_TTL

```
#define ACL_MODE_TTL 0 /* Adaptive TTL countdown; 0 → calls ACL-RECHECK */
```

7.263.1.3 ACL_TTL_OPEN

```
#define ACL_TTL_OPEN UINT32_MAX /* Permissive TTL before ACL.4th loads */
```

7.263.1.4 ACL_USER_DEFAULT

```
#define ACL_USER_DEFAULT 0x01 /* Default access:  users can execute and compile */
```

7.263.1.5 BLK_VM_SLOTS

```
#define BLK_VM_SLOTS 4
```

7.263.1.6 BLK_VM_WINDOW_BASE

```
#define BLK_VM_WINDOW_BASE (VM_MEMORY_SIZE - BLK_VM_WINDOW_SIZE)
```

7.263.1.7 BLK_VM_WINDOW_SIZE

```
#define BLK_VM_WINDOW_SIZE (BLK_VM_SLOTS * BLOCK_SIZE)
```

7.263.1.8 BLOCK_SIZE

```
#define BLOCK_SIZE 1024 /* 1KB per block */
```

7.263.1.9 CF_STACK_MAX

```
#define CF_STACK_MAX 64
```

7.263.1.10 CHARGE_NEUTRAL

```
#define CHARGE_NEUTRAL 0 /* Particle charge state:  neutral (execution heat acts as charge) */
```

7.263.1.11 COMPILE_BUFFER_SIZE

```
#define COMPILE_BUFFER_SIZE 1024
```

7.263.1.12 DICTIONARY_BLOCKS

```
#define DICTIONARY_BLOCKS 2048 /* First 2048 blocks (2MB) for dictionary */
```

7.263.1.13 DICTIONARY_MEMORY_SIZE

```
#define DICTIONARY_MEMORY_SIZE (DICTIONARY_BLOCKS * BLOCK_SIZE)
```

7.263.1.14 DICTIONARY_SIZE

```
#define DICTIONARY_SIZE 1024
```

7.263.1.15 HEARTBEAT_TICK_BUFFER_SIZE

```
#define HEARTBEAT_TICK_BUFFER_SIZE 100000 /* ~100K ticks = ~100 seconds at 1kHz */
```

7.263.1.16 HEAT_CACHE_DEMOTION_THRESHOLD

```
#define HEAT_CACHE_DEMOTION_THRESHOLD 10 /* Demotion from hotwords cache threshold */
```

7.263.1.17 INPUT_BUFFER_SIZE

```
#define INPUT_BUFFER_SIZE 1025
```

7.263.1.18 LIKELY

```
#define LIKELY(
             x ) __builtin_expect(!!(x), 1)
```

7.263.1.19 LOG_BLOCKS

```
#define LOG_BLOCKS (LOG_BLOCKS_END - LOG_BLOCKS_START) /* 2048 blocks (2MB) for logs */
```

7.263.1.20 LOG_BLOCKS_END

```
#define LOG_BLOCKS_END 5120 /* Log ends at block 5120 (exclusive) */
```

7.263.1.21 LOG_BLOCKS_START

```
#define LOG_BLOCKS_START 3072 /* Log starts at block 3072 */
```

7.263.1.22 LOG_LAYER1_MAX_LINES

```
#define LOG_LAYER1_MAX_LINES (LOG_BLOCKS * LOG_LINES_PER_BLOCK) /* 32768 max lines */
```

7.263.1.23 LOG_LINE_MAX

```
#define LOG_LINE_MAX 64 /* Each log line is 64 bytes */
```

7.263.1.24 LOG_LINES_PER_BLOCK

```
#define LOG_LINES_PER_BLOCK (BLOCK_SIZE / LOG_LINE_MAX) /* 16 lines per 1KB block */
```

7.263.1.25 MAX_BLOCKS

```
#define MAX_BLOCKS (VM_MEMORY_SIZE / BLOCK_SIZE) /* 5120 blocks from 5MB */
```

7.263.1.26 PHYSICS_STATE_COMPILED

```
#define PHYSICS_STATE_COMPILED 0x08
```

7.263.1.27 PHYSICS_STATE_HIDDEN

```
#define PHYSICS_STATE_HIDDEN 0x04
```

7.263.1.28 PHYSICS_STATE_IMMEDIATE

```
#define PHYSICS_STATE_IMMEDIATE 0x01
```

7.263.1.29 PHYSICS_STATE_PINNED

```
#define PHYSICS_STATE_PINNED 0x02
```

7.263.1.30 SPIN_IDLE

```
#define SPIN_IDLE 0 /* Particle spin state:  idle */
```

7.263.1.31 STACK_SIZE

```
#define STACK_SIZE 1024
```

7.263.1.32 UNLIKELY

```
#define UNLIKELY(
              x ) __builtin_expect(!!(x), 0)
```

7.263.1.33 USER_BLOCKS_START

```
#define USER_BLOCKS_START DICTIONARY_BLOCKS /* User blocks start at block 2048 */
```

7.263.1.34 VM_MEMORY_SIZE

```
#define VM_MEMORY_SIZE (5 * 1024 * 1024) /* 5 MB total VM memory */
```

7.263.1.35 VM_POP

```
#define VM_POP(
              vm ) vm_pop_fast(vm)
```

7.263.1.36 VM_PUSH

```
#define VM_PUSH(
              vm,
              val ) vm_push_fast(vm, val)
```

7.263.1.37 VM_RPOP

```
#define VM_RPOP(
              vm ) vm_rpop_fast(vm)
```

7.263.1.38 VM_RPUSH

```
#define VM_RPUSH(
              vm,
              val ) vm_rpush_fast(vm, val)
```

7.263.1.39 VOCAB_FC_BUCKETS

```
#define VOCAB_FC_BUCKETS 256
```

7.263.1.40 WORD_COMPILED

```
#define WORD_COMPILED 0x10 /* Word is user-compiled (not built-in) */
```

7.263.1.41 WORD_FROZEN

```
#define WORD_FROZEN 0x04 /* Word is frozen:  execution heat does not decay (Phase 2) */
```

7.263.1.42 WORD_HIDDEN

```
#define WORD_HIDDEN 0x40 /* Word is hidden from dictionary searches */
```

7.263.1.43 WORD_ID_INVALID

```
#define WORD_ID_INVALID UINT32_MAX
```

7.263.1.44 WORD_IMMEDIATE

```
#define WORD_IMMEDIATE 0x80 /* Word executes immediately even in compile mode */
```

7.263.1.45 WORD_NAME_MAX

```
#define WORD_NAME_MAX 31
```

7.263.1.46 WORD_PINNED

```
#define WORD_PINNED 0x08 /* Word's execution heat is pinned (cannot decay to zero) */
```

7.263.1.47 WORD_SMUDGED

```
#define WORD_SMUDGED 0x20 /* Word is smudged (being defined) - FORTH-79 */
```

7.263.2 Typedef Documentation

7.263.2.1 cell_t

```
typedef signed long cell_t
```

7.263.2.2 HeartbeatWorker

```
typedef struct HeartbeatWorker HeartbeatWorker
```

7.263.2.3 HotwordsCache

```
typedef struct HotwordsCache_s HotwordsCache
```

7.263.2.4 InferenceOutputs

```
typedef struct InferenceOutputs InferenceOutputs
```

7.263.2.5 ucell_t

```
typedef unsigned long ucell_t
```

7.263.2.6 vaddr_t

```
typedef uint64_t vaddr_t
```

7.263.2.7 VM

```
typedef struct VM VM
```

7.263.2.8 VMHostServices

```
typedef struct VMHostServices VMHostServices
```

7.263.2.9 word_func_t

```
typedef void(* word_func_t) (struct VM *vm)
```

7.263.2.10 WordTransitionMetrics

```
typedef struct WordTransitionMetrics WordTransitionMetrics
```

7.263.3 Enumeration Type Documentation

7.263.3.1 cf_tag_t

```
enum cf_tag_t
```

Enumerator

CF_BEGIN	
CF_IF	
CF_ELSE	
CF_WHILE	
CF_DO	
CF_CASE	
CF_OF	

7.263.3.2 vm_mode_t

```
enum vm_mode_t
```

Enumerator

MODE_INTERPRET	
MODE_COMPILE	

7.263.4 Function Documentation

7.263.4.1 __attribute__()

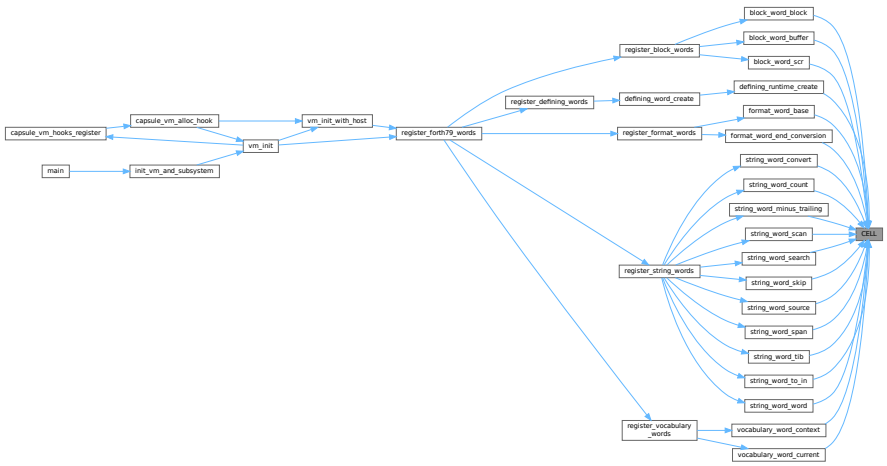
```
struct DictEntry __attribute__ (
            (aligned(8))  )
```

7.263.4.2 CELL()

```
static cell_t CELL (
            vaddr_t a )  [inline], [static]
```

Referenced by block_word_block(), block_word_buffer(), block_word_scr(), defining_runtime_create(), format_word_base(), format_word_end_conversion(), string_word_convert(), string_word_count(), string_word_minus_trailing(), string_word_scan(), string_word_search(), string_word_skip(), string_word_source(), string_word_span(), string_word_tib(), string_word_to_in(), string_word_word(), vocabulary_word_context(), and vocabulary_word_current().

Here is the caller graph for this function:



7.263.4.3 execute_colon_word()

```
void execute_colon_word (
            VM * vm )
```

Inner interpreter for threaded colon definitions.

Runs the body of a colon-defined word whose DictEntry points to a sequence of DictEntry* cells in VM memory (indirect threaded code). The interpreter pointer (IP) is maintained on the return stack so that runtime control-flow words ((BRANCH), (0BRANCH), (DO), etc.) can redirect execution by modifying the top-of-return-stack IP.

Execution loop per iteration:

1. Read DictEntry* w from *ip.
2. Run physics pre-execute hooks (heat, decay, pipelining).
3. ACL hot-path check (decrement TTL; call acl_recheck() on zero).
4. Push ip+1 onto the return stack as the resume point.
5. Call w->func(vm).
6. Run physics post-execute hooks.
7. Check vm->exit_colon (set by EXIT): pop and discard the saved IP, then return to caller.
8. Check vm->abort_requested: unwind and return immediately.
9. Pop the (possibly modified) IP from the return stack and repeat.

vm->ecw_nesting is incremented on entry and decremented on every return path to enable recursive colon calls.

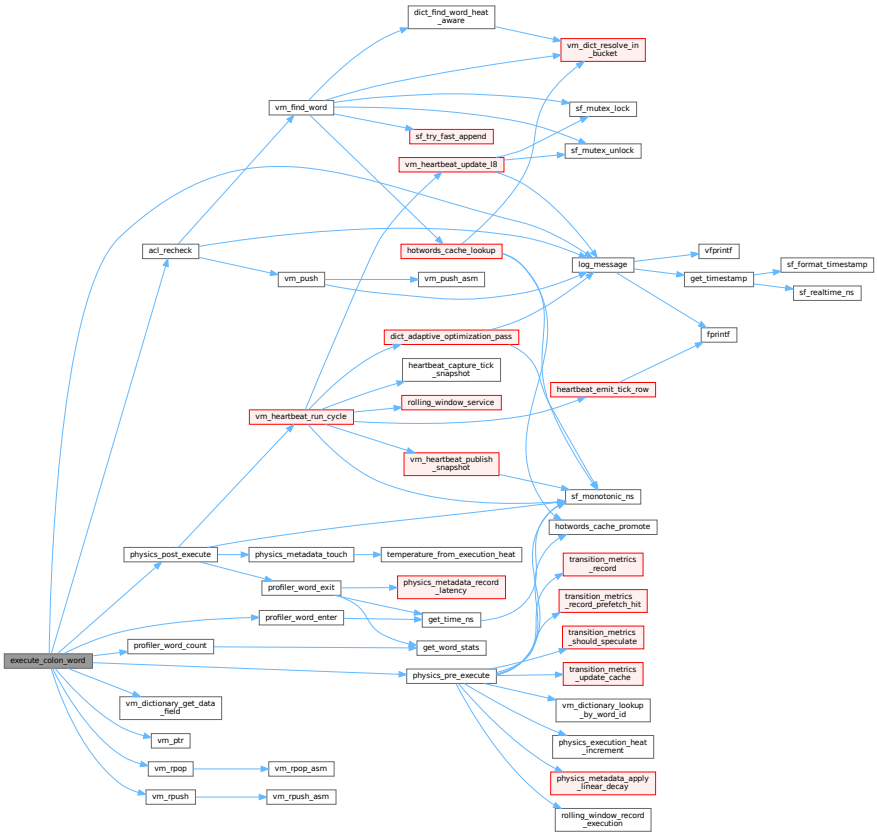
Parameters

vm	VM instance; vm->current_executing_entry must point to the DictEntry being executed.

References VM::abort_requested, DictEntry::acl_allow, acl_recheck(), DictEntry::acl_ttl, VM::current_executing_entry, VM::dsp, VM::ecw_nesting, VM::emergency_console, entry, VM::error, VM::exit_colon, DictEntry::func, LOG_ERROR, log_message(), LOG_WARN, DictEntry::name, DictEntry::name_len, physics_post_execute(), physics_pre_execute(), profiler_word_count(), profiler_word_enter(), VM::rsp, vm_dictionary_get_data_field(), vm_ptr(), vm_rpop(), and vm_rpush().

Referenced by system_word_see().

Here is the call graph for this function:



Here is the caller graph for this function:

7.263.4.4 heartbeat_capture_tick_snapshot()

```
void heartbeat_capture_tick_snapshot (
            VM * vm,
            HeartbeatTickSnapshot * snapshot )
```

Capture current VM metrics into a heartbeat tick snapshot.

Collects 11 metrics per tick for multivariate dynamics analysis. Called every heartbeat tick to record VM state.

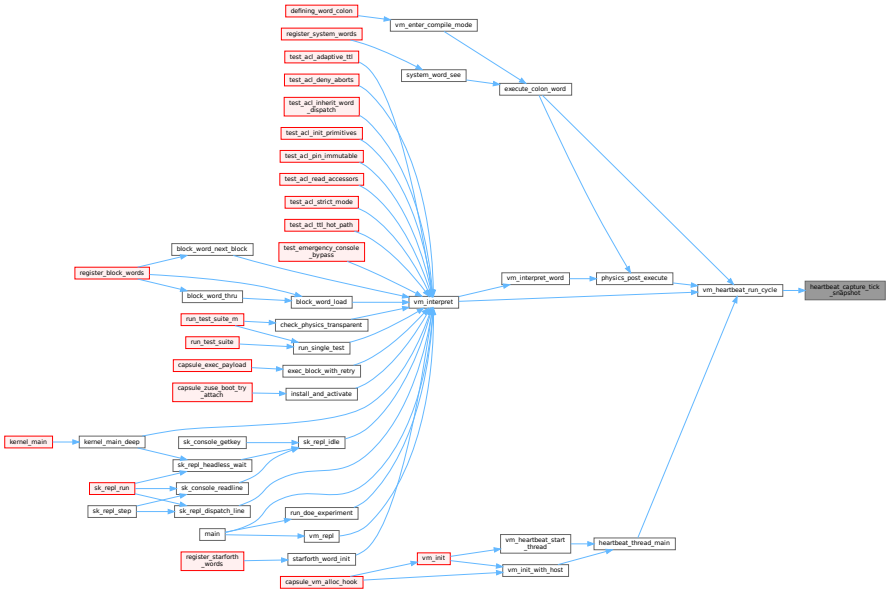
Parameters

vm	Pointer to the VM instance
snapshot	Pointer to snapshot structure to populate

References HeartbeatTickSnapshot::actual_window_size, HeartbeatTickSnapshot::avg_word_heat, HeartbeatTickSnapshot::bucket_
HeartbeatTickSnapshot::cache_hits_delta, HeartbeatState::early_exit_count, RollingWindowOfTruth::effective_window_size,
HeartbeatTickSnapshot::elapsed_ns, entry, HeartbeatTickSnapshot::estimated_jitter_ns, VM::heartbeat, HEARTBEAT_TICK_NS,
HeartbeatTickSnapshot::hot_word_count, VM::latest, HeartbeatTickSnapshot::predicted_label_hits, VM::rolling_window,
HeartbeatState::tick_count, HeartbeatTickSnapshot::tick_interval_ns, HeartbeatTickSnapshot::tick_number,
RollingWindowOfTruth::total_executions, HeartbeatTickSnapshot::window_width, word_count, HeartbeatTickSnapshot::word_executi
and HeartbeatState::words_executed.

Referenced by vm_heartbeat_run_cycle().

Here is the caller graph for this function:



7.263.4.5 heartbeat_emit_tick_row()

```
void heartbeat_emit_tick_row (
            VM * vm,
            HeartbeatTickSnapshot * snapshot )
```

Emit a heartbeat tick snapshot as CSV row to stderr.

Outputs 11 comma-separated values (no header) for real-time streaming. Called every heartbeat tick to emit metrics immediately.

Format: tick_number,elapsed_ns,tick_interval_ns,cache_hits_delta, bucket_hits_delta,word_executions_↩
delta,hot_word_count, avg_word_heat,window_width,actual_window_size,predicted_label_hits,estimated_jitter_ns

Parameters

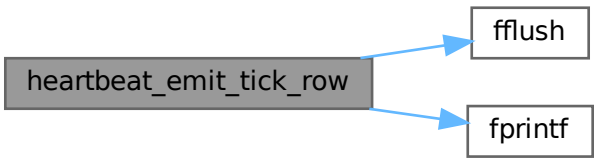
vm	Pointer to the VM instance (unused, for API consistency)
snapshot	Pointer to the snapshot to emit

References HeartbeatTickSnapshot::actual_window_size, HeartbeatTickSnapshot::avg_word_heat, HeartbeatTickSnapshot::bucket_

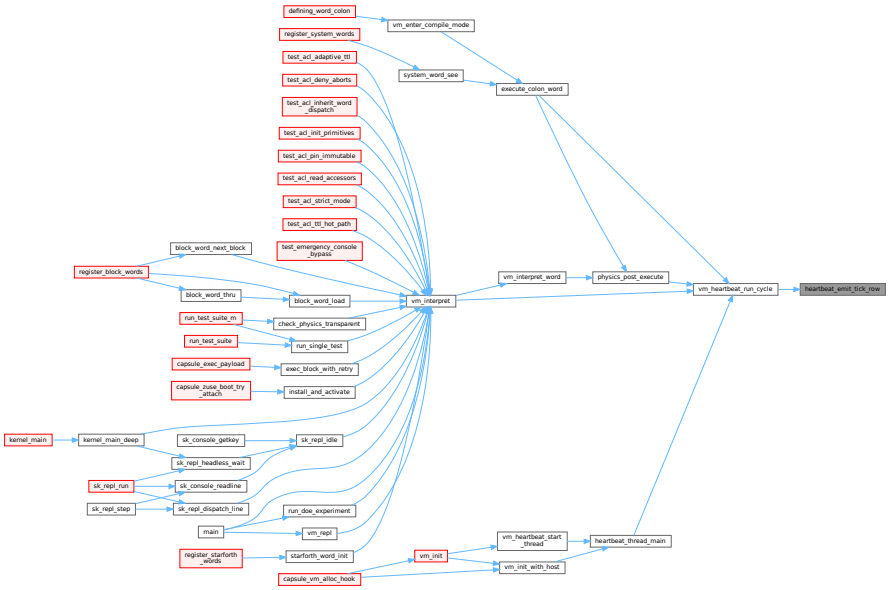
HeartbeatTickSnapshot::cache_hits_delta, HeartbeatTickSnapshot::elapsed_ns, HeartbeatTickSnapshot::estimated_jitter_ns, fflush(), fprintf(), HeartbeatTickSnapshot::hot_word_count, HeartbeatTickSnapshot::predicted_label_hits, stderr, HeartbeatTickSnapshot::tick_interval_ns, HeartbeatTickSnapshot::tick_number, HeartbeatTickSnapshot::window_width, and HeartbeatTickSnapshot::word_executions_delta.

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



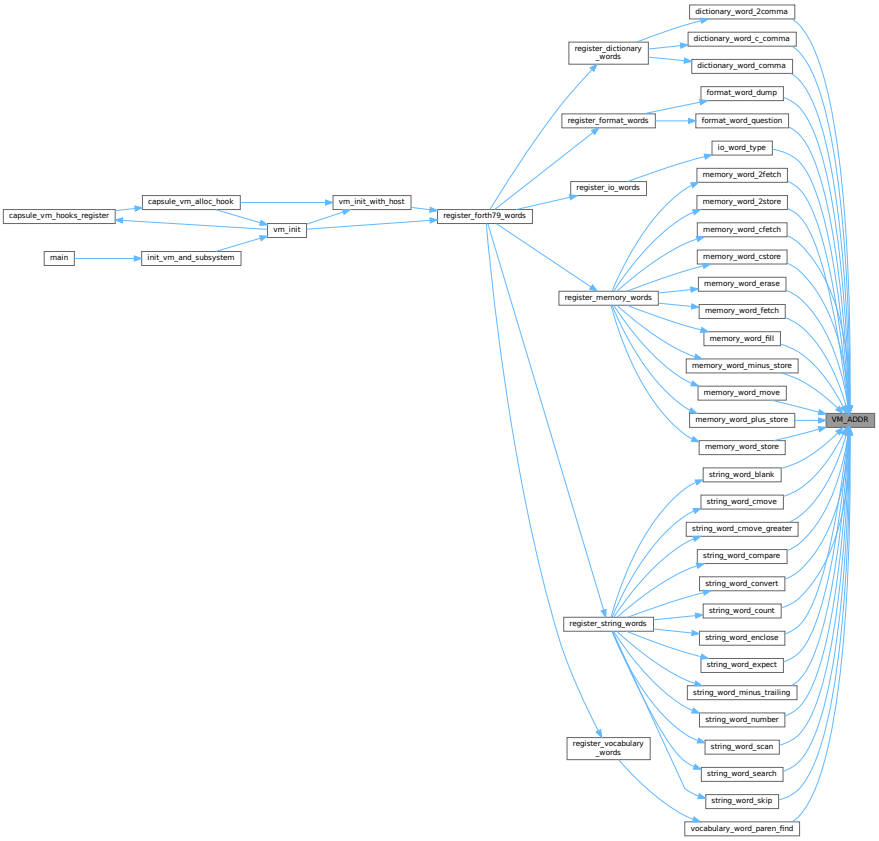
7.263.4.6 VM_ADDR()

```
static vaddr_t VM_ADDR (
            cell_t c )  [inline], [static]
```

Referenced by dictionary_word_2comma(), dictionary_word_c_comma(), dictionary_word_comma(), format_word_dump(), format_word_question(), io_word_type(), memory_word_2fetch(), memory_word_2store(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), string_word_blank(), string_word_cmove(),

string_word_cmove_greater(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_minus_trailing(), string_word_number(), string_word_scan(), string_word_search(), string_word_skip(), and vocabulary_word_paren_find().

Here is the caller graph for this function:



7.263.4.7 vm_addr_ok()

```
int vm_addr_ok (
            struct VM * vm,
            vaddr_t addr,
            size_t len )
```

Validate that a VM address range lies within `vm->memory`.

Checks that [addr, addr+len) is fully contained within the `VM_MEMORY_SIZE`-byte flat address space. The check is done with unsigned arithmetic to avoid UB on overflow: if `len` alone exceeds `VM_MEMORY_SIZE` the range is immediately rejected.

Parameters

vm	VM instance; returns 0 if NULL or memory not allocated.
addr	Start of the range (inclusive).
len	Number of bytes in the range.

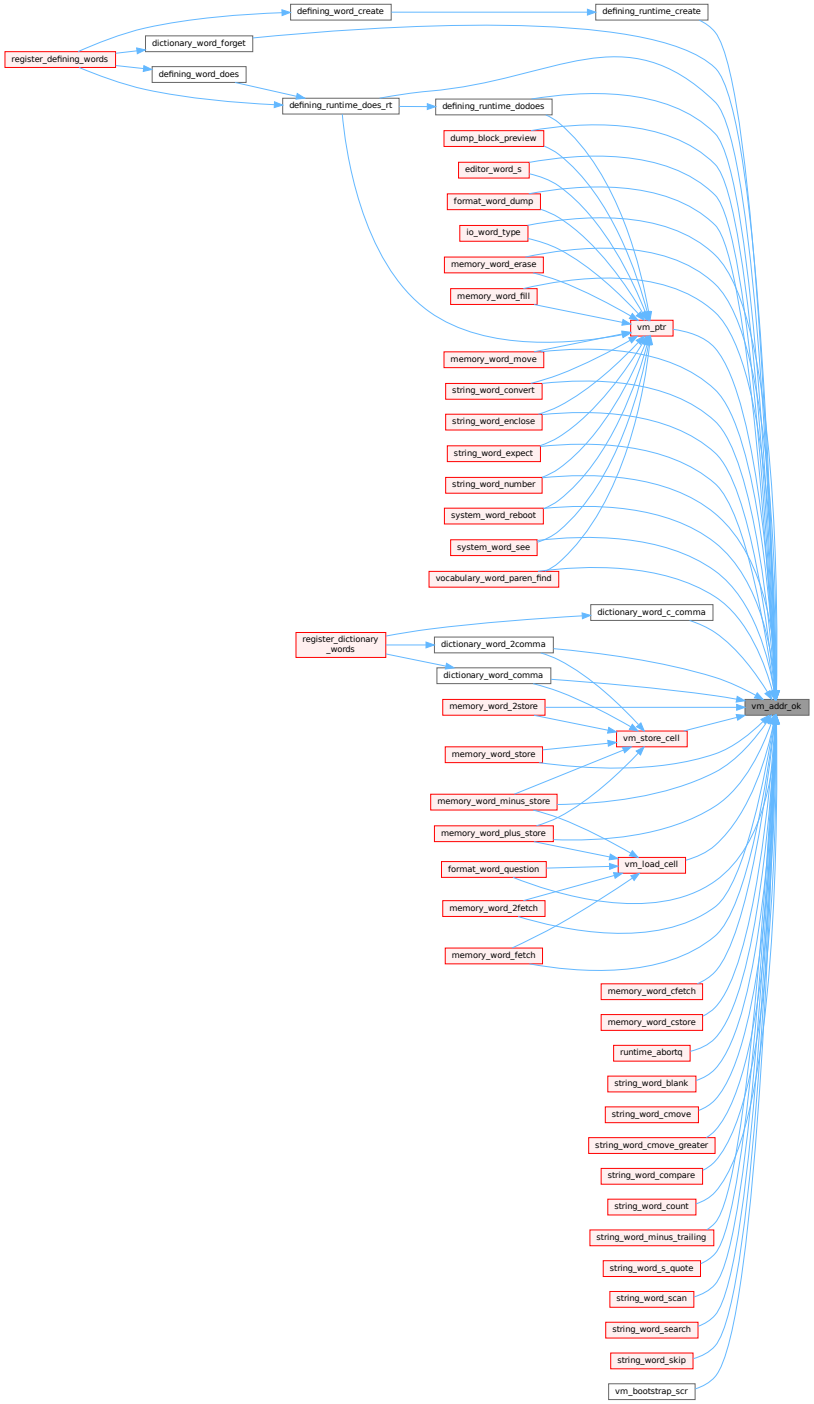
Returns

1 if the full range [addr, addr+len) is in bounds; 0 otherwise.

References VM::memory, and VM_MEMORY_SIZE.

Referenced by defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_does_rt(), dictionary_word_2comma(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_forget(), dump_block_preview(), editor_word_s(), format_word_dump(), format_word_question(), io_word_type(), memory_word_2fetch(), memory_word_2store(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), runtime_abortq(), string_word_blank(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_minus_trailing(), string_word_number(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), system_word_reboot(), system_word_see(), vm_bootstrap_scr(), vm_load_cell(), vm_ptr(), vm_store_cell(), and vocabulary_word_paren_find().

Here is the caller graph for this function:



7.263.4.8 vm_addr_to_block()

```
int vm_addr_to_block (
            VM * vm,
            void * addr )
```

Converts a memory address to its corresponding block number.

Parameters

vm	Pointer to the VM instance
addr	Memory address to convert

Returns

Block number (0 to MAX_BLOCKS-1) or -1 if address is outside VM memory

Note

Used for determining which block a memory address belongs to

References BLOCK_SIZE, VM::memory, and VM_MEMORY_SIZE.

7.263.4.9 vm_align()

```
void vm_align (
            VM * vm )
```

Aligns the VM's dictionary pointer (here) to the cell boundary.

Parameters

vm	Pointer to the VM instance

Note

Adds padding bytes if necessary to ensure proper alignment for cell_t type

7.263.4.10 vm_allot()

```
void * vm_allot (
            VM * vm,
            size_t bytes )
```

Allocates memory from the VM's dictionary space.

Parameters

vm	Pointer to the VM instance
bytes	Number of bytes to allocate

Returns

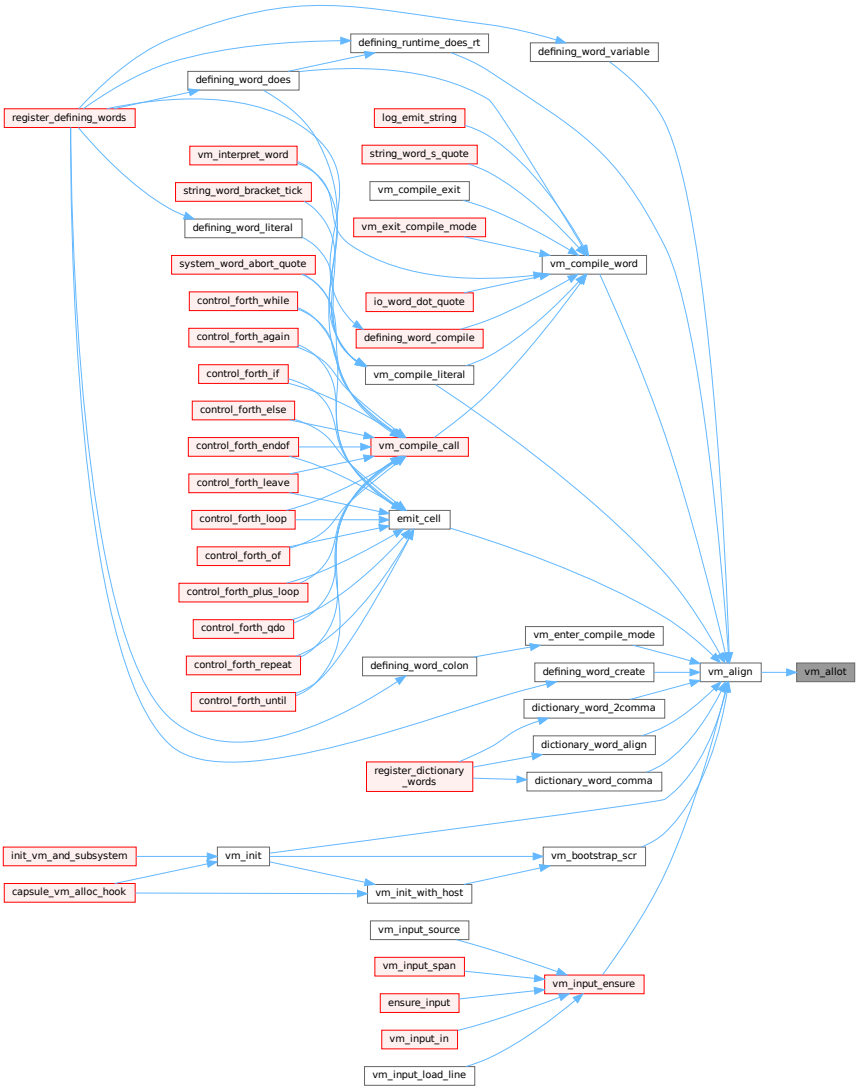
Pointer to the allocated memory or NULL if allocation fails

Note

The allocation is done from the VM's dictionary space which is limited to DICTIONARY_MEMORY_SIZE

Referenced by vm_align().

Here is the caller graph for this function:



7.263.4.11 vm_bootstrap_root_vocabulary()

```
void vm_bootstrap_root_vocabulary (
            VM * vm,
            const char * name )
```

Bootstrap a root vocabulary for the VM.

Creates a vocabulary by name, finds its dictionary entry, executes it to make it current, then calls DEFINITIONS.

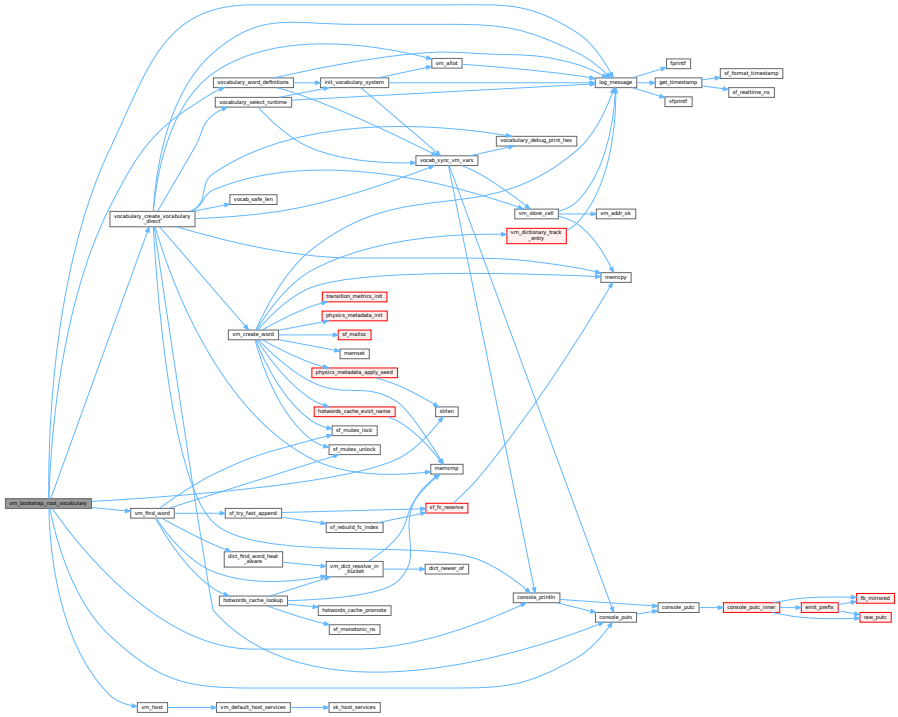
Parameters

vm	Pointer to VM structure
name	Vocabulary name to bootstrap

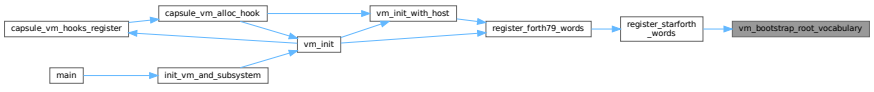
References console_println(), console_puts(), entry, VM::error, VMHostServices::is_executable_ptr, LOG_ERROR, log_message(), name, VMHostServices::panic, strlen(), vm_find_word(), vm_host(), vocabulary_create_vocabulary_direct(), and vocabulary_word_definitions().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.12 vm_cleanup()

```
void vm_cleanup (
            VM * vm )
```

Clean up and free VM resources.

Frees allocated memory and resets VM state.

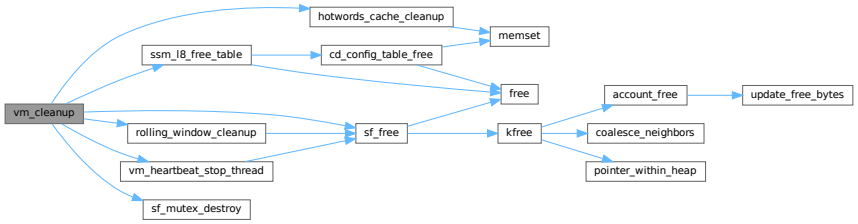
Parameters

vm	Pointer to VM structure to clean up

References VM::dict_lock, VM::here, VM::hotwords_cache, hotwords_cache_cleanup(), VM::latest, DictEntry::link, VM::memory, VM::rolling_window, rolling_window_cleanup(), sf_free(), sf_mutex_destroy(), VM::ssm_config, ssm_l8_free_table(), VM::ssm_l8_state, DictEntry::transition_metrics, VM::tuning_lock, and vm_heartbeat_stop_thread().

Referenced by capsule_vm_alloc_hook(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_vm_kill_all_nonmama(), and cleanup_and_exit().

Here is the call graph for this function:



Here is the caller graph for this function:

7.263.4.13 vm_compile_call()

```
void vm_compile_call (
            VM * vm,
            word_func_t func )
```

Compile a call to a word identified by its C function pointer.

Looks up the `DictEntry` whose `func` field matches `func` via `vm_dictionary_find_by_func()`, then delegates to `vm_compile_word()`. Used by generating words (e.g., `DOES>`) that need to embed a specific runtime routine into a compiled definition without knowing its name.

Sets `vm->error` if not in compile mode or if no matching entry is found.

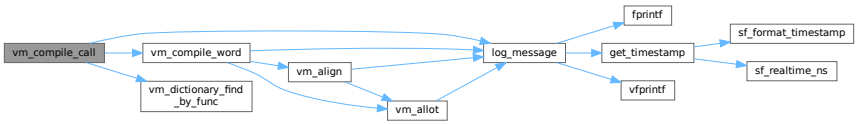
Parameters

vm	VM instance in compile mode.
func	C function pointer identifying the word to compile.

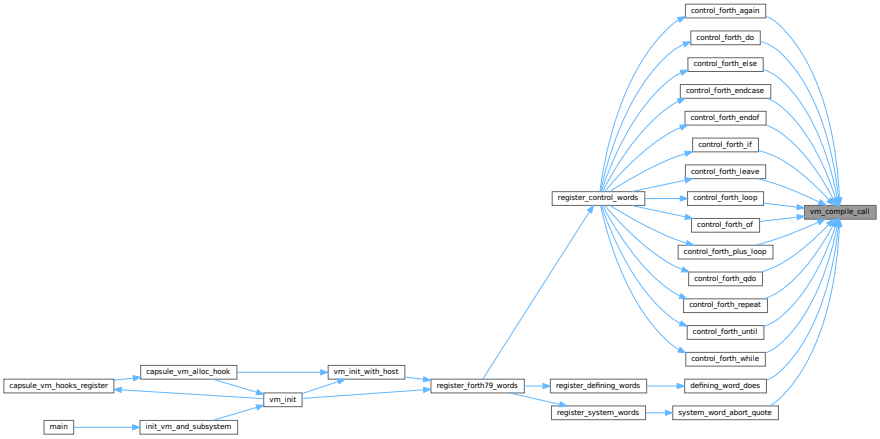
References entry, VM::error, func, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_compile_word(), and vm_dictionary_find_by_func().

Referenced by control_forth_again(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_if(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_until(), control_forth_while(), defining_word_does(), and system_word_abort_quote().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.14 vm_compile_exit()

```
void vm_compile_exit (
            VM * vm )
```

Compile an `EXIT` reference into the current threaded body.

Appends the `EXIT` word entry so that when the threaded body is executed, `execute_colon_word()` sets `vm->exit_colon` and unwinds the inner loop. Used by control-flow generating words (e.g., `IF` optimisation paths) that need an explicit early-return.

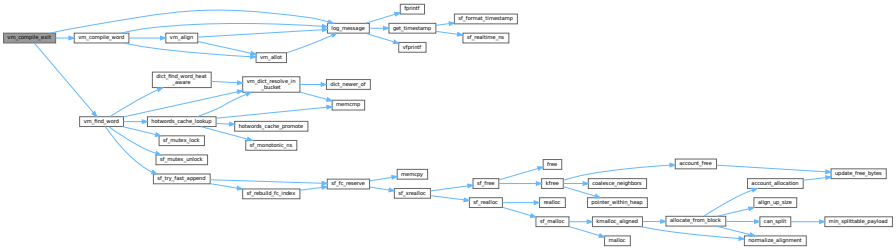
Sets `vm->error` if not in compile mode or if `EXIT` is not found.

Parameters

vm	VM instance in compile mode.

References VM::error, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_compile_word(), and vm_find_word().

Here is the call graph for this function:



7.263.4.15 vm_compile_literal()

```
void vm_compile_literal (
            VM * vm,
            cell_t value )
```

Compile a literal value into the current threaded body (or push it).

In compile mode, appends the `LIT` word reference followed by the literal value so the inner interpreter will push `value` at runtime. In interpret mode, pushes `value` directly onto the data stack.

Sets `vm->error` if `LIT` cannot be found in the dictionary.

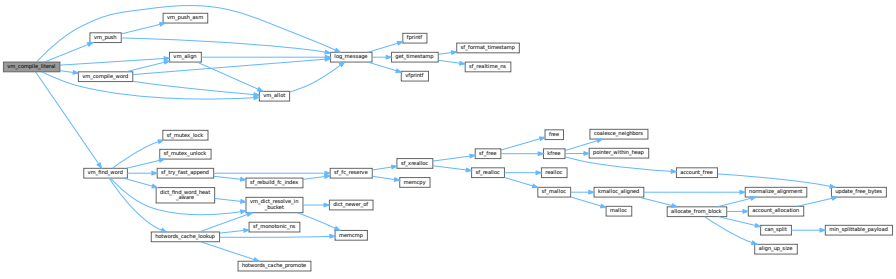
Parameters

vm	VM instance.
value	Cell value to compile or push.

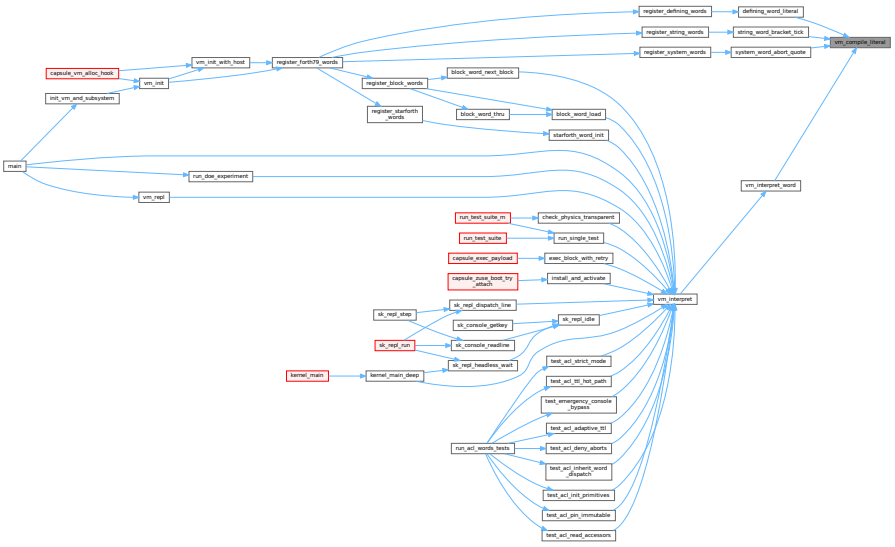
References VM::error, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_align(), vm_allot(), vm_compile_word(), vm_find_word(), and vm_push().

Referenced by defining_word_literal(), string_word_bracket_tick(), system_word_abort_quote(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.16 vm_compile_word()

```
void vm_compile_word (
            VM * vm,
            DictEntry * entry )
```

Append a word reference (DictEntry∗) to the current threaded body.

Aligns `HERE`, then allots one `cell_t` slot and stores the `DictEntry∗` as a cell. This is the fundamental compilation step used by the inner interpreter: each slot in the threaded body is a pointer to the `DictEntry` of the word to call.

Sets `vm->error` if the VM is not in compile mode, `entry` is NULL, or allotment fails.

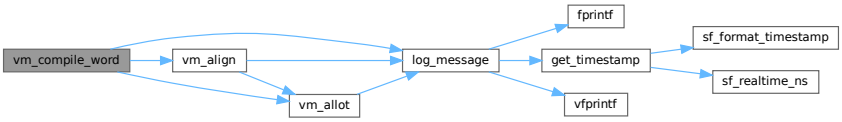
Parameters

vm	VM instance in compile mode.
entry	Dictionary entry of the word to compile; must not be NULL.

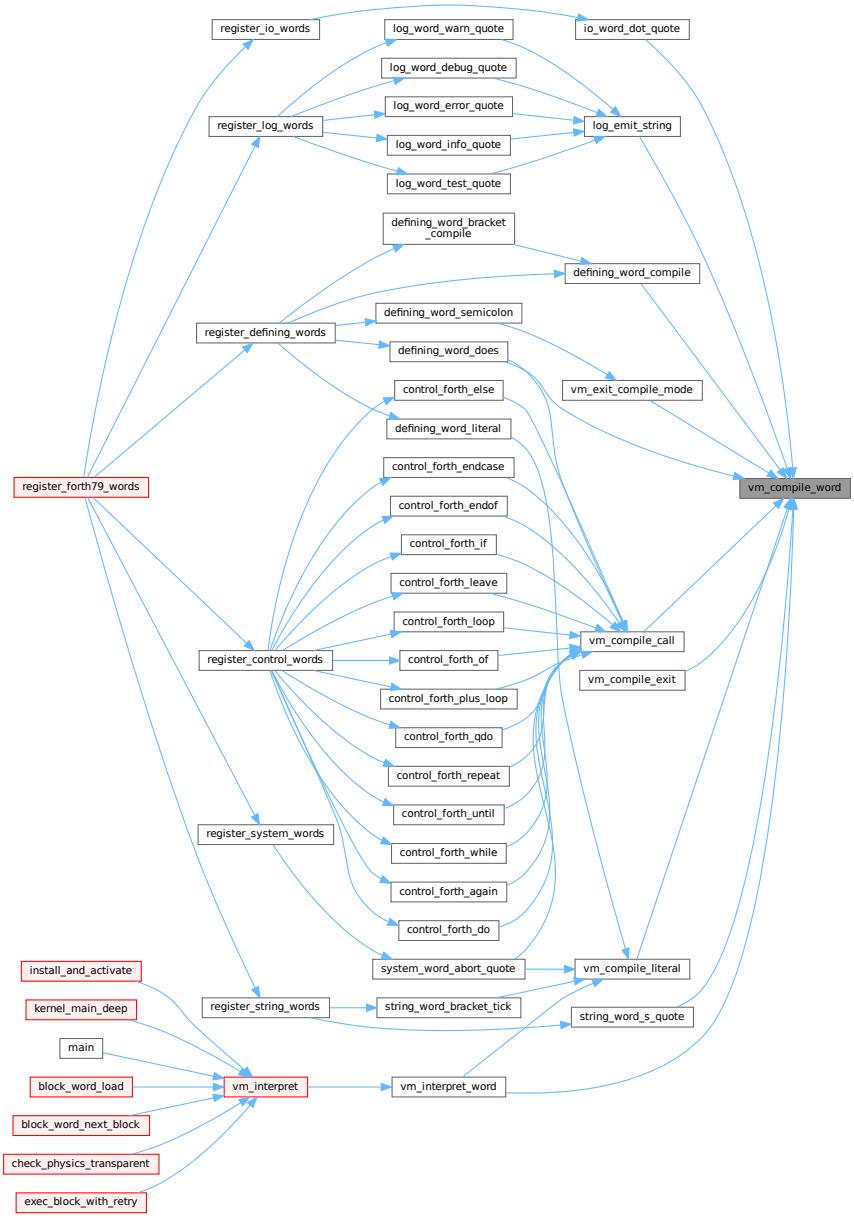
References entry, VM::error, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_align(), and vm_allot().

Referenced by defining_word_compile(), defining_word_does(), io_word_dot_quote(), log_emit_string(), string_word_s_quote(), vm_compile_call(), vm_compile_exit(), vm_compile_literal(), vm_exit_compile_mode(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.17 vm_create_word()

```
DictEntry * vm_create_word (
            VM * vm,
            const char * name,
            size_t len,
            word_func_t func )
```

Creates a new word in the dictionary.

Allocates and initializes a new dictionary entry with the given name and function. The word is added to the dictionary but index update is deferred until next lookup.

Parameters

vm	The virtual machine context
name	Name for the new word
len	Length of the name string
func	Function pointer for the word's behavior

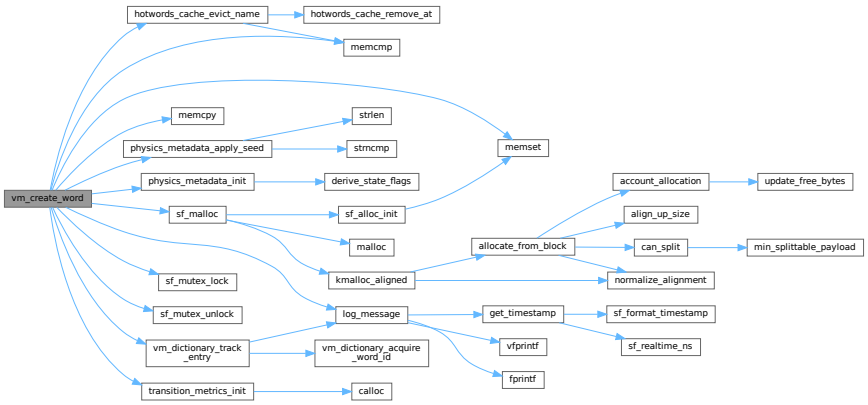
Returns

DictEntry* Pointer to new dictionary entry or NULL on failure

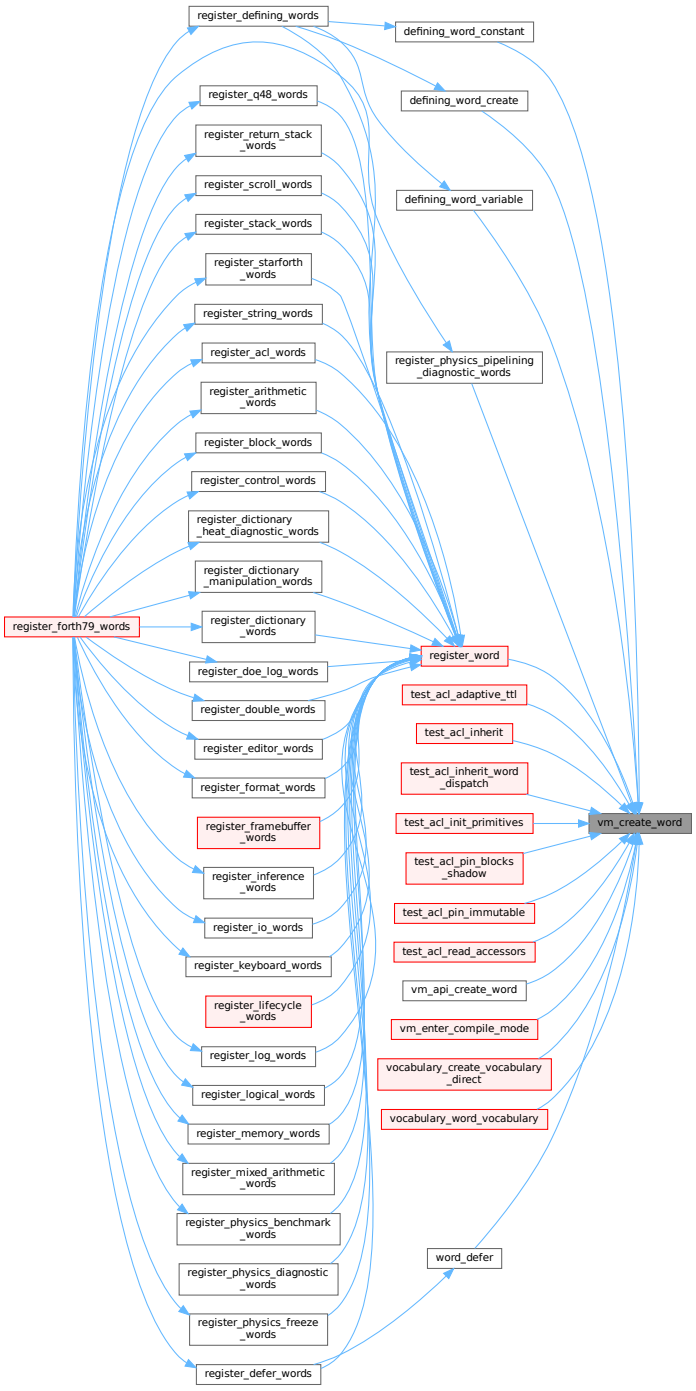
References ACL_MODE_TTL, ACL_USER_DEFAULT, base, VM::dict_lock, entry, VM::error, func, DictEntry::func, VM::hotwords_cache, hotwords_cache_evict_name(), VM::latest, LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, memcmp(), memcpy(), memset(), name, physics_metadata_apply_seed(), physics_metadata_init(), sf_malloc(), sf_mutex_lock(), sf_mutex_unlock(), transition_metrics_init(), vm_dictionary_track_entry(), WORD_ID_INVALID, and WORD_NAME_MAX.

Referenced by defining_word_constant(), defining_word_create(), defining_word_variable(), register_physics_pipelining_diagnostic_w register_word(), test_acl_adaptive_ttl(), test_acl_inherit(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_blocks_shadow(), test_acl_pin_immutable(), test_acl_read_accessors(), vm_api_create_word(), vm_enter_compile_mode(), vocabulary_create_vocabulary_direct(), vocabulary_word_vocabulary(), and word_defer().

Here is the call graph for this function:



Here is the caller graph for this function:



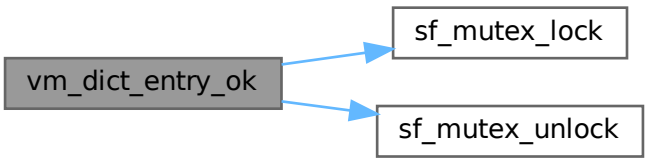
7.263.4.18 vm_dict_entry_ok()

```
int vm_dict_entry_ok (
            VM * vm,
            DictEntry * candidate )
```

References VM::dict_lock, VM::latest, sf_mutex_lock(), and sf_mutex_unlock().

Referenced by starforth_word_execution_heat_fetch(), starforth_word_execution_heat_store(), and system_word_execute().

Here is the call graph for this function:



Here is the caller graph for this function:

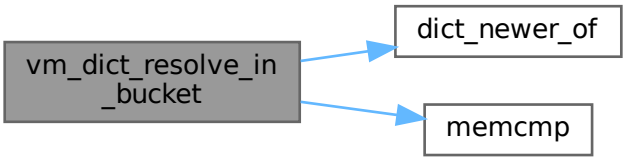
7.263.4.19 vm_dict_resolve_in_bucket()

```
DictEntry * vm_dict_resolve_in_bucket (
            VM * vm,
            DictEntry ** bucket,
            size_t n,
            const char * name,
            size_t len )
```

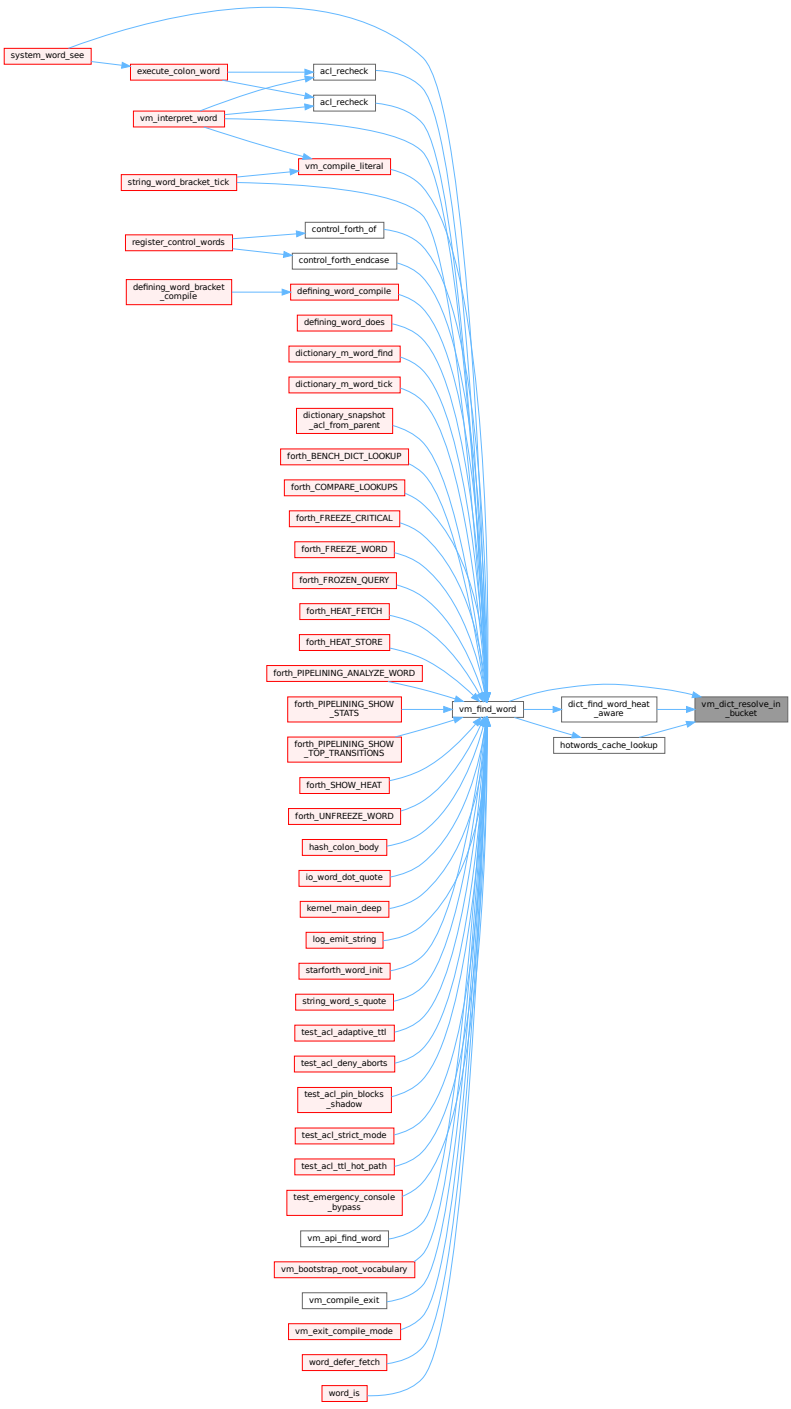
References dict_newer_of(), DictEntry::flags, memcmp(), name, DictEntry::name, DictEntry::name_len, UNLIKELY, WORD_HIDDEN, and WORD_SMUDGED.

Referenced by dict_find_word_heat_aware(), hotwords_cache_lookup(), and vm_find_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.20 vm_dictionary_find_by_func()

```
DictEntry * vm_dictionary_find_by_func (
            VM * vm,
            word_func_t func )
```

Finds a word by its function pointer.

Performs a linear scan of the dictionary to find an entry with matching function.

Parameters

vm	The virtual machine context
func	Function pointer to search for

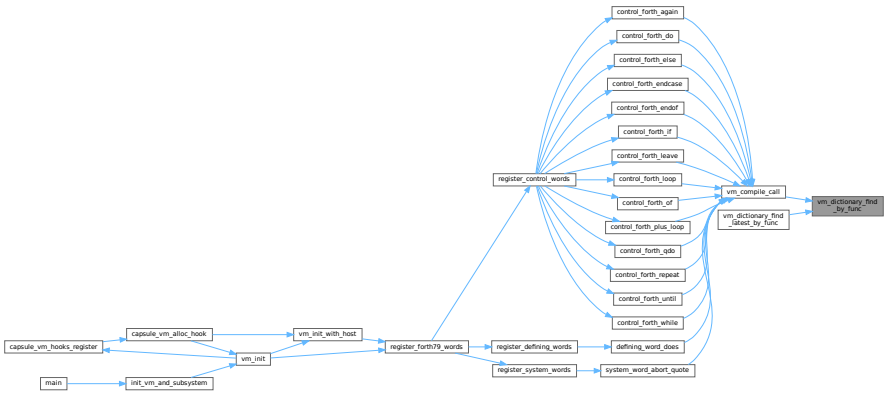
Returns

DictEntry* Pointer to found dictionary entry or NULL if not found

References func, and VM::latest.

Referenced by vm_compile_call(), and vm_dictionary_find_latest_by_func().

Here is the caller graph for this function:

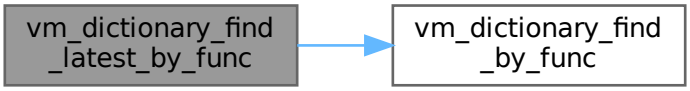


7.263.4.21 vm_dictionary_find_latest_by_func()

```
DictEntry * vm_dictionary_find_latest_by_func (
            VM * vm,
            word_func_t func )
```

References func, and vm_dictionary_find_by_func().

Here is the call graph for this function:



7.263.4.22 vm_dictionary_get_data_field()

```
cell_t * vm_dictionary_get_data_field (
            DictEntry * entry )
```

Gets pointer to a word's data field.

Calculates and returns pointer to the aligned data field following the name field.

Parameters

entry	Dictionary entry to get data field from

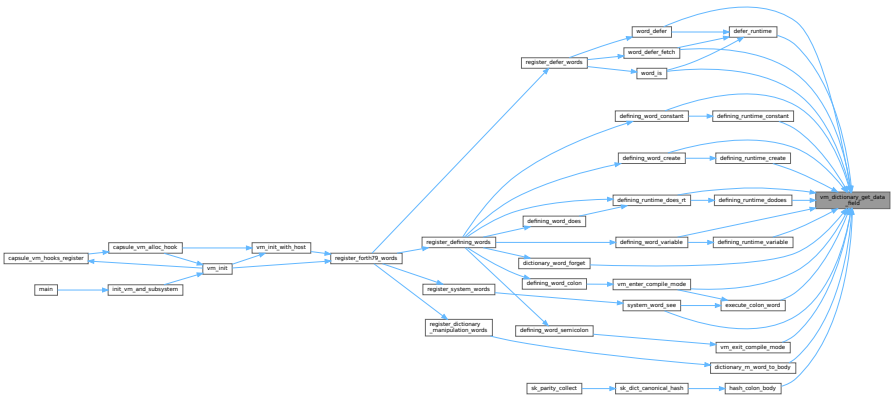
Returns

cell_t* Pointer to data field or NULL if entry is NULL

References base, entry, and name.

Referenced by defer_runtime(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_does_rt(), defining_runtime_variable(), defining_word_constant(), defining_word_create(), defining_word_variable(), dictionary_m_word_to_body(), dictionary_word_forget(), execute_colon_word(), hash_colon_body(), system_word_see(), vm_enter_compile_mode(), vm_exit_compile_mode(), word_defer(), word_defer_fetch(), and word_is().

Here is the caller graph for this function:



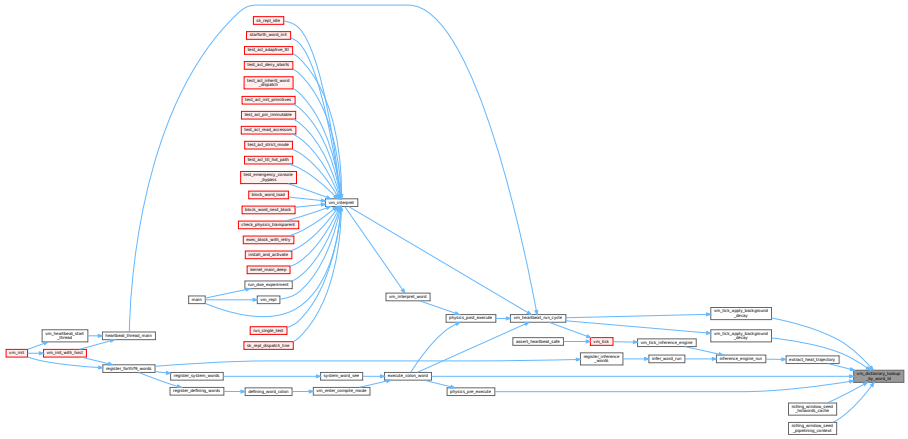
7.263.4.23 vm_dictionary_lookup_by_word_id()

```
DictEntry * vm_dictionary_lookup_by_word_id (
            VM * vm,
            uint32_t word_id )
```

References DICTIONARY_SIZE, word_id, and VM::word_id_map.

Referenced by execute_colon_word(), extract_heat_trajectory(), physics_pre_execute(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

Here is the caller graph for this function:



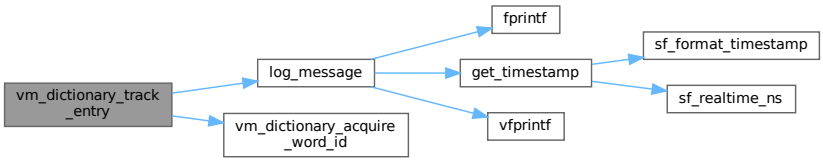
7.263.4.24 vm_dictionary_track_entry()

```
void vm_dictionary_track_entry (
            VM * vm,
            DictEntry * entry )
```

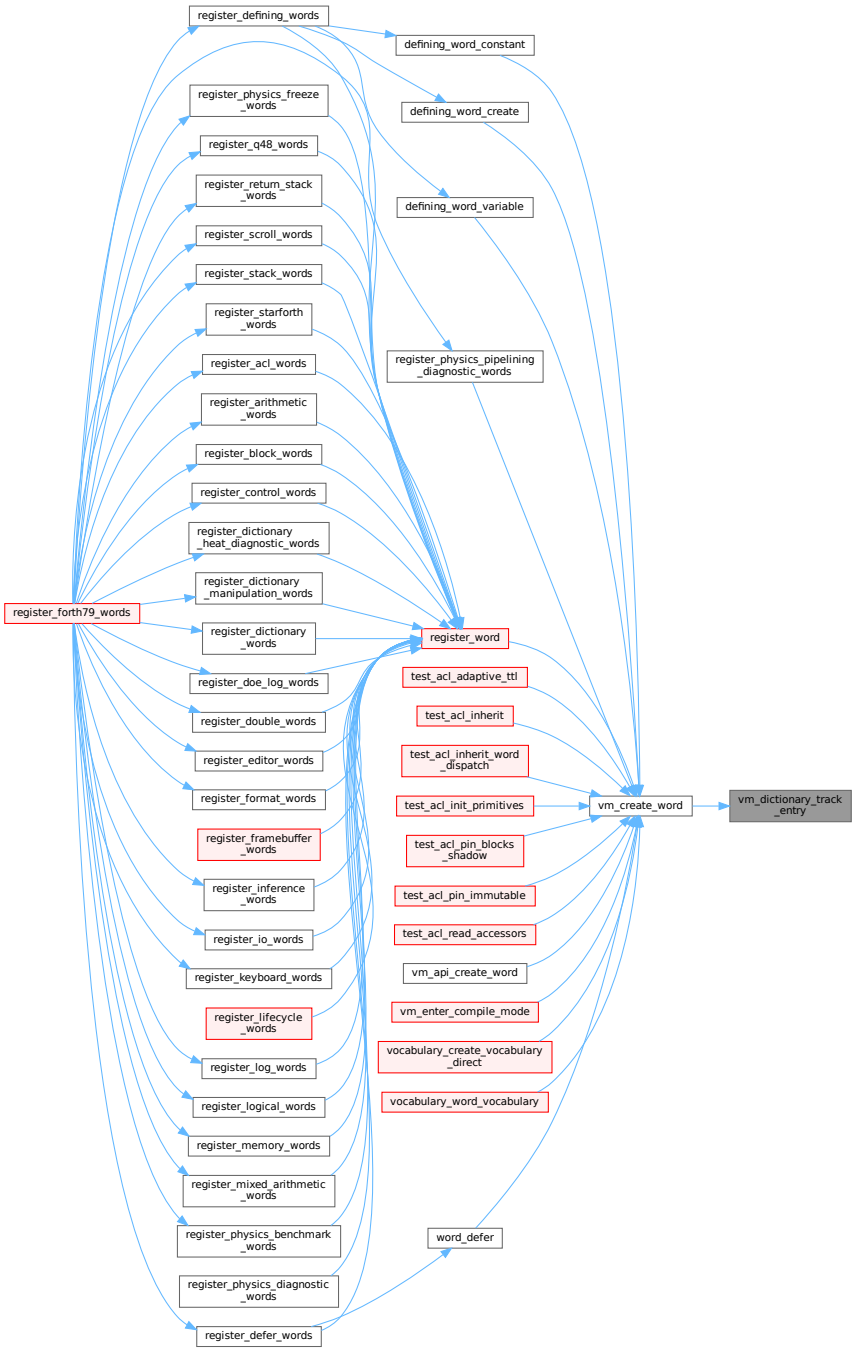
References DICTIONARY_SIZE, entry, log_message(), LOG_WARN, vm_dictionary_acquire_word_id(), word_id, WORD_ID_INVALID, and VM::word_id_map.

Referenced by vm_create_word().

Here is the call graph for this function:



Here is the caller graph for this function:



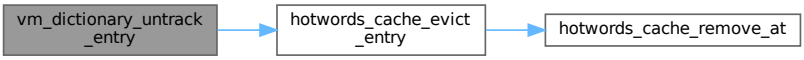
7.263.4.25 vm_dictionary_untrack_entry()

```
void vm_dictionary_untrack_entry (
            VM * vm,
            DictEntry * entry )
```

References DICTIONARY_SIZE, entry, VM::hotwords_cache, hotwords_cache_evict_entry(), VM::recycled_word_id_count, VM::recycled_word_ids, word_id, WORD_ID_INVALID, and VM::word_id_map.

Referenced by dictionary_word_forget().

Here is the call graph for this function:



Here is the caller graph for this function:

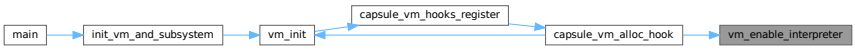
7.263.4.26 vm_enable_interpreter()

```
void vm_enable_interpreter (
            VM * vm )
```

References VM::interpreter_enabled.

Referenced by capsule_vm_alloc_hook().

Here is the caller graph for this function:



7.263.4.27 vm_enter_compile_mode()

```
void vm_enter_compile_mode (
            VM * vm,
            const char * name,
            size_t len )
```

Begin a colon definition and switch the VM to compile mode.

Creates a new dictionary entry for `name` via `vm_create_word()`, marks it `WORD_SMUDGED` so it is invisible to lookups during compilation, sets the VM mode to `MODE_COMPILE`, and writes the current `HERE` address into the entry's data field as the start of the threaded body.

The SMUDGE flag is cleared by `vm_exit_compile_mode()` when the semicolon is encountered.

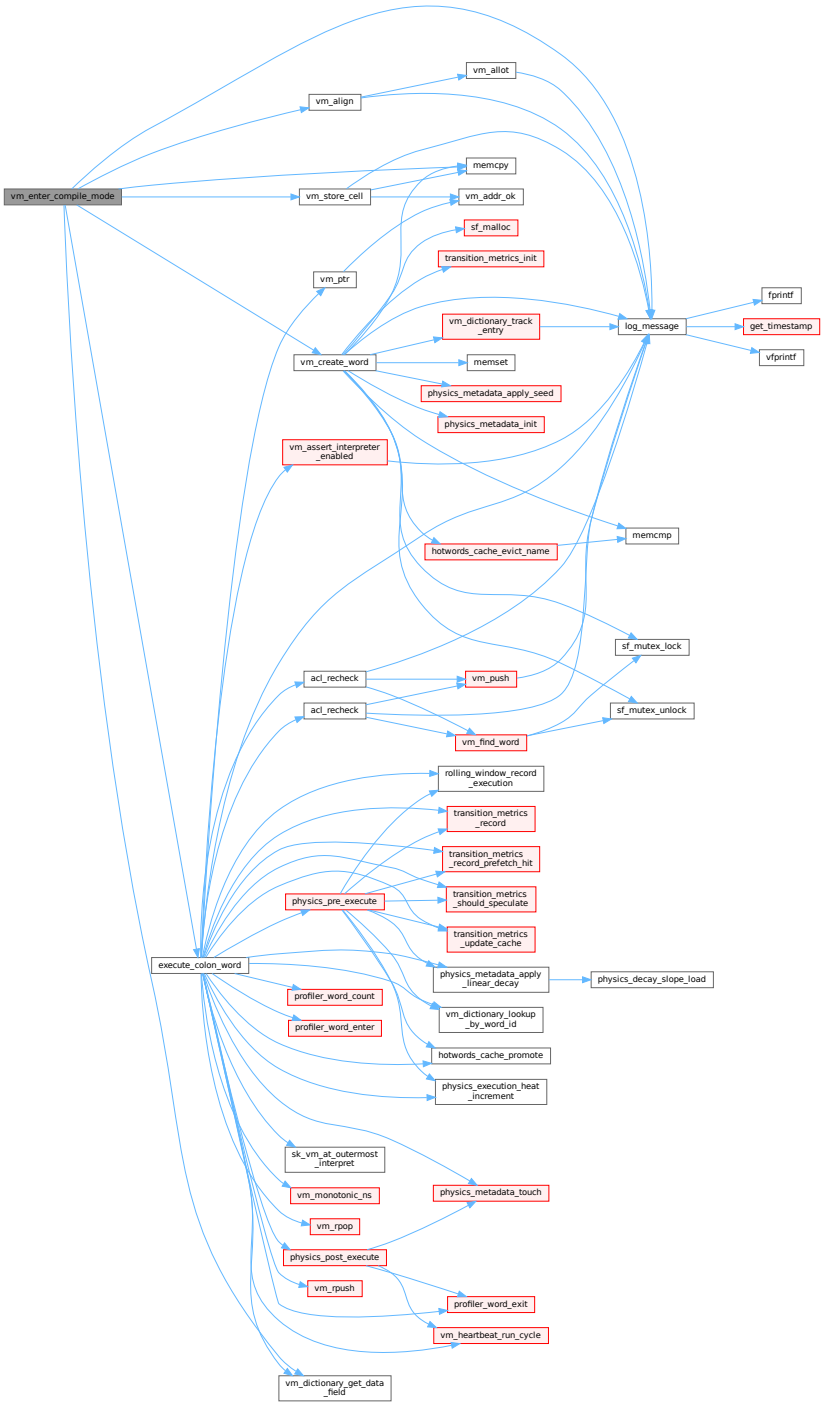
Parameters

vm	VM instance.
name	Word name string (does not need to be NUL-terminated).
len	Length of `name`; clamped to `WORD_NAME_MAX`.

References VM::compiling_word, VM::current_word_name, VM::error, execute_colon_word(), DictEntry::flags, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), memcpy(), VM::mode, MODE_COMPILE, name, VM::state_addr, VM::state_var, vm_align(), vm_create_word(), vm_dictionary_get_data_field(), vm_store_cell(), WORD_NAME_MAX, and WORD_SMUDGED.

Referenced by defining_word_colon().

Here is the call graph for this function:



Here is the caller graph for this function:

7.263.4.28 vm_exit_compile_mode()

```
void vm_exit_compile_mode (
            VM * vm )
```

Finalise a colon definition and return to interpret mode.

Appends a final `EXIT` reference, clears the `WORD_SMUDGED` flag so the new word becomes visible in the dictionary, and sets `WORD_COMPILED`. Also computes the word's mass (header + body bytes) for the physics engine and calls `physics_metadata_refresh_state()`.

Resets `vm->mode` to `MODE_INTERPRET`, clears `vm->state_var`, and NULLs `vm->compiling_word`.

Sets `vm->error` if `vm` has no active `compiling_word` or if `EXIT` cannot be found.

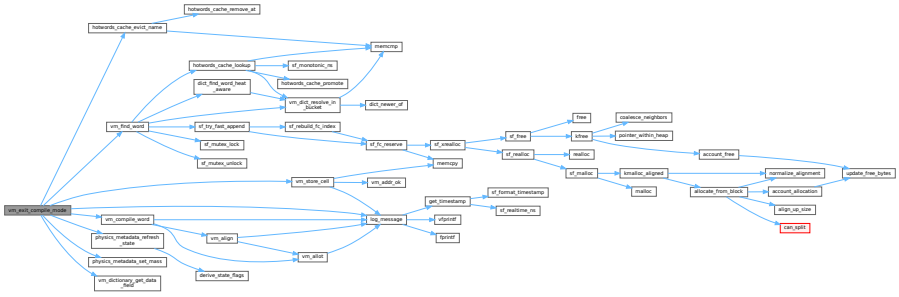
Parameters

vm	VM instance currently in compile mode.

References DictEntry::acl_allow, DictEntry::acl_ttl, ACL_TTL_OPEN, VM::compiling_word, VM::error, DictEntry::flags, VM::here, VM::hotwords_cache, hotwords_cache_evict_name(), LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_INTERPRET, DictEntry::name, DictEntry::name_len, physics_metadata_refresh_state(), physics_metadata_set_mass(), VM::state_addr, VM::state_var, vm_compile_word(), vm_dictionary_get_data_field(), vm_find_word(), vm_store_cell(), WORD_COMPILED, and VM::zuse_session.

Referenced by defining_word_semicolon().

Here is the call graph for this function:



Here is the caller graph for this function:

7.263.4.29 vm_find_word()

```
DictEntry * vm_find_word (
            VM * vm,
```

```
const char * name,
size_t len )
```

Finds a word in the dictionary by name.

Searches for a word in the dictionary using an optimized first-character index. The search is performed newest-first within the matching first-character bucket.

Parameters

vm	The virtual machine context
name	The name to search for
len	Length of the name string

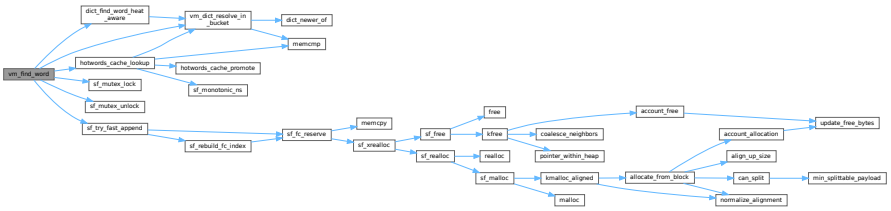
Returns

DictEntry* Pointer to found dictionary entry or NULL if not found

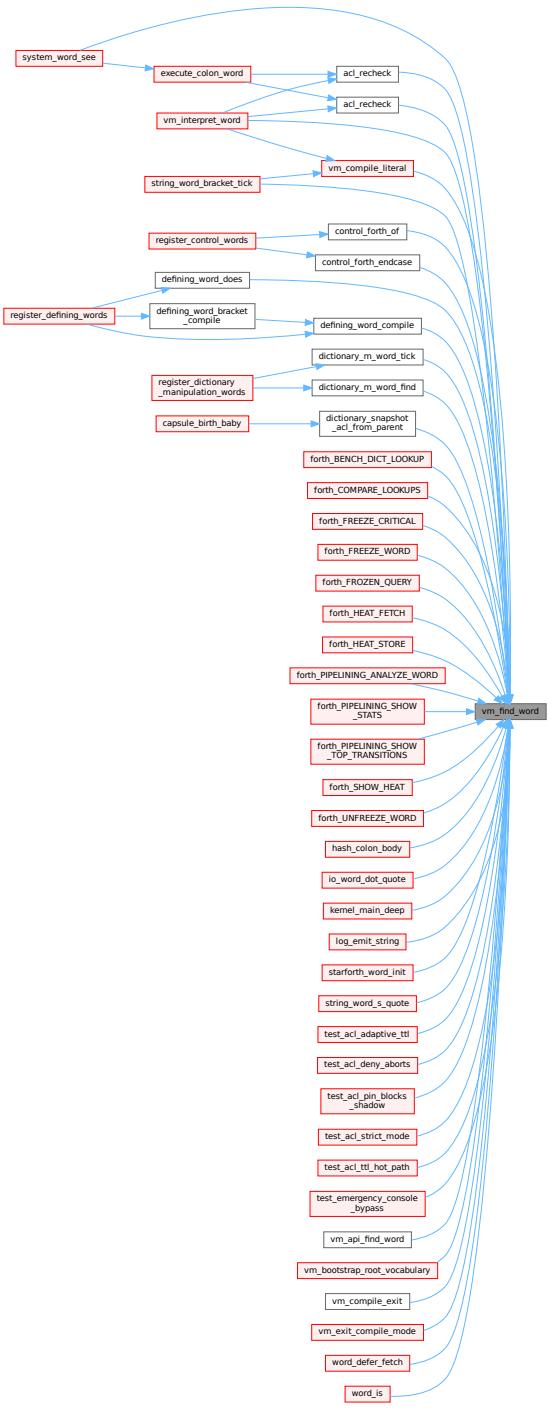
References dict_find_word_heat_aware(), VM::dict_lock, HeartbeatState::dictionary_lookups, VM::heartbeat, VM::hotwords_cache, hotwords_cache_lookup(), VM::latest, VM::lookup_strategy, name, sf_cached_latest, sf_fc_count, sf_fc_list, sf_mutex_lock(), sf_mutex_unlock(), sf_try_fast_append(), UNLIKELY, and vm_dict_resolve_in_bucket().

Referenced by acl_recheck(), acl_recheck(), control_forth_endcase(), control_forth_of(), defining_word_compile(), defining_word_does(), dictionary_m_word_find(), dictionary_m_word_tick(), dictionary_snapshot_acl_from_parent(), forth_BENCH_DICT_LOOKUP(), forth_COMPARE_LOOKUPS(), forth_FREEZE_CRITICAL(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_STORE(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), hash_colon_body(), io_word_dot_quote(), kernel_main_deep(), log_emit_string(), starforth_word_init(), string_word_bracket_tick(), string_word_s_quote(), system_word_see(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_pin_blocks_shadow(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_byp vm_api_find_word(), vm_bootstrap_root_vocabulary(), vm_compile_exit(), vm_compile_literal(), vm_exit_compile_mode(), vm_interpret_word(), word_defer_fetch(), and word_is().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.30 vm_get_base()

```
unsigned vm_get_base (
            const struct VM * vm )
```

7.263.4.31 vm_get_block_addr()

```
void * vm_get_block_addr (
            VM * vm,
            int block_num )
```

Gets the memory address for a specified block number.

Parameters

vm	Pointer to the VM instance
block_num	Block number to get the address for

Returns

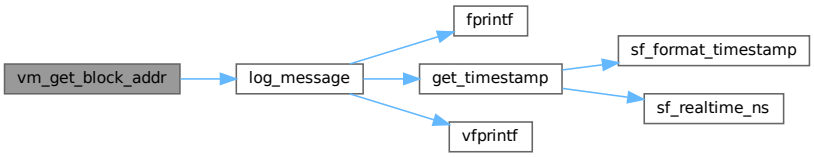
Pointer to the start of the block or NULL if block number is invalid

Note

Blocks are fixed-size memory regions of BLOCK_SIZE bytes

References BLOCK_SIZE, LOG_ERROR, log_message(), MAX_BLOCKS, and VM::memory.

Here is the call graph for this function:



7.263.4.32 vm_heartbeat_start()

```
void vm_heartbeat_start (
            VM * vm )
```

7.263.4.33 vm_hide_word()

```
void vm_hide_word (
            VM * vm )
```

Marks the latest word as hidden.

Sets the WORD_HIDDEN flag on the most recently defined word.

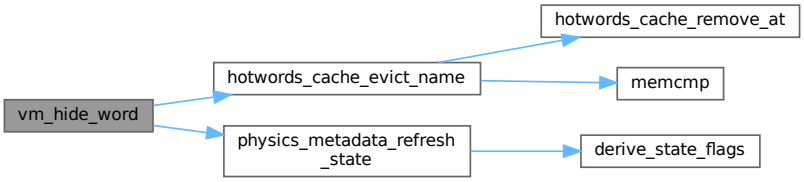
Parameters

vm	The virtual machine context

References DictEntry::flags, VM::hotwords_cache, hotwords_cache_evict_name(), VM::latest, DictEntry::name, DictEntry::name_len, physics_metadata_refresh_state(), and WORD_HIDDEN.

Referenced by vm_api_hide_word().

Here is the call graph for this function:



Here is the caller graph for this function:

7.263.4.34 vm_init()

```
void vm_init (
            VM * vm )
```

Initialize a new virtual machine instance.

Allocates memory and initializes all VM structures including:

- Memory array
- Data and return stacks
- System variables (SCR, STATE, BASE)
- Dictionary
- Forth-79 wordset

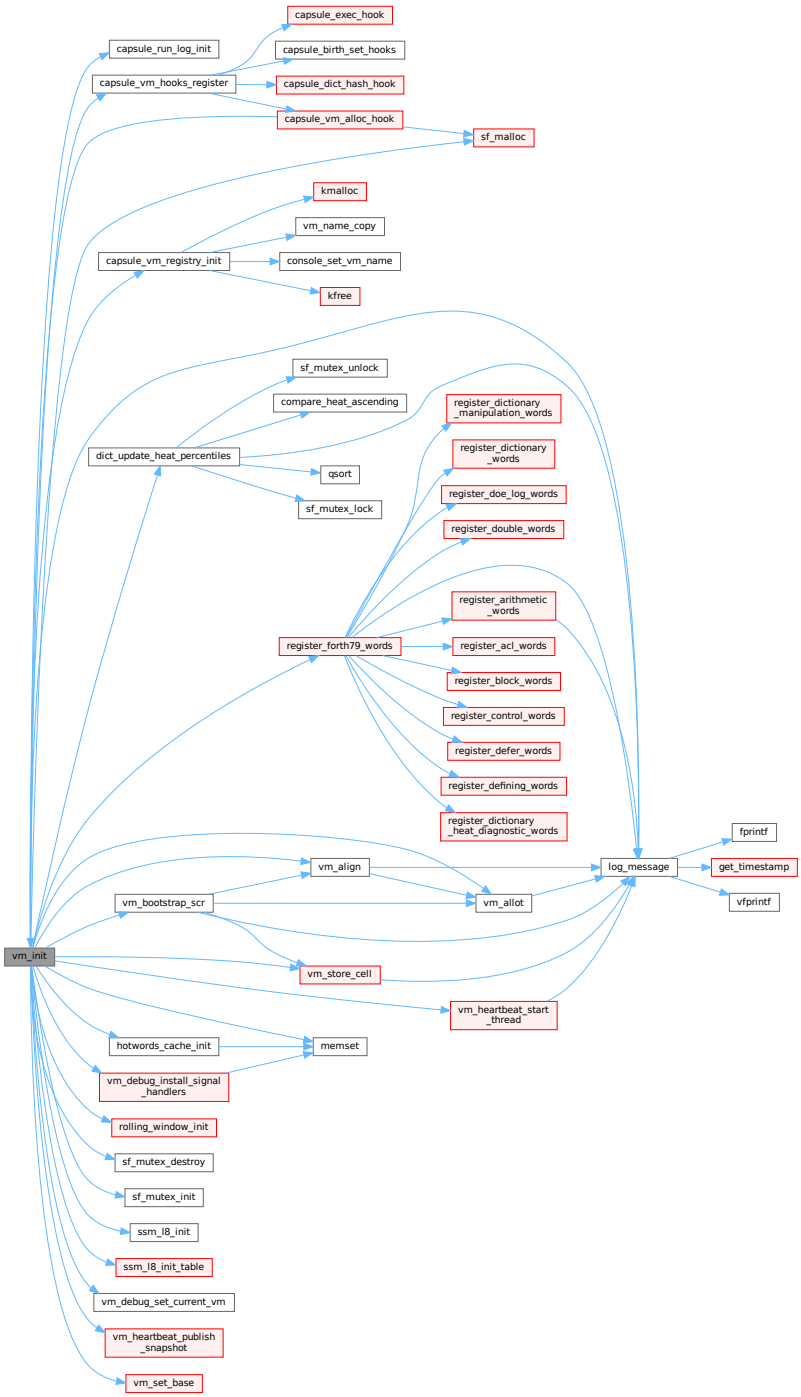
Parameters

vm	Pointer to VM structure to initialize

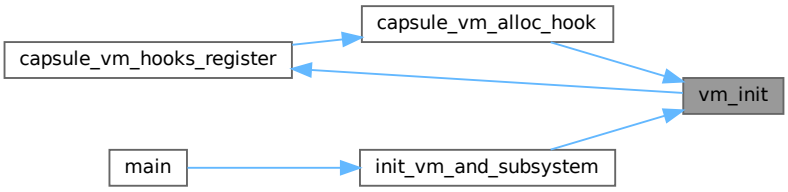
References VM::abort_requested, VM::base_addr, capsule_run_log_init(), capsule_vm_hooks_register(), capsule_vm_registry_init(), VM::cf, ControlFlowState::cf_last_mode, ControlFlowState::cf_sp, HeartbeatState::check_counter, VM::compiling_word, VM::current_executing_entry, VM::decay_slope_direction, VM::decay_slope_q48, VM::dict_fence_here, VM::dict_fence_latest, VM::dict_lock, dict_update_heat_percentiles(), VM::dsp, RollingWindowOfTruth::effective_window_size, ControlFlowState::endof_mark_sp, ControlFlowState::endof_sp, VM::error, VM::exit_colon, VM::halted, VM::heartbeat, VM::heartbeat_decay_cursor_id, HeartbeatState::heartbeat_enabled, HEARTBEAT_TICK_NS, VM::here, VM::hold_addr, VM::hold_pos, VM::hotwords_cache, hotwords_cache_init(), VM::input_length, VM::input_pos, VM::last_bucket_reorg_ns, PipelineGlobalMetrics::last_checked_accuracy, PipelineGlobalMetrics::last_checked_window_size, VM::last_decay_check_ns, HeartbeatState::last_inference_tick, VM::latest, ControlFlowState::leave_mark_sp, ControlFlowState::leave_sp, LOG_ERROR, log_message(), VM::lookup_strategy, VM::memory, memset(), VM::mode, MODE_INTERPRET, VM::next_word_id, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::recycled_word_id_count, register_forth79_words(), VM::rolling_window, rolling_window_init(), VM::rsp, VM::scr_addr, sf_malloc(), sf_mutex_destroy(), sf_mutex_init(), HeartbeatState::snapshot_index, VM::ssm_config, ssm_l8_init(), ssm_l8_init_table(), VM::ssm_l8_state, SSM_MODE_C0, VM::stale_word_count_at_check, VM::state_addr, VM::state_var, PipelineGlobalMetrics::suggested_next_size, HeartbeatState::tick_count, HeartbeatState::tick_target_ns, VM::total_heat_at_last_check, VM::tuning_lock, vm_align(), vm_allot(), vm_bootstrap_scr(), vm_debug_install_signal_handlers(), vm_debug_set_current_vm(), vm_heartbeat_publish_snapshot(), vm_heartbeat_start_thread(), VM_MEMORY_SIZE, vm_set_base(), vm_store_cell(), PipelineGlobalMetrics::window_tuning_checks, WORD_ID_INVALID, and HeartbeatState::worker.

Referenced by capsule_vm_alloc_hook(), and init_vm_and_subsystem().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.35 vm_init_with_host()

```
void vm_init_with_host (
            VM * vm,
            const VMHostServices * host )
```

Initialize VM with explicit host services.

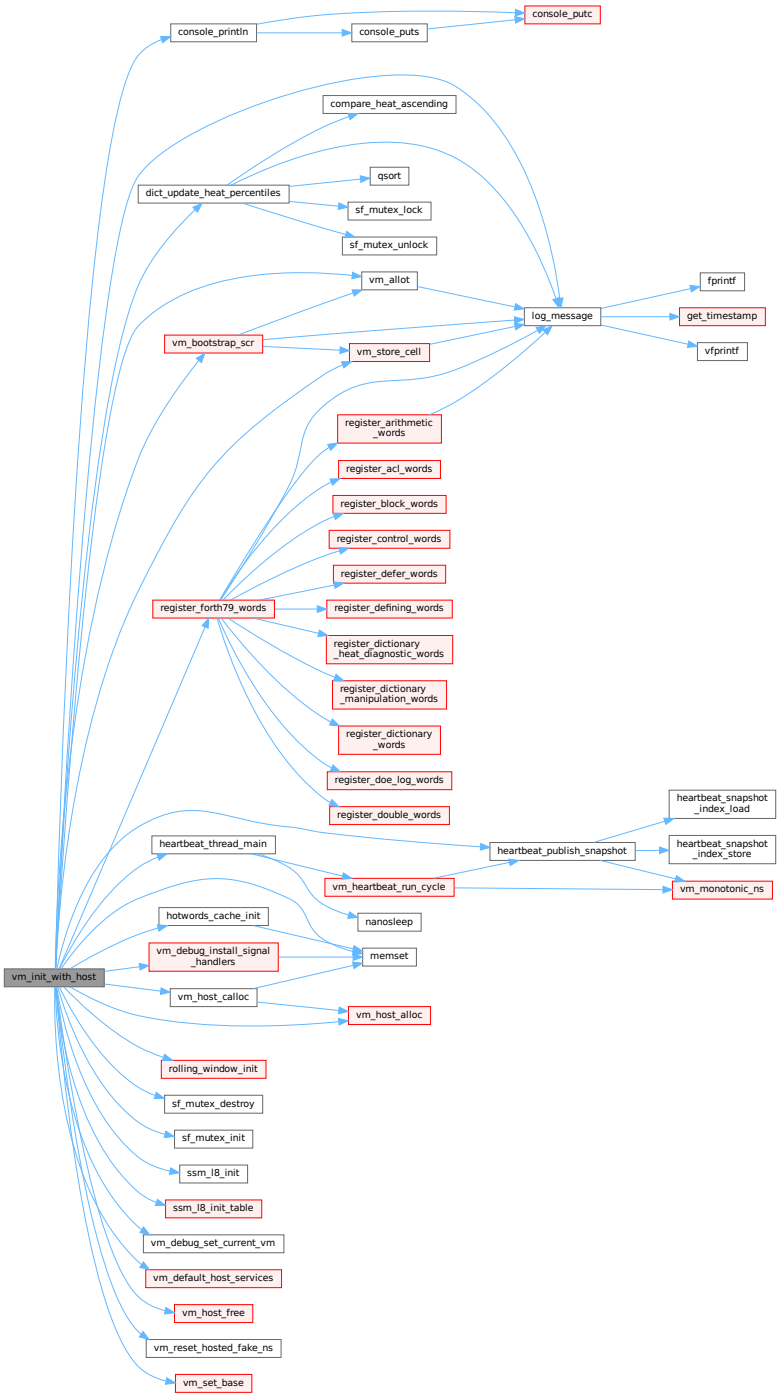
Parameters

vm	Pointer to VM structure to initialize
host	Host services (allocator/time). If NULL, uses build default.

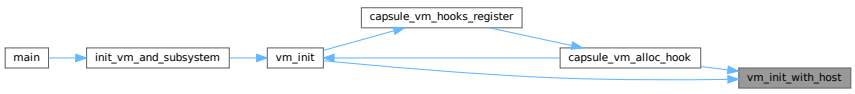
References VM::abort_requested, VM::base_addr, VM::cf, ControlFlowState::cf_last_mode, ControlFlowState::cf_sp, HeartbeatState::check_counter, VM::compiling_word, console_println(), VM::current_executing_entry, VM::decay_slope_direction, VM::decay_slope_q48, VM::dict_fence_here, VM::dict_fence_latest, VM::dict_lock, dict_update_heat_percentiles(), VM::dsp, RollingWindowOfTruth::effective_window_size, ControlFlowState::endof_mark_sp, ControlFlowState::endof_sp, errno, VM::error, VM::exit_colon, VM::halted, VM::heartbeat, VM::heartbeat_decay_cursor_id, HeartbeatState::heartbeat_enabled, heartbeat_publish_snapshot(), heartbeat_thread_main(), HEARTBEAT_TICK_NS, VM::here, VM::hold_addr, VM::hold_pos, VM::host, VM::hotwords_cache, hotwords_cache_init(), VM::input_length, VM::input_pos, VM::interpreter_enabled, VM::last_bucket_reorg_ns, PipelineGlobalMetrics::last_checked_accuracy, PipelineGlobalMetrics::last_chec VM::last_decay_check_ns, HeartbeatState::last_inference_tick, VM::latest, ControlFlowState::leave_mark_sp, ControlFlowState::leave_sp, LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, VM::lookup_strategy, VM::memory, memset(), VM::mode, MODE_INTERPRET, VM::next_word_id, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_ PipelineGlobalMetrics::prefetch_hits, VM::recycled_word_id_count, register_forth79_words(), VM::rolling_window, rolling_window_init(), VM::rsp, sf_mutex_destroy(), sf_mutex_init(), HeartbeatState::snapshot_index, VM::ssm_config, ssm_l8_init(), ssm_l8_init_table(), VM::ssm_l8_state, SSM_MODE_C0, VM::stale_word_count_at_check, VM::state_addr, VM::state_var, PipelineGlobalMetrics::suggested_next_size, HeartbeatState::tick_count, HeartbeatWorker::tick_ns, HeartbeatState::tick_target_ns, VM::total_heat_at_last_check, VM::tuning_lock, vm_allot(), vm_bootstrap_scr(), vm_debug_install_signal_handlers(), vm_debug_set_current_vm(), vm_default_host_services(), vm_host_alloc(), vm_host_calloc(), vm_host_free(), VM_MEMORY_SIZE, vm_reset_hosted_fake_ns(), vm_set_base(), vm_store_cell(), PipelineGlobalMetrics::window_tuning_checks, WORD_ID_INVALID, and HeartbeatState::worker.

Referenced by capsule_vm_alloc_hook(), and vm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



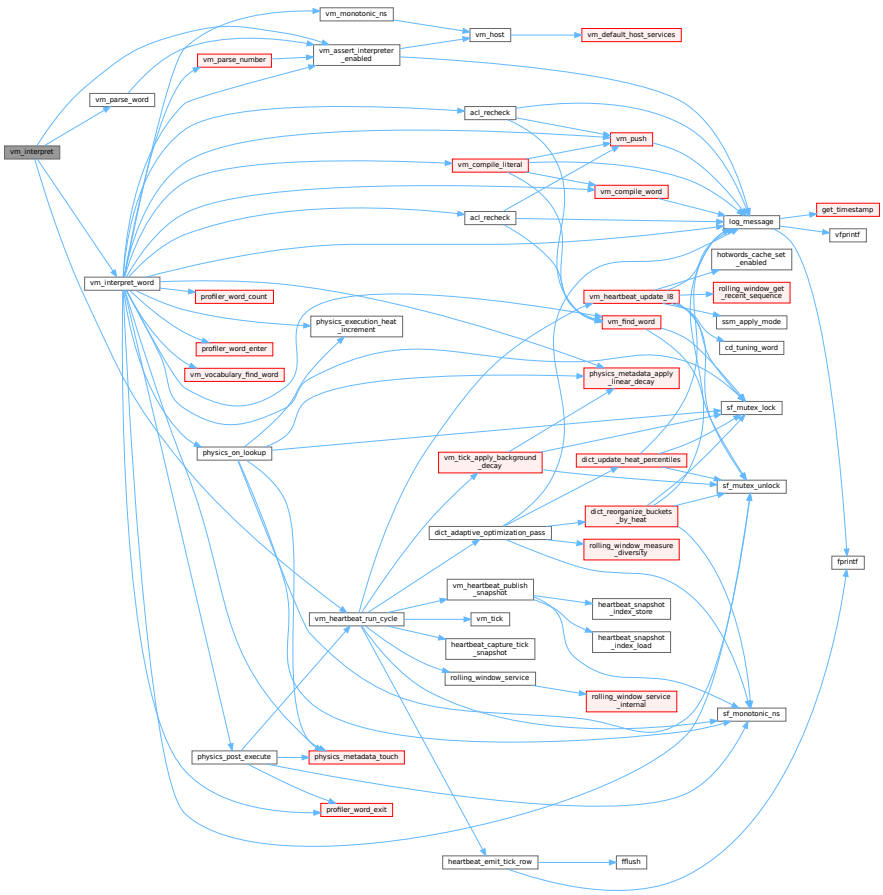
7.263.4.36 vm_interpret()

```
void vm_interpret (
            VM * vm,
            const char * input )
```

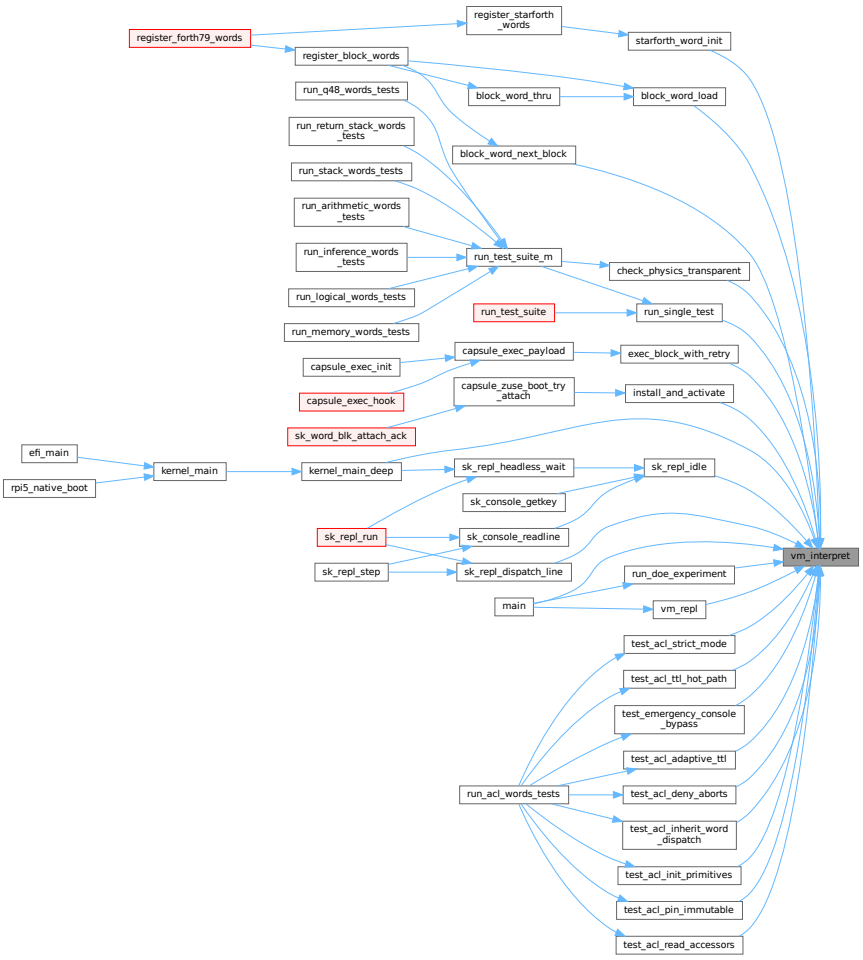
References VM::abort_requested, HeartbeatState::check_counter, VM::error, g_log_attrib_vm, g_vm_interpret_depth, g_vm_interpret_depth, VM::heartbeat, HEARTBEAT_CHECK_FREQUENCY, VM::input_buffer, INPUT_BUFFER_SIZE, VM::input_length, VM::input_pos, vm_assert_interpreter_enabled(), vm_heartbeat_run_cycle(), vm_interpret_word(), vm_parse_word(), and HeartbeatState::worker.

Referenced by block_word_load(), block_word_next_block(), check_physics_transparent(), exec_block_with_retry(), install_and_activate(), kernel_main_deep(), main(), run_doe_experiment(), run_single_test(), sk_repl_dispatch_line(), sk_repl_idle(), starforth_word_init(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_bypass(), and vm_repl().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.37 vm_interpret_word()

```
void vm_interpret_word (
            VM * vm,
            const char * word_str,
            size_t len )
```

Interpret or compile a single word token.

This is the outer interpreter's per-token dispatch:

1. **Dictionary lookup**: vocabulary-aware search, falling back to the canonical flat dictionary. Physics heat is recorded via `physics_on_lookup()`.
2. **Compile mode, non-immediate**: append the entry to the current threaded body via `vm_compile_word()`.
3. **Interpret mode, or immediate in compile mode**: ACL check (hot-path TTL decrement / cold-path `acl_recheck()`), then call `entry->func()`.
4. **Number literal**: if no dictionary match, attempt `vm_parse_number()`. In compile mode, compile as a literal; in interpret mode, push directly.
5. **Error**: unknown token — sets `vm->error`.

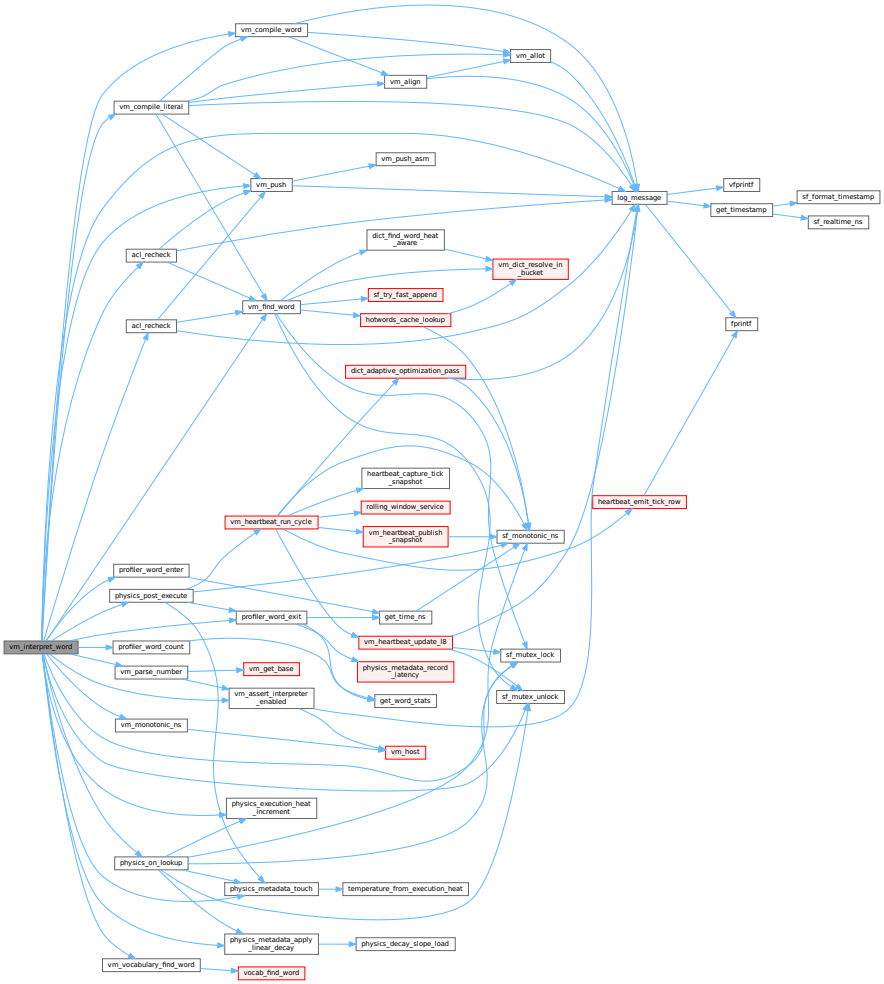
Parameters

vm	VM instance.
word_str	Token string (need not be NUL-terminated).
len	Length of `word_str` in bytes.

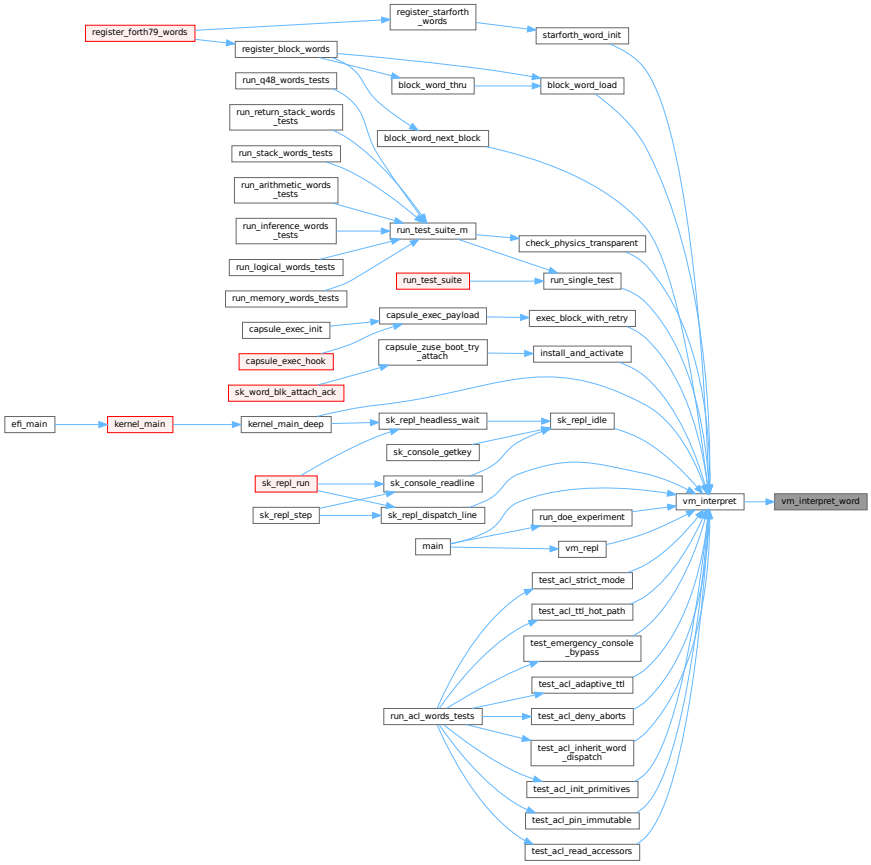
References acl_recheck(), acl_recheck(), VM::current_executing_entry, VM::dict_lock, VM::dsp, VM::emergency_console, entry, VM::error, DictEntry::execution_heat, DictEntry::flags, g_sk_fault_word, VM::heartbeat, DictPhysics::last_active_ns, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, VM::mode, MODE_COMPILE, name, DictEntry::physics, physics_execution_heat_increment(), physics_metadata_apply_linear_decay(), physics_metadata_touch(), physics_on_lookup(), physics_post_execute(), profiler_word_count(), profiler_word_enter(), profiler_word_exit(), VM::rsp, sf_mutex_lock(), sf_mutex_unlock(), HeartbeatState::tick_count, vm_assert_interpreter_enabled(), vm_compile_literal(), vm_compile_word(), vm_find_word(), vm_monotonic_ns(), vm_parse_number(), vm_push(), vm_vocabulary_find_word(), DictEntry::word_id, WORD_IMMEDIATE, and HeartbeatState::words_executed.

Referenced by vm_interpret().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.38 vm_load_cell()

```
cell_t vm_load_cell (
            struct VM * vm,
            vaddr_t addr )
```

Load an aligned cell from VM memory.

Validates that `addr` is within bounds for a full `cell_t` read and that it is naturally aligned (addr % sizeof(cell_t) == 0). Uses `memcpy()` rather than a direct pointer dereference to avoid strict aliasing and alignment UB. Sets `vm->error` and returns 0 on failure.

Parameters

vm	VM instance.
addr	Aligned VM byte offset to read.

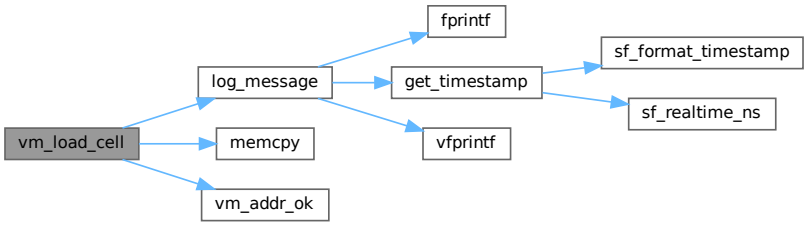
Returns

Cell value at `addr`, or 0 on alignment or bounds error.

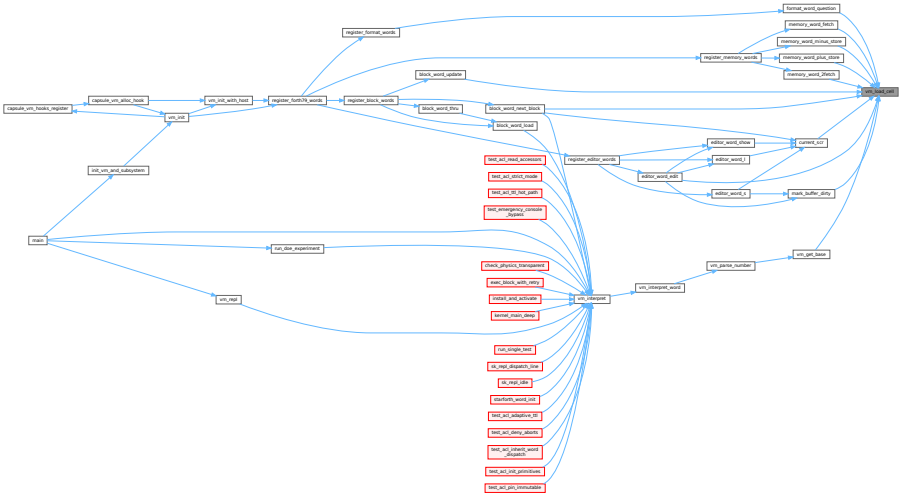
References VM::error, LOG_ERROR, log_message(), memcpy(), VM::memory, and vm_addr_ok().

Referenced by block_word_next_block(), block_word_update(), current_scr(), editor_word_edit(), format_word_question(), mark_buffer_dirty(), memory_word_2fetch(), memory_word_fetch(), memory_word_minus_store(), memory_word_plus_store(), and vm_get_base().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.39 vm_load_u8()

```
uint8_t vm_load_u8 (
            struct VM * vm,
            vaddr_t addr )
```

Load an unsigned byte from VM memory.

Resolves `addr` via `vm_ptr()`. Sets `vm->error` and returns 0 if the address is out of bounds.

Parameters

vm	VM instance.
addr	VM byte offset to read.

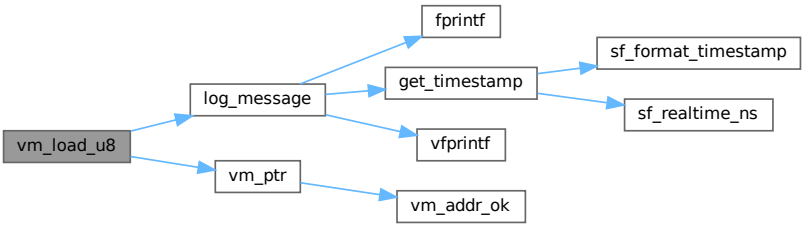
Returns

Byte value at `addr`, or 0 on error.

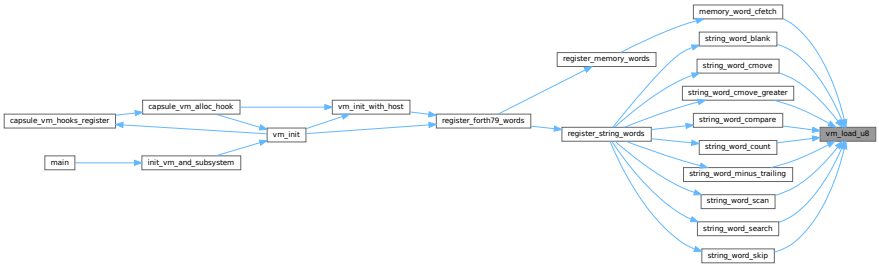
References VM::error, LOG_ERROR, log_message(), and vm_ptr().

Referenced by memory_word_cfetch(), string_word_blank(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_count(), string_word_minus_trailing(), string_word_scan(), string_word_search(), and string_word_skip().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.40 vm_make_immediate()

```
void vm_make_immediate (
            VM * vm )
```

Mark the most recently created word as IMMEDIATE (FORTH-79).

Sets `WORD_IMMEDIATE` on `vm->latest` so the word executes during compilation rather than being compiled into the current definition. Also calls `physics_metadata_refresh_state()` to update the physics engine's record of the entry's flags.

Sets `vm->error` if `vm` has no `latest` entry (i.e., the dictionary is empty or no word has been created yet).

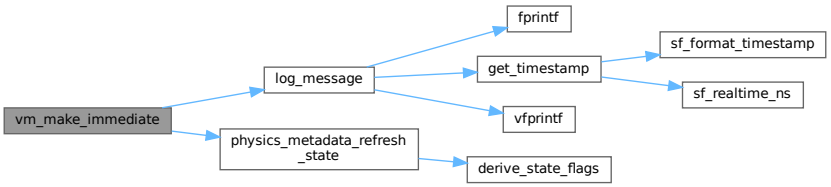
Parameters

vm	VM instance.

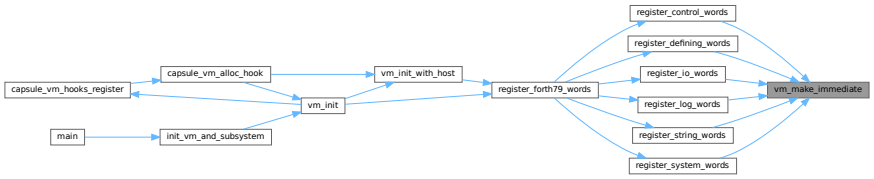
References VM::error, DictEntry::flags, VM::latest, LOG_DEBUG, LOG_ERROR, log_message(), DictEntry::name, DictEntry::name_len, physics_metadata_refresh_state(), and WORD_IMMEDIATE.

Referenced by register_control_words(), register_defining_words(), register_io_words(), register_log_words(), register_string_words(), and register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.41 vm_parse_number()

```
int vm_parse_number (
            VM * vm,
            const char * s,
            cell_t * out )
```

Attempt to parse a string as an integer in the current number base.

Reads the current base from `vm_get_base()`, then converts each character of `s` (case-insensitive, digits 0–9 and letters A–Z/a–z) to a digit value. Returns 0 immediately on any character that is not a valid digit in the current base, or on an empty string after the optional sign.

Handles an optional leading `'+'` or `'-'` sign. The result is a full-width `cell_t` (sign extended for negative values).

Parameters

vm	VM instance (used for base only; may be NULL, in which case base defaults to 10).
s	NUL-terminated string to parse.
out	Pointer to receive the parsed value on success.

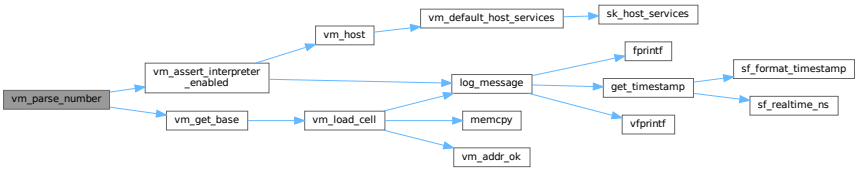
Returns

1 on successful parse, 0 on any parse failure.

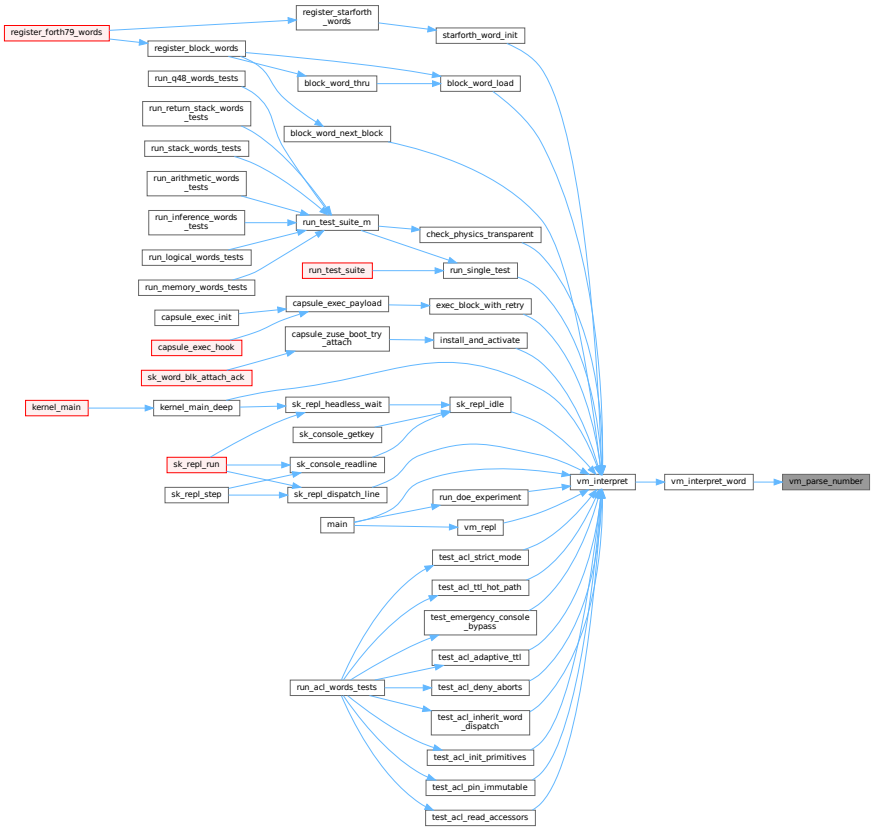
References base, vm_assert_interpreter_enabled(), and vm_get_base().

Referenced by vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.42 vm_parse_word()

```
int vm_parse_word (
            VM * vm,
```

```
char * word,
size_t max_len )
```

Parse the next whitespace-delimited token from the input buffer.

Skips leading whitespace (space, tab, CR, LF), then extracts characters until the next whitespace or end of buffer, storing up to `max_len` ? 1 characters in `word` followed by a NUL terminator. Advances `vm->input_pos` past the consumed token.

Parameters

vm	VM instance; `vm->input_buffer` / `input_pos` / `input_length` must be populated (typically by `vm_interpret()`).
word	Output buffer for the parsed token.
max_len	Size of `word` in bytes (including NUL terminator).

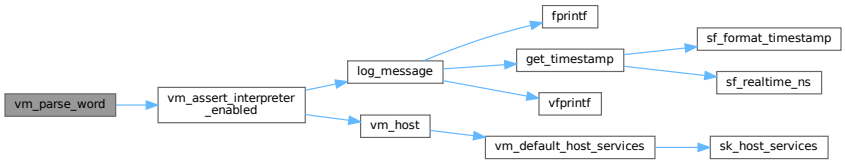
Returns

Number of characters written (excluding NUL), or 0 if the buffer is exhausted or `vm` / `word` is NULL.

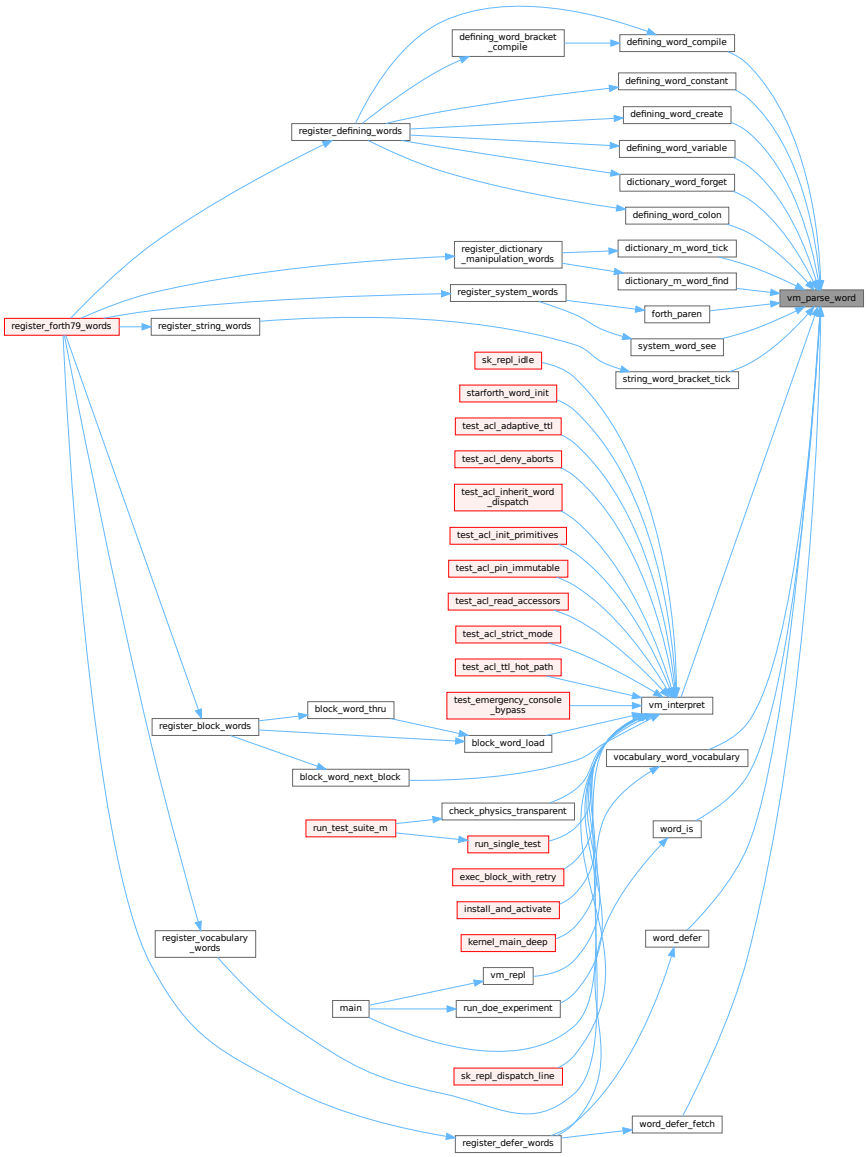
References VM::input_buffer, VM::input_length, VM::input_pos, and vm_assert_interpreter_enabled().

Referenced by defining_word_colon(), defining_word_compile(), defining_word_constant(), defining_word_create(), defining_word_variable(), dictionary_m_word_find(), dictionary_m_word_tick(), dictionary_word_forget(), forth_paren(), string_word_bracket_tick(), system_word_see(), vm_interpret(), vocabulary_word_vocabulary(), word_defer(), word_defer_fetch(), and word_is().

Here is the call graph for this function:



Here is the caller graph for this function:

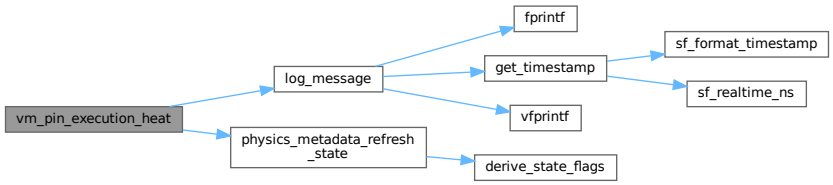


7.263.4.43 vm_pin_execution_heat()

```
void vm_pin_execution_heat (
            VM * vm )
```

References VM::error, DictEntry::flags, VM::latest, LOG_ERROR, log_message(), physics_metadata_refresh_state(), and WORD_PINNED.

Here is the call graph for this function:



7.263.4.44 vm_pop()

```
cell_t vm_pop (
            VM * vm )
```

Pop a value from the data stack.

Parameters

vm	Pointer to the VM structure

Returns

The popped value

Note

Returns 0 and sets error flag if stack underflow occurs

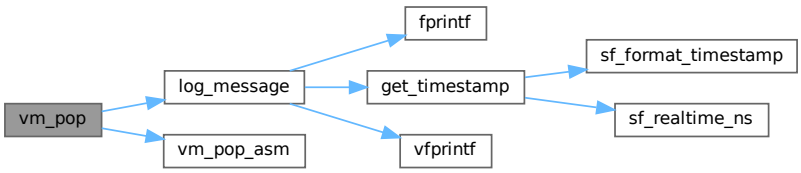
Uses assembly optimization if available for x86_64 or ARM64

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), PROFILE_INC_STACK_OP, and vm_pop_asm().

Referenced by arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), block_word_acl_allow_fetch(), block_word_acl_allow_store(), block_word_acl_ttl_fetch(), block_word_acl_ttl_store(), block_word_blk_attach(), block_word_block(), block_word_buffer(), block_word_confirm_format(), block_word_list(), block_word_load(), block_word_owner_fetch(), block_word_relocate(), block_word_thru(), control_forth_0branch(), control_forth_runtime_do(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), defining_word_constant(), defining_word_literal(), dictionary_m_word_cfa(), dictionary_m_word_link_from(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), dictionary_m_word_traverse(), dictionary_word_2comma(), dictionary_word_allot(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_sp_store(), double_word_2drop(), double_word_2rot(), double_word_2swap(), double_word_2to_r(), double_word_d_equals(), double_word_d_less(), double_word_d_minus(), double_word_d_plus(), double_word_d_two_slash(), double_word_d_two_star(), double_word_d_zero_equals(), double_word_d_zero_less(), double_word_dmax(), double_word_dmin(), double_word_dnegate(), double_word_s_to_d(), editor_word_edit(), editor_word_l(), editor_word_s(), fbw_plot(), format_word_d_dot(), format_word_d_dot_r(), format_word_dot(),

format_word_dot_r(), format_word_dump(), format_word_end_conversion(), format_word_hash(), format_word_hash_s(), format_word_hold(), format_word_question(), format_word_sign(), format_word_u_dot(), format_word_u_dot_r(), forth_acl_allow_store(), forth_acl_mode_store(), forth_acl_ttl_store(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), hosted_lifecycle_birth(), hosted_lifecycle_kill(), hosted_lifecycle_pause(), hosted_lifecycle_resume(), hosted_lifecycle_use(), io_word_type(), log_str_emit(), log_word_set_level(), logical_word_and(), logical_word_equals(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), memory_word_2fetch(), memory_word_2store(), memory_word_cells(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), mixed_math_word_m_minus(), mixed_math_word_m_plus(), mixed_math_word_m_slash_mod(), mixed_math_word_m_star(), mixed_math_word_mod(), mixed_math_word_mod(), mixed_math_word_slash_mod(), mixed_math_word_star_slash(), mixed_math_word_star_slash_mod(), pop_xt(), return_stack_word_to_r(), runtime_abortq(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), stack_word_pick(), starforth_word_execution_heat_fetc starforth_word_execution_heat_store(), starforth_word_random(), starforth_word_seed(), starforth_word_top_words(), starforth_word_wait(), starforth_word_zuse_pubkey_fetch(), string_word_blank(), string_word_cmove(), string_word_cmove_greater() string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_minus_trailing(), string_word_number(), string_word_scan(), string_word_search(), string_word_skip(), string_word_word(), system_word_abort_quote(), system_word_execute(), system_word_reboot(), test_acl_adaptive_ttl(), test_acl_read_accessors(), vm_api_pop(), word_divide(), word_divmod(), and word_is().

Here is the call graph for this function:



7.263.4.45 vm_pop_fast()

```
static cell_t vm_pop_fast (
            VM * vm )  [inline], [static]
```

References VM::data_stack, and VM::dsp.

7.263.4.46 vm_ptr()

```
uint8_t * vm_ptr (
            struct VM * vm,
            vaddr_t addr )
```

Return a host C pointer into VM memory at `addr`.

Validates the address with `vm_addr_ok()` (minimum 1-byte range) and returns `NULL` if the address is out of bounds or `vm` is uninitialised. The returned pointer is valid for at least one byte; callers that need multi-byte access should perform their own range check first.

Parameters

vm	VM instance.
addr	VM byte offset to dereference.

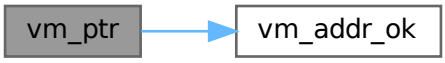
Returns

Host pointer into `vm->memory`, or NULL on bounds failure.

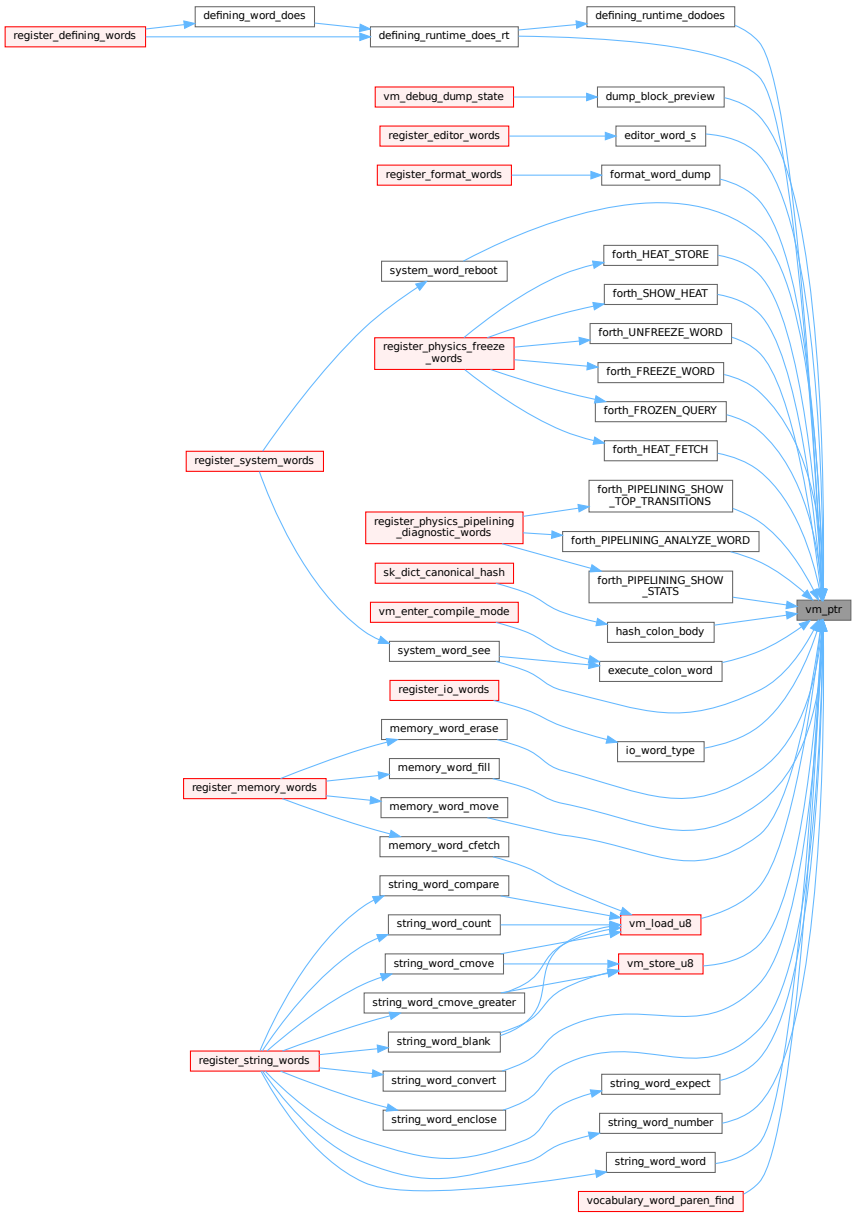
References VM::memory, and vm_addr_ok().

Referenced by defining_runtime_dodoes(), defining_runtime_does_rt(), dump_block_preview(), editor_word_s(), execute_colon_word(), format_word_dump(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_STORE(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TO forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), hash_colon_body(), io_word_type(), memory_word_erase(), memory_word_fill(), memory_word_move(), string_word_convert(), string_word_enclose(), string_word_expect(), string_word_number(), string_word_word(), system_word_reboot(), system_word_see(), vm_load_u8(), vm_store_u8(), and vocabulary_word_paren_find().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.47 vm_push()

```
void vm_push (
            VM * vm,
            cell_t value )
```

Push a value onto the data stack.

Parameters

vm	Pointer to the VM structure
value	Value to push onto the stack

Note

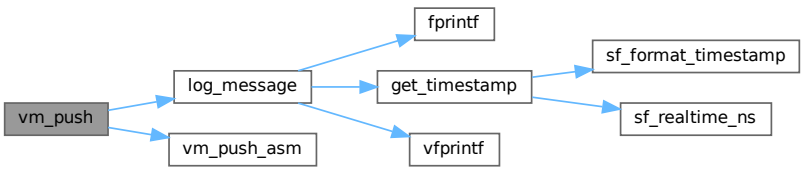
Performs bounds checking to prevent stack overflow

Uses assembly optimization if available for x86_64 or ARM64

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), PROFILE_INC_STACK_OP, STACK_SIZE, and vm_push_asm().

Referenced by acl_recheck(), acl_recheck(), arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), block_word_acl_allow_fetch(), block_word_acl_ttl_fetch(), block_word_blk_attach(), block_word_block(), block_word_buffer(), block_word_owner_fetch(), block_word_scr(), block_word_thru(), control_forth_I(), control_forth_J(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_lit(), defining_runtime_variable(), defining_word_state(), dictionary_m_word_cfa(), dictionary_m_word_find(), dictionary_m_word_link_from(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_tick(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), dictionary_m_word_traverse(), dictionary_word_here(), dictionary_word_latest(), dictionary_word_pad(), dictionary_word_sp_fetch(), double_word_2dup(), double_word_2over(), double_word_2r_fetch(), double_word_2r_from(), double_word_2rot(), double_word_2swap(), double_word_d_equals(), double_word_d_less(), double_word_d_minus(), double_word_d_plus(), double_word_d_two_slash(), double_word_d_two_star(), double_word_d_zero_equals(), double_word_d_zero_less(), double_word_dmax(), double_word_dmin(), double_word_dnegate(), double_word_s_to_d(), editor_word_edit(), emit_cell(), fbw_height(), fbw_width(), format_word_base(), format_word_end_conversion(), format_word_hash(), format_word_hash_s(), format_word_sign(), forth_acl_allow_fetch(), forth_acl_heat_fetch(), forth_acl_mode_fetch(), forth_acl_pinned_query(), forth_acl_ttl_fetch(), forth_acl_word_id(), kbw_debug(), kbw_key_event(), kbw_scan(), kbw_vdebug(), kbw_vevent(), log_word_debug(), log_word_error(), log_word_get_level(), log_word_info(), log_word_test(), log_word_warn(), logical_word_and(), logical_word_equals(), logical_word_false(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_true(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), memory_word_2fetch(), memory_word_cells(), memory_word_cfetch(), memory_word_fetch(), mixed_math_word_m_minus(), mixed_math_word_m_plus(), mixed_math_word_m_slash_mod(), mixed_math_word_m_star(), mixed_math_word_mod(), mixed_math_word_mod(), mixed_math_word_slash_mod(), mixed_math_word_star_slash(), mixed_math_word_star_slash_mod(), return_stack_word_r_fetch(), return_stack_word_r_from(), sk_word_kh_blk_attach_send(), stack_word_pick(), starforth_word_execution_heat_fetch(), starforth_word_heartbeat_ticks(), starforth_word_init(), starforth_word_random(), starforth_word_zuse_cert_installed_query(), starforth_word_zuse_pubkey_fetch(), starforth_word_zuse_session(), string_runtime_s_quote(), string_word_bl(), string_word_bracket_tick(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_minus_trailing(), string_word_number(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), string_word_source(), string_word_span(), string_word_tib(), string_word_to_in(), string_word_word(), system_word_79_standard(), test_acl_adaptive_ttl(), test_acl_inherit_word_dispatch(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), vm_api_push(), vm_compile_literal(), vm_interpret_word(), vocabulary_word_context(), vocabulary_word_current(), vocabulary_word_paren_find(), word_defer_fetch(), word_divide(), and word_divmod().

Here is the call graph for this function:



7.263.4.48 vm_push_fast()

```
static void vm_push_fast (
            VM * vm,
            cell_t value )  [inline], [static]
```

References VM::data_stack, and VM::dsp.

7.263.4.49 vm_repl()

```
void vm_repl (
            VM * vm,
            int script_mode )
```

Starts the Forth REPL (Read-Eval-Print Loop)

Parameters

vm	Pointer to the VM structure
script_mode	If 1, suppress prompts and "ok" output (for piped input)

This function implements an interactive REPL for the Forth interpreter. It continuously reads input from the user, interprets it, and prints the result until the VM is halted or an error occurs.

In interactive mode (script_mode=0):

- Prints colored prompt "ok> "
- Acknowledges successful commands with " ok"
- Reports errors with " ERROR"

In script mode (script_mode=1):

- Suppresses all prompts and status messages
- Suitable for piped input (heredoc, pipes, redirects)

7.263.4.50 vm_rpop()

```
cell_t vm_rpop (
            VM * vm )
```

Pop a value from the return stack.

Parameters

vm	Pointer to the VM structure

Returns

The popped value

Note

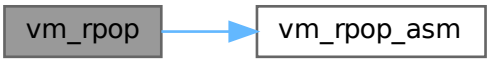
Returns 0 and sets error flag if stack underflow occurs

Uses assembly optimization if available for x86_64 or ARM64

References VM::error, VM::return_stack, VM::rsp, and vm_rpop_asm().

Referenced by execute_colon_word(), return_stack_word_r_from(), and vm_api_rpop().

Here is the call graph for this function:



Here is the caller graph for this function:

7.263.4.51 vm_rpop_fast()

```
static cell_t vm_rpop_fast (
            VM * vm )  [inline], [static]
```

References VM::return_stack, and VM::rsp.

7.263.4.52 vm_rpush()

```
void vm_rpush (
            VM * vm,
            cell_t value )
```

Push a value onto the return stack.

Parameters

vm	Pointer to the VM structure
value	Value to push onto the return stack

Note

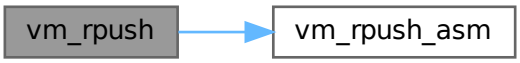
Sets error flag if stack overflow occurs

Uses assembly optimization if available for x86_64 or ARM64

References VM::error, VM::return_stack, VM::rsp, STACK_SIZE, and vm_rpush_asm().

Referenced by execute_colon_word(), return_stack_word_to_r(), and vm_api_rpush().

Here is the call graph for this function:



Here is the caller graph for this function:

7.263.4.53 vm_rpush_fast()

```
static void vm_rpush_fast (
            VM * vm,
            cell_t value )  [inline], [static]
```

References VM::return_stack, and VM::rsp.

7.263.4.54 vm_run_smoke_tests()

```
void vm_run_smoke_tests (
            VM * vm )
```

7.263.4.55 vm_set_base()

```
void vm_set_base (
            VM * vm,
            unsigned b )
```

Set the VM number base and update both the cell and the host mirror.

Clamps `b` to [2, 36] (replacing out-of-range values with 10), then writes to the FORTH `BASE` variable cell at `vm->base_addr` and to the host-mirror field `vm->base` so both sources agree.

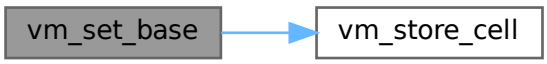
Parameters

vm	VM instance to update.
b	Desired number base; clamped to [2, 36].

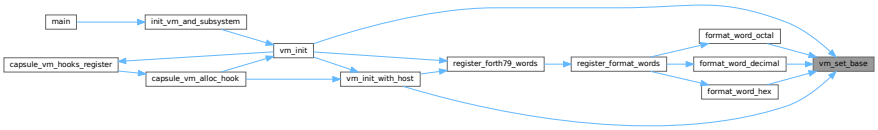
References VM::base, VM::base_addr, and vm_store_cell().

Referenced by format_word_decimal(), format_word_hex(), format_word_octal(), vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



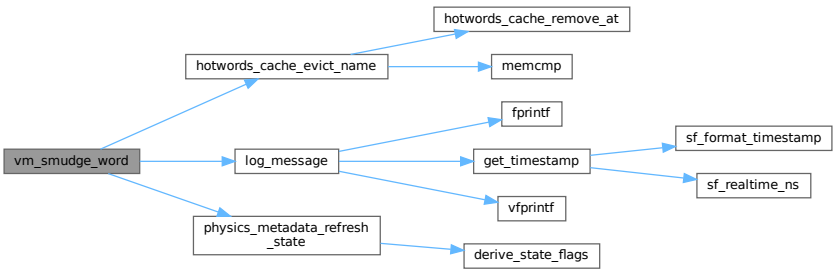
7.263.4.56 vm_smudge_word()

```
void vm_smudge_word (
            VM * vm )
```

References VM::error, DictEntry::flags, VM::hotwords_cache, hotwords_cache_evict_name(), VM::latest, LOG_ERROR, log_message(), DictEntry::name, DictEntry::name_len, physics_metadata_refresh_state(), and WORD_SMUDGED.

Referenced by vm_api_smudge_word().

Here is the call graph for this function:



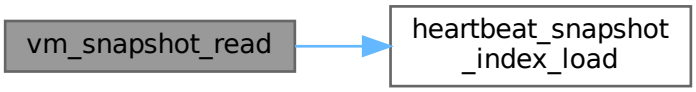
Here is the caller graph for this function:

7.263.4.57 vm_snapshot_read()

```
void vm_snapshot_read (
            const VM * vm,
            HeartbeatSnapshot * out_snapshot )
```

References VM::heartbeat, heartbeat_snapshot_index_load(), HeartbeatState::snapshot_index, and HeartbeatState::snapshots.

Here is the call graph for this function:



7.263.4.58 vm_store_cell()

```
void vm_store_cell (
            struct VM * vm,
            vaddr_t addr,
            cell_t v )
```

Store an aligned cell to VM memory.

Validates alignment and bounds, then copies `v` into VM memory via `memcpy()`. Sets `vm->error` on alignment or bounds failure.

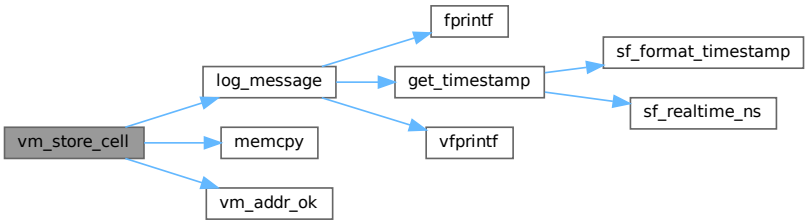
Parameters

vm	VM instance.
addr	Aligned VM byte offset to write.
v	Cell value to store.

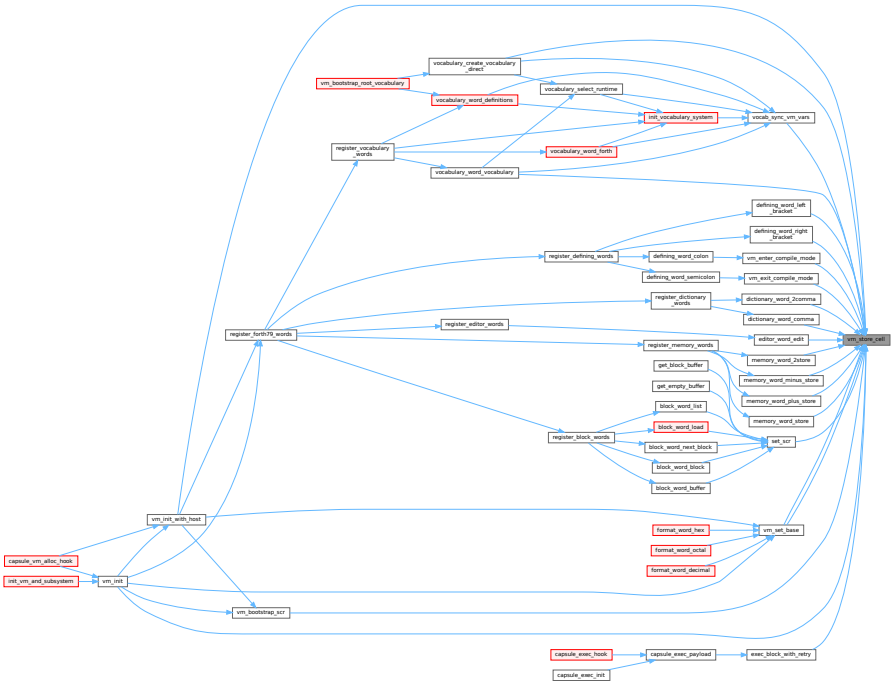
References VM::error, LOG_ERROR, log_message(), memcpy(), VM::memory, and vm_addr_ok().

Referenced by defining_word_left_bracket(), defining_word_right_bracket(), dictionary_word_2comma(), dictionary_word_comma(), editor_word_edit(), exec_block_with_retry(), memory_word_2store(), memory_word_minus_store(), memory_word_plus_store(), memory_word_store(), set_scr(), vm_bootstrap_scr(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_init(), vm_init_with_host(), vm_set_base(), vm_set_base(), vocab_sync_vm_vars(), vocabulary_create_vocabulary_direct(), and vocabulary_word_vocabulary().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.59 vm_store_u8()

```
void vm_store_u8 (
            struct VM * vm,
            vaddr_t addr,
            uint8_t v )
```

Store an unsigned byte to VM memory.

Resolves `addr` via `vm_ptr()`. Sets `vm->error` if the address is out of bounds; otherwise writes `v`.

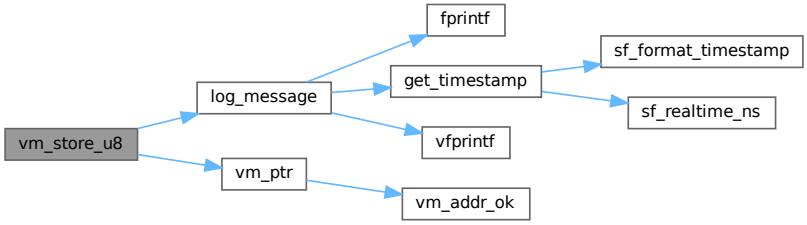
Parameters

vm	VM instance.
addr	VM byte offset to write.
v	Byte value to store.

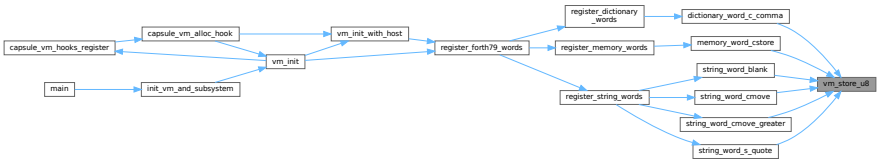
References VM::error, LOG_ERROR, log_message(), and vm_ptr().

Referenced by dictionary_word_c_comma(), memory_word_cstore(), string_word_blank(), string_word_cmove(), string_word_cmove_greater(), and string_word_s_quote().

Here is the call graph for this function:



Here is the caller graph for this function:



7.263.4.60 vm_tick()

```
void vm_tick (
            VM * vm )
```

Central heartbeat dispatcher for time-driven tuning operations.

Aggregates all periodic optimization tasks (Loop #3 and Loop #5) into one place. Designed as plugin architecture - new tuning operations can be added as plugins.

Options for integration:

- Synchronous (now): Called from main execution loop, every N executions
- Background thread (future): Runs in separate thread, decoupled from VM execution

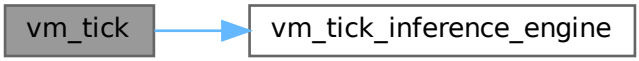
Parameters

vm	Pointer to VM instance

References VM::heartbeat, HeartbeatState::heartbeat_enabled, HEARTBEAT_INFERENCE_FREQUENCY, HeartbeatState::last_inference_tick, HeartbeatState::tick_count, and vm_tick_inference_engine().

Referenced by assert_heartbeat_safe(), and starforth_word_wait().

Here is the call graph for this function:



Here is the caller graph for this function:

7.263.4.61 vm_tick_inference_engine()

```
void vm_tick_inference_engine (
            VM * vm )
```

Phase 2: Unified Inference Engine - Adaptive Window & Decay Slope Tuning.

Coordinates inference on rolling window of truth to determine:

- Optimal adaptive window width (via variance inflection detection)
- Optimal decay slope (via exponential regression on heat trajectory)

Uses ANOVA early-exit to skip full inference when variance is stable (<5% change). All math uses Q48.16 fixed-point (integer-only, no floating-point).

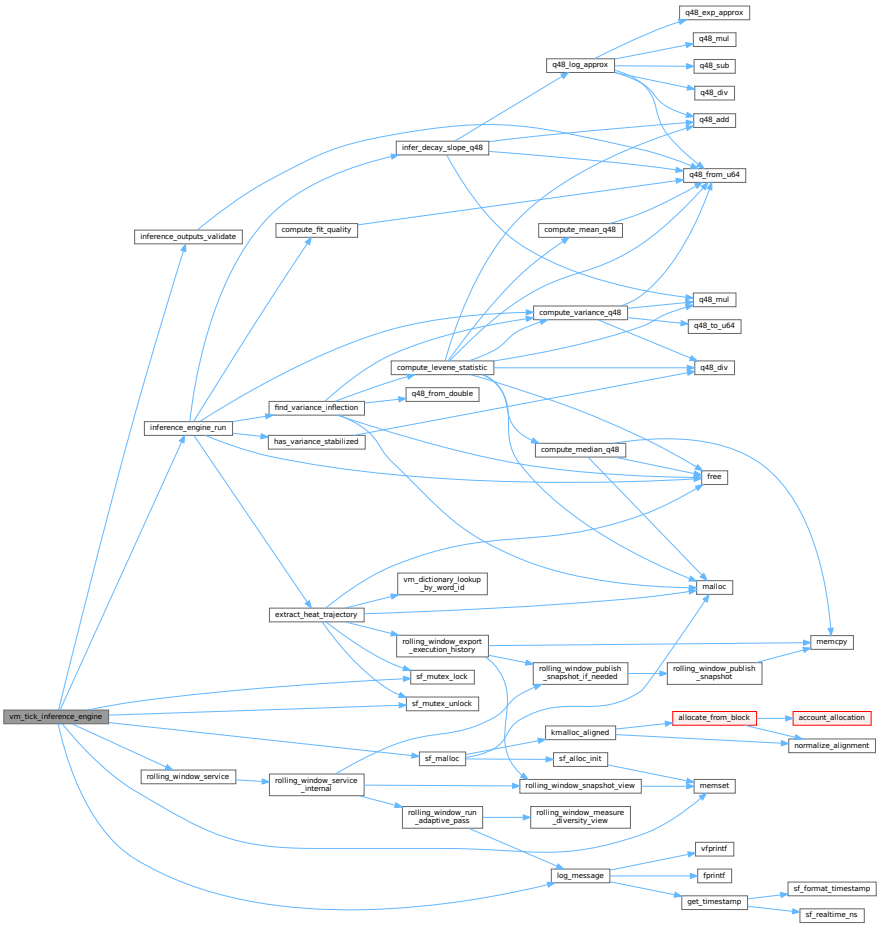
Replaces legacy vm_tick_window_tuner() and vm_tick_slope_validator().

Parameters

vm	Pointer to VM instance

References InferenceOutputs::adaptive_decay_slope, InferenceOutputs::adaptive_window_width, VM::decay_slope_q48, VM::dict_lock, HeartbeatState::early_exit_count, InferenceOutputs::early_exited, RollingWindowOfTruth::effective_window_size, VM::heartbeat, HeartbeatState::heartbeat_enabled, HEARTBEAT_TICK_NS, HOTWORDS_EXECUTION_HEAT_THRESHOLD, inference_engine_run(), inference_outputs_validate(), HeartbeatState::inference_run_count, RollingWindowOfTruth::is_warm, VM::last_inference_outputs, HeartbeatState::last_inference_tick, VM::latest, LOG_DEBUG, LOG_ERROR, LOG_INFO, log_message(), memset(), VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::rolling_window, rolling_window_service(), sf_malloc(), sf_mutex_lock(), sf_mutex_unlock(), InferenceOutputs::slope_fit_quality_q48, VM::stale_word_count_at_check, HeartbeatState::tick_count, HeartbeatWorker::tick_ns, HeartbeatState::tick_target_ns, RollingWindowOfTruth::total_executions, VM::total_heat_at_last_check, VM::tuning_lock, InferenceInputs::window, RollingWindowOfTruth::window_pos, InferenceOutputs::window_variance_q48, word_count, VM::word_count_at_check, and HeartbeatState::worker.

Here is the call graph for this function:



7.263.4.62 vm_tick_slope_validator()

```
void vm_tick_slope_validator (
            VM * vm )
```

Loop #3: Heat decay slope validation via periodic measurement.

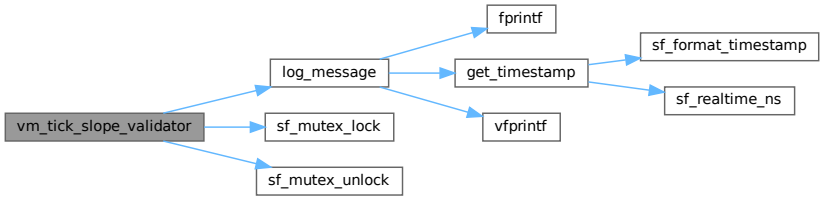
Validates that linear decay is actually helping optimize dictionary caching. Measures stale word ratio, hot word count, and average heat distribution.

Parameters

vm	Pointer to VM instance

References VM::decay_slope_direction, VM::decay_slope_q48, VM::dict_lock, VM::hot_word_count_at_check, HOTWORDS_EXECUTION_HEAT_THRESHOLD, VM::latest, LOG_DEBUG, log_message(), sf_mutex_lock(), sf_mutex_unlock(), VM::stale_word_count_at_check, VM::total_heat_at_last_check, VM::tuning_lock, word_count, and VM::word_count_at_check.

Here is the call graph for this function:



7.263.4.63 vm_tick_window_tuner()

```
void vm_tick_window_tuner (
            VM * vm )
```

Loop #5: Context-aware window tuning via binary chop search.

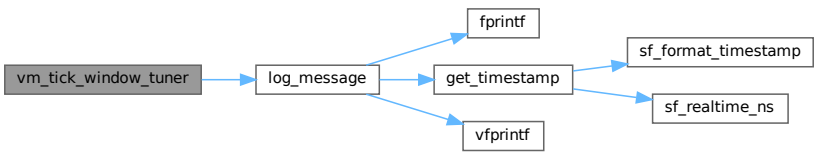
Uses prefetch accuracy to guide window size adaptation. Binary search converges on optimal effective_window_↩ size for current workload.

Parameters

vm	Pointer to VM instance

References ADAPTIVE_MIN_WINDOW_SIZE, RollingWindowOfTruth::effective_window_size, ENABLE_PIPELINING, RollingWindowOfTruth::is_warm, PipelineGlobalMetrics::last_checked_accuracy, PipelineGlobalMetrics::last_checked_window_size, LOG_DEBUG, log_message(), VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::rolling_window, ROLLING_WINDOW_SIZE, and PipelineGlobalMetrics::window_tuning_checks.

Here is the call graph for this function:

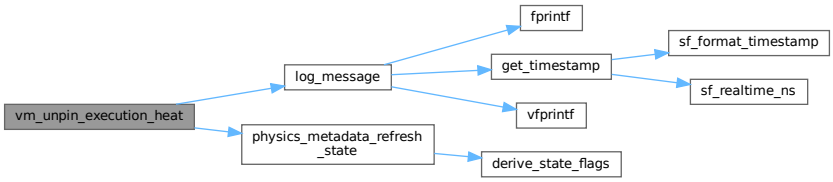


7.263.4.64 vm_unpin_execution_heat()

```
void vm_unpin_execution_heat (
            VM * vm )
```

References VM::error, DictEntry::flags, VM::latest, LOG_ERROR, log_message(), and physics_metadata_refresh_state().

Here is the call graph for this function:



7.263.4.65 vm_zuse_cert_install()

```
int vm_zuse_cert_install (
            VM * vm,
            const uint8_t seed[32],
            const uint8_t pubkey[32] )
```

One-time write of the Zuse cert (seed + derived pubkey) – blows the fuse.

Deliberately not backed by a dictionary CONSTANT: ACL-PIN only blocks redefinition (vm_create_word shadowing), not a >BODY-then-store on the word's data field, so a pinned CONSTANT is not actually immutable against FORTH-level tampering. Keeping the value in a VM struct field with no corresponding FORTH store word closes that path entirely – see FABRIC-2.md's Milestone 4 mint-then-pin writeup for the finding.

Parameters

vm	VM instance.
seed	Ed25519 seed, 32 bytes (the private identity).
pubkey	Ed25519 public key derived from seed, 32 bytes.

Returns

0 on success; -1 if already installed (fuse already blown).

References memcpy(), VM::zuse_cert_installed, VM::zuse_cert_pubkey, and VM::zuse_cert_seed.

Referenced by install_and_activate().

Here is the call graph for this function:

Here is the caller graph for this function:

7.263.5 Variable Documentation

7.263.5.1 __attribute__

```
struct idtr __attribute__
```

7.263.5.2 acl_allow

```
uint8_t acl_allow
```

7.263.5.3 acl_default

```
uint8_t acl_default
```

7.263.5.4 acl_mode

```
uint8_t acl_mode
```

7.263.5.5 acl_pinned

```
uint8_t acl_pinned
```

7.263.5.6 acl_ttl

```
uint32_t acl_ttl
```

7.263.5.7 execution_heat

```
cell_t execution_heat
```

Referenced by forth_ALL_HEATS(), hotwords_bucket_reorder(), physics_metadata_touch(), starforth_word_top_words(), and temperature_from_execution_heat().

7.263.5.8 flags

```
cell_t flags
```

Referenced by blk_flags_set(), capsule_find_mama_init(), contrib_capsule_refused(), decode_composite_glyph(), decode_simple_glyph(), derive_state_flags(), ioapic_init(), ioapic_route_legacy_irq(), make_pte(), vmm_map_page(), and vmm_map_range().

7.263.5.9 func

```
word_func_t func
```

Referenced by register_word(), vm_compile_call(), vm_create_word(), vm_dictionary_find_by_func(), and vm_dictionary_find_latest_by_func().

7.263.5.10 link

```
struct DictEntry* link
```

7.263.5.11 name

```
char name[]
```

Referenced by capsule_birth_baby(), capsule_birth_mama(), capsule_find_by_name(), capsule_parity_log_kill(), capsule_run_experiment(), capsule_vm_find_by_name(), capsule_vm_find_by_name_nocase(), capsule_vm_kill(), capsule_vm_registry_set_name(), cell_prop(), console_set_vm_name(), dict_find_word_heat_aware(), doe_log_heat_by_name(), fdt_find_child(), fdt_find_node_by_name(), fdt_find_prop(), fdt_find_prop_in_node(), fdt_prop_u32(), find_and_cache_pubkey(), forth_FREEZE_CRITICAL(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_STORE(), forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), gpt_write_one(), hosted_lifecycle_birth(), hosted_lifecycle_kill(), hosted_lifecycle_pause(), hosted_lifecycle_resume(), hosted_lifecycle_use(), hotwords_cache_evict_name(), hotwords_cache_lookup(), mb_cell_prop(), node_cells(), register_word(), sig_handler(), vm_bootstrap_root_vocabulary(), vm_create_word(), vm_dict_resolve_in_bucket(), vm_dictionary_get_data_field(), vm_enter_compile_mode(), vm_find_word(), vm_interpret_word(), vm_vocabulary_find_word(), vocab_find_word(), vocabulary_create_vocabulary_direct(), vocabulary_word_paren_find(), vocabulary_word_vocabulary(), word_defer(), word_defer_fetch(), and word_is().

7.263.5.12 name_len

```
uint8_t name_len
```

Referenced by fdt_find_child(), profiler_generate_report(), profiler_print_hotspots(), and traverse_name_field().

7.263.5.13 physics

```
DictPhysics physics
```

7.263.5.14 transition_metrics

```
WordTransitionMetrics* transition_metrics
```

7.263.5.15 word_id

```
uint32_t word_id
```

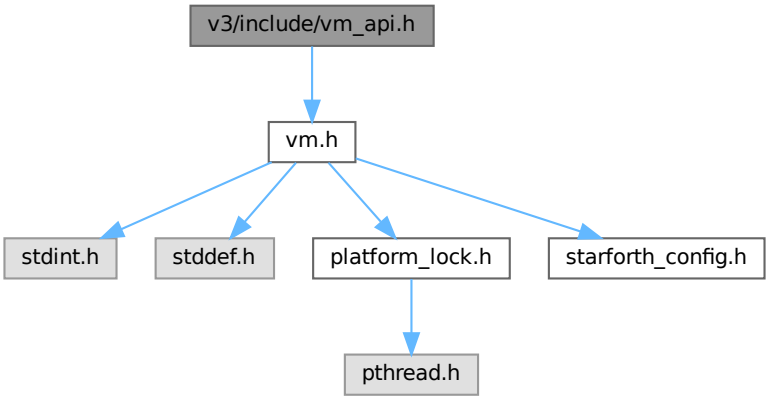
Referenced by execute_colon_word(), extract_heat_trajectory(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), physics_pre_execute(), rolling_window_find_hottest_word(), rolling_window_record_execution(), rolling_window_seed_hotwords_cac transition_metrics_update_context_window(), vm_dictionary_lookup_by_word_id(), vm_dictionary_track_entry(), and vm_dictionary_untrack_entry().

7.264 v3/include/vm_api.h File Reference

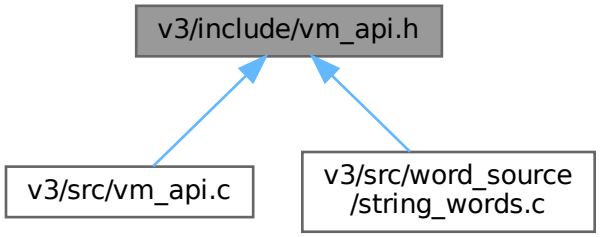
StarForth Virtual Machine Public API.

```
#include "vm.h"
```

Include dependency graph for vm_api.h:



This graph shows which files directly or indirectly include this file:



Functions

- int vm_input_ensure (VM *vm)

  Ensure input buffers are allocated.
- unsigned char * vm_input_tib (VM *vm)

  Get pointer to terminal input buffer (TIB)
- cell_t * vm_input_in (VM *vm)

  Get pointer to input offset variable.
- cell_t * vm_input_span (VM *vm)

  Get pointer to input span variable.
- void vm_input_load_line (VM *vm, const char *src, size_t n)

  Load input line into TIB.
- void vm_input_source (VM *vm, cell_t *out_addr, cell_t *out_len)

  Get current input source buffer info.
- cell_t vm_addr_from_ptr (VM *vm, void *p)

  Convert native pointer to Forth memory address.

7.264.1 Detailed Description

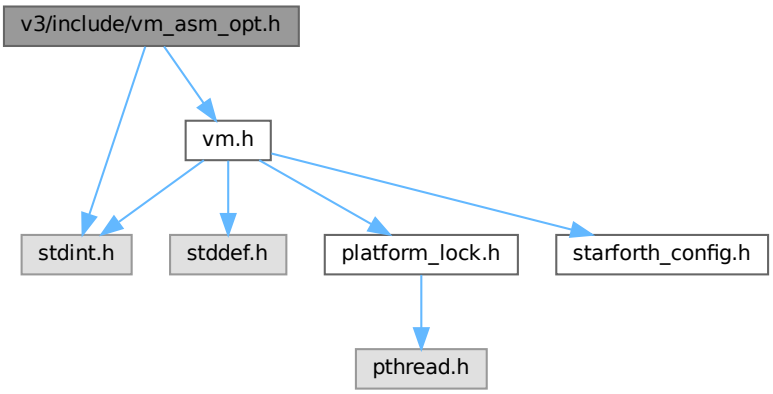
StarForth Virtual Machine Public API.

This header provides the public interface for interacting with the StarForth Virtual Machine, particularly focusing on input handling and memory management.

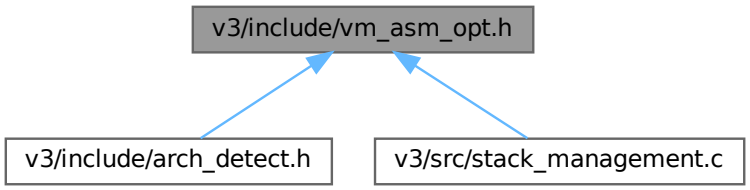
7.265 v3/include/vm_asm_opt.h File Reference

```
#include "vm.h"
#include <stdint.h>
```

Include dependency graph for vm_asm_opt.h:



This graph shows which files directly or indirectly include this file:



Functions

- static void vm_push_asm (VM *vm, cell_t value)

  Optimized data stack push operation.
- static cell_t vm_pop_asm (VM *vm)

  Optimized data stack pop operation.
- static void vm_rpush_asm (VM *vm, cell_t value)

  Optimized return stack push operation.
- static cell_t vm_rpop_asm (VM *vm)

  Optimized return stack pop operation.
- static int vm_add_check_overflow (cell_t a, cell_t b, cell_t *result)

  Performs addition with hardware overflow detection.
- static int vm_sub_check_overflow (cell_t a, cell_t b, cell_t *result)

  Performs subtraction with hardware overflow detection.
- static void vm_mul_double (cell_t a, cell_t b, cell_t *hi, cell_t *lo)

  Performs double-width multiplication.
- static void vm_divmod (cell_t dividend, cell_t divisor, cell_t *quotient, cell_t *remainder)

Performs division with remainder.

- static int vm_strcmp_asm (const char *s1, const char *s2, size_t len)

  Hardware-accelerated string comparison.
- static void vm_memcpy_asm (void *dest, const void *src, size_t len)

  Hardware-accelerated memory copy.
- static void vm_memzero_asm (void *dest, size_t len)

  Hardware-accelerated memory zero.
- static cell_t vm_min_asm (cell_t a, cell_t b)

  Branchless minimum calculation.
- static cell_t vm_max_asm (cell_t a, cell_t b)

  Branchless maximum calculation.
- static void vm_cpuid (uint32_t leaf, uint32_t *eax, uint32_t *ebx, uint32_t *ecx, uint32_t *edx)

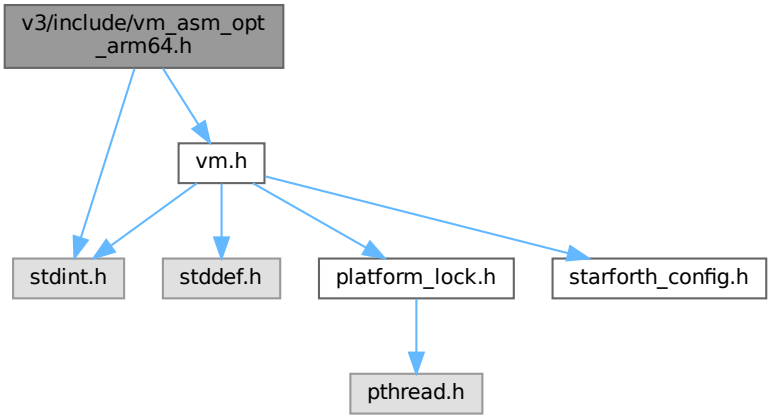
  Wrapper for CPUID instruction.
- static int vm_has_sse42 (void)

  Checks for SSE4.2 support.

7.266 v3/include/vm_asm_opt_arm64.h File Reference

```
#include "vm.h"
#include <stdint.h>
```

Include dependency graph for vm_asm_opt_arm64.h:



Functions

- static void vm_push_asm (VM *vm, cell_t value)

  Optimized assembly implementation of data stack push operation.
- static cell_t vm_pop_asm (VM *vm)

  Optimized assembly implementation of data stack pop operation.
- static void vm_rpush_asm (VM *vm, cell_t value)

  Optimized assembly implementation of return stack push operation.
- static cell_t vm_rpop_asm (VM *vm)

Optimized assembly implementation of return stack pop operation.

- static int vm_add_check_overflow (cell_t a, cell_t b, cell_t *result)

  Fast addition with overflow detection using ARM64 ADDS instruction.
- static int vm_sub_check_overflow (cell_t a, cell_t b, cell_t *result)

  Fast subtraction with overflow detection using ARM64 SUBS instruction.
- static void vm_mul_double (cell_t a, cell_t b, cell_t *hi, cell_t *lo)

  Fast 128-bit multiplication using ARM64 SMULH instruction.
- static void vm_divmod (cell_t dividend, cell_t divisor, cell_t *quotient, cell_t *remainder)

  Fast division with remainder using ARM64 SDIV and MSUB instructions.
- static int vm_strcmp_short_asm (const char *s1, const char *s2, size_t len)

  Optimized string comparison for short strings (<=32 bytes)
- static int vm_strcmp_asm (const char *s1, const char *s2, size_t len)
- static void vm_memcpy_asm (void *dest, const void *src, size_t len)

  Optimized memory copy using ARM64 load/store pair instructions.
- static void vm_memzero_asm (void *dest, size_t len)

  Fast memory zeroing using ARM64 store pair instructions.
- static cell_t vm_min_asm (cell_t a, cell_t b)
- static cell_t vm_max_asm (cell_t a, cell_t b)
- static cell_t vm_abs_asm (cell_t a)
- static int vm_clz (cell_t x)
- static int vm_ctz (cell_t x)
- static int vm_popcnt (cell_t x)
- static uint64_t vm_read_midr (void)
- static int vm_has_neon (void)
- static int vm_is_cortex_a72 (void)
- static void vm_prefetch (const void *addr)
- static void vm_prefetch_stream (const void *addr)
- static void vm_dc_zva (void *addr)

7.266.1 Function Documentation

7.266.1.1 vm_abs_asm()

```
static cell_t vm_abs_asm (
            cell_t a )  [inline], [static]
```

7.266.1.2 vm_add_check_overflow()

```
static int vm_add_check_overflow (
            cell_t a,
            cell_t b,
            cell_t * result )  [inline], [static]
```

Fast addition with overflow detection using ARM64 ADDS instruction.

Parameters

a	First operand
b	Second operand
result	Pointer to store result

Returns

1 if overflow occurred, 0 otherwise

7.266.1.3 vm_clz()

```
static int vm_clz (
            cell_t x )  [inline], [static]
```

7.266.1.4 vm_ctz()

```
static int vm_ctz (
            cell_t x )  [inline], [static]
```

7.266.1.5 vm_dc_zva()

```
static void vm_dc_zva (
            void * addr )  [inline], [static]
```

7.266.1.6 vm_divmod()

```
static void vm_divmod (
            cell_t dividend,
            cell_t divisor,
            cell_t * quotient,
            cell_t * remainder )  [inline], [static]
```

Fast division with remainder using ARM64 SDIV and MSUB instructions.

Parameters

dividend	Dividend value
divisor	Divisor value
quotient	Pointer to store quotient
remainder	Pointer to store remainder

7.266.1.7 vm_has_neon()

```
static int vm_has_neon (
            void  )  [inline], [static]
```

7.266.1.8 vm_is_cortex_a72()

```
static int vm_is_cortex_a72 (
            void  )  [inline], [static]
```

7.266.1.9 vm_max_asm()

```
static cell_t vm_max_asm (
             cell_t a,
             cell_t b )  [inline], [static]
```

7.266.1.10 vm_memcpy_asm()

```
static void vm_memcpy_asm (
             void * dest,
             const void * src,
             size_t len )  [inline], [static]
```

Optimized memory copy using ARM64 load/store pair instructions.

Parameters

dest	Destination buffer
src	Source buffer
len	Number of bytes to copy

7.266.1.11 vm_memzero_asm()

```
static void vm_memzero_asm (
             void * dest,
             size_t len )  [inline], [static]
```

Fast memory zeroing using ARM64 store pair instructions.

Parameters

dest	Buffer to zero
len	Number of bytes to zero

7.266.1.12 vm_min_asm()

```
static cell_t vm_min_asm (
             cell_t a,
             cell_t b )  [inline], [static]
```

7.266.1.13 vm_mul_double()

```
static void vm_mul_double (
             cell_t a,
             cell_t b,
             cell_t * hi,
             cell_t * lo )  [inline], [static]
```

Fast 128-bit multiplication using ARM64 SMULH instruction.

Parameters

a	First operand
b	Second operand
hi	Pointer to store high 64 bits of result
lo	Pointer to store low 64 bits of result

7.266.1.14 vm_pop_asm()

```
static cell_t vm_pop_asm (
            VM * vm )  [inline], [static]
```

Optimized assembly implementation of data stack pop operation.

Parameters

vm	Pointer to VM instance

Returns

Popped value, or 0 on underflow

Note

Sets vm->error on stack underflow

References VM::data_stack, VM::dsp, and VM::error.

7.266.1.15 vm_popcnt()

```
static int vm_popcnt (
            cell_t x )  [inline], [static]
```

7.266.1.16 vm_prefetch()

```
static void vm_prefetch (
            const void * addr )  [inline], [static]
```

7.266.1.17 vm_prefetch_stream()

```
static void vm_prefetch_stream (
            const void * addr )  [inline], [static]
```

7.266.1.18 vm_push_asm()

```
static void vm_push_asm (
            VM * vm,
            cell_t value )  [inline], [static]
```

Optimized assembly implementation of data stack push operation.

Parameters

vm	Pointer to VM instance
value	Value to push onto data stack

Note

Sets vm->error on stack overflow

References VM::data_stack, VM::dsp, and VM::error.

7.266.1.19 vm_read_midr()

```
static uint64_t vm_read_midr (
            void  )  [inline], [static]
```

7.266.1.20 vm_rpop_asm()

```
static cell_t vm_rpop_asm (
            VM * vm )  [inline], [static]
```

Optimized assembly implementation of return stack pop operation.

Parameters

vm	Pointer to VM instance

Returns

Popped value, or 0 on underflow

Note

Sets vm->error on stack underflow

References VM::error, VM::return_stack, and VM::rsp.

7.266.1.21 vm_rpush_asm()

```
static void vm_rpush_asm (
            VM * vm,
            cell_t value )  [inline], [static]
```

Optimized assembly implementation of return stack push operation.

Parameters

vm	Pointer to VM instance
value	Value to push onto return stack

Note

Sets vm->error on stack overflow

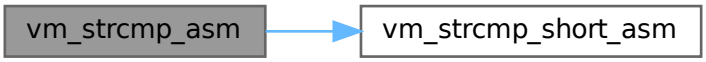
References VM::error, VM::return_stack, and VM::rsp.

7.266.1.22 vm_strcmp_asm()

```
static int vm_strcmp_asm (
            const char * s1,
            const char * s2,
            size_t len )  [inline], [static]
```

References vm_strcmp_short_asm().

Here is the call graph for this function:



7.266.1.23 vm_strcmp_short_asm()

```
static int vm_strcmp_short_asm (
            const char * s1,
            const char * s2,
            size_t len )  [inline], [static]
```

Optimized string comparison for short strings (<=32 bytes)

Parameters

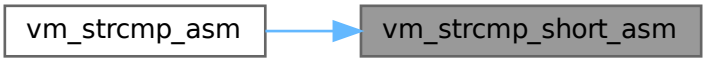
s1	First string
s2	Second string
len	Length to compare

Returns

0 if equal, 1 if different

Referenced by vm_strcmp_asm().

Here is the caller graph for this function:



7.266.1.24 vm_sub_check_overflow()

```
static int vm_sub_check_overflow (
            cell_t a,
            cell_t b,
            cell_t * result )  [inline], [static]
```

Fast subtraction with overflow detection using ARM64 SUBS instruction.

Parameters

a	First operand
b	Second operand
result	Pointer to store result

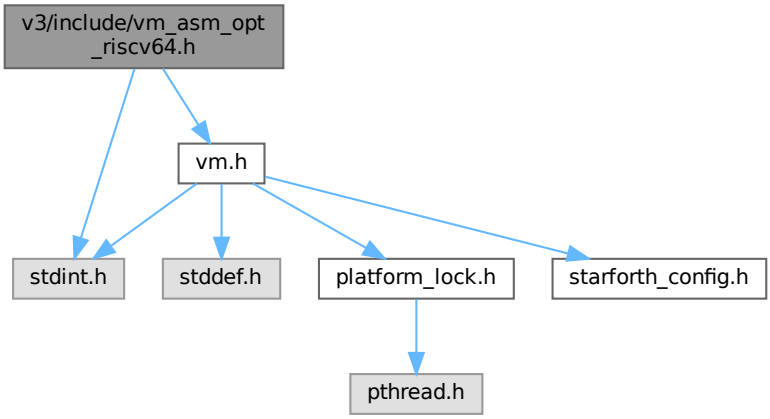
Returns

1 if overflow occurred, 0 otherwise

7.267 v3/include/vm_asm_opt_riscv64.h File Reference

```
#include "vm.h"
#include <stdint.h>
```

Include dependency graph for vm_asm_opt_riscv64.h:



Functions

- static void vm_push_asm (VM *vm, cell_t value)
- static cell_t vm_pop_asm (VM *vm)
- static void vm_rpush_asm (VM *vm, cell_t value)
- static cell_t vm_rpop_asm (VM *vm)
- static int vm_add_check_overflow (cell_t a, cell_t b, cell_t *result)
- static int vm_sub_check_overflow (cell_t a, cell_t b, cell_t *result)
- static void vm_mul_double (cell_t a, cell_t b, cell_t *hi, cell_t *lo)
- static void vm_divmod (cell_t dividend, cell_t divisor, cell_t *quotient, cell_t *remainder)
- static int vm_strcmp_asm (const char *s1, const char *s2, size_t len)
- static void vm_memcpy_asm (void *dest, const void *src, size_t len)
- static void vm_memzero_asm (void *dest, size_t len)
- static cell_t vm_min_asm (cell_t a, cell_t b)
- static cell_t vm_max_asm (cell_t a, cell_t b)
- static cell_t vm_abs_asm (cell_t a)
- static int vm_clz (cell_t x)
- static int vm_ctz (cell_t x)
- static int vm_popcnt (cell_t x)
- static void vm_prefetch (const void *addr)
- static void vm_prefetch_stream (const void *addr)
- static int vm_has_riscv_m_ext (void)
- static int vm_has_riscv_b_ext (void)

7.267.1 Function Documentation

7.267.1.1 vm_abs_asm()

```
static cell_t vm_abs_asm (
            cell_t a )  [inline], [static]
```

7.267.1.2 vm_add_check_overflow()

```
static int vm_add_check_overflow (
            cell_t a,
            cell_t b,
            cell_t * result )  [inline], [static]
```

7.267.1.3 vm_clz()

```
static int vm_clz (
            cell_t x )  [inline], [static]
```

7.267.1.4 vm_ctz()

```
static int vm_ctz (
            cell_t x )  [inline], [static]
```

7.267.1.5 vm_divmod()

```
static void vm_divmod (
            cell_t dividend,
            cell_t divisor,
            cell_t * quotient,
            cell_t * remainder )  [inline], [static]
```

7.267.1.6 vm_has_riscv_b_ext()

```
static int vm_has_riscv_b_ext (
            void  )  [inline], [static]
```

7.267.1.7 vm_has_riscv_m_ext()

```
static int vm_has_riscv_m_ext (
            void  )  [inline], [static]
```

7.267.1.8 vm_max_asm()

```
static cell_t vm_max_asm (
            cell_t a,
            cell_t b )  [inline], [static]
```

7.267.1.9 vm_memcpy_asm()

```
static void vm_memcpy_asm (
            void * dest,
            const void * src,
            size_t len )  [inline], [static]
```

7.267.1.10 vm_memzero_asm()

```
static void vm_memzero_asm (
            void * dest,
            size_t len )  [inline], [static]
```

7.267.1.11 vm_min_asm()

```
static cell_t vm_min_asm (
            cell_t a,
            cell_t b )  [inline], [static]
```

7.267.1.12 vm_mul_double()

```
static void vm_mul_double (
            cell_t a,
            cell_t b,
            cell_t * hi,
            cell_t * lo )  [inline], [static]
```

7.267.1.13 vm_pop_asm()

```
static cell_t vm_pop_asm (
            VM * vm )  [inline], [static]
```

References VM::data_stack, VM::dsp, and VM::error.

7.267.1.14 vm_popcnt()

```
static int vm_popcnt (
            cell_t x )  [inline], [static]
```

7.267.1.15 vm_prefetch()

```
static void vm_prefetch (
            const void * addr )  [inline], [static]
```

7.267.1.16 vm_prefetch_stream()

```
static void vm_prefetch_stream (
            const void * addr )  [inline], [static]
```

7.267.1.17 vm_push_asm()

```
static void vm_push_asm (
            VM * vm,
            cell_t value )  [inline], [static]
```

References VM::data_stack, VM::dsp, and VM::error.

7.267.1.18 vm_rpop_asm()

```
static cell_t vm_rpop_asm (
            VM * vm )  [inline], [static]
```

References VM::error, VM::return_stack, and VM::rsp.

7.267.1.19 vm_rpush_asm()

```
static void vm_rpush_asm (
            VM * vm,
            cell_t value )  [inline], [static]
```

References VM::error, VM::return_stack, and VM::rsp.

7.267.1.20 vm_strcmp_asm()

```
static int vm_strcmp_asm (
            const char * s1,
            const char * s2,
            size_t len )  [inline], [static]
```

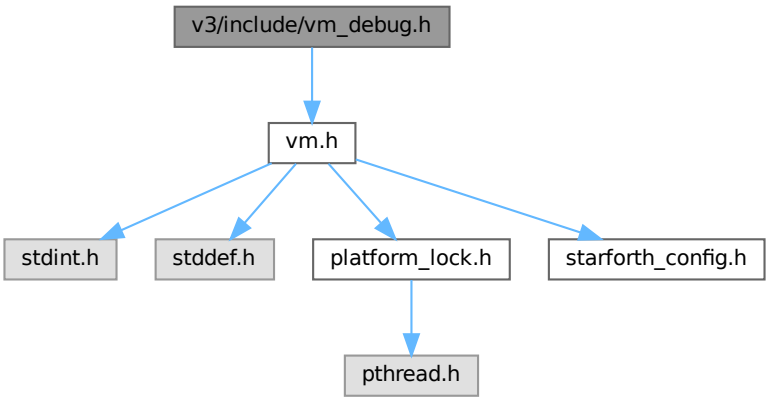
7.267.1.21 vm_sub_check_overflow()

```
static int vm_sub_check_overflow (
            cell_t a,
            cell_t b,
            cell_t * result )  [inline], [static]
```

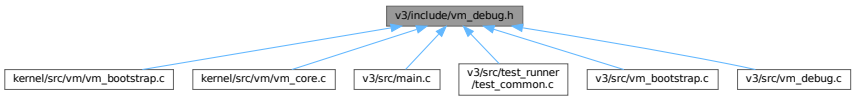
7.268 v3/include/vm_debug.h File Reference

```
#include "vm.h"
```

Include dependency graph for vm_debug.h:



This graph shows which files directly or indirectly include this file:



Functions

- void vm_debug_set_current_vm (VM *vm)

  Sets the VM instance to dump state for when signals occur.
- void vm_debug_install_signal_handlers (void)

  Installs signal handlers for SIGSEGV and SIGABRT.
- void vm_debug_dump_state (const VM *vm, const char *reason)

  Manually dumps the state of a VM instance.

7.268.1 Function Documentation

7.268.1.1 vm_debug_dump_state()

```
void vm_debug_dump_state (
            const VM * vm,
            const char * reason )
```

Manually dumps the state of a VM instance.

Parameters

vm	The VM instance to dump state for
reason	Description of why the dump was triggered

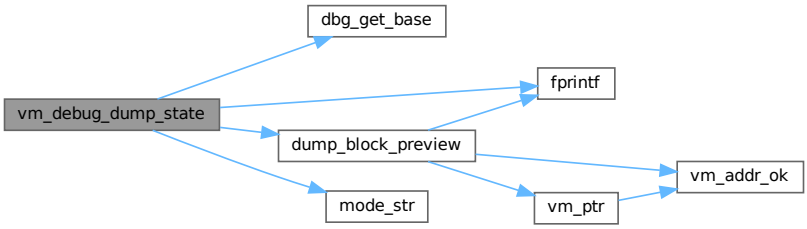
Note

Typically called before exit(1) on test failure

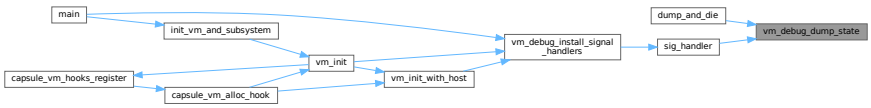
References base, VM::current_executing_entry, dbg_get_base(), VM::dsp, dump_block_preview(), VM::error, fprintf(), VM::here, VM::ip, VM::memory, VM::mode, mode_str(), DictEntry::name, VM::rsp, VM::scr_addr, stderr, and VM_MEMORY_SIZE.

Referenced by dump_and_die(), and sig_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.268.1.2 vm_debug_install_signal_handlers()

```
void vm_debug_install_signal_handlers (
            void  )
```

Installs signal handlers for SIGSEGV and SIGABRT.

The handlers will print VM state before re-raising the signal

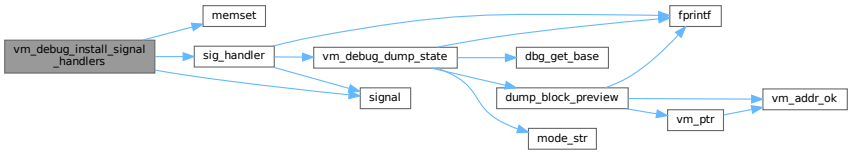
Installs signal handlers for SIGSEGV and SIGABRT.

Sets up handlers for SIGSEGV and SIGABRT to dump VM state on crash

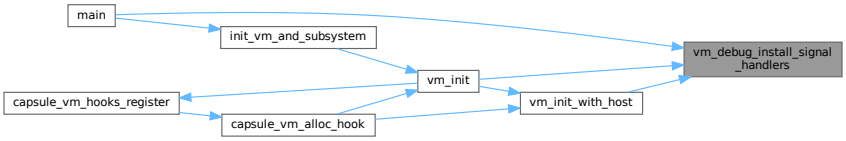
References memset(), sigaction::sa_flags, sigaction::sa_handler, SA_RESETHAND, sig_handler(), SIGABRT, signal(), and SIGSEGV.

Referenced by main(), vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.268.1.3 vm_debug_set_current_vm()

```
void vm_debug_set_current_vm (
            VM * vm )
```

Sets the VM instance to dump state for when signals occur.

Parameters

vm	Pointer to the VM instance to monitor

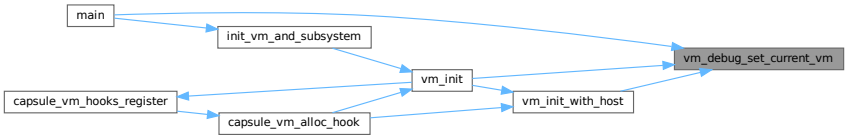
Note

Should be called once after vm_init

References g_vm.

Referenced by main(), vm_init(), and vm_init_with_host().

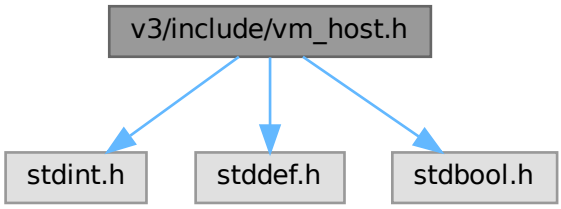
Here is the caller graph for this function:



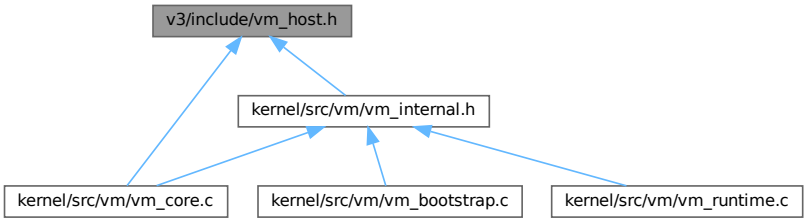
7.269 v3/include/vm_host.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
```

Include dependency graph for vm_host.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct VMHostServices

  VMHostServices - pluggable host operations.

Macros

- #define SK_ALLOC(size) (sk_host_services()->alloc((size), sizeof(void*)))

  Initialize host services for kernel build.
- #define SK_ALLOC_ALIGNED(s, a) (sk_host_services()->alloc((s), (a)))
- #define SK_FREE(ptr) (sk_host_services()->free(ptr))
- #define SK_TIME_NS() (sk_host_services()->monotonic_ns())
- #define SK_PUTS(str) (sk_host_services()->puts(str))
- #define SK_PUTC(c) (sk_host_services()->putc(c))
- #define SK_MUTEX_INIT(m) (sk_host_services()->mutex_init(m))

  Mutex wrapper macros.
- #define SK_MUTEX_LOCK(m) (sk_host_services()->mutex_lock(m))
- #define SK_MUTEX_UNLOCK(m) (sk_host_services()->mutex_unlock(m))
- #define SK_MUTEX_DESTROY(m) (sk_host_services()->mutex_destroy(m))

Typedefs

- typedef void *(* sk_alloc_fn) (size_t size, size_t align)

  Host allocator function signature Returns NULL on failure.
- typedef void(* sk_free_fn) (void *ptr)

  Host free function signature.
- typedef uint64_t(* sk_time_fn) (void)

  Host monotonic time function signature Returns nanoseconds since arbitrary epoch In PARITY_MODE, returns deterministic fake time.
- typedef int(* sk_mutex_init_fn) (void *mutex)

  Host mutex operations (no-op in kernel single-threaded mode)
- typedef int(* sk_mutex_lock_fn) (void *mutex)
- typedef int(* sk_mutex_unlock_fn) (void *mutex)
- typedef void(* sk_mutex_destroy_fn) (void *mutex)
- typedef int(* sk_puts_fn) (const char *str)

  Host console output (for words that print) Returns number of bytes written, or -1 on error.
- typedef int(* sk_putc_fn) (int c)
- typedef bool(* sk_xt_validate_fn) (const void *ptr)

  Host execution-token validation Returns 1 if pointer is executable, 0 otherwise.
- typedef bool(* sk_xt_entry_owns_fn) (const void *entry, size_t bytes)

  Host dictionary entry ownership check.
- typedef void(* sk_panic_fn) (const char *message)

  Host fatal handler (panic)
- typedef struct VMHostServices VMHostServices

  VMHostServices - pluggable host operations.

Functions

- const VMHostServices * sk_host_services (void)

  Get the current host services instance.

7.269.1 Macro Definition Documentation

7.269.1.1 SK_ALLOC

```
#define SK_ALLOC(
              size ) (sk_host_services()->alloc((size), sizeof(void*)))
```

Initialize host services for kernel build.

Must be called before any VM operations. Convenience macros for common operations Use these instead of direct malloc/free/etc.

7.269.1.2 SK_ALLOC_ALIGNED

```
#define SK_ALLOC_ALIGNED(
              s,
              a ) (sk_host_services()->alloc((s), (a)))
```

7.269.1.3 SK_FREE

```
#define SK_FREE(
              ptr ) (sk_host_services()->free(ptr))
```

7.269.1.4 SK_MUTEX_DESTROY

```
#define SK_MUTEX_DESTROY(
              m ) (sk_host_services()->mutex_destroy(m))
```

7.269.1.5 SK_MUTEX_INIT

```
#define SK_MUTEX_INIT(
              m ) (sk_host_services()->mutex_init(m))
```

Mutex wrapper macros.

7.269.1.6 SK_MUTEX_LOCK

```
#define SK_MUTEX_LOCK(
              m ) (sk_host_services()->mutex_lock(m))
```

7.269.1.7 SK_MUTEX_UNLOCK

```
#define SK_MUTEX_UNLOCK(
              m ) (sk_host_services()->mutex_unlock(m))
```

7.269.1.8 SK_PUTC

```
#define SK_PUTC(
              c ) (sk_host_services()->putc(c))
```

7.269.1.9 SK_PUTS

```
#define SK_PUTS(
              str ) (sk_host_services()->puts(str))
```

7.269.1.10 SK_TIME_NS

```
#define SK_TIME_NS( ) (sk_host_services()->monotonic_ns())
```

7.269.2 Typedef Documentation

7.269.2.1 sk_alloc_fn

```
typedef void *(* sk_alloc_fn) (size_t size, size_t align)
```

Host allocator function signature Returns NULL on failure.

7.269.2.2 sk_free_fn

```
typedef void(* sk_free_fn) (void *ptr)
```

Host free function signature.

7.269.2.3 sk_mutex_destroy_fn

```
typedef void(* sk_mutex_destroy_fn) (void *mutex)
```

7.269.2.4 sk_mutex_init_fn

```
typedef int(* sk_mutex_init_fn) (void *mutex)
```

Host mutex operations (no-op in kernel single-threaded mode)

7.269.2.5 sk_mutex_lock_fn

```
typedef int(* sk_mutex_lock_fn) (void *mutex)
```

7.269.2.6 sk_mutex_unlock_fn

```
typedef int(* sk_mutex_unlock_fn) (void *mutex)
```

7.269.2.7 sk_panic_fn

```
typedef void(* sk_panic_fn) (const char *message)
```

Host fatal handler (panic)

7.269.2.8 sk_putc_fn

```
typedef int(* sk_putc_fn) (int c)
```

7.269.2.9 sk_puts_fn

```
typedef int(* sk_puts_fn) (const char *str)
```

Host console output (for words that print) Returns number of bytes written, or -1 on error.

7.269.2.10 sk_time_fn

```
typedef uint64_t(* sk_time_fn) (void)
```

Host monotonic time function signature Returns nanoseconds since arbitrary epoch In PARITY_MODE, returns deterministic fake time.

7.269.2.11 sk_xt_entry_owns_fn

```
typedef bool(* sk_xt_entry_owns_fn) (const void *entry, size_t bytes)
```

Host dictionary entry ownership check.

7.269.2.12 sk_xt_validate_fn

```
typedef bool(* sk_xt_validate_fn) (const void *ptr)
```

Host execution-token validation Returns 1 if pointer is executable, 0 otherwise.

7.269.2.13 VMHostServices

```
typedef struct VMHostServices VMHostServices
```

VMHostServices - pluggable host operations.

Hosted builds point these at libc functions. Kernel builds point these at kernel implementations.

7.269.3 Function Documentation

7.269.3.1 sk_host_services()

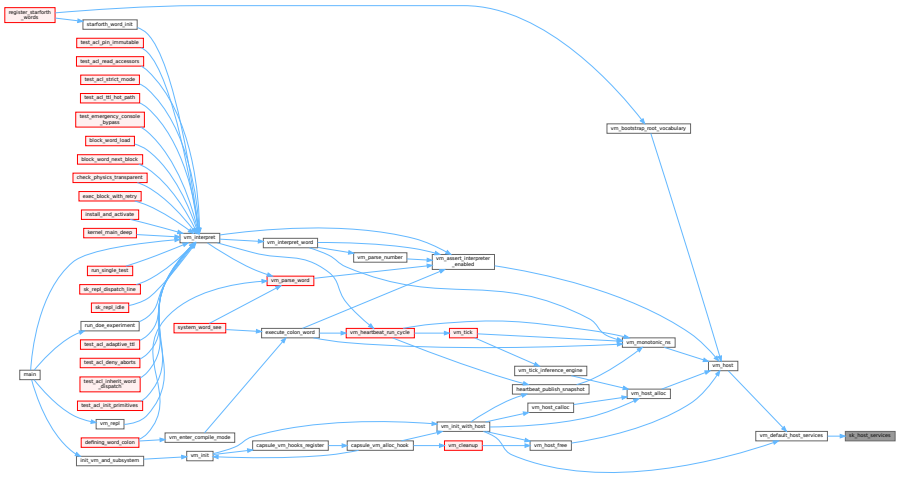
```
const VMHostServices * sk_host_services (
            void  )
```

Get the current host services instance.

Returns hosted or kernel services depending on build.

Referenced by vm_default_host_services().

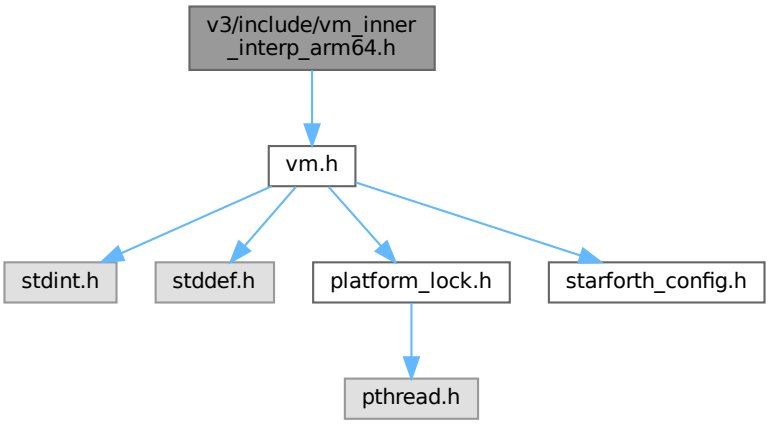
Here is the caller graph for this function:



7.270 v3/include/vm_inner_interp_arm64.h File Reference

```
#include "vm.h"
```

Include dependency graph for vm_inner_interp_arm64.h:



Macros

- #define NEXT_ARM64()
- #define PRIM_DUP_ARM64()

  DUP primitive - duplicates top of stack item.
- #define PRIM_DROP_ARM64()
- #define PRIM_SWAP_ARM64()
- #define PRIM_OVER_ARM64()
- #define PRIM_ROT_ARM64()
- #define PRIM_PLUS_ARM64()
- #define PRIM_MINUS_ARM64()
- #define PRIM_STAR_ARM64()
- #define PRIM_SLASH_ARM64()
- #define PRIM_MOD_ARM64()
- #define PRIM_AND_ARM64()
- #define PRIM_OR_ARM64()
- #define PRIM_XOR_ARM64()
- #define PRIM_INVERT_ARM64()
- #define PRIM_NEGATE_ARM64()
- #define PRIM_FETCH_ARM64()
- #define PRIM_STORE_ARM64()
- #define PRIM_C_FETCH_ARM64()
- #define PRIM_C_STORE_ARM64()
- #define PRIM_TO_R_ARM64()
- #define PRIM_R_FROM_ARM64()
- #define PRIM_R_FETCH_ARM64()
- #define PRIM_2DUP_ARM64()
- #define PRIM_2DROP_ARM64()
- #define PRIM_ZERO_EQUALS_ARM64()
- #define PRIM_ZERO_LESS_ARM64()
- #define PRIM_EQUALS_ARM64()
- #define PRIM_LESS_ARM64()
- #define PRIM_GREATER_ARM64()
- #define PRIM_BRANCH_ARM64()
- #define PRIM_ZBRANCH_ARM64()
- #define PRIM_EXECUTE_ARM64()

Functions

- static void vm_docol_arm64 (VM *vm)
- static void vm_exit_arm64 (void)
- static void vm_setup_registers_arm64 (VM *vm)
- static void vm_save_registers_arm64 (VM *vm)

7.270.1 Macro Definition Documentation

7.270.1.1 NEXT_ARM64

```
#define NEXT_ARM64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ldr     x0, [x20], #8\n\t"       /* Load word ptr from [IP], IP += 8 */ \
        "br      x0\n\t"                  /* Branch to word */ \
        ::: "x0", "memory" \
    )
```

7.270.2 Function Documentation

7.270.2.1 vm_docol_arm64()

```
static void vm_docol_arm64 (
            VM * vm )  [inline], [static]
```

References VM::current_executing_entry, and entry.

7.270.2.2 vm_exit_arm64()

```
static void vm_exit_arm64 (
            void  )  [inline], [static]
```

7.270.2.3 vm_save_registers_arm64()

```
static void vm_save_registers_arm64 (
            VM * vm )  [inline], [static]
```

References VM::data_stack, VM::dsp, VM::ip, VM::return_stack, and VM::rsp.

7.270.2.4 vm_setup_registers_arm64()

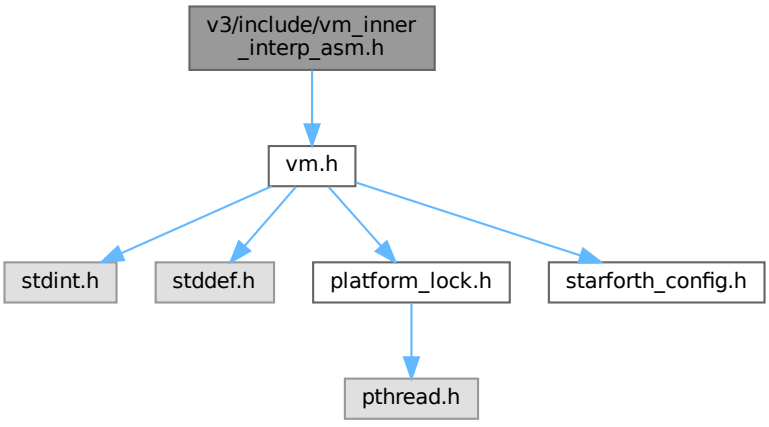
```
static void vm_setup_registers_arm64 (
            VM * vm )  [inline], [static]
```

References VM::data_stack, VM::dsp, VM::ip, VM::return_stack, and VM::rsp.

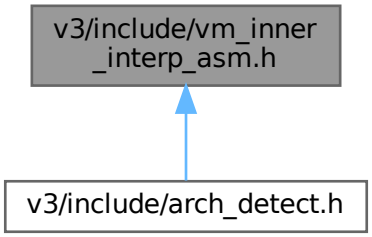
7.271 v3/include/vm_inner_interp_asm.h File Reference

```
#include "vm.h"
```

Include dependency graph for vm_inner_interp_asm.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define NEXT_ASM()

  Register allocation strategy for optimized execution.
- #define PRIM_DUP()
- #define PRIM_DROP()
- #define PRIM_SWAP()
- #define PRIM_PLUS()
- #define PRIM_MINUS()
- #define PRIM_STAR()
- #define PRIM_FETCH()
- #define PRIM_STORE()
- #define PRIM_TO_R()
- #define PRIM_R_FROM()
- #define PRIM_R_FETCH()
- #define PRIM_BRANCH()
- #define PRIM_ZBRANCH()
- #define PRIM_OVER()
- #define PRIM_ROT()
- #define PRIM_SLASH()
- #define PRIM_MOD()
- #define PRIM_AND()
- #define PRIM_OR()
- #define PRIM_XOR()
- #define PRIM_INVERT()
- #define PRIM_NEGATE()
- #define PRIM_C_FETCH()
- #define PRIM_C_STORE()
- #define PRIM_2DUP()
- #define PRIM_2DROP()
- #define PRIM_ZERO_EQUALS()
- #define PRIM_ZERO_LESS()
- #define PRIM_EQUALS()
- #define PRIM_LESS()
- #define PRIM_GREATER()
- #define PRIM_EXECUTE()

Functions

- static void vm_docol_asm (VM *vm)

  Enter a colon definition (DOCOL)
- static void vm_exit_asm (void)

  Return from a colon definition (EXIT)
- static void vm_setup_registers (VM *vm)

  Main interpreter loop implementation.
- static void vm_save_registers (VM *vm)

7.271.1 Macro Definition Documentation

7.271.1.1 NEXT_ASM

```
#define NEXT_ASM( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r13), %%rax\n\t"      /* Load word pointer from [IP] */ \
        "addq    $8, %%r13\n\t"           /* IP++ */ \
        "jmpq    *%%rax\n\t"              /* Jump to word's code */ \
        ::: "rax", "memory" \
    )
```

Register allocation strategy for optimized execution.

Register usage:

- r12 VM pointer (callee-saved, preserved across calls)
- r13 Instruction Pointer (IP) - points to current cell
- r14 Data Stack Pointer (DSP) - points to vm->data_stack[vm->dsp]
- r15 Return Stack Pointer (RSP) - points to vm->return_stack[vm->rsp]

Callee-saved registers are used because:

- They're preserved across function calls
- Allows mixing C and assembly code
- No need to save/restore on each word execution

Core inner interpreter macro - executes next word

Fetches next word pointer from IP, increments IP, and jumps to word

7.271.1.2 PRIM_2DROP

```
#define PRIM_2DROP( )
```

Value:

```
    __asm__ __volatile__ ( \
        "subq    $16, %%r14\n\t"          /* DSP -= 2 */ \
        ::: "r14" \
    )
```

7.271.1.3 PRIM_2DUP

```
#define PRIM_2DUP( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    -8(%%r14), %%rax\n\t"    /* Load NOS (a) */ \
        "movq    (%%r14), %%rcx\n\t"      /* Load TOS (b) */ \
        "movq    %%rax, 8(%%r14)\n\t"     /* Store a above TOS */ \
        "movq    %%rcx, 16(%%r14)\n\t"    /* Store b above that */ \
        "addq    $16, %%r14\n\t"          /* DSP += 2 */ \
        ::: "rax", "rcx", "r14", "memory" \
    )
```

7.271.1.4 PRIM_AND

```
#define PRIM_AND( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "andq    %%rax, -8(%%r14)\n\t"    /* NOS &= TOS */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "r14", "cc", "memory" \
    )
```

7.271.1.5 PRIM_BRANCH

```
#define PRIM_BRANCH( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r13), %%rax\n\t"      /* Load offset from IP */ \
        "addq    %%rax, %%r13\n\t"        /* IP += offset */ \
        ::: "rax", "r13", "memory" \
    )
```

7.271.1.6 PRIM_C_FETCH

```
#define PRIM_C_FETCH( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load address from TOS */ \
        "movq    (%%r12), %%rcx\n\t"      /* Load VM->memory */ \
        "movzbq  (%%rcx, %%rax, 1), %%rax\n\t"  /* Load byte, zero-extend */ \
        "movq    %%rax, (%%r14)\n\t"      /* Store byte value to TOS */ \
        ::: "rax", "rcx", "memory" \
    )
```

7.271.1.7 PRIM_C_STORE

```
#define PRIM_C_STORE( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load address from TOS */ \
        "movq    -8(%%r14), %%rcx\n\t"    /* Load byte value from NOS */ \
        "movq    (%%r12), %%rdx\n\t"      /* Load VM->memory */ \
        "movb    %%cl, (%%rdx, %%rax, 1)\n\t"  /* Store byte */ \
        "subq    $16, %%r14\n\t"          /* DSP -= 2 */ \
        ::: "rax", "rcx", "rdx", "r14", "memory" \
    )
```

7.271.1.8 PRIM_DROP

```
#define PRIM_DROP( )
```

Value:

```
    __asm__ __volatile__ ( \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "r14" \
    )
```

7.271.1.9 PRIM_DUP

```
#define PRIM_DUP( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "addq    $8, %%r14\n\t"           /* DSP++ */ \
        "movq    %%rax, (%%r14)\n\t"      /* Store duplicate */ \
        ::: "rax", "r14", "memory" \
    )
```

7.271.1.10 PRIM_EQUALS

```
#define PRIM_EQUALS( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS (b) */ \
        "cmpq    %%rax, -8(%%r14)\n\t"    /* Compare NOS (a) with TOS (b) */ \
        "sete    %%al\n\t"                 /* AL = 1 if equal */ \
        "movzbq  %%al, %%rax\n\t"         /* Zero-extend */ \
        "negq    %%rax\n\t"               /* -1 if true, 0 if false */ \
        "movq    %%rax, -8(%%r14)\n\t"    /* Store result to NOS slot */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "r14", "cc", "memory" \
    )
```

7.271.1.11 PRIM_EXECUTE

```
#define PRIM_EXECUTE( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load xt from TOS */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        "jmpq    *%%rax\n\t"              /* Jump to execution token */ \
        ::: "rax", "r14", "memory" \
    )
```

7.271.1.12 PRIM_FETCH

```
#define PRIM_FETCH( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load address from TOS */ \
        "movq    (%%r12), %%rcx\n\t"      /* Load VM->memory */ \
        "movq    (%%rcx, %%rax, 1), %%rax\n\t"  /* Load cell from memory */ \
        "movq    %%rax, (%%r14)\n\t"      /* Store to TOS */ \
        ::: "rax", "rcx", "memory" \
    )
```

7.271.1.13 PRIM_GREATER

```
#define PRIM_GREATER( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rcx\n\t"      /* Load TOS (b) */ \
        "movq    -8(%%r14), %%rax\n\t"    /* Load NOS (a) */ \
        "cmpq    %%rcx, %%rax\n\t"        /* rax - rcx: signed compare a vs b */ \
        "setg    %%al\n\t"                 /* AL = 1 if a > b */ \
        "movzbq  %%al, %%rax\n\t"         /* Zero-extend */ \
        "negq    %%rax\n\t"               /* -1 if true, 0 if false */ \
        "movq    %%rax, -8(%%r14)\n\t"    /* Store result to NOS slot */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "rcx", "r14", "cc", "memory" \
    )
```

7.271.1.14 PRIM_INVERT

```
#define PRIM_INVERT( )
```

Value:

```
    __asm__ __volatile__ ( \
        "notq    (%%r14)\n\t"             /* TOS = ~TOS */ \
        ::: "memory" \
    )
```

7.271.1.15 PRIM_LESS

```
#define PRIM_LESS( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rcx\n\t"      /* Load TOS (b) */ \
        "movq    -8(%%r14), %%rax\n\t"    /* Load NOS (a) */ \
        "cmpq    %%rcx, %%rax\n\t"        /* rax - rcx: signed compare a vs b */ \
        "setl    %%al\n\t"                 /* AL = 1 if a < b */ \
        "movzbq  %%al, %%rax\n\t"         /* Zero-extend */ \
        "negq    %%rax\n\t"               /* -1 if true, 0 if false */ \
        "movq    %%rax, -8(%%r14)\n\t"    /* Store result to NOS slot */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "rcx", "r14", "cc", "memory" \
    )
```

7.271.1.16 PRIM_MINUS

```
#define PRIM_MINUS( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "subq    %%rax, -8(%%r14)\n\t"    /* Subtract from NOS */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "r14", "cc", "memory" \
    )
```

7.271.1.17 PRIM_MOD

```
#define PRIM_MOD( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    -8(%%r14), %%rax\n\t"    /* Load dividend (NOS) */ \
        "cqo\n\t"                          /* Sign-extend */ \
        "idivq   (%%r14)\n\t"              /* rdx = remainder */ \
        "movq    %%rdx, -8(%%r14)\n\t"    /* Store remainder to NOS slot */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "rdx", "r14", "cc", "memory" \
    )
```

7.271.1.18 PRIM_NEGATE

```
#define PRIM_NEGATE( )
```

Value:

```
    __asm__ __volatile__ ( \
        "negq    (%%r14)\n\t"            /* TOS = -TOS */ \
        ::: "cc", "memory" \
    )
```

7.271.1.19 PRIM_OR

```
#define PRIM_OR( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "orq     %%rax, -8(%%r14)\n\t"    /* NOS |= TOS */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "r14", "cc", "memory" \
    )
```

7.271.1.20 PRIM_OVER

```
#define PRIM_OVER( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    -8(%%r14), %%rax\n\t"    /* Load NOS */ \
        "addq    $8, %%r14\n\t"           /* DSP++ */ \
        "movq    %%rax, (%%r14)\n\t"      /* Store NOS as new TOS */ \
        ::: "rax", "r14", "memory" \
    )
```

7.271.1.21 PRIM_PLUS

```
#define PRIM_PLUS( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "addq    %%rax, -8(%%r14)\n\t"    /* Add to NOS */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "r14", "cc", "memory" \
    )
```

7.271.1.22 PRIM_R_FETCH

```
#define PRIM_R_FETCH( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    -8(%%r15), %%rax\n\t"    /* Load from top of return stack */ \
        "addq    $8, %%r14\n\t"           /* DSP++ */ \
        "movq    %%rax, (%%r14)\n\t"      /* Store to TOS */ \
        ::: "rax", "r14", "memory" \
    )
```

7.271.1.23 PRIM_R_FROM

```
#define PRIM_R_FROM( )
```

Value:

```
    __asm__ __volatile__ ( \
        "subq    $8, %%r15\n\t"           /* RSP-- */ \
        "movq    (%%r15), %%rax\n\t"      /* Load from return stack */ \
        "addq    $8, %%r14\n\t"           /* DSP++ */ \
        "movq    %%rax, (%%r14)\n\t"      /* Store to TOS */ \
        ::: "rax", "r14", "r15", "memory" \
    )
```

7.271.1.24 PRIM_ROT

```
#define PRIM_ROT( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* c = TOS */ \
        "movq    -8(%%r14), %%rcx\n\t"    /* b = NOS */ \
        "movq    -16(%%r14), %%rdx\n\t"   /* a = third */ \
        "movq    %%rdx, -8(%%r14)\n\t"    /* a -> NOS */ \
        "movq    %%rax, -16(%%r14)\n\t"   /* c -> third */ \
        "movq    %%rcx, (%%r14)\n\t"      /* b -> TOS */ \
        ::: "rax", "rcx", "rdx", "memory" \
    )
```

7.271.1.25 PRIM_SLASH

```
#define PRIM_SLASH( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    -8(%%r14), %%rax\n\t"    /* Load dividend (NOS) */ \
        "cqo\n\t"                          /* Sign-extend rax to rdx:rax */ \
        "idivq   (%%r14)\n\t"              /* Divide by TOS */ \
        "movq    %%rax, -8(%%r14)\n\t"    /* Store quotient to NOS slot */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "rdx", "r14", "cc", "memory" \
    )
```

7.271.1.26 PRIM_STAR

```
#define PRIM_STAR( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "imulq   -8(%%r14), %%rax\n\t"    /* Multiply with NOS */ \
        "movq    %%rax, -8(%%r14)\n\t"    /* Store result */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "rdx", "r14", "cc", "memory" \
    )
```

7.271.1.27 PRIM_STORE

```
#define PRIM_STORE( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load address from TOS */ \
        "movq    -8(%%r14), %%rcx\n\t"    /* Load value from NOS */ \
        "movq    (%%r12), %%rdx\n\t"      /* Load VM->memory */ \
        "movq    %%rcx, (%%rdx, %%rax, 1)\n\t"  /* Store to memory */ \
        "subq    $16, %%r14\n\t"          /* DSP -= 2 */ \
        ::: "rax", "rcx", "rdx", "r14", "memory" \
    )
```

7.271.1.28 PRIM_SWAP

```
#define PRIM_SWAP( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "movq    -8(%%r14), %%rcx\n\t"    /* Load NOS */ \
        "movq    %%rcx, (%%r14)\n\t"      /* Store NOS to TOS */ \
        "movq    %%rax, -8(%%r14)\n\t"    /* Store TOS to NOS */ \
        ::: "rax", "rcx", "memory" \
    )
```

7.271.1.29 PRIM_TO_R

```
#define PRIM_TO_R( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load from TOS */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        "movq    %%rax, (%%r15)\n\t"      /* Store to return stack */ \
        "addq    $8, %%r15\n\t"           /* RSP++ */ \
        ::: "rax", "r14", "r15", "memory" \
    )
```

7.271.1.30 PRIM_XOR

```
#define PRIM_XOR( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "xorq    %%rax, -8(%%r14)\n\t"    /* NOS ^= TOS */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        ::: "rax", "r14", "cc", "memory" \
    )
```

7.271.1.31 PRIM_ZBRANCH

```
#define PRIM_ZBRANCH( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "subq    $8, %%r14\n\t"           /* DSP-- */ \
        "testq   %%rax, %%rax\n\t"        /* Test if zero */ \
        "jnz     1f\n\t"                  /* Skip if non-zero */ \
        "movq    (%%r13), %%rax\n\t"      /* Load offset */ \
        "addq    %%rax, %%r13\n\t"        /* IP += offset */ \
        "jmp     2f\n\t" \
        "1:\n\t" \
        "addq    $8, %%r13\n\t"           /* Skip offset */ \
        "2:\n\t" \
        ::: "rax", "r13", "r14", "cc", "memory" \
    )
```

7.271.1.32 PRIM_ZERO_EQUALS

```
#define PRIM_ZERO_EQUALS( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "testq   %%rax, %%rax\n\t"        /* Test for zero */ \
        "sete    %%al\n\t"                 /* AL = 1 if zero */ \
        "movzbq  %%al, %%rax\n\t"         /* Zero-extend */ \
        "negq    %%rax\n\t"               /* -1 if true, 0 if false */ \
        "movq    %%rax, (%%r14)\n\t"      /* Store result */ \
        ::: "rax", "cc", "memory" \
    )
```

7.271.1.33 PRIM_ZERO_LESS

```
#define PRIM_ZERO_LESS( )
```

Value:

```
    __asm__ __volatile__ ( \
        "movq    (%%r14), %%rax\n\t"      /* Load TOS */ \
        "sarq    $63, %%rax\n\t"          /* Arithmetic shift: -1 if negative, 0 otherwise */ \
        "movq    %%rax, (%%r14)\n\t"      /* Store result */ \
        ::: "rax", "cc", "memory" \
    )
```

7.271.2 Function Documentation

7.271.2.1 vm_docol_asm()

```
static void vm_docol_asm (
            VM * vm )  [inline], [static]
```

Enter a colon definition (DOCOL)

Parameters

vm	Pointer to VM instance

Called when executing a user-defined word

References VM::current_executing_entry, and entry.

7.271.2.2 vm_exit_asm()

```
static void vm_exit_asm (
            void  )  [inline], [static]
```

Return from a colon definition (EXIT)

Pops IP from return stack and continues execution

7.271.2.3 vm_save_registers()

```
static void vm_save_registers (
            VM * vm )  [inline], [static]
```

References VM::data_stack, VM::dsp, VM::ip, VM::return_stack, and VM::rsp.

7.271.2.4 vm_setup_registers()

```
static void vm_setup_registers (
            VM * vm )  [inline], [static]
```

Main interpreter loop implementation.

Replaces execute_colon_word function with optimized assembly

Usage example:

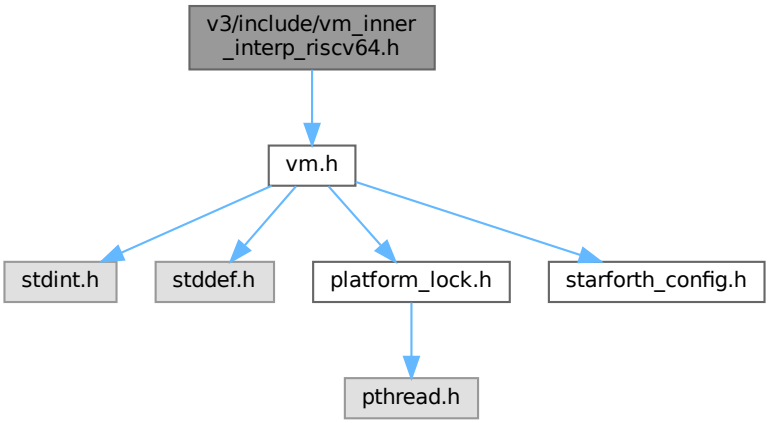
```
vm_setup_registers(vm);
VM_INNER_LOOP();
```

References VM::data_stack, VM::dsp, VM::ip, VM::return_stack, and VM::rsp.

7.272 v3/include/vm_inner_interp_riscv64.h File Reference

```
#include "vm.h"
```

Include dependency graph for vm_inner_interp_riscv64.h:



Macros

- #define NEXT_RV64()
- #define PRIM_DUP_RV64()
- #define PRIM_DROP_RV64()
- #define PRIM_SWAP_RV64()
- #define PRIM_OVER_RV64()
- #define PRIM_ROT_RV64()
- #define PRIM_PLUS_RV64()
- #define PRIM_MINUS_RV64()
- #define PRIM_STAR_RV64()
- #define PRIM_SLASH_RV64()
- #define PRIM_MOD_RV64()
- #define PRIM_AND_RV64()
- #define PRIM_OR_RV64()
- #define PRIM_XOR_RV64()
- #define PRIM_INVERT_RV64()
- #define PRIM_NEGATE_RV64()
- #define PRIM_FETCH_RV64()
- #define PRIM_STORE_RV64()
- #define PRIM_C_FETCH_RV64()
- #define PRIM_C_STORE_RV64()
- #define PRIM_TO_R_RV64()
- #define PRIM_R_FROM_RV64()
- #define PRIM_R_FETCH_RV64()
- #define PRIM_2DUP_RV64()
- #define PRIM_2DROP_RV64()
- #define PRIM_ZERO_EQUALS_RV64()
- #define PRIM_ZERO_LESS_RV64()

- #define PRIM_EQUALS_RV64()
- #define PRIM_LESS_RV64()
- #define PRIM_GREATER_RV64()
- #define PRIM_BRANCH_RV64()
- #define PRIM_ZBRANCH_RV64()
- #define PRIM_EXECUTE_RV64()

Functions

- static void vm_docol_rv64 (VM *vm)
- static void vm_exit_rv64 (void)
- static void vm_setup_registers_rv64 (VM *vm)
- static void vm_save_registers_rv64 (VM *vm)

7.272.1 Macro Definition Documentation

7.272.1.1 NEXT_RV64

```
#define NEXT_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s3)\n\t"    /* a0 = *IP                */ \
        "addi    s3, s3, 8\n\t"    /* IP++                    */ \
        "jr      a0\n\t"           /* jump to code field      */ \
        ::: "a0", "memory" \
    )
```

7.272.1.2 PRIM_2DROP_RV64

```
#define PRIM_2DROP_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      s6, -8(s4)\n\t"   /* new TOS = item3         */ \
        "addi    s4, s4, -16\n\t" \
        ::: "s4", "s6", "memory" \
    )
```

7.272.1.3 PRIM_2DUP_RV64

```
#define PRIM_2DUP_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = NOS                */ \
        "addi    s4, s4, 8\n\t" \
        "sd      a0, 0(s4)\n\t"    /* push NOS                */ \
        "addi    s4, s4, 8\n\t" \
        "sd      s6, 0(s4)\n\t"    /* push TOS copy           */ \
        ::: "a0", "s4", "memory" \
    )
```

7.272.1.4 PRIM_AND_RV64

```
#define PRIM_AND_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        "and     s6, s6, a0\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.5 PRIM_BRANCH_RV64

```
#define PRIM_BRANCH_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s3)\n\t"    /* offset                  */ \
        "add     s3, s3, a0\n\t"   /* IP += offset            */ \
        ::: "a0", "s3", "memory" \
    )
```

7.272.1.6 PRIM_C_FETCH_RV64

```
#define PRIM_C_FETCH_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s2)\n\t" \
        "add     a0, a0, s6\n\t" \
        "lbu     s6, 0(a0)\n\t"    /* zero-extend byte        */ \
        ::: "a0", "s6", "memory" \
    )
```

7.272.1.7 PRIM_C_STORE_RV64

```
#define PRIM_C_STORE_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = addr               */ \
        "addi    s4, s4, -8\n\t" \
        "ld      a1, 0(s2)\n\t" \
        "add     a1, a1, a0\n\t" \
        "sb      s6, 0(a1)\n\t" \
        "ld      s6, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        ::: "a0", "a1", "s4", "s6", "memory" \
    )
```

7.272.1.8 PRIM_DROP_RV64

```
#define PRIM_DROP_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      s6, -8(s4)\n\t"   /* new TOS = NOS           */ \
        "addi    s4, s4, -8\n\t"   /* DSP--                   */ \
        ::: "s4", "s6", "memory" \
    )
```

7.272.1.9 PRIM_DUP_RV64

```
#define PRIM_DUP_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "addi    s4, s4, 8\n\t"    /* DSP++                    */ \
        "sd      s6, 0(s4)\n\t"    /* push TOS copy            */ \
        ::: "s4", "memory" \
    )
```

7.272.1.10 PRIM_EQUALS_RV64

```
#define PRIM_EQUALS_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        "sub     s6, a0, s6\n\t" \
        "seqz    s6, s6\n\t" \
        "neg     s6, s6\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.11 PRIM_EXECUTE_RV64

```
#define PRIM_EXECUTE_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "mv      a0, s6\n\t"       /* a0 = xt                  */ \
        "ld      s6, 0(s4)\n\t"    /* new TOS                  */ \
        "addi    s4, s4, -8\n\t" \
        "jr      a0\n\t"           /* jump to xt               */ \
        ::: "a0", "s4", "s6", "memory" \
    )
```

7.272.1.12 PRIM_FETCH_RV64

```
#define PRIM_FETCH_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s2)\n\t"    /* a0 = vm->memory          */ \
        "add     a0, a0, s6\n\t"   /* a0 = &memory[TOS]        */ \
        "ld      s6, 0(a0)\n\t"    /* TOS = memory[TOS]        */ \
        ::: "a0", "s6", "memory" \
    )
```

7.272.1.13 PRIM_GREATER_RV64

```
#define PRIM_GREATER_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        "slt     s6, s6, a0\n\t"   /* 1 if TOS < NOS, i.e. NOS > TOS */ \
        "neg     s6, s6\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.14 PRIM_INVERT_RV64

```
#define PRIM_INVERT_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "not     s6, s6\n\t" \
        ::: "s6" \
    )
```

7.272.1.15 PRIM_LESS_RV64

```
#define PRIM_LESS_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = NOS                */ \
        "addi    s4, s4, -8\n\t" \
        "slt     s6, a0, s6\n\t"   /* 1 if NOS < TOS          */ \
        "neg     s6, s6\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.16 PRIM_MINUS_RV64

```
#define PRIM_MINUS_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = NOS                */ \
        "addi    s4, s4, -8\n\t" \
        "sub     s6, a0, s6\n\t"   /* TOS = NOS - TOS         */ \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.17 PRIM_MOD_RV64

```
#define PRIM_MOD_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        "rem     s6, a0, s6\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.18 PRIM_NEGATE_RV64

```
#define PRIM_NEGATE_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "neg     s6, s6\n\t" \
        ::: "s6" \
    )
```

7.272.1.19 PRIM_OR_RV64

```
#define PRIM_OR_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        "or      s6, s6, a0\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.20 PRIM_OVER_RV64

```
#define PRIM_OVER_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = NOS               */ \
        "addi    s4, s4, 8\n\t"    /* DSP++                  */ \
        "sd      s6, 0(s4)\n\t"    /* push old TOS           */ \
        "mv      s6, a0\n\t"       /* TOS = NOS              */ \
        ::: "a0", "s4", "s6", "memory" \
    )
```

7.272.1.21 PRIM_PLUS_RV64

```
#define PRIM_PLUS_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        "add     s6, s6, a0\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.22 PRIM_R_FETCH_RV64

```
#define PRIM_R_FETCH_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "addi    s4, s4, 8\n\t" \
        "sd      s6, 0(s4)\n\t" \
        "ld      s6, 0(s5)\n\t"    /* TOS = RS top (no pop)   */ \
        ::: "s4", "s6", "memory" \
    )
```

7.272.1.23 PRIM_R_FROM_RV64

```
#define PRIM_R_FROM_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "addi    s4, s4, 8\n\t" \
        "sd      s6, 0(s4)\n\t"    /* push old TOS           */ \
        "ld      s6, 0(s5)\n\t"    /* TOS = RS top           */ \
        "addi    s5, s5, -8\n\t" \
        ::: "s4", "s5", "s6", "memory" \
    )
```

7.272.1.24 PRIM_ROT_RV64

```
#define PRIM_ROT_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = item2 (NOS)         */ \
        "ld      a1, -8(s4)\n\t"   /* a1 = item3               */ \
        "sd      a1, 0(s4)\n\t"    /* item3 -> NOS position    */ \
        "sd      s6, -8(s4)\n\t"   /* TOS -> item3 position    */ \
        "mv      s6, a0\n\t"       /* item2 -> TOS             */ \
        ::: "a0", "a1", "s6", "memory" \
    )
```

7.272.1.25 PRIM_SLASH_RV64

```
#define PRIM_SLASH_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = NOS (dividend)      */ \
        "addi    s4, s4, -8\n\t" \
        "div     s6, a0, s6\n\t"   /* TOS = NOS / TOS          */ \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.26 PRIM_STAR_RV64

```
#define PRIM_STAR_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        "mul     s6, s6, a0\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.27 PRIM_STORE_RV64

```
#define PRIM_STORE_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = addr (NOS)          */ \
        "addi    s4, s4, -8\n\t" \
        "ld      a1, 0(s2)\n\t"    /* a1 = vm->memory          */ \
        "add     a1, a1, a0\n\t" \
        "sd      s6, 0(a1)\n\t"    /* memory[addr] = TOS       */ \
        "ld      s6, 0(s4)\n\t"    /* load new TOS             */ \
        "addi    s4, s4, -8\n\t" \
        ::: "a0", "a1", "s4", "s6", "memory" \
    )
```

7.272.1.28 PRIM_SWAP_RV64

```
#define PRIM_SWAP_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t"    /* a0 = NOS                 */ \
        "sd      s6, 0(s4)\n\t"    /* NOS = old TOS            */ \
        "mv      s6, a0\n\t"       /* TOS = old NOS            */ \
        ::: "a0", "s6", "memory" \
    )
```

7.272.1.29 PRIM_TO_R_RV64

```
#define PRIM_TO_R_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "addi    s5, s5, 8\n\t" \
        "sd      s6, 0(s5)\n\t"    /* RS[top] = TOS           */ \
        "ld      s6, 0(s4)\n\t"    /* new TOS = NOS           */ \
        "addi    s4, s4, -8\n\t" \
        ::: "s4", "s5", "s6", "memory" \
    )
```

7.272.1.30 PRIM_XOR_RV64

```
#define PRIM_XOR_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s4)\n\t" \
        "addi    s4, s4, -8\n\t" \
        "xor     s6, s6, a0\n\t" \
        ::: "a0", "s4", "s6" \
    )
```

7.272.1.31 PRIM_ZBRANCH_RV64

```
#define PRIM_ZBRANCH_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "ld      a0, 0(s3)\n\t"    /* offset                  */ \
        "ld      s6, 0(s4)\n\t"    /* new TOS                 */ \
        "addi    s4, s4, -8\n\t" \
        /* if old TOS was zero: IP += offset; else IP += 8 */ \
        "bnez    s6, 1f\n\t"       /* skip branch if nonzero  */ \
        "add     s3, s3, a0\n\t" \
        "j       2f\n\t" \
        "1:\n\t" \
        "addi    s3, s3, 8\n\t" \
        "2:\n\t" \
        ::: "a0", "s3", "s4", "s6", "memory" \
    )
```

7.272.1.32 PRIM_ZERO_EQUALS_RV64

```
#define PRIM_ZERO_EQUALS_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "seqz    s6, s6\n\t"       /* 1 if zero, else 0       */ \
        "neg     s6, s6\n\t"       /* convert to -1 (true)    */ \
        ::: "s6" \
    )
```

7.272.1.33 PRIM_ZERO_LESS_RV64

```
#define PRIM_ZERO_LESS_RV64( )
```

Value:

```
    __asm__ __volatile__ ( \
        "srai    s6, s6, 63\n\t"   /* -1 if negative, 0 if >= 0 */ \
        ::: "s6" \
    )
```

7.272.2 Function Documentation

7.272.2.1 vm_docol_rv64()

```
static void vm_docol_rv64 (
            VM * vm )  [inline], [static]
```

References VM::current_executing_entry, and entry.

7.272.2.2 vm_exit_rv64()

```
static void vm_exit_rv64 (
            void  )  [inline], [static]
```

7.272.2.3 vm_save_registers_rv64()

```
static void vm_save_registers_rv64 (
            VM * vm )  [inline], [static]
```

References VM::data_stack, VM::dsp, VM::ip, VM::return_stack, and VM::rsp.

7.272.2.4 vm_setup_registers_rv64()

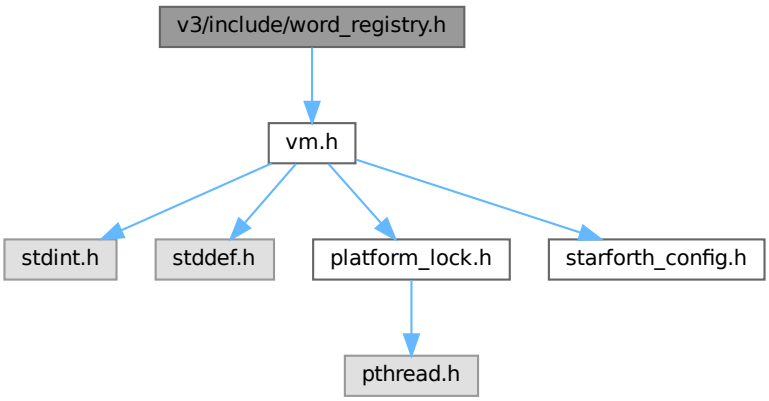
```
static void vm_setup_registers_rv64 (
            VM * vm )  [inline], [static]
```

References VM::data_stack, VM::dsp, VM::ip, VM::return_stack, and VM::rsp.

7.273 v3/include/word_registry.h File Reference

```
#include "vm.h"
```

Include dependency graph for word_registry.h:



This graph shows which files directly or indirectly include this file:

Functions

- void register_word (VM *vm, const char *name, word_func_t func)

  Registers a single FORTH word in the virtual machine.
- void register_forth79_words (VM *vm)

  Registers all standard FORTH-79 words in the virtual machine.

7.273.1 Function Documentation

7.273.1.1 register_forth79_words()

```
void register_forth79_words (
            VM * vm )
```

Registers all standard FORTH-79 words in the virtual machine.

Parameters

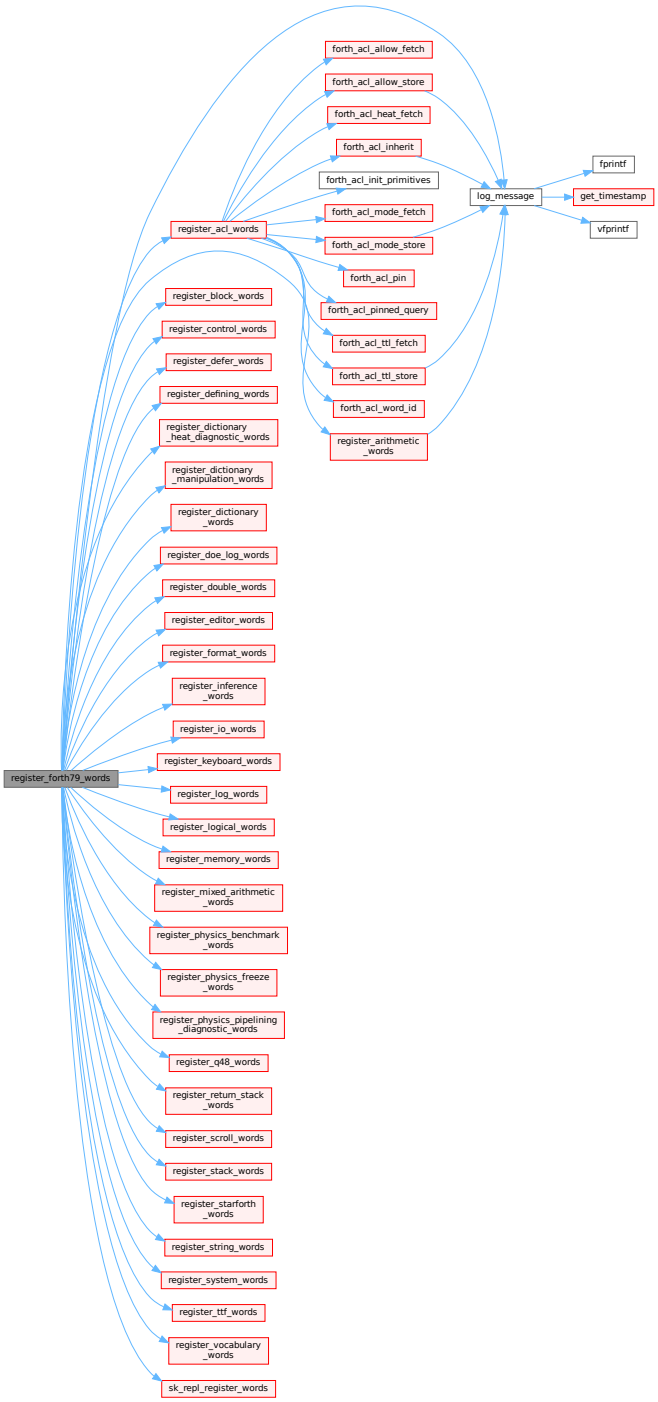
vm	Pointer to the virtual machine instance

Registers all 18 FORTH-79 word modules in dependency order, starting with fundamental stack operations and building up to more complex functionality like control flow and editing.

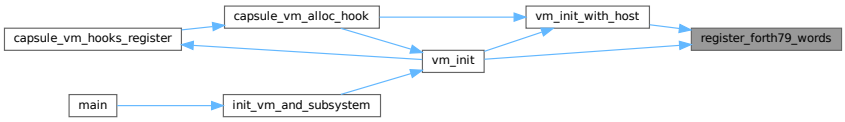
References LOG_INFO, log_message(), register_acl_words(), register_arithmetic_words(), register_block_words(), register_control_words(), register_defer_words(), register_defining_words(), register_dictionary_heat_diagnostic_words(), register_dictionary_manipulation_words(), register_dictionary_words(), register_doe_log_words(), register_double_words(), register_editor_words(), register_format_words(), register_inference_words(), register_io_words(), register_keyboard_words(), register_log_words(), register_logical_words(), register_memory_words(), register_mixed_arithmetic_words(), register_physics_benchmark_words(), register_physics_freeze_words(), register_physics_pipelining_diagnostic_words(), register_q48_words(), register_return_stack_words(), register_scroll_words(), register_stack_words(), register_starforth_words(), register_string_words(), register_system_words(), register_ttf_words(), register_vocabulary_words(), and sk_repl_register_words().

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.273.1.2 register_word()

```
void register_word (
            VM * vm,
            const char * name,
            word_func_t func )
```

Registers a single FORTH word in the virtual machine.

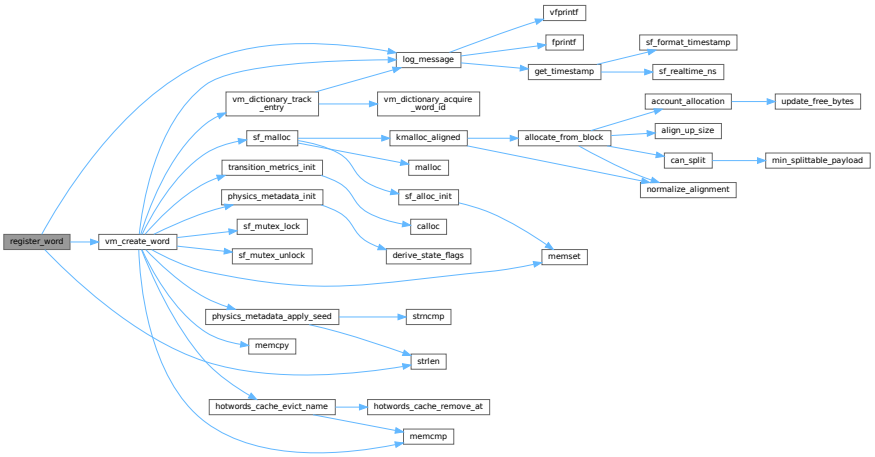
Parameters

vm	Pointer to the virtual machine instance
name	Name of the FORTH word to register
func	Function pointer to the word's implementation

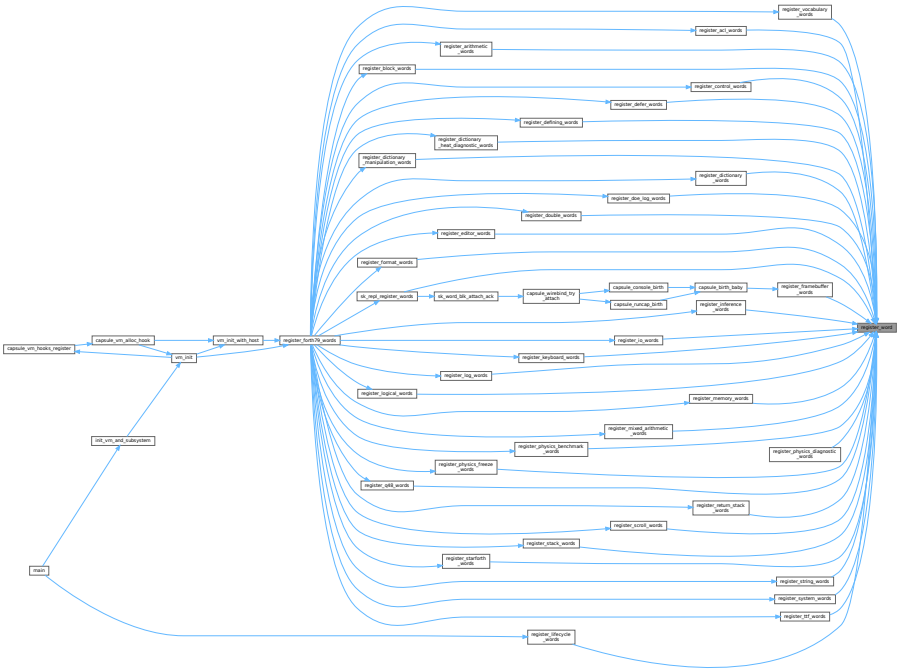
References entry, func, LOG_DEBUG, LOG_ERROR, log_message(), name, strlen(), and vm_create_word().

Referenced by register_acl_words(), register_arithmetic_words(), register_block_words(), register_control_words(), register_defer_words(), register_defining_words(), register_dictionary_heat_diagnostic_words(), register_dictionary_manipulation_wo register_dictionary_words(), register_doe_log_words(), register_double_words(), register_editor_words(), register_format_words(), register_framebuffer_words(), register_inference_words(), register_io_words(), register_keyboard_words(), register_lifecycle_words(), register_log_words(), register_logical_words(), register_memory_words(), register_mixed_arithmetic_word register_physics_benchmark_words(), register_physics_diagnostic_words(), register_physics_freeze_words(), register_q48_words(), register_return_stack_words(), register_scroll_words(), register_stack_words(), register_starforth_words(), register_string_words(), register_system_words(), register_ttf_words(), register_vocabulary_words(), and sk_repl_register_words().

Here is the call graph for this function:



Here is the caller graph for this function:

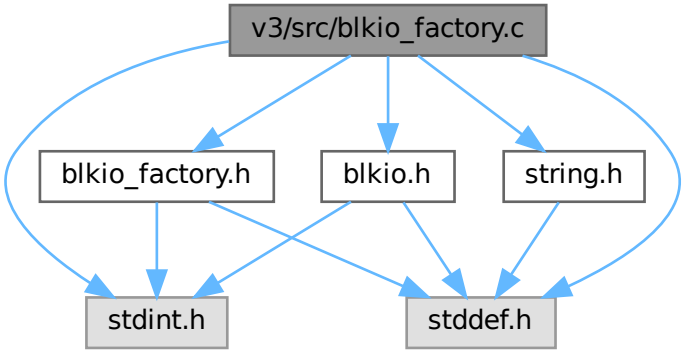


7.274 v3/src/blkio_factory.c File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "blkio.h"
```

```
#include "blkio_factory.h"
```

Include dependency graph for blkio_factory.c:



Functions

- size_t blkio_file_state_size (void)

  Return the byte size required for a `blkio_file_state_t`.
- const blkio_vtable_t * blkio_file_vtable (void)

  Return the vtable for the file-backed block I/O backend.
- int blkio_file_init_state (void *state_mem, size_t state_len, const char *path, uint32_t total_blocks, uint32_t fbs, uint8_t read_only, uint8_t create_if_missing, uint8_t truncate_to_size, void **out_opaque)

  Initialise a `blkio_file_state_t` in caller-provided memory.
- size_t blkio_ram_state_size (void)

  Return the byte size required for a `blkio_ram_state_t`.
- const blkio_vtable_t * blkio_ram_vtable (void)

  Return the vtable for the RAM block I/O backend.
- int blkio_ram_init_state (void *state_mem, size_t state_len, uint8_t *base, uint32_t total_blocks, uint32_t fbs, uint8_t read_only, void **out_opaque)

  Initialise a `blkio_ram_state_t` in caller-provided memory.
- static int nonempty (const char *s)

  Return 1 if string `s` is non-NULL and non-empty.
- int blkio_factory_open (blkio_dev_t *dev, const char *disk_img_path, uint8_t read_only, uint32_t total_↩ blocks, uint32_t fbs, void *state_mem, size_t state_len, uint8_t *ram_base, uint32_t ram_blocks, uint8_t *out_used_file)

  Open a block I/O device using the best available backend.

7.274.1 Function Documentation

7.274.1.1 blkio_factory_open()

```
int blkio_factory_open (
            blkio_dev_t * dev,
            const char * disk_img_path,
```

```
    uint8_t read_only,
    uint32_t total_blocks,
    uint32_t fbs,
    void * state_mem,
    size_t state_len,
    uint8_t * ram_base,
    uint32_t ram_blocks,
    uint8_t * out_used_file )
```

Open a block I/O device using the best available backend.

Open a block device using either a file-backed image (if disk_img_path is non-empty) or a RAM-backed buffer (fallback).

Selects the file backend when `disk_img_path` is non-empty, otherwise falls back to the RAM backend using `ram_base` / `ram_blocks`.

File backend behaviour:

- Creates the file if it does not exist (unless `read_only` is set).
- Derives `total_blocks` from the file size when `total_blocks` == 0.
- Sets `*out_used_file` = 1 on success.

RAM backend behaviour:

- `ram_base` must point to at least `ram_blocks * fbs` bytes of caller-owned memory that outlives the device.

Parameters

dev	Output device handle to populate
disk_img_path	Path to disk image file, or NULL/empty for RAM
read_only	1 = open read-only, 0 = read-write
total_blocks	Desired block count (0 = derive from file size)
fbs	FORTH block size in bytes (0 = use `BLKIO_FORTH_BLOCK_SIZE`)
state_mem	Caller-allocated state buffer (must be large enough)
state_len	Size of `state_mem` in bytes
ram_base	Base of RAM block array (used only when no disk path)
ram_blocks	Number of blocks in `ram_base`
out_used_file	Optional output: set to 1 if the file backend was used

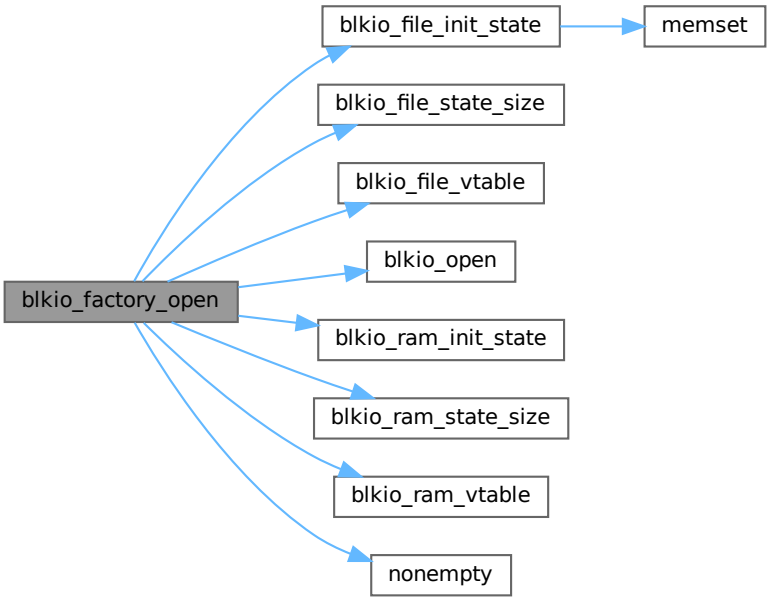
Returns

`BLKIO_OK` on success, or a `BLKIO_E*` error code

References BLKIO_EINVAL, BLKIO_ENOSPACE, blkio_file_init_state(), blkio_file_state_size(), blkio_file_vtable(), BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_open(), blkio_ram_init_state(), blkio_ram_state_size(), blkio_ram_vtable(), blkio_params_t::forth_block_size, nonempty(), and ram_base.

Referenced by init_blkio().

Here is the call graph for this function:



Here is the caller graph for this function:

7.274.1.2 blkio_file_init_state()

```
int blkio_file_init_state (
            void * state_mem,
            size_t state_len,
            const char * path,
            uint32_t total_blocks,
            uint32_t fbs,
            uint8_t read_only,
            uint8_t create_if_missing,
            uint8_t truncate_to_size,
            void ** out_opaque )  [extern]
```

Initialise a `blkio_file_state_t` in caller-provided memory.

Copies the path into state (truncated to PATH_MAX-1), sets `fbs`, `total_blocks`, and `read_only`, and writes the state pointer to `*out_opaque`. The file descriptor is initialised to -1; the actual `open()` call happens in `file_open()` (the vtable open method). `create_if_missing` and `truncate_to_size` are accepted for API compatibility but currently unused (creation is handled in `file_open()`).

Parameters

state_mem	Caller-allocated state buffer
state_len	Size of `state_mem` (must be >= blkio_file_state_size())
path	Path to the disk image file
total_blocks	Total block count (0 = derive from file size at open)
fbs	FORTH block size in bytes
read_only	1 = open read-only, 0 = read-write
create_if_missing	Ignored (reserved for future use)
truncate_to_size	Ignored (reserved for future use)
out_opaque	Output: set to `state_mem` on success

Returns

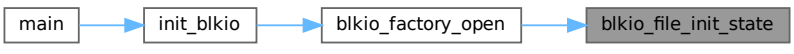
`BLKIO_OK` on success, `BLKIO_EINVAL` on bad arguments

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_file_state_t::fbs, blkio_file_state_t::fd, memset(), blkio_file_state_t::path, PATH_MAX, blkio_file_state_t::read_only, and blkio_file_state_t::total_blocks.

Referenced by blkio_factory_open().

Here is the call graph for this function:

Here is the caller graph for this function:



7.274.1.3 blkio_file_state_size()

```
size_t blkio_file_state_size (
            void  )  [extern]
```

Return the byte size required for a `blkio_file_state_t`.

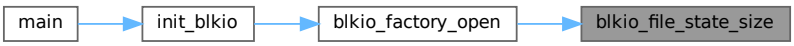
Used by `blkio_factory_open()` to verify the caller's state buffer is large enough before calling `blkio_file_init_state()`.

Returns

sizeof(blkio_file_state_t)

Referenced by blkio_factory_open().

Here is the caller graph for this function:



7.274.1.4 blkio_file_vtable()

```
const blkio_vtable_t * blkio_file_vtable (
            void  )  [extern]
```

Return the vtable for the file-backed block I/O backend.

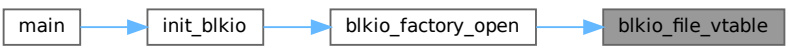
Returns

Pointer to the static `blkio_vtable_t` for file devices

References BLKIO_FILE_VT.

Referenced by blkio_factory_open().

Here is the caller graph for this function:



7.274.1.5 blkio_ram_init_state()

```
int blkio_ram_init_state (
            void * state_mem,
            size_t state_len,
            uint8_t * base,
            uint32_t total_blocks,
            uint32_t fbs,
            uint8_t read_only,
            void ** out_opaque )  [extern]
```

Initialise a `blkio_ram_state_t` in caller-provided memory.

Stores `base`, `total_blocks`, `fbs` (defaulting to `BLKIO_FORTH_BLOCK_SIZE` if zero), and `read_only` into the state struct at `state_mem`. Sets `*out_opaque` to point to the state. The caller owns the `base` memory and must ensure it remains valid for the device lifetime.

Parameters

state_mem	Caller-allocated buffer (must be >= blkio_ram_state_size())
state_len	Size of `state_mem` in bytes
base	Pointer to the RAM block array
total_blocks	Number of FORTH blocks in `base`
fbs	FORTH block size in bytes (0 = use default)
read_only	1 = deny writes, 0 = allow reads and writes
out_opaque	Output: set to `state_mem` on success

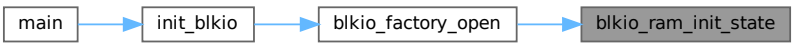
Returns

`BLKIO_OK` on success, `BLKIO_EINVAL` on bad arguments

References base, blkio_ram_state_t::base, BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_ram_state_t::fbs, blkio_ram_state_t::read_only, and blkio_ram_state_t::total_blocks.

Referenced by blkio_factory_open().

Here is the caller graph for this function:



7.274.1.6 blkio_ram_state_size()

```
size_t blkio_ram_state_size (
            void  )  [extern]
```

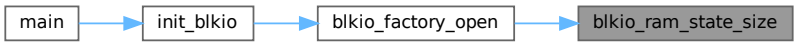
Return the byte size required for a `blkio_ram_state_t`.

Returns

sizeof(blkio_ram_state_t)

Referenced by blkio_factory_open().

Here is the caller graph for this function:



7.274.1.7 blkio_ram_vtable()

```
const blkio_vtable_t * blkio_ram_vtable (
            void  )  [extern]
```

Return the vtable for the RAM block I/O backend.

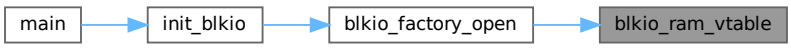
Returns

Pointer to the static `blkio_vtable_t` for RAM devices

References BLKIO_RAM_VT.

Referenced by blkio_factory_open().

Here is the caller graph for this function:



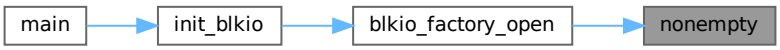
7.274.1.8 nonempty()

```
static int nonempty (
            const char * s )  [static]
```

Return 1 if string `s` is non-NULL and non-empty.

Referenced by blkio_factory_open().

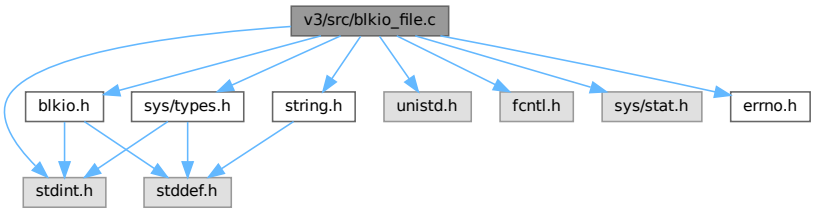
Here is the caller graph for this function:



7.275 v3/src/blkio_file.c File Reference

```
#include "blkio.h"
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <errno.h>
#include <string.h>
#include <stdint.h>
```

Include dependency graph for blkio_file.c:



Data Structures

- struct blkio_file_state_t

Macros

- #define PATH_MAX 4096

Functions

- size_t blkio_file_state_size (void)

  Return the byte size required for a `blkio_file_state_t`.
- int blkio_file_init_state (void *state_mem, size_t state_len, const char *path, uint32_t total_blocks, uint32_t fbs, uint8_t read_only, uint8_t create_if_missing, uint8_t truncate_to_size, void **out_opaque)

  Initialise a `blkio_file_state_t` in caller-provided memory.
- static int full_read_at (int fd, void *buf, size_t len, off_t off_abs)

  Perform a complete read of `len` bytes at absolute offset `off_abs`.
- static int full_write_at (int fd, const void *buf, size_t len, off_t off_abs)

  Perform a complete write of `len` bytes at absolute offset `off_abs`.
- static int file_open (blkio_dev_t *dev, const blkio_params_t *p)

  Open the file and attach it to the device handle.
- static int file_close (blkio_dev_t *dev)

  Close the file descriptor held by the device.
- static int file_read (blkio_dev_t *dev, uint32_t fblock, void *dst)

  Read one FORTH block from the file into `dst`.
- static int file_write (blkio_dev_t *dev, uint32_t fblock, const void *src)

  Write one FORTH block from `src` into the file.

- static int file_flush (blkio_dev_t *dev)

  Flush (fsync) pending writes to the file.
- static int file_info (blkio_dev_t *dev, blkio_info_t *out)

  Return device info (block size, count, physical size, read-only flag).
- const blkio_vtable_t * blkio_file_vtable (void)

  Return the vtable for the file-backed block I/O backend.

Variables

- static const blkio_vtable_t BLKIO_FILE_VT

7.275.1 Macro Definition Documentation

7.275.1.1 PATH_MAX

```
#define PATH_MAX 4096
```

7.275.2 Function Documentation

7.275.2.1 blkio_file_init_state()

```
int blkio_file_init_state (
            void * state_mem,
            size_t state_len,
            const char * path,
            uint32_t total_blocks,
            uint32_t fbs,
            uint8_t read_only,
            uint8_t create_if_missing,
            uint8_t truncate_to_size,
            void ** out_opaque )
```

Initialise a `blkio_file_state_t` in caller-provided memory.

Copies the path into state (truncated to PATH_MAX-1), sets `fbs`, `total_blocks`, and `read_only`, and writes the state pointer to `*out_opaque`. The file descriptor is initialised to -1; the actual `open()` call happens in `file_open()` (the vtable open method). `create_if_missing` and `truncate_to_size` are accepted for API compatibility but currently unused (creation is handled in `file_open()`).

Parameters

state_mem	Caller-allocated state buffer
state_len	Size of `state_mem` (must be >= blkio_file_state_size())
path	Path to the disk image file
total_blocks	Total block count (0 = derive from file size at open)
fbs	FORTH block size in bytes
read_only	1 = open read-only, 0 = read-write
create_if_missing	Ignored (reserved for future use)
truncate_to_size	Ignored (reserved for future use)
out_opaque	Output: set to `state_mem` on success

Returns

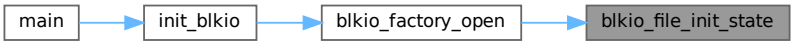
`BLKIO_OK` on success, `BLKIO_EINVAL` on bad arguments

References BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_file_state_t::fbs, blkio_file_state_t::fd, memset(), blkio_file_state_t::path, PATH_MAX, blkio_file_state_t::read_only, and blkio_file_state_t::total_blocks.

Referenced by blkio_factory_open().

Here is the call graph for this function:

Here is the caller graph for this function:



7.275.2.2 blkio_file_state_size()

```
size_t blkio_file_state_size (
            void  )
```

Return the byte size required for a `blkio_file_state_t`.

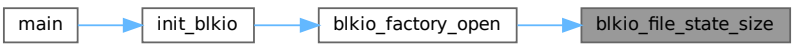
Used by `blkio_factory_open()` to verify the caller's state buffer is large enough before calling `blkio_file_init_state()`.

Returns

sizeof(blkio_file_state_t)

Referenced by blkio_factory_open().

Here is the caller graph for this function:



7.275.2.3 blkio_file_vtable()

```
const blkio_vtable_t * blkio_file_vtable (
            void  )
```

Return the vtable for the file-backed block I/O backend.

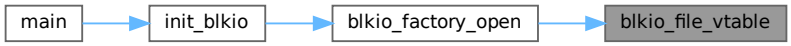
Returns

Pointer to the static `blkio_vtable_t` for file devices

References BLKIO_FILE_VT.

Referenced by blkio_factory_open().

Here is the caller graph for this function:



7.275.2.4 file_close()

```
static int file_close (
            blkio_dev_t * dev )  [static]
```

Close the file descriptor held by the device.

Parameters

dev	Device handle (must have valid state)

Returns

`BLKIO_OK` on success, `BLKIO_EIO` if `close()` fails

References BLKIO_EINVAL, BLKIO_EIO, BLKIO_OK, blkio_file_state_t::fd, and blkio_dev::state.

7.275.2.5 file_flush()

```
static int file_flush (
            blkio_dev_t * dev )  [static]
```

Flush (fsync) pending writes to the file.

Flush pending writes via `fsync()` on Linux/POSIX; no-op elsewhere.

Parameters

dev	Open file device

Returns

`BLKIO_OK` on success, `BLKIO_EIO` if `fsync()` fails

References BLKIO_ECLOSED, BLKIO_EINVAL, BLKIO_EIO, BLKIO_OK, blkio_file_state_t::fd, and blkio_dev::state.

7.275.2.6 file_info()

```
static int file_info (
            blkio_dev_t * dev,
            blkio_info_t * out )  [static]
```

Return device info (block size, count, physical size, read-only flag).

Return device info for the file backend.

Physical sector size is reported as 512 (best-effort guess; no ioctl used). Physical size in bytes comes from `fstat()`.

Parameters

dev	Open file device
out	Output struct to populate

Returns

`BLKIO_OK` on success, `BLKIO_EIO` if `fstat()` fails

References BLKIO_EINVAL, BLKIO_EIO, BLKIO_OK, blkio_file_state_t::fbs, blkio_file_state_t::fd, blkio_info_t::forth_block_size, blkio_info_t::phys_sector_size, blkio_info_t::phys_size_bytes, blkio_info_t::read_only, blkio_file_state_t::read_only, blkio_dev::state, blkio_info_t::total_blocks, and blkio_file_state_t::total_blocks.

Referenced by load_kernel_from_esp().

Here is the caller graph for this function:

7.275.2.7 file_open()

```
static int file_open (
            blkio_dev_t * dev,
            const blkio_params_t * p )  [static]
```

Open the file and attach it to the device handle.

Open the underlying file and populate the device handle.

Opens `st->path` with `O_RDONLY` or `O_RDWR|O_CREAT` depending on `st->read_only`. Derives `total_blocks` from the file size if the state value is 0 — the file size must then be a multiple of `fbs`. Sets `dev->state`, `dev->forth_block_size`, and `dev->total_blocks` on success.

Parameters

dev	Uninitialised device handle
p	Parameters including the pre-initialised opaque state pointer

Returns

`BLKIO_OK` on success, `BLKIO_EIO` or `BLKIO_EINVAL` on failure

References BLKIO_EINVAL, BLKIO_EIO, BLKIO_OK, blkio_file_state_t::fbs, blkio_file_state_t::fd, blkio_dev::forth_block_size, blkio_params_t::opaque, blkio_file_state_t::path, blkio_file_state_t::read_only, blkio_dev::state, blkio_dev::total_blocks, and blkio_file_state_t::total_blocks.

7.275.2.8 file_read()

```
static int file_read (
            blkio_dev_t * dev,
            uint32_t fblock,
            void * dst )  [static]
```

Read one FORTH block from the file into `dst`.

Read one FORTH block from the file backend.

Parameters

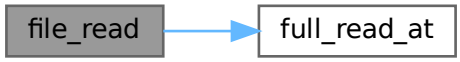
dev	Open file device
fblock	Logical block number (must be < `dev->total_blocks`)
dst	Destination buffer (must be >= `fbs` bytes)

Returns

`BLKIO_OK`, `BLKIO_EINVAL` if `fblock` is out of range, or `BLKIO_EIO`

References BLKIO_ECLOSED, BLKIO_EINVAL, BLKIO_EIO, BLKIO_OK, blkio_file_state_t::fbs, blkio_file_state_t::fd, full_read_at(), blkio_dev::state, and blkio_dev::total_blocks.

Here is the call graph for this function:



7.275.2.9 file_write()

```
static int file_write (
            blkio_dev_t * dev,
            uint32_t fblock,
            const void * src )  [static]
```

Write one FORTH block from `src` into the file.

Write one FORTH block to the file backend.

Parameters

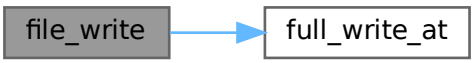
dev	Open file device (must not be read-only)
fblock	Logical block number (must be < `dev->total_blocks`)
src	Source buffer (must be >= `fbs` bytes)

Returns

`BLKIO_OK`, `BLKIO_ENOSUP` if read-only, `BLKIO_EINVAL` if out of range, or `BLKIO_EIO` on write failure

References BLKIO_ECLOSED, BLKIO_EINVAL, BLKIO_EIO, BLKIO_ENOSUP, BLKIO_OK, blkio_file_state_t::fbs, blkio_file_state_t::fd, full_write_at(), blkio_file_state_t::read_only, blkio_dev::state, and blkio_dev::total_blocks.

Here is the call graph for this function:



7.275.2.10 full_read_at()

```
static int full_read_at (
             int fd,
             void * buf,
             size_t len,
             off_t off_abs )  [static]
```

Perform a complete read of `len` bytes at absolute offset `off_abs`.

Uses `lseek()` + `read()` loops to handle short reads and EINTR. Not safe for concurrent readers/writers on the same file descriptor. Returns -1 on any I/O error or unexpected EOF.

Parameters

fd	Open file descriptor
buf	Destination buffer (must be >= `len` bytes)
len	Number of bytes to read
off_abs	Absolute byte offset within the file

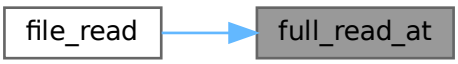
Returns

0 on success, -1 on error (errno set)

References errno, and SEEK_SET.

Referenced by file_read().

Here is the caller graph for this function:



7.275.2.11 full_write_at()

```
static int full_write_at (
             int fd,
             const void * buf,
             size_t len,
             off_t off_abs )  [static]
```

Perform a complete write of `len` bytes at absolute offset `off_abs`.

Uses `lseek()` + `write()` loops to handle short writes and EINTR. Not safe for concurrent readers/writers on the same file descriptor. Returns -1 on any I/O error.

Parameters

fd	Open file descriptor
buf	Source buffer (must be >= `len` bytes)
len	Number of bytes to write
off_abs	Absolute byte offset within the file

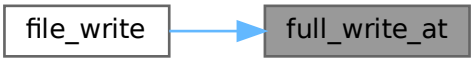
Returns

0 on success, -1 on error (errno set)

References errno, and SEEK_SET.

Referenced by file_write().

Here is the caller graph for this function:



7.275.3 Variable Documentation

7.275.3.1 BLKIO_FILE_VT

```
const blkio_vtable_t BLKIO_FILE_VT  [static]
```

Initial value:

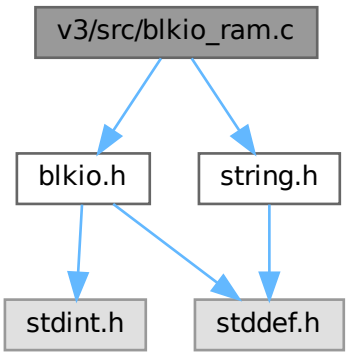
```
= {
    .open = file_open,
    .close = file_close,
    .read = file_read,
    .write = file_write,
    .flush = file_flush,
    .info = file_info
}
```

Referenced by blkio_file_vtable().

7.276 v3/src/blkio_ram.c File Reference

```
#include "blkio.h"
#include <string.h>
```

Include dependency graph for blkio_ram.c:



Data Structures

- struct blkio_ram_state_t

Functions

- size_t blkio_ram_state_size (void)

  Return the byte size required for a `blkio_ram_state_t`.
- int blkio_ram_init_state (void *state_mem, size_t state_len, uint8_t *base, uint32_t total_blocks, uint32_t fbs, uint8_t read_only, void **out_opaque)

  Initialise a `blkio_ram_state_t` in caller-provided memory.
- static int ram_open (blkio_dev_t *dev, const blkio_params_t *p)

  Attach state to the device handle and validate parameters.
- static int ram_close (blkio_dev_t *dev)

  Close the RAM device (no-op; caller owns the memory).
- static int ram_read (blkio_dev_t *dev, uint32_t fblock, void *dst)

  Read one FORTH block from the RAM array into `dst`.
- static int ram_write (blkio_dev_t *dev, uint32_t fblock, const void *src)

  Write one FORTH block from `src` into the RAM array.
- static int ram_flush (blkio_dev_t *dev)

  Flush the RAM device — always a no-op since data is in memory.
- static int ram_info (blkio_dev_t *dev, blkio_info_t *out)

  Return device info (block size, count, physical size, read-only flag).
- const blkio_vtable_t * blkio_ram_vtable (void)

  Return the vtable for the RAM block I/O backend.
- static int in_bounds (const blkio_ram_state_t *st, uint32_t fb)

  Test whether `fb` is a valid block number for the RAM device.

Variables

- static const blkio_vtable_t BLKIO_RAM_VT

7.276.1 Function Documentation

7.276.1.1 blkio_ram_init_state()

```
int blkio_ram_init_state (
            void * state_mem,
            size_t state_len,
            uint8_t * base,
            uint32_t total_blocks,
            uint32_t fbs,
            uint8_t read_only,
            void ** out_opaque )
```

Initialise a `blkio_ram_state_t` in caller-provided memory.

Stores `base`, `total_blocks`, `fbs` (defaulting to `BLKIO_FORTH_BLOCK_SIZE` if zero), and `read_only` into the state struct at `state_mem`. Sets `*out_opaque` to point to the state. The caller owns the `base` memory and must ensure it remains valid for the device lifetime.

Parameters

state_mem	Caller-allocated buffer (must be >= blkio_ram_state_size())
state_len	Size of `state_mem` in bytes
base	Pointer to the RAM block array
total_blocks	Number of FORTH blocks in `base`
fbs	FORTH block size in bytes (0 = use default)
read_only	1 = deny writes, 0 = allow reads and writes
out_opaque	Output: set to `state_mem` on success

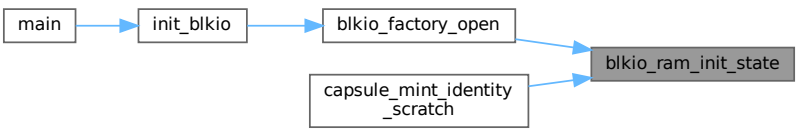
Returns

`BLKIO_OK` on success, `BLKIO_EINVAL` on bad arguments

References base, blkio_ram_state_t::base, BLKIO_EINVAL, BLKIO_FORTH_BLOCK_SIZE, BLKIO_OK, blkio_ram_state_t::fbs, blkio_ram_state_t::read_only, and blkio_ram_state_t::total_blocks.

Referenced by blkio_factory_open(), and capsule_mint_identity_scratch().

Here is the caller graph for this function:



7.276.1.2 blkio_ram_state_size()

```
size_t blkio_ram_state_size (
            void  )
```

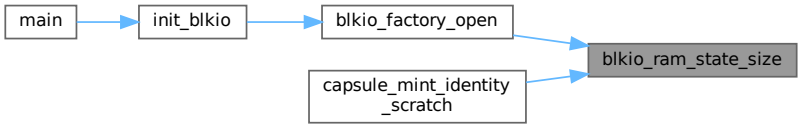
Return the byte size required for a `blkio_ram_state_t`.

Returns

sizeof(blkio_ram_state_t)

Referenced by blkio_factory_open(), and capsule_mint_identity_scratch().

Here is the caller graph for this function:



7.276.1.3 blkio_ram_vtable()

```
const blkio_vtable_t * blkio_ram_vtable (
            void  )
```

Return the vtable for the RAM block I/O backend.

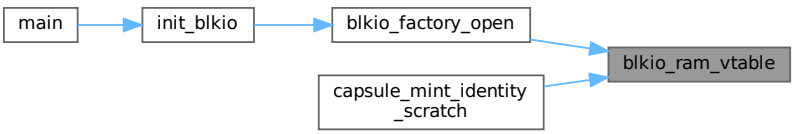
Returns

Pointer to the static `blkio_vtable_t` for RAM devices

References BLKIO_RAM_VT.

Referenced by blkio_factory_open(), and capsule_mint_identity_scratch().

Here is the caller graph for this function:



7.276.1.4 in_bounds()

```
static int in_bounds (
            const blkio_ram_state_t * st,
            uint32_t fb )  [inline], [static]
```

Test whether `fb` is a valid block number for the RAM device.

Parameters

st	RAM state (may be NULL)
fb	Block number to test

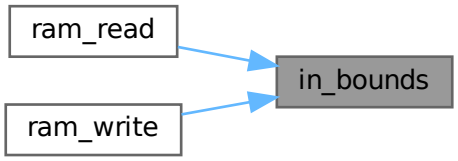
Returns

1 if in bounds, 0 otherwise

References blkio_ram_state_t::total_blocks.

Referenced by ram_read(), and ram_write().

Here is the caller graph for this function:



7.276.1.5 ram_close()

```
static int ram_close (
            blkio_dev_t * dev )  [static]
```

Close the RAM device (no-op; caller owns the memory).

Close the RAM device — caller retains ownership of the RAM buffer.

Parameters

dev	Device handle

Returns

`BLKIO_OK` always (no resources to release)

References BLKIO_EINVAL, BLKIO_OK, and blkio_dev::state.

7.276.1.6 ram_flush()

```
static int ram_flush (
            blkio_dev_t * dev )  [static]
```

Flush the RAM device — always a no-op since data is in memory.

Flush the RAM device — no-op since data is always in memory.

Parameters

dev	Device handle (unused)

Returns

`BLKIO_OK` always

References BLKIO_OK.

7.276.1.7 ram_info()

```
static int ram_info (
            blkio_dev_t * dev,
            blkio_info_t * out )  [static]
```

Return device info (block size, count, physical size, read-only flag).

Return device info for the RAM backend.

Physical sector size equals `fbs` (RAM has no physical sector constraint). Physical size in bytes = `fbs * total_blocks`.

Parameters

dev	Open RAM device
out	Output struct to populate

Returns

`BLKIO_OK` on success, `BLKIO_EINVAL` if `dev` or `out` is NULL

References BLKIO_EINVAL, BLKIO_OK, blkio_ram_state_t::fbs, blkio_info_t::forth_block_size, blkio_info_t::phys_sector_size, blkio_info_t::phys_size_bytes, blkio_info_t::read_only, blkio_ram_state_t::read_only, blkio_dev::state, blkio_info_t::total_blocks, and blkio_ram_state_t::total_blocks.

7.276.1.8 ram_open()

```
static int ram_open (
            blkio_dev_t * dev,
            const blkio_params_t * p )  [static]
```

Attach state to the device handle and validate parameters.

Attach the RAM state to the device handle.

Validates that `base` and `fbs` are non-zero, then sets `dev->state`, `dev->forth_block_size`, and `dev->total_blocks`.

Parameters

dev	Uninitialised device handle
p	Parameters with pre-initialised opaque RAM state pointer

Returns

`BLKIO_OK` on success, `BLKIO_EINVAL` on bad arguments

References blkio_ram_state_t::base, BLKIO_EINVAL, BLKIO_OK, blkio_ram_state_t::fbs, blkio_dev::forth_block_size, blkio_params_t::opaque, blkio_dev::state, blkio_dev::total_blocks, and blkio_ram_state_t::total_blocks.

7.276.1.9 ram_read()

```
static int ram_read (
            blkio_dev_t * dev,
            uint32_t fblock,
            void * dst )  [static]
```

Read one FORTH block from the RAM array into `dst`.

Copy one FORTH block from the RAM array to `dst`.

Parameters

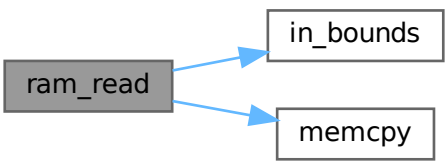
dev	Open RAM device
fblock	Logical block number (must be < `total_blocks`)
dst	Destination buffer (must be >= `fbs` bytes)

Returns

`BLKIO_OK` on success, `BLKIO_EINVAL` if out of range or NULL pointer

References blkio_ram_state_t::base, BLKIO_EINVAL, BLKIO_OK, blkio_ram_state_t::fbs, in_bounds(), memcpy(), and blkio_dev::state.

Here is the call graph for this function:



7.276.1.10 ram_write()

```
static int ram_write (
            blkio_dev_t * dev,
            uint32_t fblock,
            const void * src )  [static]
```

Write one FORTH block from `src` into the RAM array.

Copy one FORTH block from `src` into the RAM array.

Parameters

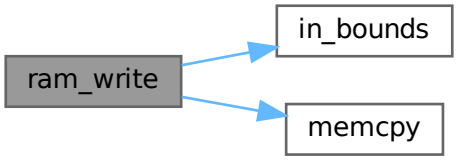
dev	Open RAM device (must not be read-only)
fblock	Logical block number (must be < `total_blocks`)
src	Source buffer (must be >= `fbs` bytes)

Returns

`BLKIO_OK` on success, `BLKIO_ENOSUP` if read-only, `BLKIO_EINVAL` if out of range or NULL pointer

References blkio_ram_state_t::base, BLKIO_EINVAL, BLKIO_ENOSUP, BLKIO_OK, blkio_ram_state_t::fbs, in_bounds(), memcpy(), blkio_ram_state_t::read_only, and blkio_dev::state.

Here is the call graph for this function:



7.276.2 Variable Documentation

7.276.2.1 BLKIO_RAM_VT

```
const blkio_vtable_t BLKIO_RAM_VT  [static]
```

Initial value:

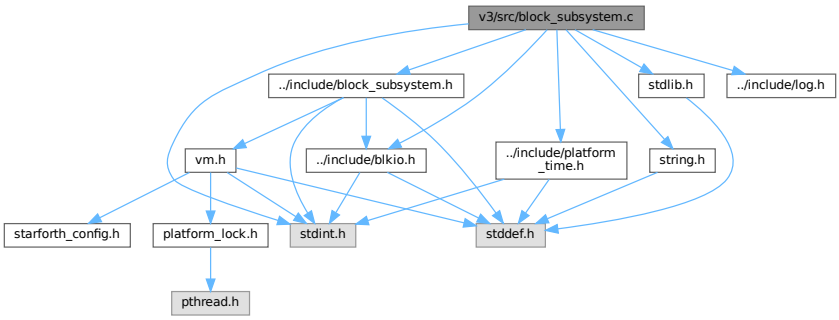
```
= {
    .open = ram_open,
    .close = ram_close,
    .read = ram_read,
    .write = ram_write,
    .flush = ram_flush,
    .info = ram_info
}
```

Referenced by blkio_ram_vtable().

7.277 v3/src/block_subsystem.c File Reference

```
#include "../include/blkio.h"
#include "../include/block_subsystem.h"
#include "../include/log.h"
#include "../include/platform_time.h"
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
```

Include dependency graph for block_subsystem.c:



Data Structures

- struct blk_reloc_entry_t
- struct cache_slot_t
- struct blk_dev_slot

Macros

- #define BLK_FORTH_SYS_RESERVED 32u
- #define BLK_DISK_SYS_RESERVED 32u
- #define META_REGION_OFFSET (BLK_PACK_RATIO ∗ BLK_FORTH_SIZE)
- #define META_REGION_SIZE (BLK_DEVICE_SECTOR - META_REGION_OFFSET)
- #define META_PER_BLOCK (META_REGION_SIZE / BLK_PACK_RATIO)
- #define DISK_CACHE_SLOTS 8
- #define BLK_RELOC_MAX_ENTRIES 500u
- #define BLK_FIRSTTOUCH_MAX_CLAIM 256u
- #define MIGRATION_WEAR_THRESHOLD 10000u
- #define MIGRATION_SCAN_BUDGET 256u

Typedefs

- typedef struct blk_dev_slot blk_dev_slot_t

Functions

- static size_t minzu (size_t a, size_t b)
- static uint32_t udiv_floor (uint32_t a, uint32_t b)
- static void crc64_init (void)
- uint64_t compute_crc64 (const uint8_t *data, size_t len)
- static int pbam_test (const uint8_t *bm, uint32_t i)
- static void pbam_set (uint8_t *bm, uint32_t i)
- static void pbam_clr (uint8_t *bm, uint32_t i)
- static uint32_t resolve_lbn (uint32_t lbn)
- static blk_dev_slot_t * lbn_to_slot (uint32_t lbn)
- static void chain_append (blk_dev_slot_t *slot)
- static uint32_t devblock4k_to_lba1k (blk_dev_slot_t *slot, uint32_t dev4k)
- static uint32_t lbn_to_slot_pbn (blk_dev_slot_t *slot, uint32_t lbn)
- static uint32_t slot_pbn_to_devblock (uint32_t rel_pbn)
- static uint32_t slot_pbn_pack_offset (uint32_t rel_pbn)
- static int read_devblock_4k (blk_dev_slot_t *slot, uint32_t dev4k, uint8_t *buf4k)
- static int write_devblock_4k (blk_dev_slot_t *slot, uint32_t dev4k, const uint8_t *buf4k)
- static void meta_from_slice (blk_meta_t *dst, const uint8_t *slice)
- static void meta_to_slice (const blk_meta_t *src, uint8_t *slice)
- static int cache_writeback (blk_dev_slot_t *slot, cache_slot_t *s)
- static cache_slot_t * cache_get_slot (blk_dev_slot_t *slot, uint32_t dev4k)
- static cache_slot_t * cache_load_devblock (blk_dev_slot_t *slot, uint32_t dev4k)
- static void volmeta_from_buf (blk_volume_meta_t *out, const uint8_t *buf)
- static void volmeta_to_buf (const blk_volume_meta_t *in, uint8_t *buf)
- static int read_header_4k (blk_dev_slot_t *slot, uint8_t *buf4k)
- static int write_header_4k (blk_dev_slot_t *slot, const uint8_t *buf4k)
- static uint32_t choose_B (uint64_t total_devblocks_4k)
- static void compute_totals_from_B (blk_volume_meta_t *m)
- void blk_volume_compute_totals (blk_volume_meta_t *m)
- void blk_subsys_set_format_bam_for (uint64_t total_devblocks)
- static uint32_t bam_pages_used (const blk_dev_slot_t *slot)
- static int bam_sync_from_disk (blk_dev_slot_t *slot)
- static int bam_flush_to_disk (blk_dev_slot_t *slot)
- static int reloc_load_from_disk (blk_dev_slot_t *slot)
- static int reloc_flush_to_disk (blk_dev_slot_t *slot)
- static void blk_compute_fresh_geometry (blk_dev_slot_t *slot)
- static int blk_commit_format (blk_dev_slot_t *slot)
- static int blk_format_or_load_disk (blk_dev_slot_t *slot)
- static uint64_t blk_get_timestamp (void)
- int blk_subsys_init (uint8_t *ram_base, size_t ram_size)
- int blk_subsys_add_raw_device (uint8_t *buf, uint32_t nblocks)
- static blk_dev_slot_t * first_disk_slot (void)
- int blk_subsys_attach_device (struct blkio_dev *dev)
- int blk_subsys_detach_device (struct blkio_dev *dev)
- int blk_subsys_relocate_block (uint32_t home_lbn, uint32_t target_lbn)
- uint64_t blk_subsys_epoch (void)
- int blk_subsys_shutdown (void)
- int blk_subsys_confirm_format (uint32_t lbn)
- uint8_t * blk_get_buffer (uint32_t block_num, int writable)
- uint8_t * blk_get_empty_buffer (uint32_t block_num)
- int blk_update (uint32_t block_num)
- int blk_flush (uint32_t block_num)
- int blk_is_allocated (uint32_t block_num)
- int blk_mark_allocated (uint32_t block_num)

- int blk_mark_free (uint32_t block_num)
- static int devblock_claimed_by_lbn (blk_dev_slot_t *slot, uint32_t lbn)
- int blk_allocate (uint32_t *block_num)
- int blk_get_volume_meta (blk_volume_meta_t *meta)
- int blk_set_volume_meta (const blk_volume_meta_t *meta)
- static blk_dev_slot_t * slot_by_dev (struct blkio_dev *dev)
- int blk_get_device_range (struct blkio_dev *dev, uint32_t *out_start_lbn, uint32_t *out_count)
- const void * blk_lbn_device_handle (uint32_t lbn)
- int blk_get_device_free_blocks (struct blkio_dev *dev, uint64_t *out_free, uint64_t *out_total)
- int blk_get_first_disk_range (uint32_t *out_start_lbn, uint32_t *out_count)
- int blk_meta_zone_read (uint32_t devblock_from_top, uint8_t buf[4096])
- int blk_meta_zone_write (uint32_t devblock_from_top, const uint8_t buf[4096])
- int blk_is_valid (uint32_t block_num)
- uint32_t blk_get_total_blocks (void)
- int blk_get_meta (uint32_t block_num, blk_meta_t *meta)
- int blk_set_meta (uint32_t block_num, const blk_meta_t *meta)
- int blk_owner_fp_get (uint32_t block_num, uint8_t out_fp[8])
- int blk_owner_fp_set (uint32_t block_num, const uint8_t fp[8])
- int blk_acl_allow_get (uint32_t block_num, uint8_t *out_allow)
- int blk_acl_allow_set (uint32_t block_num, uint8_t allow)
- int blk_acl_ttl_get (uint32_t block_num, uint32_t *out_ttl)
- int blk_acl_ttl_set (uint32_t block_num, uint32_t ttl)
- int blk_flags_get (uint32_t block_num, uint64_t *out_flags)
- int blk_flags_set (uint32_t block_num, uint64_t flags)
- static int devblock_is_free (blk_dev_slot_t *slot, uint32_t rep_lbn, const blk_meta_t *m)
- int blk_firsttouch_claim (const uint8_t owner_fp[8], uint32_t count, uint32_t *out_chain_head)
- int blk_owner_has_claim (const uint8_t owner_fp[8])
- int blk_meta_relocate_devblock (uint32_t home_devblock, uint32_t target_devblock)
- static void blk_migration_idle_check_slot (blk_dev_slot_t *slot)
- void blk_migration_idle_check (void)
- __attribute__ ((weak))

Variables

- static uint64_t crc64_table [256]
- static int crc64_inited = 0
- struct {
  uint8_t * ram_base
  size_t ram_size
  uint8_t dirty_ram [BLK_RAM_BLOCKS]
  uint32_t ram_user
  uint64_t total_user_lbn
  blk_dev_slot_t * head
  uint64_t epoch
  blk_reloc_entry_t reloc [500u]
  uint32_t reloc_count
  int initialized
  } g = {0}

- static uint64_t g_format_bam_for

7.277.1 Macro Definition Documentation

7.277.1.1 BLK_DISK_SYS_RESERVED

```
#define BLK_DISK_SYS_RESERVED 32u
```

7.277.1.2 BLK_FIRSTTOUCH_MAX_CLAIM

```
#define BLK_FIRSTTOUCH_MAX_CLAIM 256u
```

7.277.1.3 BLK_FORTH_SYS_RESERVED

```
#define BLK_FORTH_SYS_RESERVED 32u
```

7.277.1.4 BLK_RELOC_MAX_ENTRIES

```
#define BLK_RELOC_MAX_ENTRIES 500u
```

7.277.1.5 DISK_CACHE_SLOTS

```
#define DISK_CACHE_SLOTS 8
```

7.277.1.6 META_PER_BLOCK

```
#define META_PER_BLOCK (META_REGION_SIZE / BLK_PACK_RATIO)
```

7.277.1.7 META_REGION_OFFSET

```
#define META_REGION_OFFSET (BLK_PACK_RATIO * BLK_FORTH_SIZE)
```

7.277.1.8 META_REGION_SIZE

```
#define META_REGION_SIZE (BLK_DEVICE_SECTOR - META_REGION_OFFSET)
```

7.277.1.9 MIGRATION_SCAN_BUDGET

```
#define MIGRATION_SCAN_BUDGET 256u
```

7.277.1.10 MIGRATION_WEAR_THRESHOLD

```
#define MIGRATION_WEAR_THRESHOLD 10000u
```

7.277.2 Typedef Documentation

7.277.2.1 blk_dev_slot_t

```
typedef struct blk_dev_slot blk_dev_slot_t
```

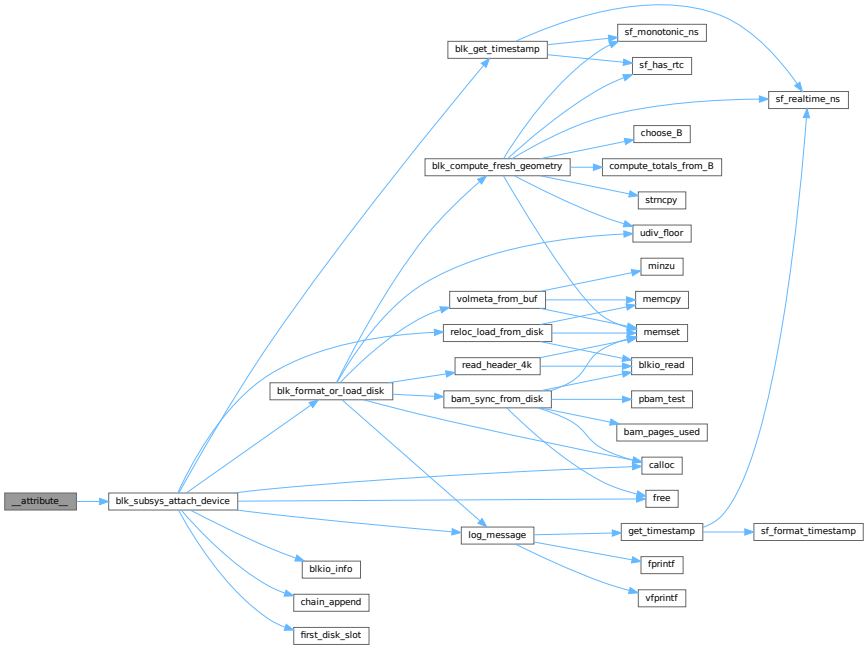
7.277.3 Function Documentation

7.277.3.1 __attribute__()

```
__attribute__ (
            (weak)  )
```

References blk_subsys_attach_device().

Here is the call graph for this function:



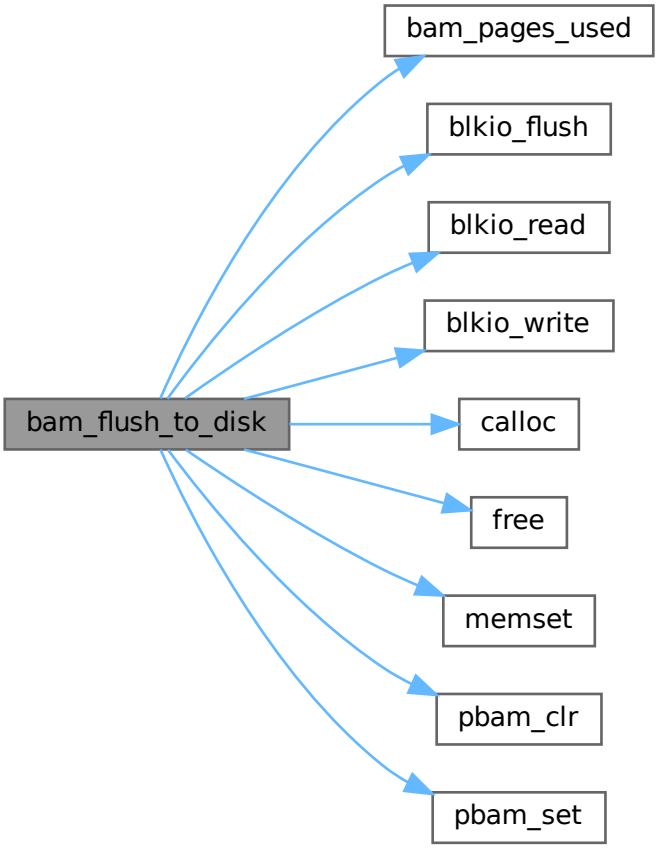
7.277.3.2 bam_flush_to_disk()

```
static int bam_flush_to_disk (
            blk_dev_slot_t * slot )  [static]
```

References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_volume_meta_t::bam_devblocks, blk_dev_slot::bam_dirty, bam_pages_used(), blk_volume_meta_t::bam_start, BLK_DISK_SYS_RESERVED, BLK_EIO, BLK_ENOMEM, BLK_OK, blkio_flush(), BLKIO_OK, blkio_read(), blkio_write(), calloc(), blk_dev_slot::dev, free(), memset(), pbam_clr(), pbam_set(), blk_dev_slot::user_blocks, and blk_dev_slot::vol_meta.

Referenced by blk_commit_format(), and blk_flush().

Here is the call graph for this function:



Here is the caller graph for this function:

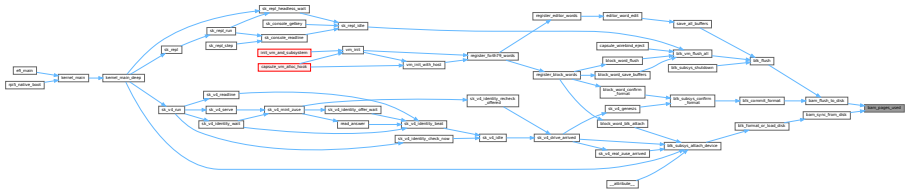
7.277.3.3 bam_pages_used()

```
static uint32_t bam_pages_used (
            const blk_dev_slot_t * slot )  [static]
```

References blk_volume_meta_t::bam_devblocks, BLK_DISK_SYS_RESERVED, blk_dev_slot::user_blocks, and blk_dev_slot::vol_meta.

Referenced by bam_flush_to_disk(), and bam_sync_from_disk().

Here is the caller graph for this function:



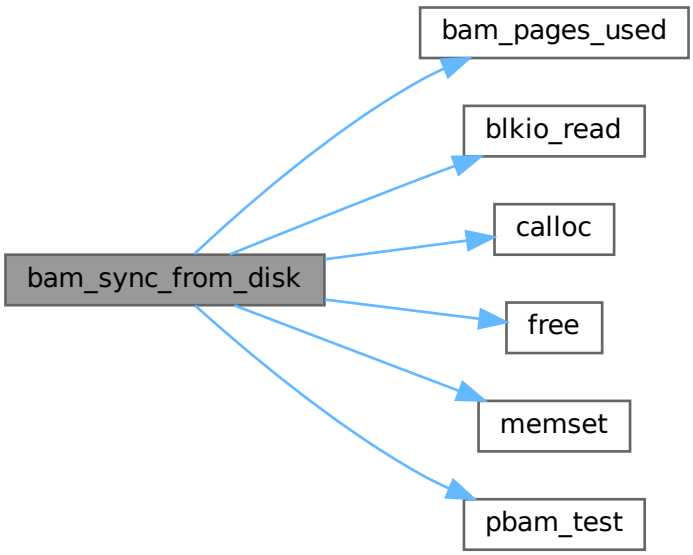
7.277.3.4 bam_sync_from_disk()

```
static int bam_sync_from_disk (
            blk_dev_slot_t * slot )  [static]
```

References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_volume_meta_t::bam_devblocks, blk_dev_slot::bam_dirty, bam_pages_used(), blk_volume_meta_t::bam_start, BLK_DISK_SYS_RESERVED, BLK_EINVAL, BLK_ENOMEM, BLK_OK, BLKIO_OK, blkio_read(), calloc(), blk_dev_slot::dev, free(), memset(), pbam_test(), blk_dev_slot::user_blocks, and blk_dev_slot::vol_meta.

Referenced by blk_format_or_load_disk().

Here is the call graph for this function:



Here is the caller graph for this function:

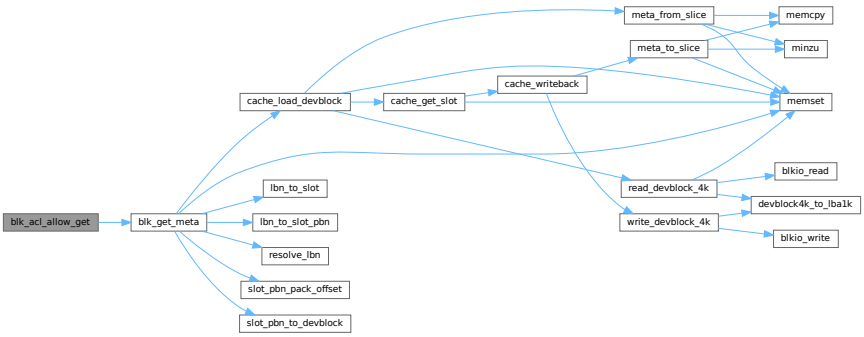
7.277.3.5 blk_acl_allow_get()

```
int blk_acl_allow_get (
            uint32_t block_num,
            uint8_t * out_allow )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, and meta.

Referenced by block_word_acl_allow_fetch().

Here is the call graph for this function:



Here is the caller graph for this function:

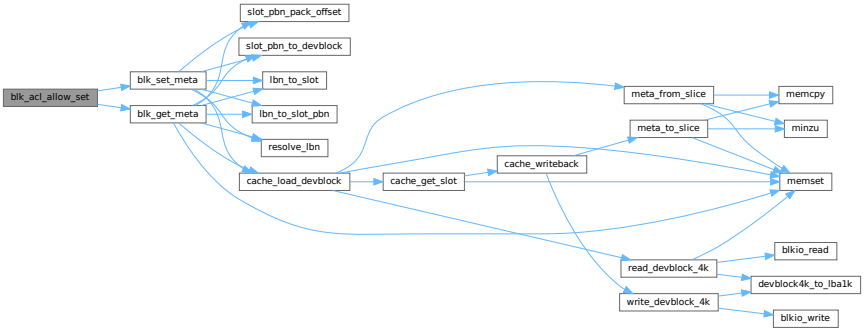
7.277.3.6 blk_acl_allow_set()

```
int blk_acl_allow_set (
            uint32_t block_num,
            uint8_t allow )
```

References blk_get_meta(), BLK_OK, blk_set_meta(), and meta.

Referenced by block_word_acl_allow_store().

Here is the call graph for this function:



Here is the caller graph for this function:

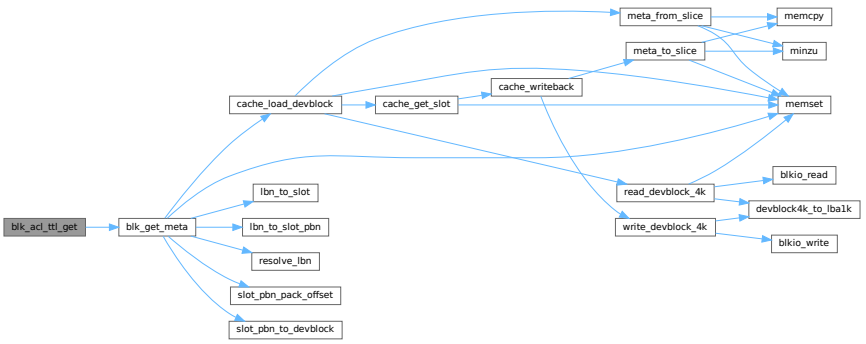
7.277.3.7 blk_acl_ttl_get()

```
int blk_acl_ttl_get (
            uint32_t block_num,
            uint32_t * out_ttl )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, and meta.

Referenced by block_word_acl_ttl_fetch().

Here is the call graph for this function:



Here is the caller graph for this function:

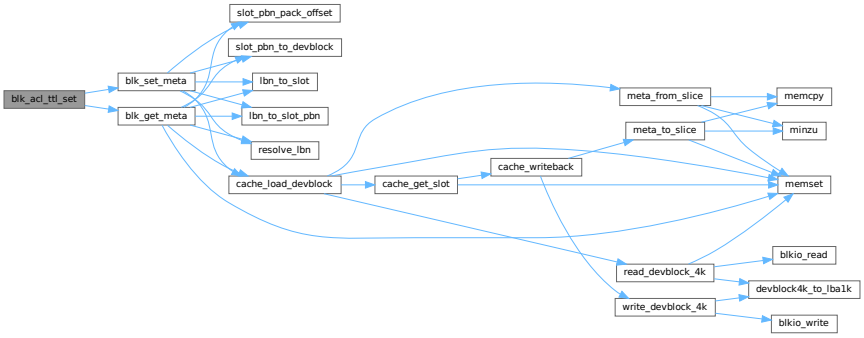
7.277.3.8 blk_acl_ttl_set()

```
int blk_acl_ttl_set (
            uint32_t block_num,
            uint32_t ttl )
```

References blk_get_meta(), BLK_OK, blk_set_meta(), and meta.

Referenced by block_word_acl_ttl_store().

Here is the call graph for this function:



Here is the caller graph for this function:

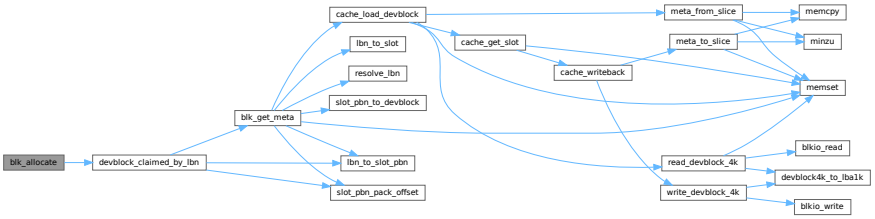
7.277.3.9 blk_allocate()

```
int blk_allocate (
            uint32_t * block_num )
```

References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_dev_slot::bam_dirty, BLK_EINVAL, BLK_ENODEV, BLK_ERANGE, BLK_FMT_PROVISIONAL, BLK_OK, blk_dev_slot::dev, devblock_claimed_by_lbn(), blk_volume_meta_t::first_free, blk_dev_slot::format_state, blk_volume_meta_t::free_blocks, g, blk_volume_meta_t::last_allocated, limit,

blk_dev_slot::next, blk_dev_slot::raw_base, blk_dev_slot::start_lbn, blk_dev_slot::user_blocks, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Here is the call graph for this function:



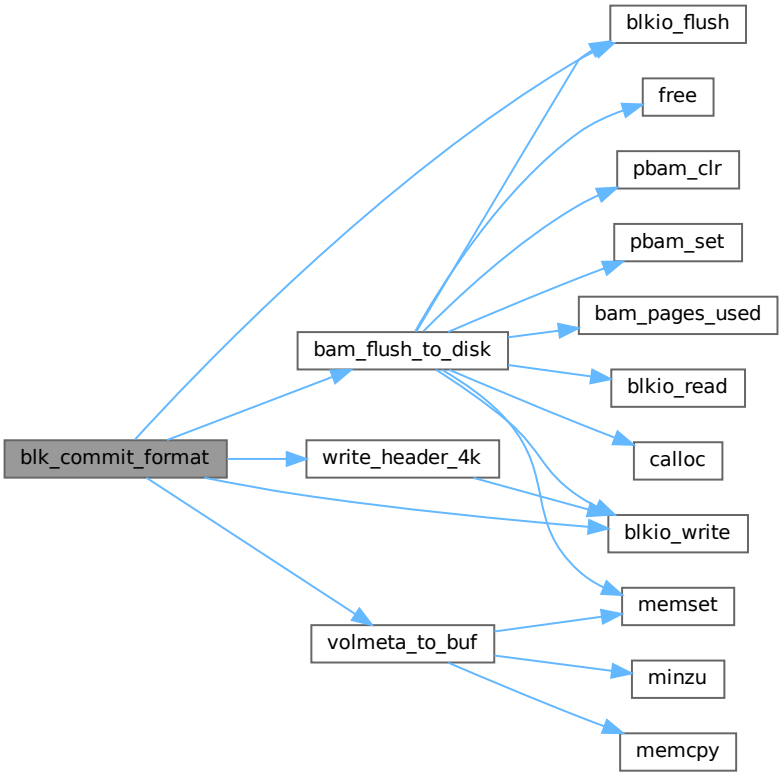
7.277.3.10 blk_commit_format()

```
static int blk_commit_format (
            blk_dev_slot_t * slot )  [static]
```

References blk_volume_meta_t::bam_devblocks, blk_dev_slot::bam_dirty, bam_flush_to_disk(), blk_volume_meta_t::bam_start, BLK_DEVICE_SECTOR, BLK_EINVAL, BLK_FMT_FORMATTED, BLK_OK, blkio_flush(), blkio_write(), blk_dev_slot::dev, blk_dev_slot::format_state, blk_volume_meta_t::reloc_devblocks, blk_volume_meta_t::reloc_start, blk_dev_slot::vol_meta, volmeta_to_buf(), and write_header_4k().

Referenced by blk_subsys_confirm_format().

Here is the call graph for this function:



Here is the caller graph for this function:

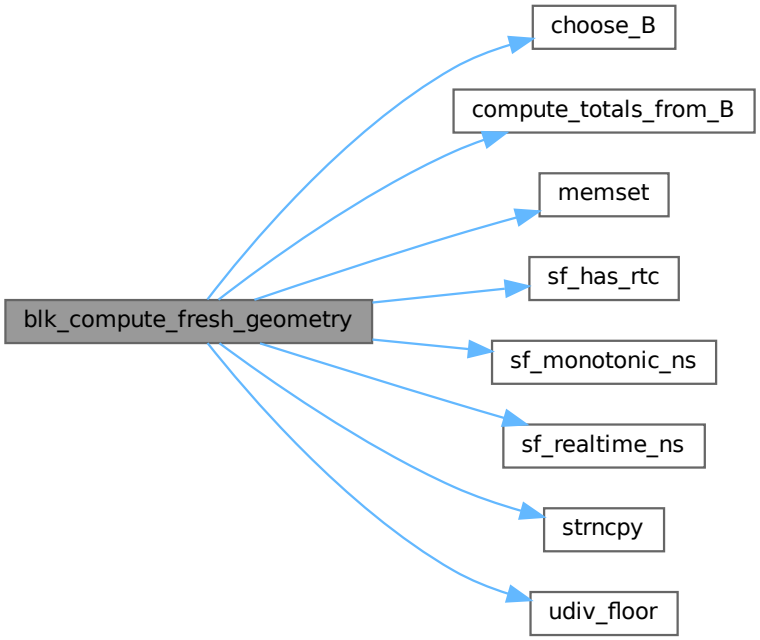
7.277.3.11 blk_compute_fresh_geometry()

```
static void blk_compute_fresh_geometry (
            blk_dev_slot_t * slot )  [static]
```

References blk_volume_meta_t::bam_devblocks, blk_volume_meta_t::bam_start, BLK_DISK_SYS_RESERVED, BLK_FORTH_SYS_RESERVED, BLK_META_FENCE_INIT, choose_B(), compute_totals_from_B(), blk_volume_meta_t::created_tim blk_volume_meta_t::devblock_base, blk_dev_slot::devblock_base_4k, blk_volume_meta_t::first_free, blk_volume_meta_t::free_block g_format_bam_for, blk_volume_meta_t::label, blk_volume_meta_t::last_allocated, blk_volume_meta_t::magic, memset(), blk_volume_meta_t::meta_fence_blocks, blk_volume_meta_t::reloc_devblocks, blk_volume_meta_t::reloc_start, blk_volume_meta_t::reserved_disk_lo, blk_volume_meta_t::reserved_ram_lo, sf_has_rtc(), sf_monotonic_ns(), sf_realtime_ns(), blk_dev_slot::start_lbn, strncpy(), blk_dev_slot::total_blkio_blocks_1k, blk_volume_meta_t::total_blocks, blk_volume_meta_t::total_devblocks, blk_volume_meta_t::total_volumes, udiv_floor(), blk_dev_slot::user_blocks, blk_volume_meta_t::version, and blk_dev_slot::vol_meta.

Referenced by blk_format_or_load_disk().

Here is the call graph for this function:



Here is the caller graph for this function:

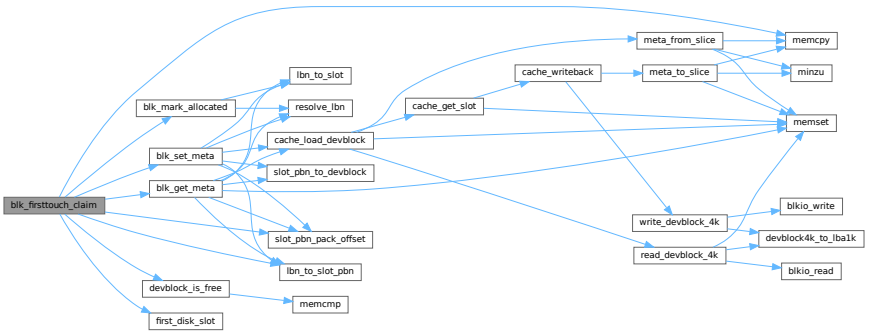
7.277.3.12 blk_firsttouch_claim()

```
int blk_firsttouch_claim (
            const uint8_t owner_fp[8],
            uint32_t count,
            uint32_t * out_chain_head )
```

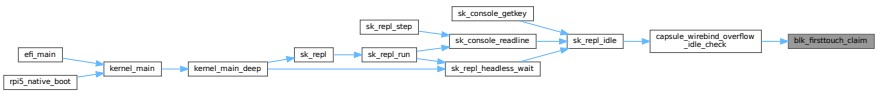
References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ENOSPC, BLK_FIRSTTOUCH_MAX_CLAIM, BLK_FLAG_CLAIMED, blk_get_meta(), blk_mark_allocated(), BLK_OK, BLK_PACK_RATIO, blk_set_meta(), count, devblock_is_free(), first_disk_slot(), blk_volume_meta_t::flags, lbn_to_slot_pbn(), memcpy(), meta, slot_pbn_pack_offset(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by capsule_wirebind_overflow_idle_check().

Here is the call graph for this function:



Here is the caller graph for this function:

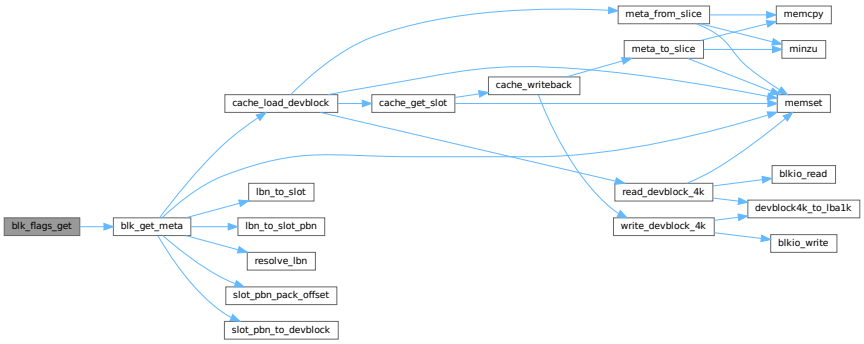


7.277.3.13 blk_flags_get()

```
int blk_flags_get (
            uint32_t block_num,
            uint64_t * out_flags )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, blk_volume_meta_t::flags, and meta.

Here is the call graph for this function:

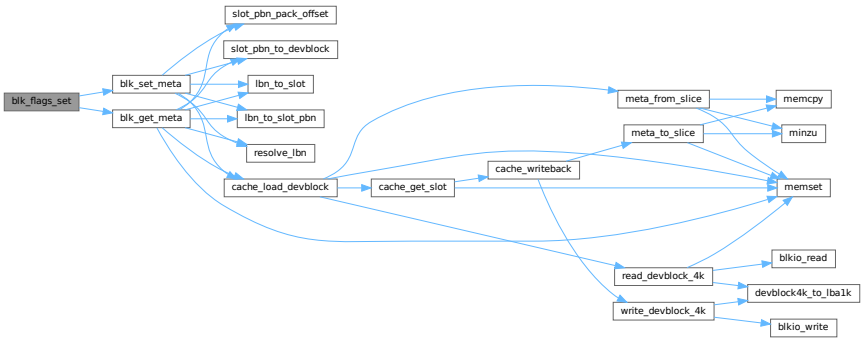


7.277.3.14 blk_flags_set()

```
int blk_flags_set (
            uint32_t block_num,
            uint64_t flags )
```

References blk_get_meta(), BLK_OK, blk_set_meta(), blk_volume_meta_t::flags, flags, and meta.

Here is the call graph for this function:



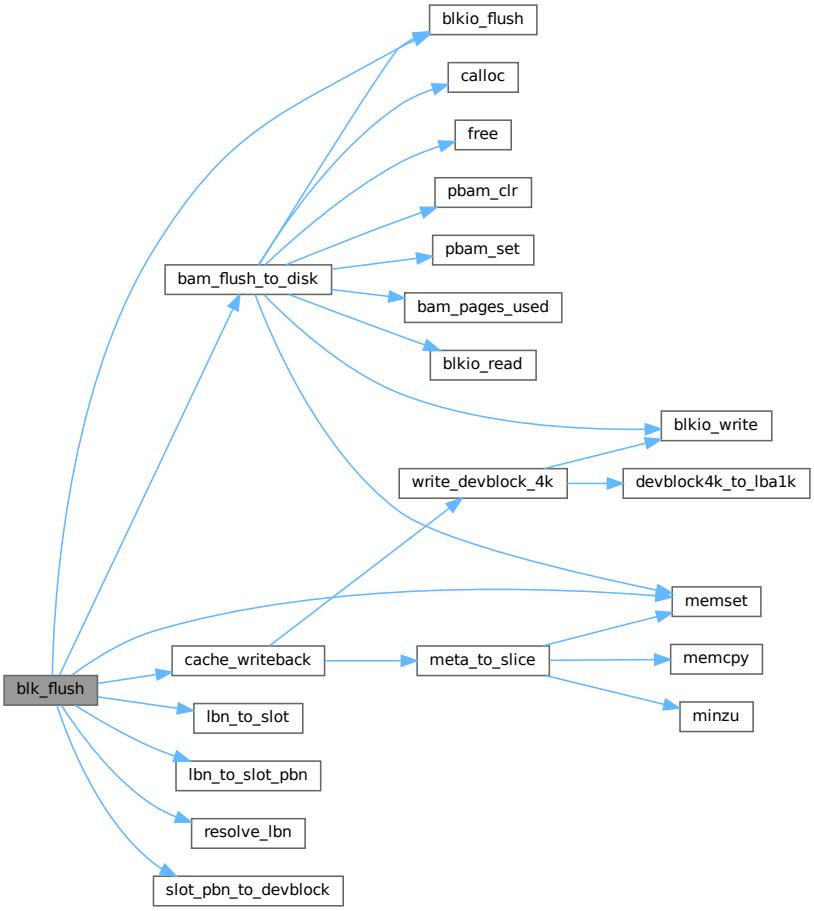
7.277.3.15 blk_flush()

```
int blk_flush (
            uint32_t block_num )
```

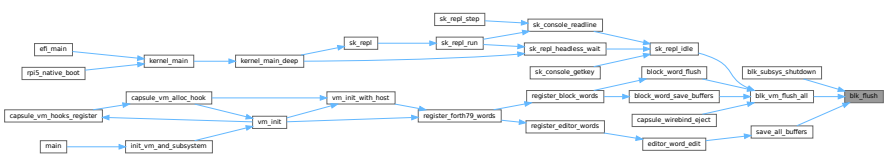
References blk_dev_slot::bam, bam_flush_to_disk(), BLK_ENODEV, BLK_ERANGE, BLK_FORTH_SYS_RESERVED, BLK_OK, BLK_RAM_BLOCKS, blkio_flush(), blk_dev_slot::cache, cache_writeback(), blk_dev_slot::dev, cache_slot_t::devblock, blk_bam_entry_t::dirty, cache_slot_t::dirty, DISK_CACHE_SLOTS, g, lbn_to_slot(), lbn_to_slot_pbn(), memset(), cache_slot_t::meta_dirty, blk_dev_slot::next, blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_to_devblock(), blk_dev_slot::start_lbn, blk_dev_slot::user_blocks, and cache_slot_t::valid.

Referenced by blk_subsys_shutdown(), blk_vm_flush_all(), and save_all_buffers().

Here is the call graph for this function:



Here is the caller graph for this function:



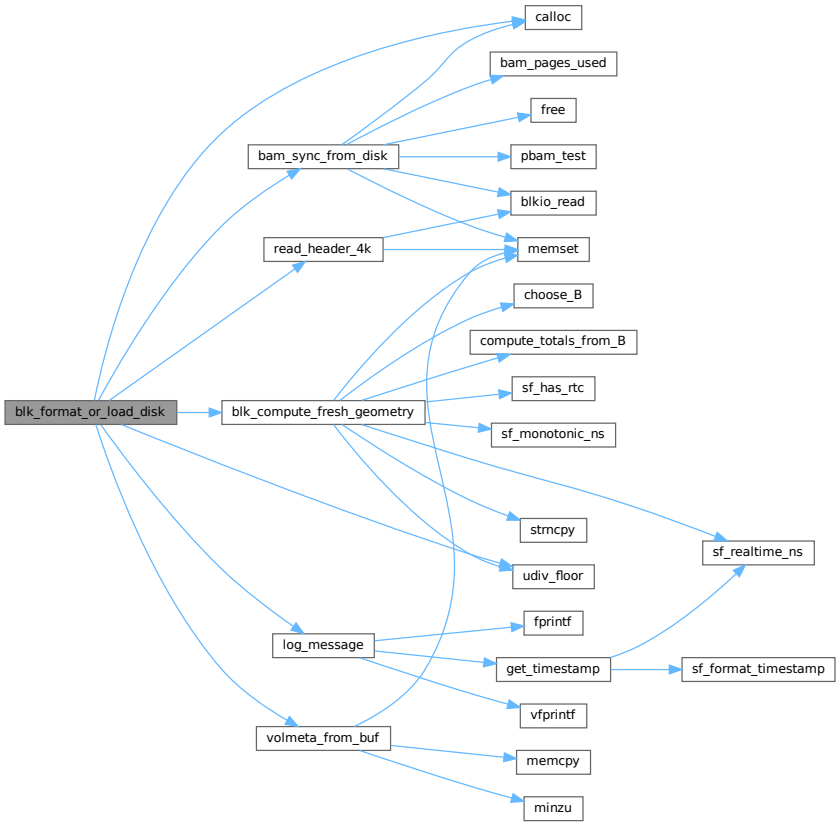
7.277.3.16 blk_format_or_load_disk()

```
static int blk_format_or_load_disk (
            blk_dev_slot_t * slot )  [static]
```

References blk_dev_slot::bam, blk_volume_meta_t::bam_devblocks, bam_sync_from_disk(), blk_compute_fresh_geometry(), BLK_DEVICE_SECTOR, BLK_DISK_SYS_RESERVED, BLK_ENOMEM, BLK_FMT_FORMATTED, BLK_FMT_PROVISIONAL, BLK_OK, calloc(), blk_volume_meta_t::devblock_base, blk_dev_slot::devblock_base_4k, blk_dev_slot::format_state, log_message(), LOG_WARN, blk_volume_meta_t::magic, read_header_4k(), blk_dev_slot::total_blkio_blocks_1k, blk_volume_meta_t::total_blocks, blk_volume_meta_t::total_devblocks, udiv_floor(), blk_dev_slot::user_blocks, blk_volume_meta_t::version, blk_dev_slot::vol_meta, and volmeta_from_buf().

Referenced by blk_subsys_attach_device().

Here is the call graph for this function:



Here is the caller graph for this function:

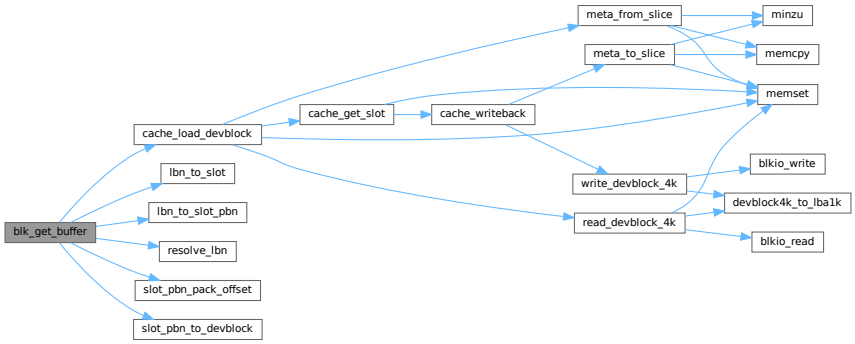
7.277.3.17 blk_get_buffer()

```
uint8_t * blk_get_buffer (
            uint32_t block_num,
            int writable )
```

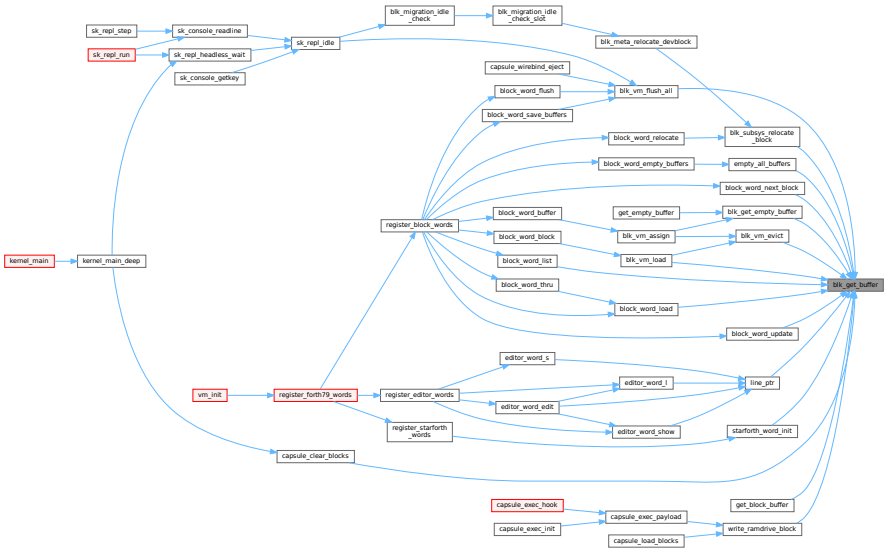
References blk_bam_entry_t::allocated, blk_dev_slot::bam, BLK_FMT_PROVISIONAL, BLK_FORTH_SIZE, BLK_FORTH_SYS_RESERVED, BLK_RAM_BLOCKS, cache_load_devblock(), cache_slot_t::data, blk_bam_entry_t::dirty, cache_slot_t::dirty, blk_dev_slot::format_state, g, lbn_to_slot(), lbn_to_slot_pbn(), blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_pack_offset(), slot_pbn_to_devblock(), and blk_dev_slot::start_lbn.

Referenced by blk_get_empty_buffer(), blk_subsys_relocate_block(), blk_vm_evict(), blk_vm_flush_all(), blk_vm_load(), block_word_list(), block_word_load(), block_word_next_block(), block_word_update(), capsule_clear_blocks(), empty_all_buffers(), get_block_buffer(), line_ptr(), starforth_word_init(), and write_ramdrive_block().

Here is the call graph for this function:



Here is the caller graph for this function:



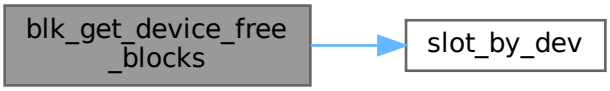
7.277.3.18 blk_get_device_free_blocks()

```
int blk_get_device_free_blocks (
            struct blkio_dev * dev,
            uint64_t * out_free,
            uint64_t * out_total )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, blk_volume_meta_t::free_blocks, blk_dev_slot::raw_base, slot_by_dev(), blk_volume_meta_t::total_blocks, and blk_dev_slot::vol_meta.

Referenced by capsule_wirebind_overflow_idle_check().

Here is the call graph for this function:



Here is the caller graph for this function:

7.277.3.19 blk_get_device_range()

```
int blk_get_device_range (
            struct blkio_dev * dev,
            uint32_t * out_start_lbn,
            uint32_t * out_count )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, slot_by_dev(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by sk_v4_genesis().

Here is the call graph for this function:

Here is the caller graph for this function:

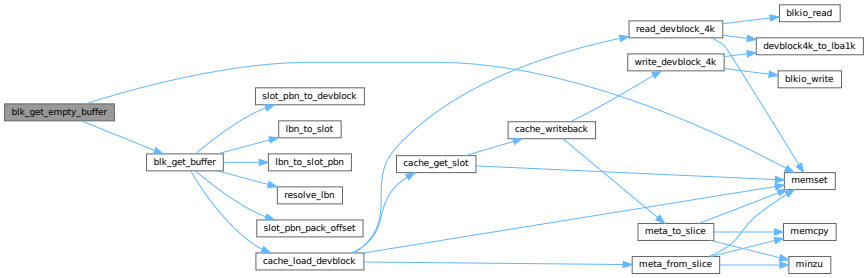
7.277.3.20 blk_get_empty_buffer()

```
uint8_t * blk_get_empty_buffer (
            uint32_t block_num )
```

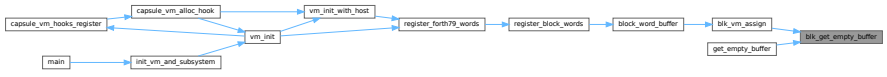
References BLK_FORTH_SIZE, blk_get_buffer(), and memset().

Referenced by blk_vm_assign(), and get_empty_buffer().

Here is the call graph for this function:



Here is the caller graph for this function:

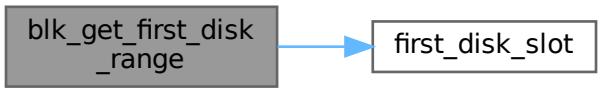


7.277.3.21 blk_get_first_disk_range()

```
int blk_get_first_disk_range (
            uint32_t * out_start_lbn,
            uint32_t * out_count )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, first_disk_slot(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Here is the call graph for this function:



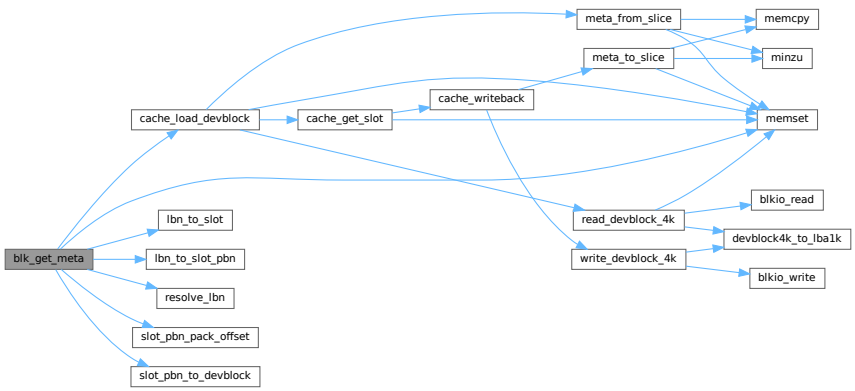
7.277.3.22 blk_get_meta()

```
int blk_get_meta (
            uint32_t block_num,
            blk_meta_t * meta )
```

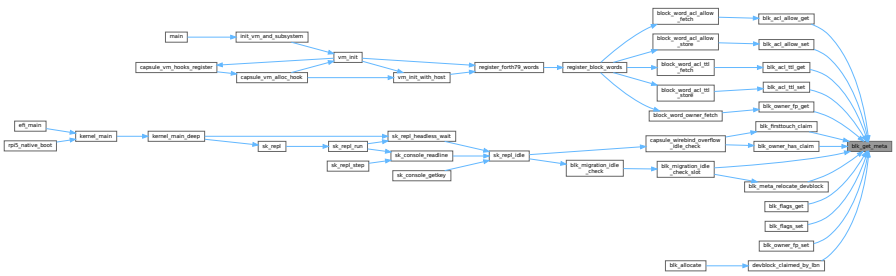
References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ERANGE, BLK_OK, cache_load_devblock(), g, lbn_to_slot(), lbn_to_slot_pbn(), blk_volume_meta_t::magic, memset(), meta, cache_slot_t::meta, blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_pack_offset(), and slot_pbn_to_devblock().

Referenced by blk_acl_allow_get(), blk_acl_allow_set(), blk_acl_ttl_get(), blk_acl_ttl_set(), blk_firsttouch_claim(), blk_flags_get(), blk_flags_set(), blk_meta_relocate_devblock(), blk_migration_idle_check_slot(), blk_owner_fp_get(), blk_owner_fp_set(), blk_owner_has_claim(), and devblock_claimed_by_lbn().

Here is the call graph for this function:



Here is the caller graph for this function:



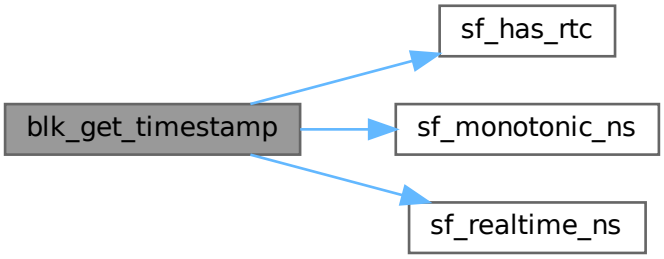
7.277.3.23 blk_get_timestamp()

```
static uint64_t blk_get_timestamp (
            void  )  [inline], [static]
```

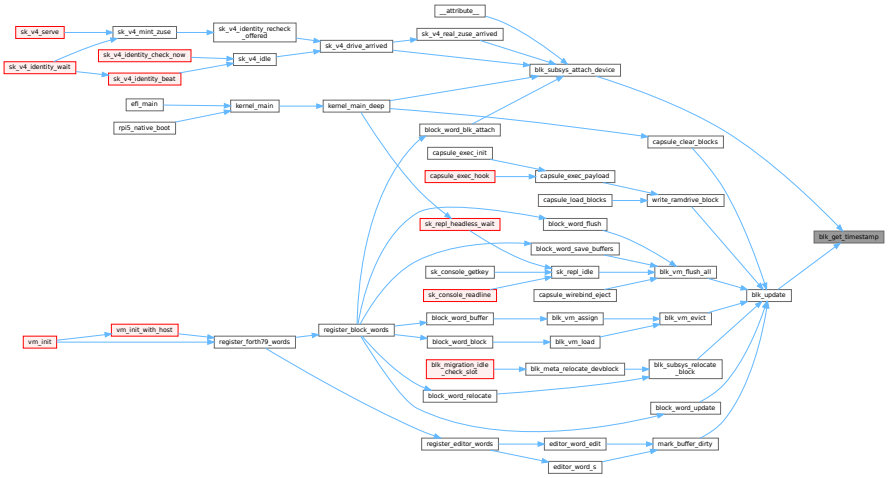
References sf_has_rtc(), sf_monotonic_ns(), and sf_realtime_ns().

Referenced by blk_subsys_attach_device(), and blk_update().

Here is the call graph for this function:



Here is the caller graph for this function:



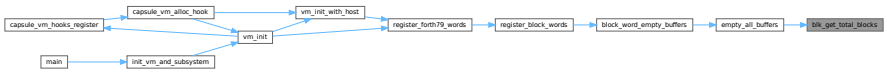
7.277.3.24 blk_get_total_blocks()

```
uint32_t blk_get_total_blocks (
            void  )
```

References g.

Referenced by empty_all_buffers().

Here is the caller graph for this function:

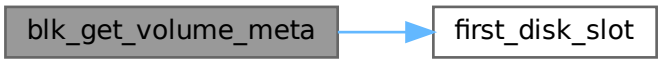


7.277.3.25 blk_get_volume_meta()

```
int blk_get_volume_meta (
            blk_volume_meta_t * meta )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, first_disk_slot(), meta, and blk_dev_slot::vol_meta.

Here is the call graph for this function:

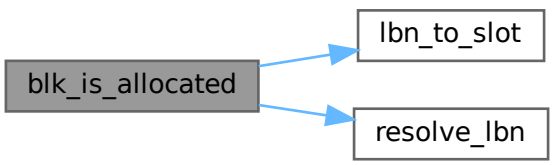


7.277.3.26 blk_is_allocated()

```
int blk_is_allocated (
            uint32_t block_num )
```

References blk_bam_entry_t::allocated, blk_dev_slot::bam, BLK_EINVAL, BLK_ENODEV, g, lbn_to_slot(), resolve_lbn(), and blk_dev_slot::start_lbn.

Here is the call graph for this function:



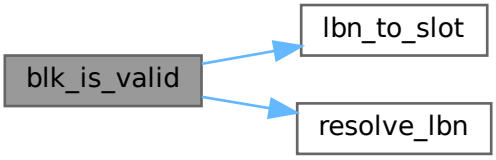
7.277.3.27 blk_is_valid()

```
int blk_is_valid (
            uint32_t block_num )
```

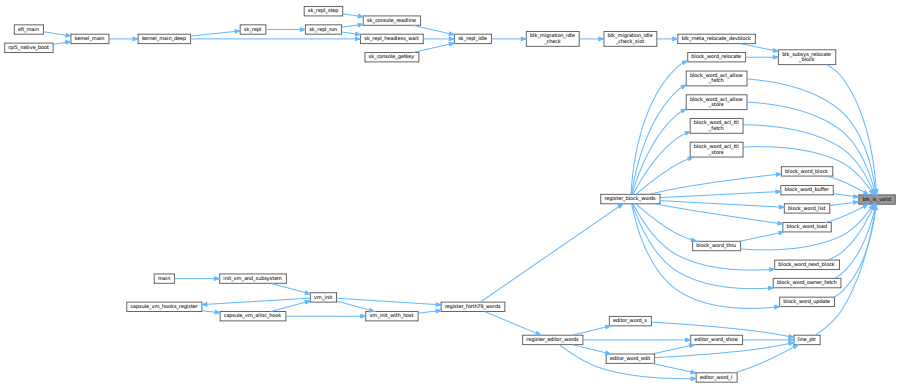
References g, lbn_to_slot(), and resolve_lbn().

Referenced by blk_subsys_relocate_block(), block_word_acl_allow_fetch(), block_word_acl_allow_store(), block_word_acl_ttl_fetch(), block_word_acl_ttl_store(), block_word_block(), block_word_buffer(), block_word_list(), block_word_load(), block_word_next_block(), block_word_owner_fetch(), block_word_thru(), block_word_update(), and line_ptr().

Here is the call graph for this function:



Here is the caller graph for this function:



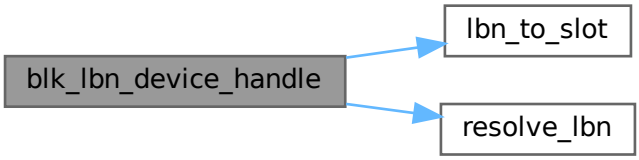
7.277.3.28 blk_lbn_device_handle()

```
const void * blk_lbn_device_handle (
            uint32_t lbn )
```

References lbn_to_slot(), and resolve_lbn().

Referenced by blk_vm_is_cross_device_copy().

Here is the call graph for this function:



Here is the caller graph for this function:

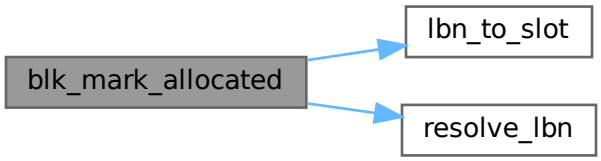
7.277.3.29 blk_mark_allocated()

```
int blk_mark_allocated (
            uint32_t block_num )
```

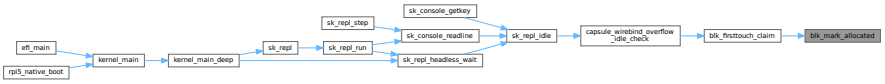
References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_dev_slot::bam_dirty, BLK_EINVAL, BLK_ENODEV, BLK_ERANGE, BLK_ERESERVED, BLK_FMT_PROVISIONAL, BLK_OK, blk_dev_slot::format_state, blk_volume_meta_t::free_blocks, lbn_to_slot(), blk_dev_slot::raw_base, resolve_lbn(), blk_dev_slot::start_lbn, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Referenced by blk_firsttouch_claim().

Here is the call graph for this function:



Here is the caller graph for this function:



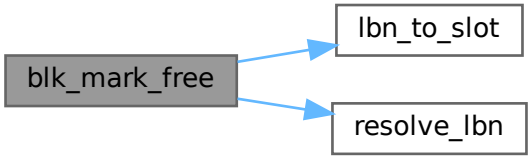
7.277.3.30 blk_mark_free()

```
int blk_mark_free (
            uint32_t block_num )
```

References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_dev_slot::bam_dirty, BLK_EINVAL, BLK_ENODEV, BLK_ERANGE, BLK_ERESERVED, BLK_FMT_PROVISIONAL, BLK_OK, blk_volume_meta_t::first_free, blk_dev_slot::format_state, blk_volume_meta_t::free_blocks, g, lbn_to_slot(), blk_dev_slot::raw_base, resolve_lbn(), blk_dev_slot::start_lbn, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Referenced by blk_subsys_relocate_block().

Here is the call graph for this function:



Here is the caller graph for this function:

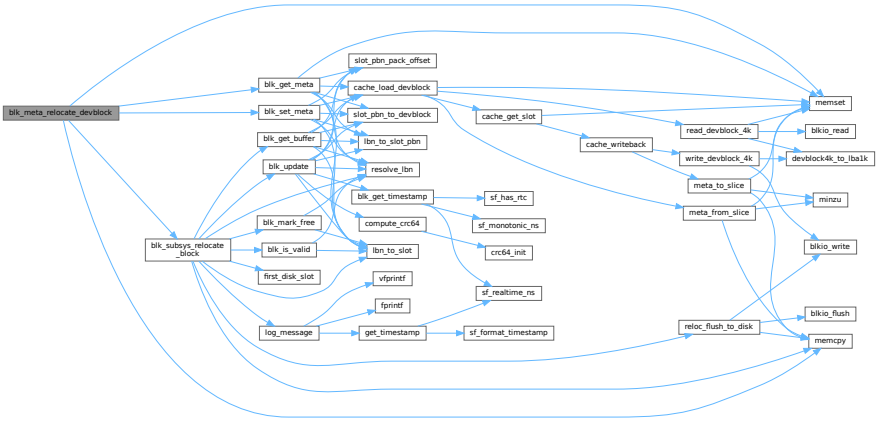
7.277.3.31 blk_meta_relocate_devblock()

```
int blk_meta_relocate_devblock (
            uint32_t home_devblock,
            uint32_t target_devblock )
```

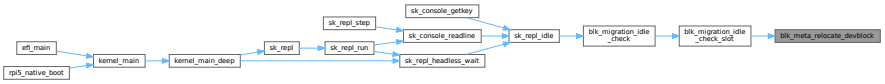
References blk_meta_t::acl_allow, blk_meta_t::acl_ttl, BLK_EINVAL, BLK_EIO, BLK_FLAG_CLAIMED, BLK_FLAG_MIGRATING, blk_get_meta(), BLK_OK, BLK_PACK_RATIO, blk_set_meta(), blk_subsys_relocate_block(), blk_meta_t::flags, memcpy(), memset(), and blk_meta_t::owner_fp.

Referenced by blk_migration_idle_check_slot().

Here is the call graph for this function:



Here is the caller graph for this function:



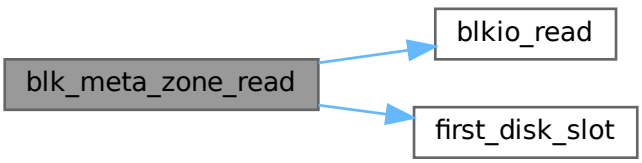
7.277.3.32 blk_meta_zone_read()

```
int blk_meta_zone_read (
            uint32_t devblock_from_top,
            uint8_t buf[4096] )
```

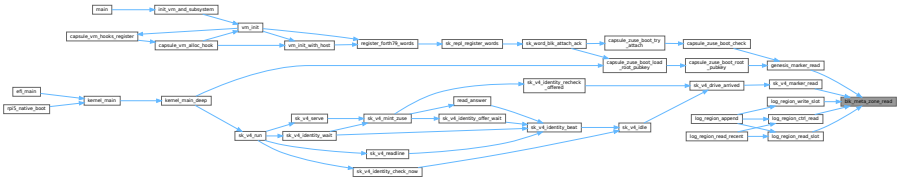
References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_OK, BLKIO_OK, blkio_read(), blk_dev_slot::dev, first_disk_slot(), blk_volume_meta_t::meta_fence_blocks, blk_volume_meta_t::total_devblocks, and blk_dev_slot::vol_meta.

Referenced by genesis_marker_read(), log_region_ctrl_read(), log_region_read_slot(), log_region_write_slot(), and sk_v4_marker_read().

Here is the call graph for this function:



Here is the caller graph for this function:



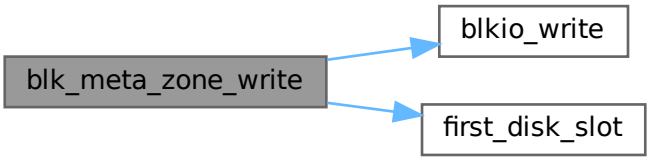
7.277.3.33 blk_meta_zone_write()

```
int blk_meta_zone_write (
            uint32_t devblock_from_top,
            const uint8_t buf[4096] )
```

References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_OK, BLKIO_OK, blkio_write(), blk_dev_slot::dev, first_disk_slot(), blk_volume_meta_t::meta_fence_blocks, blk_volume_meta_t::total_devblocks, and blk_dev_slot::vol_meta.

Referenced by capsule_zuse_boot_try_attach(), log_region_ctrl_write(), log_region_write_slot(), and sk_v4_marker_write().

Here is the call graph for this function:



Here is the caller graph for this function:

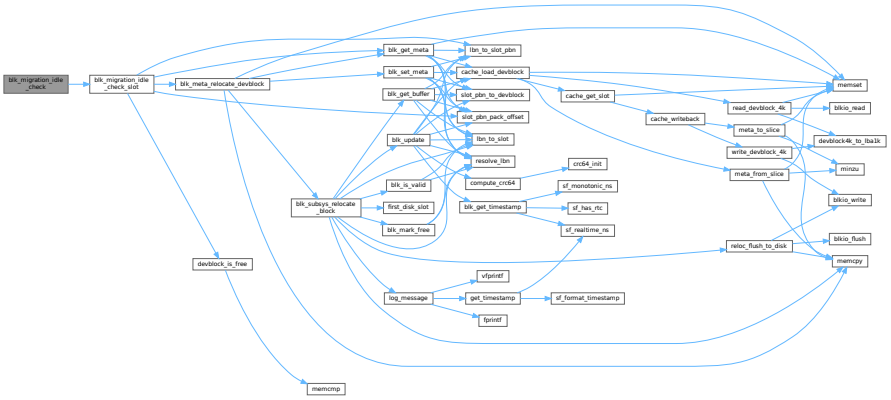
7.277.3.34 blk_migration_idle_check()

```
void blk_migration_idle_check (
            void  )
```

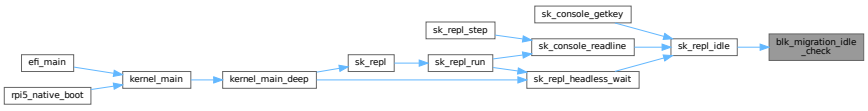
References blk_migration_idle_check_slot(), blk_dev_slot::dev, g, and blk_dev_slot::next.

Referenced by sk_repl_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



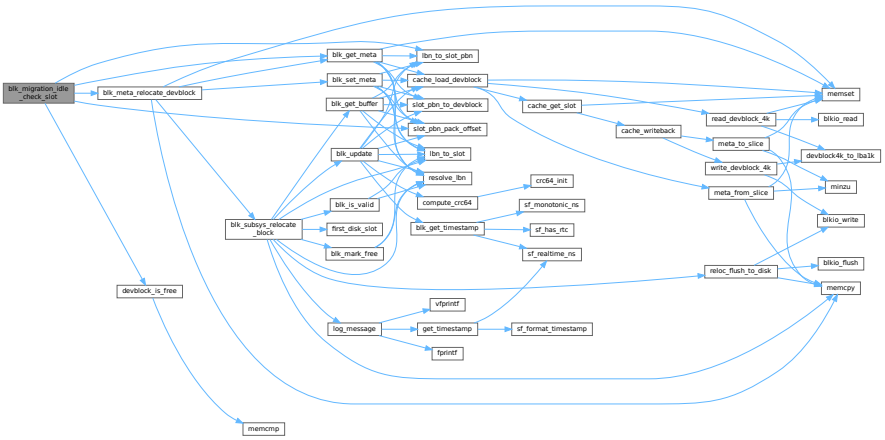
7.277.3.35 blk_migration_idle_check_slot()

```
static void blk_migration_idle_check_slot (
            blk_dev_slot_t * slot )  [static]
```

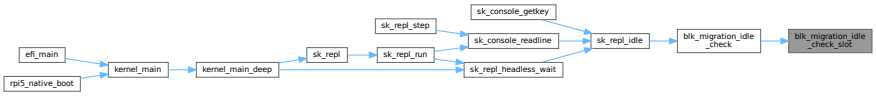
References BLK_FLAG_CLAIMED, BLK_FLAG_MIGRATING, blk_get_meta(), blk_meta_relocate_devblock(), BLK_OK, devblock_is_free(), blk_volume_meta_t::flags, blk_dev_slot::has_ever_claimed, lbn_to_slot_pbn(), meta, MIGRATION_SCAN_BUDGET, blk_dev_slot::migration_scan_lbn, MIGRATION_WEAR_THRESHOLD, slot_pbn_pack_offset(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by blk_migration_idle_check().

Here is the call graph for this function:



Here is the caller graph for this function:



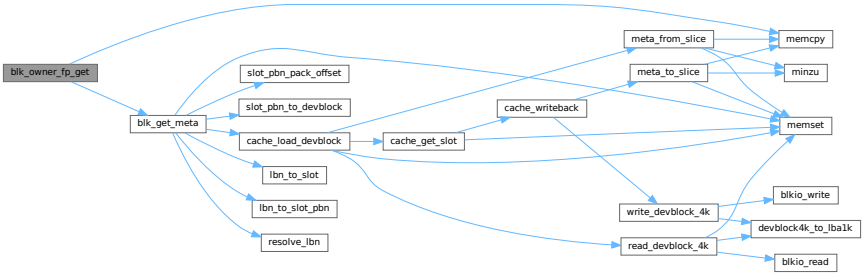
7.277.3.36 blk_owner_fp_get()

```
int blk_owner_fp_get (
            uint32_t block_num,
            uint8_t out_fp[8] )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, memcpy(), and meta.

Referenced by block_word_owner_fetch().

Here is the call graph for this function:



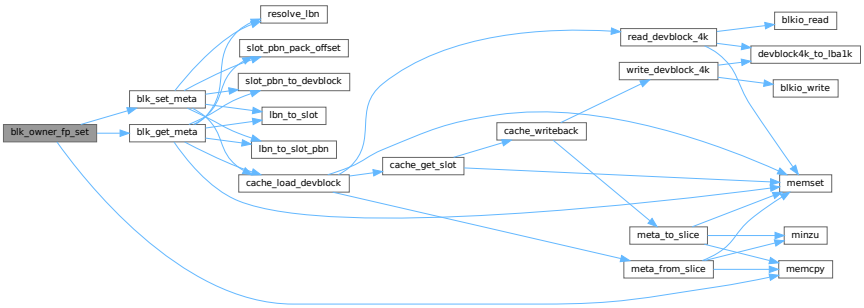
Here is the caller graph for this function:

7.277.3.37 blk_owner_fp_set()

```
int blk_owner_fp_set (
            uint32_t block_num,
            const uint8_t fp[8] )
```

References BLK_EINVAL, blk_get_meta(), BLK_OK, blk_set_meta(), memcpy(), and meta.

Here is the call graph for this function:



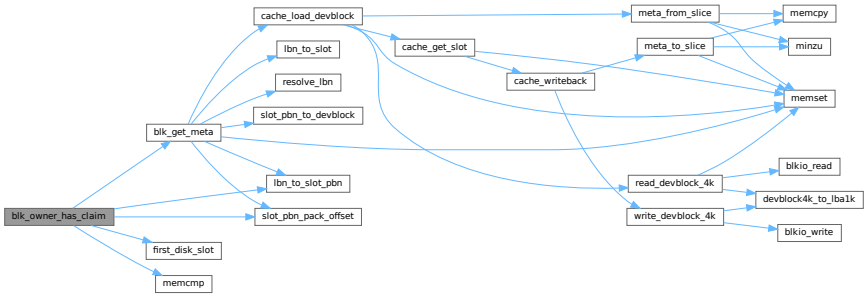
7.277.3.38 blk_owner_has_claim()

```
int blk_owner_has_claim (
            const uint8_t owner_fp[8] )
```

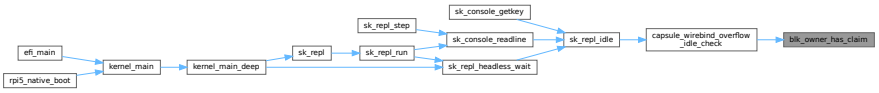
References BLK_FLAG_CLAIMED, blk_get_meta(), BLK_OK, first_disk_slot(), blk_volume_meta_t::flags, lbn_to_slot_pbn(), memcmp(), meta, slot_pbn_pack_offset(), blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by capsule_wirebind_overflow_idle_check().

Here is the call graph for this function:



Here is the caller graph for this function:



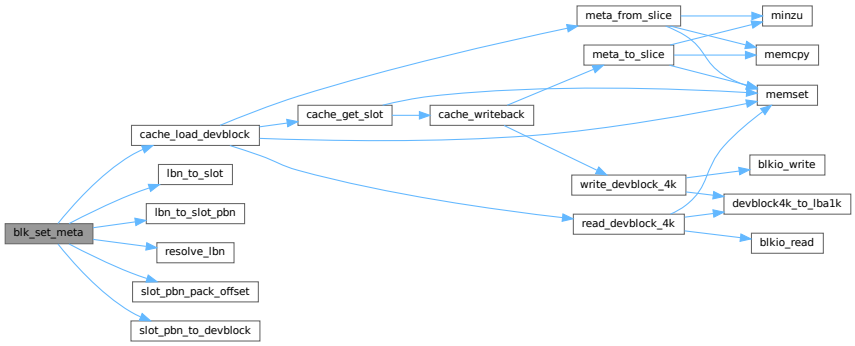
7.277.3.39 blk_set_meta()

```
int blk_set_meta (
            uint32_t block_num,
            const blk_meta_t * meta )
```

References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ERANGE, BLK_FLAG_CLAIMED, BLK_OK, cache_load_devblock(), blk_volume_meta_t::flags, g, blk_dev_slot::has_ever_claimed, lbn_to_slot(), lbn_to_slot_pbn(), meta, cache_slot_t::meta, cache_slot_t::meta_dirty, blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_pack_offset(), and slot_pbn_to_devblock().

Referenced by blk_acl_allow_set(), blk_acl_ttl_set(), blk_firsttouch_claim(), blk_flags_set(), blk_meta_relocate_devblock(), and blk_owner_fp_set().

Here is the call graph for this function:



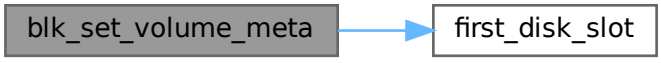
Here is the caller graph for this function:

7.277.3.40 blk_set_volume_meta()

```
int blk_set_volume_meta (
            const blk_volume_meta_t * meta )
```

References BLK_EINVAL, BLK_ENODEV, BLK_OK, first_disk_slot(), meta, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Here is the call graph for this function:

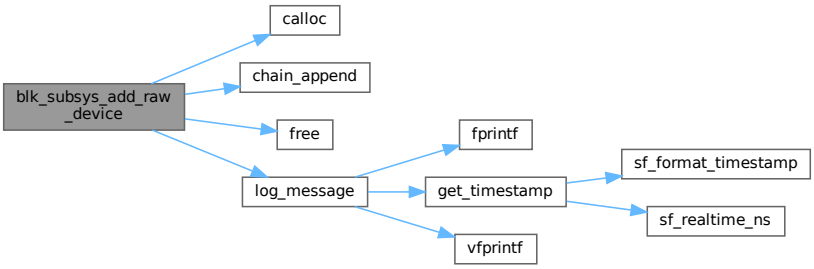


7.277.3.41 blk_subsys_add_raw_device()

```
int blk_subsys_add_raw_device (
            uint8_t * buf,
            uint32_t nblocks )
```

References blk_dev_slot::bam, BLK_EINVAL, BLK_ENODEV, BLK_ENOMEM, BLK_OK, calloc(), chain_append(), free(), g, LOG_INFO, log_message(), blk_dev_slot::raw_base, blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Here is the call graph for this function:



7.277.3.42 blk_subsys_attach_device()

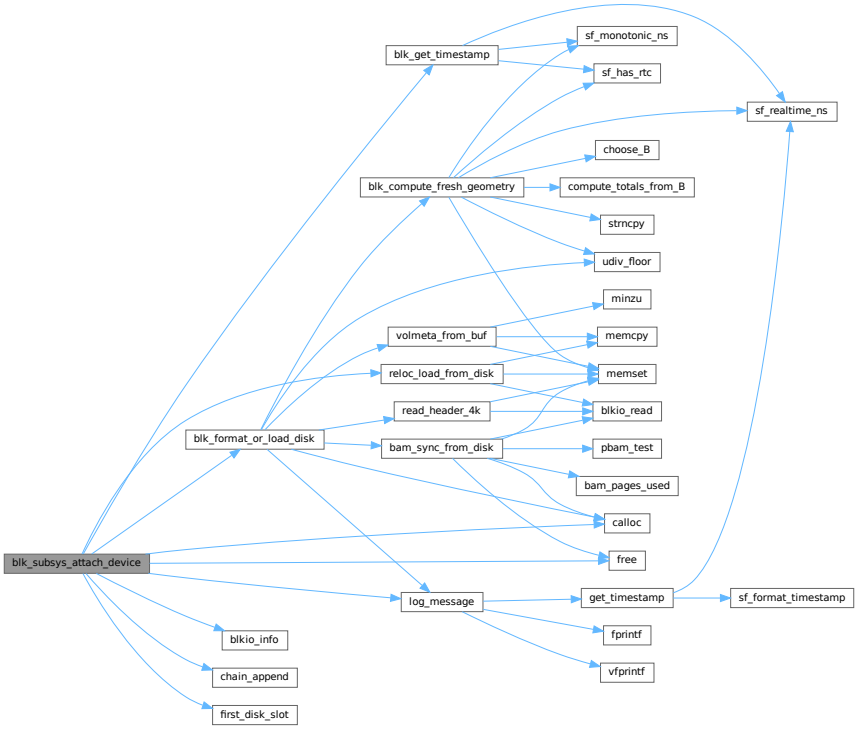
```
int blk_subsys_attach_device (
            struct blkio_dev * dev )
```

References blk_dev_slot::bam, blk_volume_meta_t::bam_devblocks, BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ENOMEM, BLK_FMT_FORMATTED, blk_format_or_load_disk(), blk_get_timestamp(), BLK_OK, blkio_info(),

BLKIO_OK, calloc(), chain_append(), blk_dev_slot::dev, blk_volume_meta_t::devblock_base, first_disk_slot(), blk_dev_slot::format_state, free(), blk_volume_meta_t::free_blocks, g, blk_volume_meta_t::label, LOG_INFO, log_message(), blk_volume_meta_t::meta_fence_blocks, blk_volume_meta_t::mounted_time, reloc_load_from_disk(), blk_dev_slot::start_lbn, blk_dev_slot::total_blkio_blocks_1k, blkio_info_t::total_blocks, blk_volume_meta_t::total_blocks, blk_volume_meta_t::total_devblocks, blk_dev_slot::user_blocks, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Referenced by __attribute__(), block_word_blk_attach(), kernel_main_deep(), sk_v4_drive_arrived(), and sk_v4_real_zuse_arrived().

Here is the call graph for this function:



Here is the caller graph for this function:

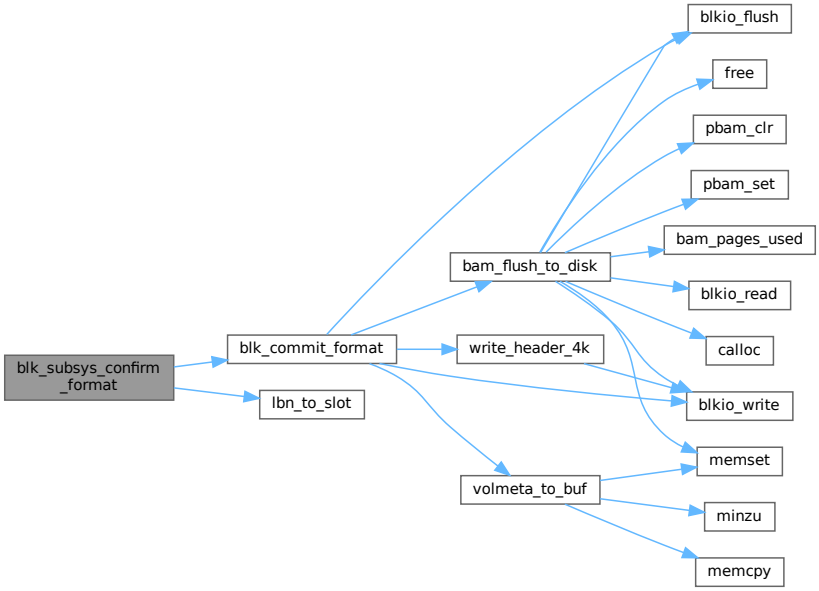
7.277.3.43 blk_subsys_confirm_format()

```
int blk_subsys_confirm_format (
            uint32_t lbn )
```

References blk_commit_format(), BLK_ENODEV, BLK_ERANGE, blk_dev_slot::dev, g, and lbn_to_slot().

Referenced by block_word_confirm_format(), and sk_v4_genesis().

Here is the call graph for this function:



Here is the caller graph for this function:

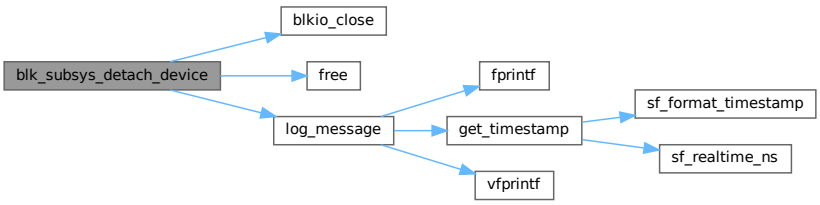
7.277.3.44 blk_subsys_detach_device()

```
int blk_subsys_detach_device (
            struct blkio_dev * dev )
```

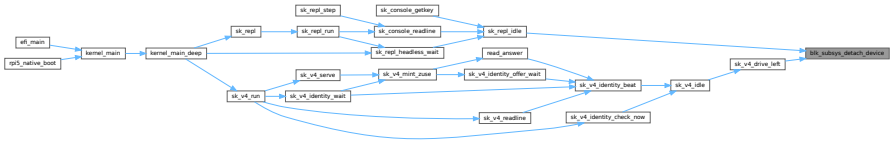
References blk_dev_slot::bam, BLK_EINVAL, BLK_ENODEV, BLK_OK, blkio_close(), blk_dev_slot::dev, free(), g, blk_volume_meta_t::label, LOG_INFO, log_message(), LOG_WARN, blk_dev_slot::next, blk_dev_slot::start_lbn, blk_dev_slot::user_blocks, and blk_dev_slot::vol_meta.

Referenced by sk_repl_idle(), and sk_v4_drive_left().

Here is the call graph for this function:



Here is the caller graph for this function:



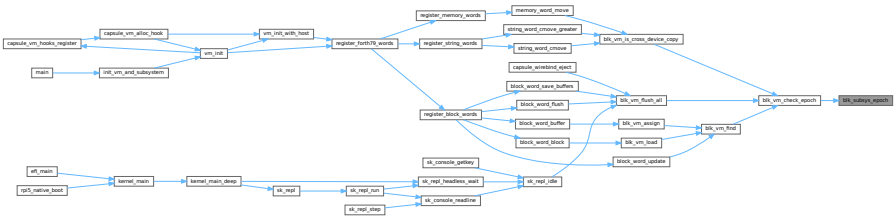
7.277.3.45 blk_subsys_epoch()

```
uint64_t blk_subsys_epoch (
            void  )
```

References g.

Referenced by blk_vm_check_epoch().

Here is the caller graph for this function:



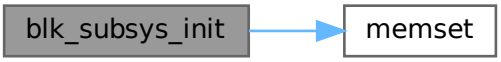
7.277.3.46 blk_subsys_init()

```
int blk_subsys_init (
            uint8_t * ram_base,
            size_t ram_size )
```

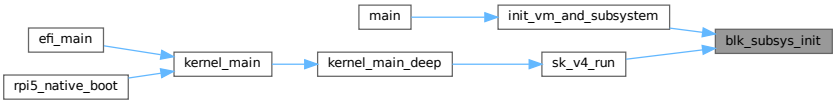
References BLK_EINVAL, BLK_FORTH_SIZE, BLK_FORTH_SYS_RESERVED, BLK_OK, BLK_RAM_BLOCKS, g, memset(), ram_base, and ram_size.

Referenced by init_vm_and_subsystem(), and sk_v4_run().

Here is the call graph for this function:



Here is the caller graph for this function:



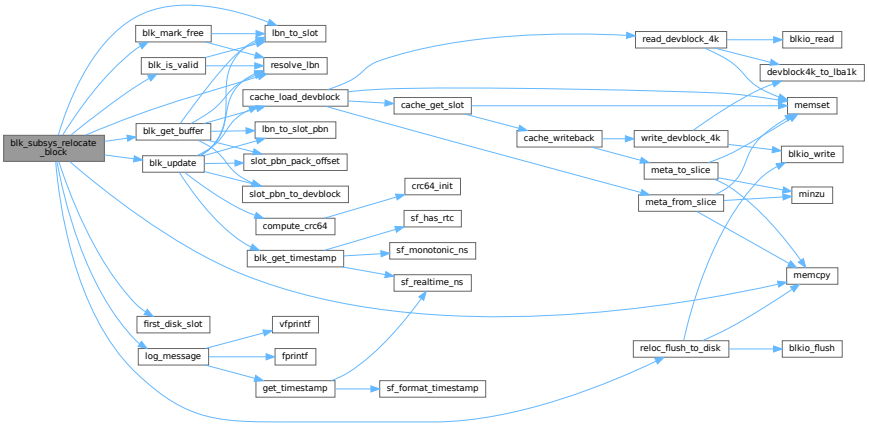
7.277.3.47 blk_subsys_relocate_block()

```
int blk_subsys_relocate_block (
            uint32_t home_lbn,
            uint32_t target_lbn )
```

References BLK_EINVAL, BLK_EIO, BLK_ENODEV, BLK_ENOMEM, BLK_ERANGE, BLK_FORTH_SIZE, blk_get_buffer(), blk_is_valid(), blk_mark_free(), BLK_OK, BLK_RELOC_MAX_ENTRIES, blk_update(), first_disk_slot(), g, lbn_to_slot(), LOG_INFO, log_message(), LOG_WARN, memcpy(), reloc_flush_to_disk(), and resolve_lbn().

Referenced by blk_meta_relocate_devblock(), and block_word_relocate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.277.3.48 blk_subsys_set_format_bam_for()

```
void blk_subsys_set_format_bam_for (
            uint64_t total_devblocks )
```

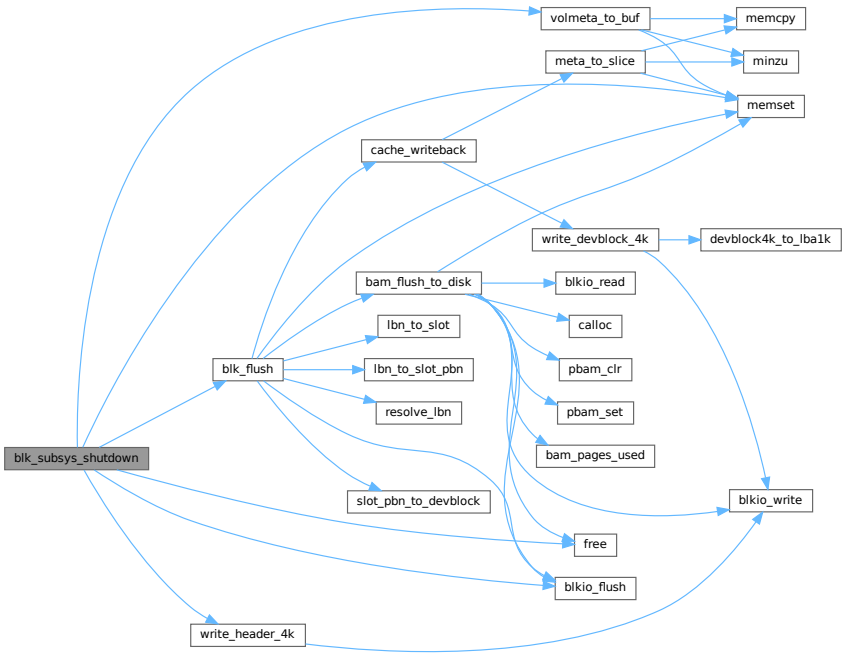
References g_format_bam_for.

7.277.3.49 blk_subsys_shutdown()

```
int blk_subsys_shutdown (
            void  )
```

References blk_dev_slot::bam, BLK_DEVICE_SECTOR, blk_flush(), BLK_FMT_FORMATTED, BLK_OK, blkio_flush(), blk_dev_slot::dev, blk_dev_slot::format_state, free(), g, memset(), blk_dev_slot::next, blk_dev_slot::vol_meta, blk_dev_slot::vol_meta_dirty, volmeta_to_buf(), and write_header_4k().

Here is the call graph for this function:



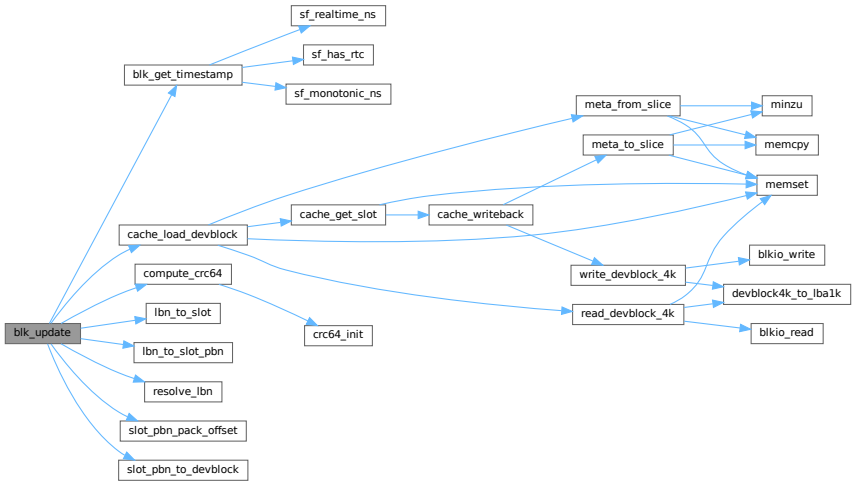
7.277.3.50 blk_update()

```
int blk_update (
            uint32_t block_num )
```

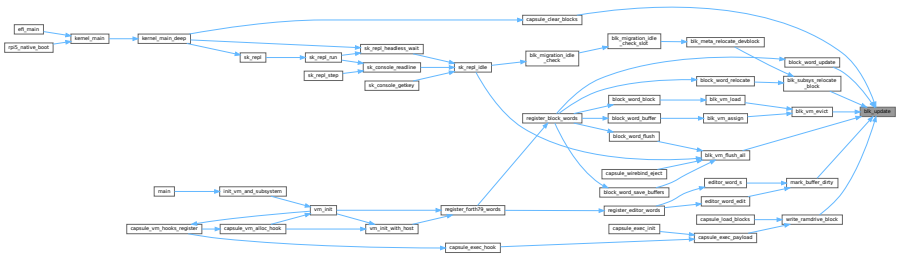
References blk_bam_entry_t::allocated, blk_dev_slot::bam, blk_dev_slot::bam_dirty, BLK_EIO, BLK_ENODEV, BLK_ERANGE, BLK_ERESERVED, BLK_FMT_PROVISIONAL, BLK_FORTH_SIZE, BLK_FORTH_SYS_RESERVED, blk_get_timestamp(), BLK_OK, BLK_RAM_BLOCKS, cache_load_devblock(), blk_meta_t::checksum, compute_crc64(), blk_meta_t::created_time, cache_slot_t::data, blk_bam_entry_t::dirty, cache_slot_t::dirty, blk_dev_slot::format_state, blk_volume_meta_t::free_blocks, g, lbn_to_slot(), lbn_to_slot_pbn(), blk_meta_t::magic, cache_slot_t::meta, cache_slot_t::meta_dirty, blk_meta_t::modified_time, blk_dev_slot::raw_base, resolve_lbn(), slot_pbn_pack_offset(), slot_pbn_to_devblock(), blk_dev_slot::start_lbn, blk_dev_slot::vol_meta, and blk_dev_slot::vol_meta_dirty.

Referenced by blk_subsys_relocate_block(), blk_vm_evict(), blk_vm_flush_all(), block_word_update(), capsule_clear_blocks(), mark_buffer_dirty(), and write_ramdrive_block().

Here is the call graph for this function:



Here is the caller graph for this function:



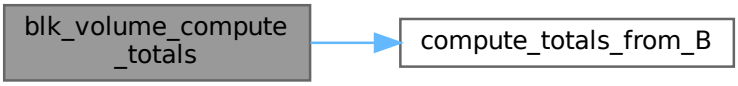
7.277.3.51 blk_volume_compute_totals()

```
void blk_volume_compute_totals (
            blk_volume_meta_t * m )
```

References compute_totals_from_B().

Referenced by sk_v4_grow_artemis().

Here is the call graph for this function:



Here is the caller graph for this function:

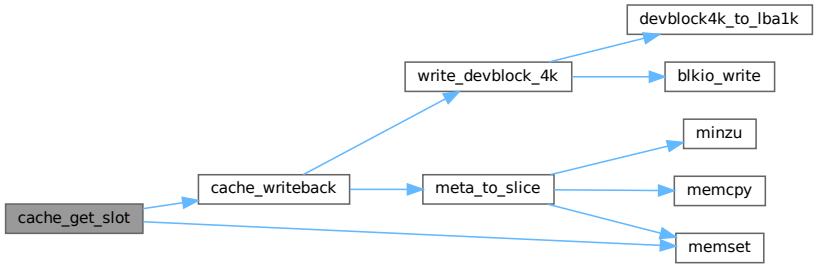
7.277.3.52 cache_get_slot()

```
static cache_slot_t * cache_get_slot (
            blk_dev_slot_t * slot,
            uint32_t dev4k )  [static]
```

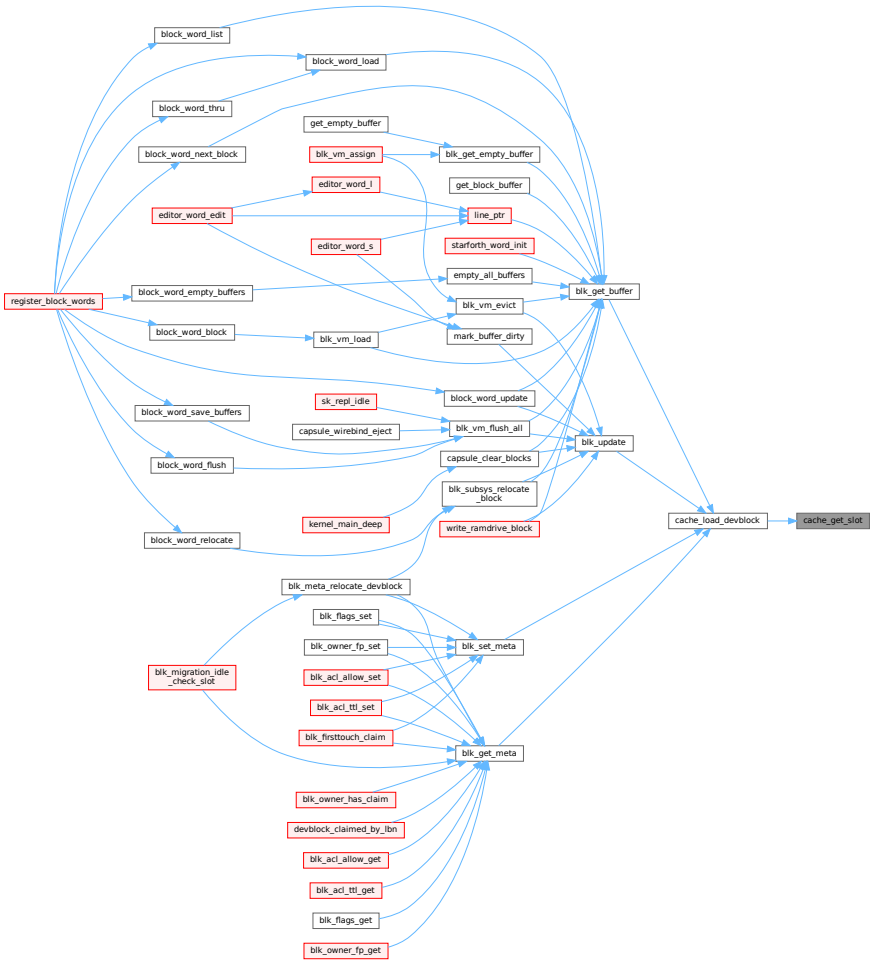
References blk_dev_slot::cache, cache_writeback(), cache_slot_t::devblock, DISK_CACHE_SLOTS, memset(), and cache_slot_t::valid.

Referenced by cache_load_devblock().

Here is the call graph for this function:



Here is the caller graph for this function:



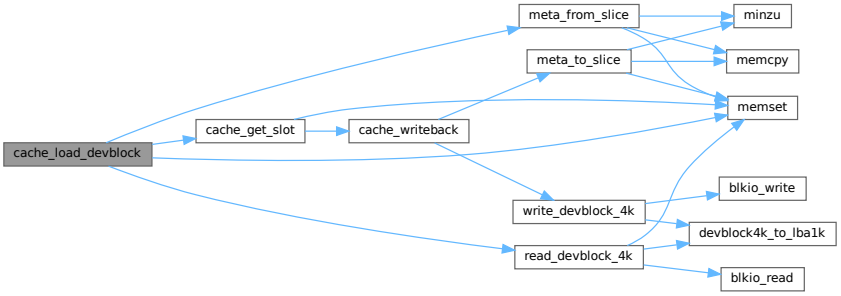
7.277.3.53 cache_load_devblock()

```
static cache_slot_t * cache_load_devblock (
            blk_dev_slot_t * slot,
            uint32_t dev4k )  [static]
```

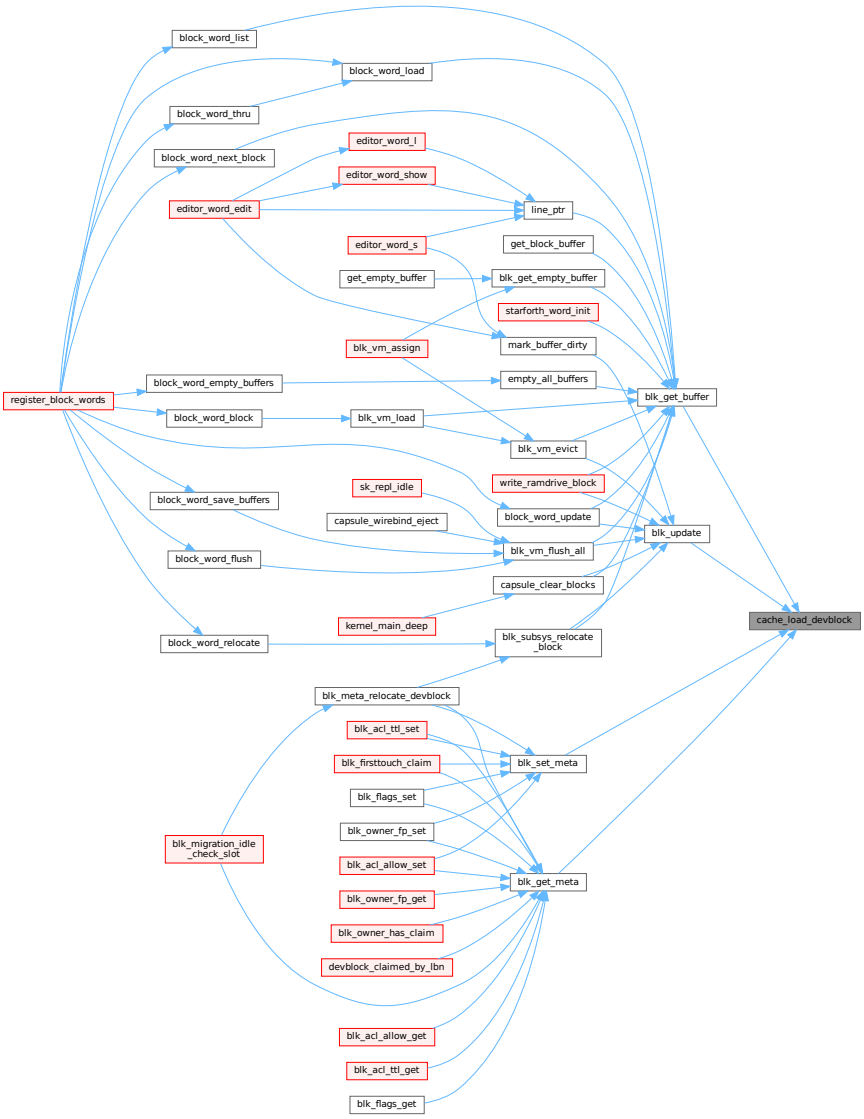
References BLK_PACK_RATIO, cache_get_slot(), cache_slot_t::data, cache_slot_t::loaded, blk_meta_t::magic, memset(), cache_slot_t::meta, meta_from_slice(), META_PER_BLOCK, META_REGION_OFFSET, and read_devblock_4k().

Referenced by blk_get_buffer(), blk_get_meta(), blk_set_meta(), and blk_update().

Here is the call graph for this function:



Here is the caller graph for this function:



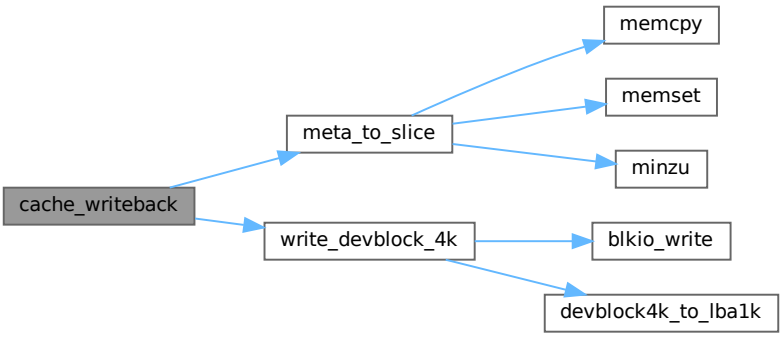
7.277.3.54 cache_writeback()

```
static int cache_writeback (
            blk_dev_slot_t * slot,
            cache_slot_t * s )  [static]
```

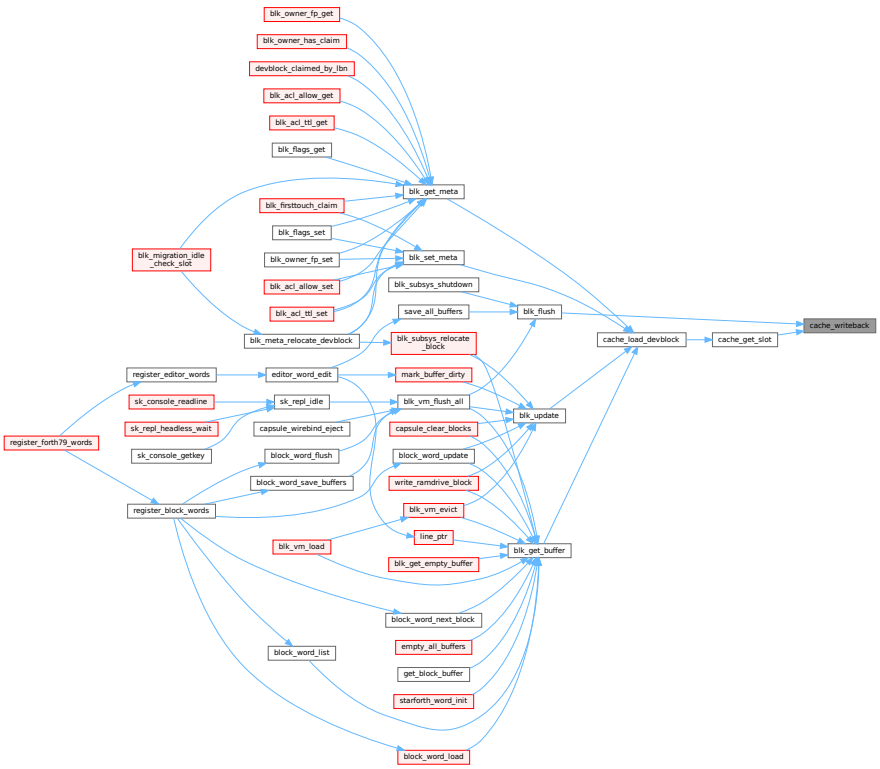
References BLK_OK, BLK_PACK_RATIO, cache_slot_t::data, blk_dev_slot::dev, cache_slot_t::devblock, cache_slot_t::dirty, cache_slot_t::meta, cache_slot_t::meta_dirty, META_PER_BLOCK, META_REGION_OFFSET, meta_to_slice(), and write_devblock_4k().

Referenced by blk_flush(), and cache_get_slot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.277.3.55 chain_append()

```
static void chain_append (
            blk_dev_slot_t * slot )  [static]
```

References g, and blk_dev_slot::next.

Referenced by blk_subsys_add_raw_device(), and blk_subsys_attach_device().

Here is the caller graph for this function:

7.277.3.56 choose_B()

```
static uint32_t choose_B (
            uint64_t total_devblocks_4k )  [static]
```

Referenced by blk_compute_fresh_geometry().

Here is the caller graph for this function:

7.277.3.57 compute_crc64()

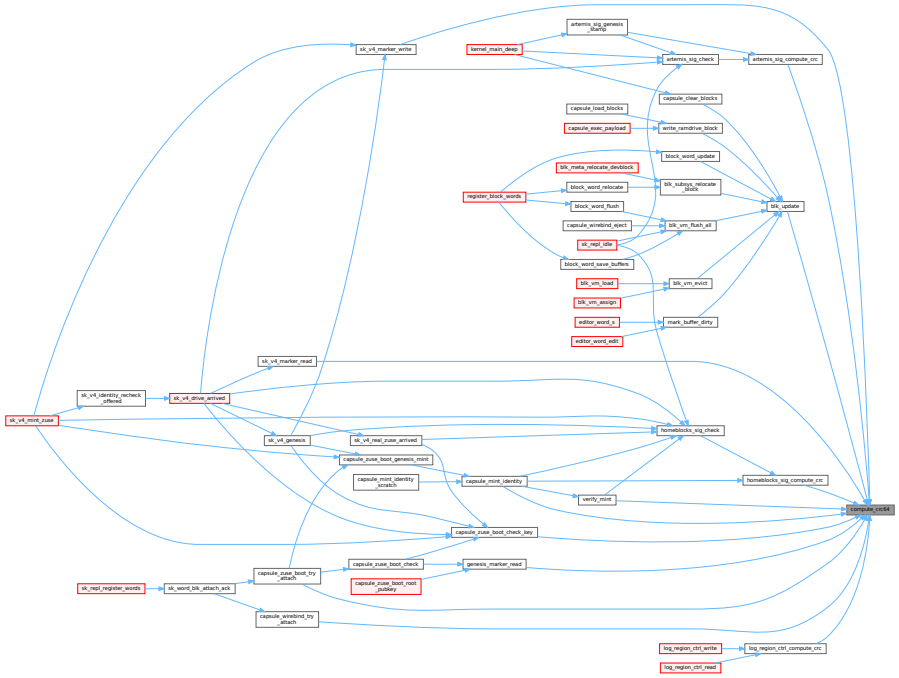
```
uint64_t compute_crc64 (
            const uint8_t * data,
            size_t len )
```

References crc64_init(), crc64_inited, and crc64_table.

Referenced by artemis_sig_compute_crc(), blk_update(), capsule_mint_identity(), capsule_wirebind_try_attach(), capsule_zuse_boot_check_key(), capsule_zuse_boot_try_attach(), genesis_marker_read(), homeblocks_sig_compute_crc(), log_region_ctrl_compute_crc(), sk_v4_marker_read(), sk_v4_marker_write(), and verify_mint().

Here is the call graph for this function:

Here is the caller graph for this function:



7.277.3.58 compute_totals_from_B()

```
static void compute_totals_from_B (
            blk_volume_meta_t * m )  [static]
```

References blk_volume_meta_t::bam_devblocks, blk_volume_meta_t::meta_fence_blocks, blk_volume_meta_t::reloc_devblocks, blk_volume_meta_t::total_blocks, blk_volume_meta_t::total_devblocks, and blk_volume_meta_t::tracked_blocks.

Referenced by blk_compute_fresh_geometry(), and blk_volume_compute_totals().

Here is the caller graph for this function:

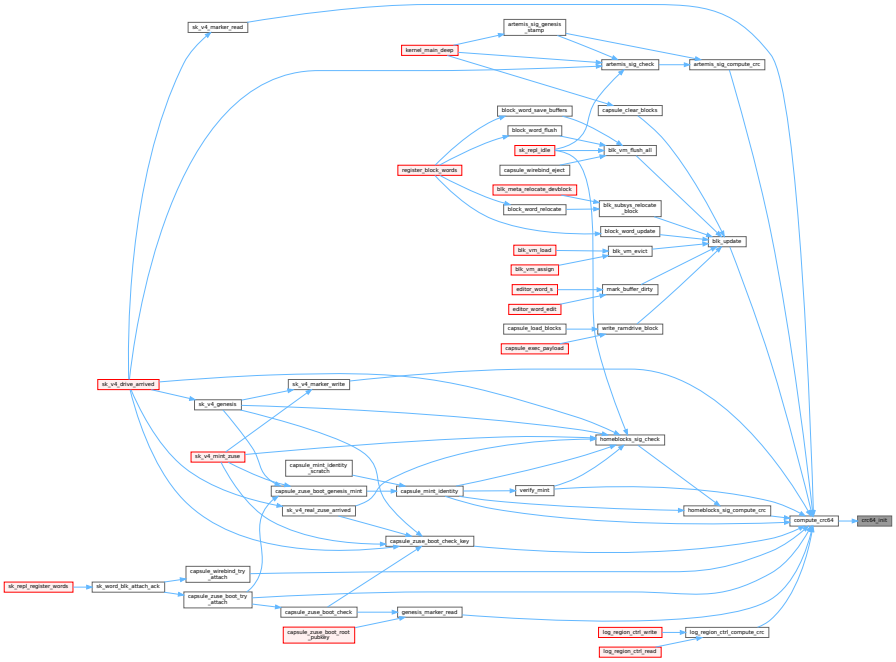
7.277.3.59 crc64_init()

```
static void crc64_init (
            void  )  [static]
```

References crc64_inited, and crc64_table.

Referenced by compute_crc64().

Here is the caller graph for this function:



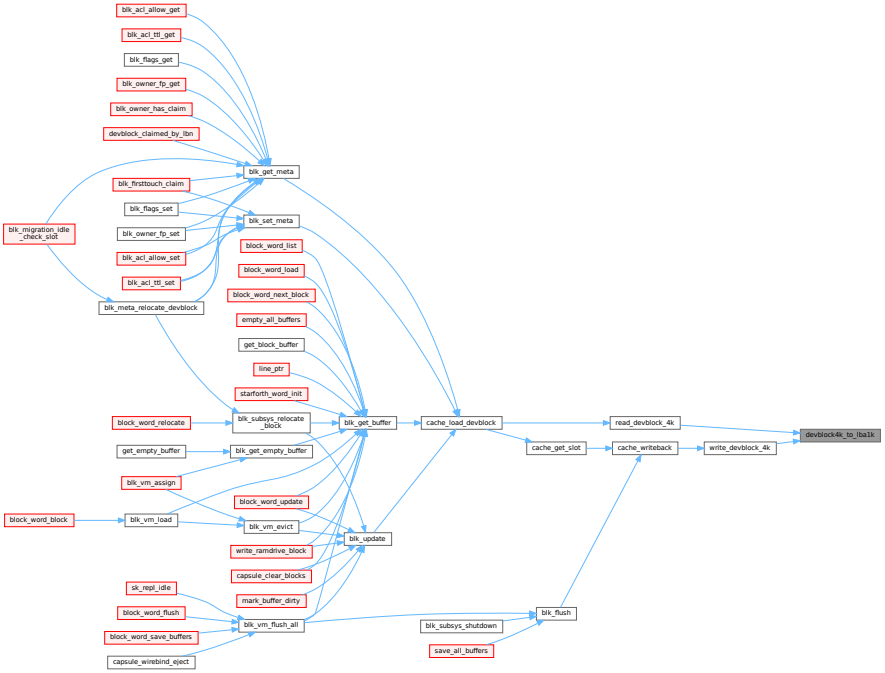
7.277.3.60 devblock4k_to_lba1k()

```
static uint32_t devblock4k_to_lba1k (
            blk_dev_slot_t * slot,
            uint32_t dev4k )  [inline], [static]
```

References blk_dev_slot::devblock_base_4k.

Referenced by read_devblock_4k(), and write_devblock_4k().

Here is the caller graph for this function:



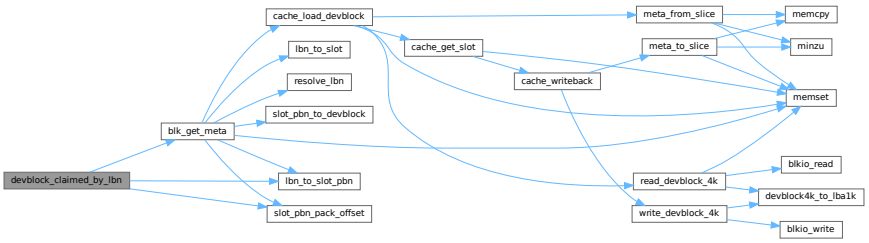
7.277.3.61 devblock_claimed_by_lbn()

```
static int devblock_claimed_by_lbn (
            blk_dev_slot_t * slot,
            uint32_t lbn )  [static]
```

References BLK_FLAG_CLAIMED, blk_get_meta(), BLK_OK, blk_volume_meta_t::flags, lbn_to_slot_pbn(), meta, slot_pbn_pack_offset(), and blk_dev_slot::start_lbn.

Referenced by blk_allocate().

Here is the call graph for this function:



Here is the caller graph for this function:

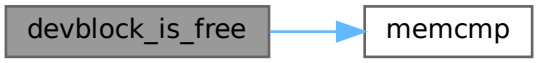
7.277.3.62 devblock_is_free()

```
static int devblock_is_free (
            blk_dev_slot_t * slot,
            uint32_t rep_lbn,
            const blk_meta_t * m )  [static]
```

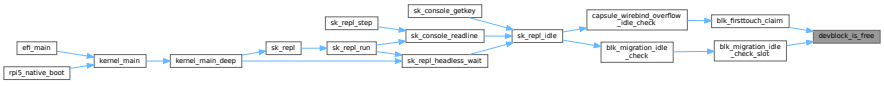
References blk_bam_entry_t::allocated, blk_dev_slot::bam, BLK_FLAG_CLAIMED, BLK_PACK_RATIO, blk_meta_t::flags, memcmp(), blk_meta_t::owner_fp, blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by blk_firsttouch_claim(), and blk_migration_idle_check_slot().

Here is the call graph for this function:



Here is the caller graph for this function:



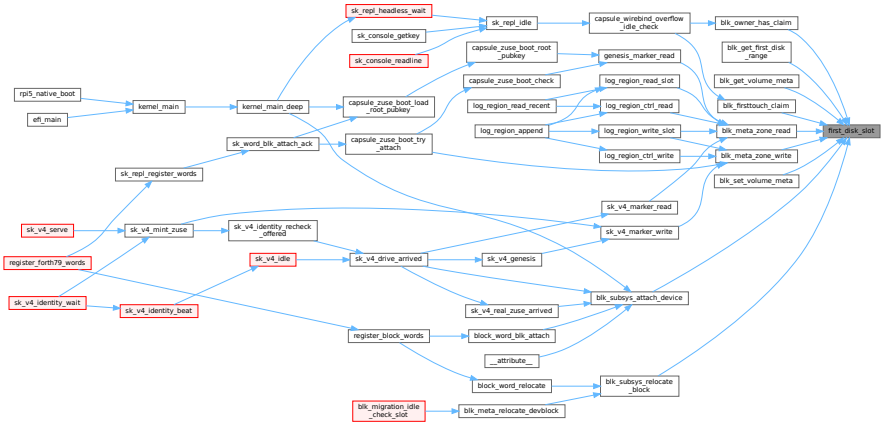
7.277.3.63 first_disk_slot()

```
static blk_dev_slot_t * first_disk_slot (
            void  )  [static]
```

References blk_dev_slot::dev, g, and blk_dev_slot::next.

Referenced by blk_firsttouch_claim(), blk_get_first_disk_range(), blk_get_volume_meta(), blk_meta_zone_read(), blk_meta_zone_write(), blk_owner_has_claim(), blk_set_volume_meta(), blk_subsys_attach_device(), and blk_subsys_relocate_block().

Here is the caller graph for this function:



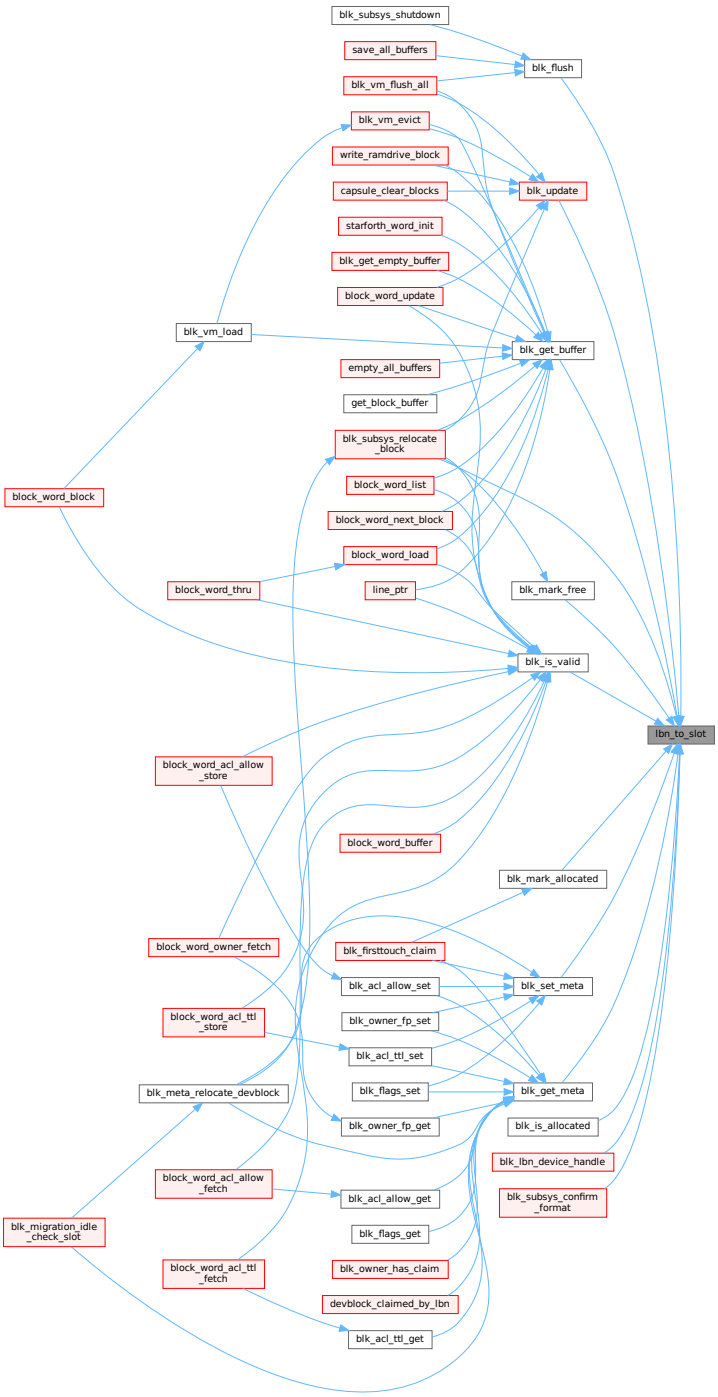
7.277.3.64 lbn_to_slot()

```
static blk_dev_slot_t * lbn_to_slot (
            uint32_t lbn )  [static]
```

References g, blk_dev_slot::next, blk_dev_slot::start_lbn, and blk_dev_slot::user_blocks.

Referenced by blk_flush(), blk_get_buffer(), blk_get_meta(), blk_is_allocated(), blk_is_valid(), blk_lbn_device_handle(), blk_mark_allocated(), blk_mark_free(), blk_set_meta(), blk_subsys_confirm_format(), blk_subsys_relocate_block(), and blk_update().

Here is the caller graph for this function:



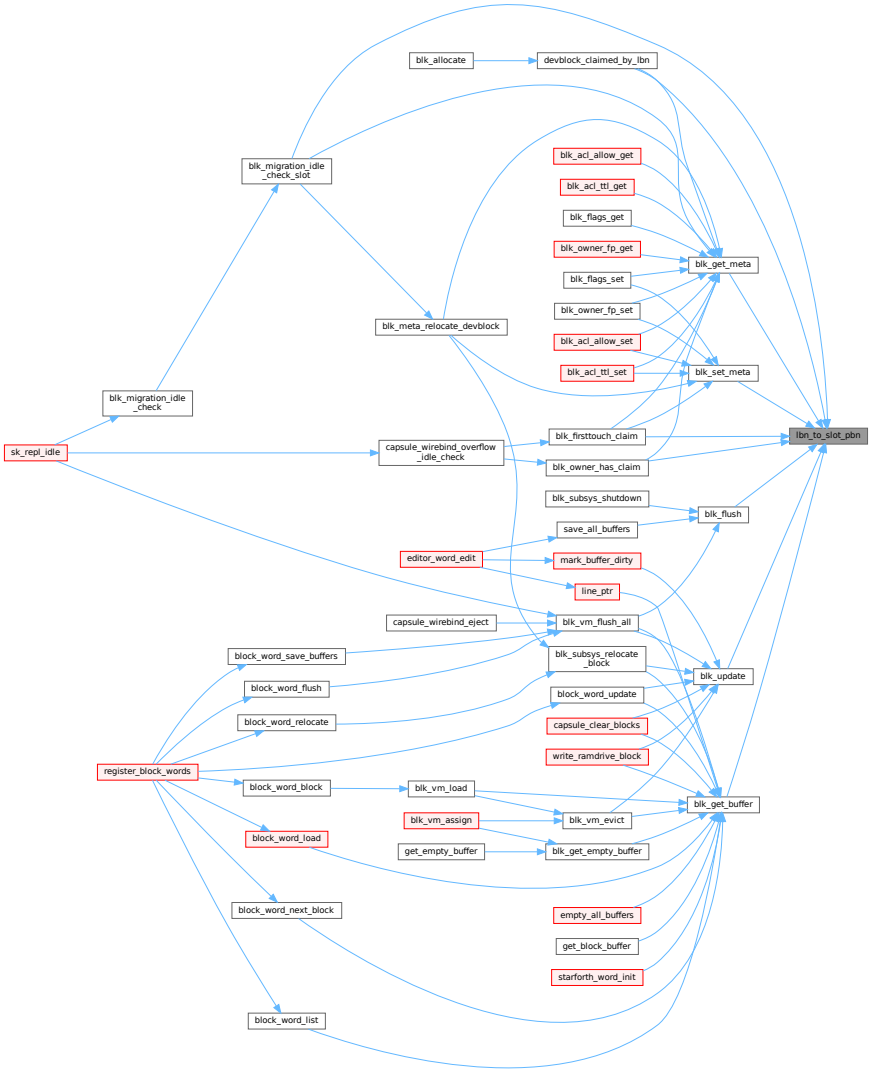
7.277.3.65 lbn_to_slot_pbn()

```
static uint32_t lbn_to_slot_pbn (
            blk_dev_slot_t * slot,
            uint32_t lbn )  [inline], [static]
```

References BLK_DISK_SYS_RESERVED, and blk_dev_slot::start_lbn.

Referenced by blk_firsttouch_claim(), blk_flush(), blk_get_buffer(), blk_get_meta(), blk_migration_idle_check_slot(), blk_owner_has_claim(), blk_set_meta(), blk_update(), and devblock_claimed_by_lbn().

Here is the caller graph for this function:



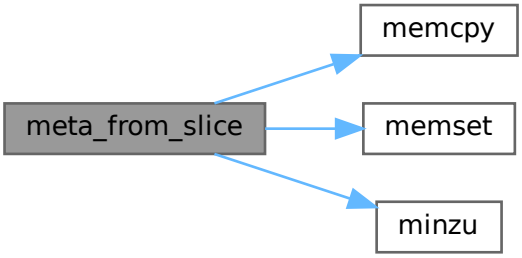
7.277.3.66 meta_from_slice()

```
static void meta_from_slice (
            blk_meta_t * dst,
            const uint8_t * slice )  [static]
```

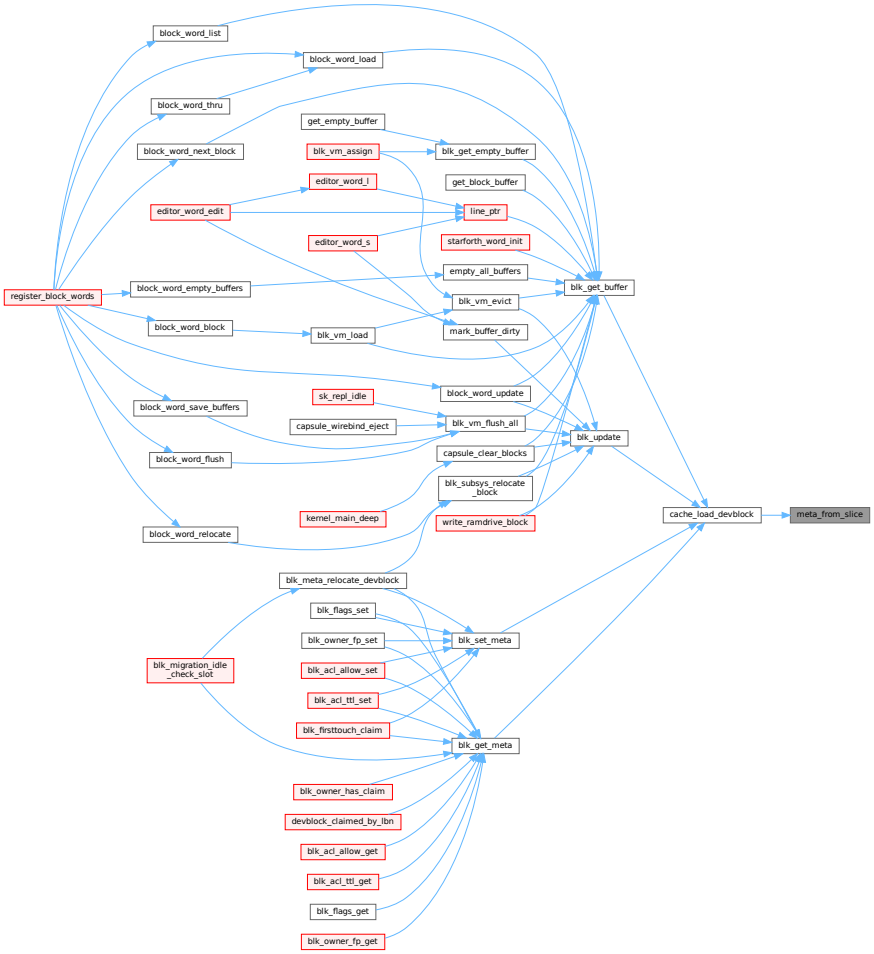
References memcpy(), memset(), META_PER_BLOCK, and minzu().

Referenced by cache_load_devblock().

Here is the call graph for this function:



Here is the caller graph for this function:



7.277.3.67 meta_to_slice()

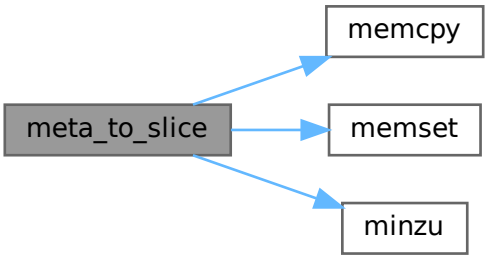
```
static void meta_to_slice (
```

```
const blk_meta_t * src,
uint8_t * slice )  [static]
```

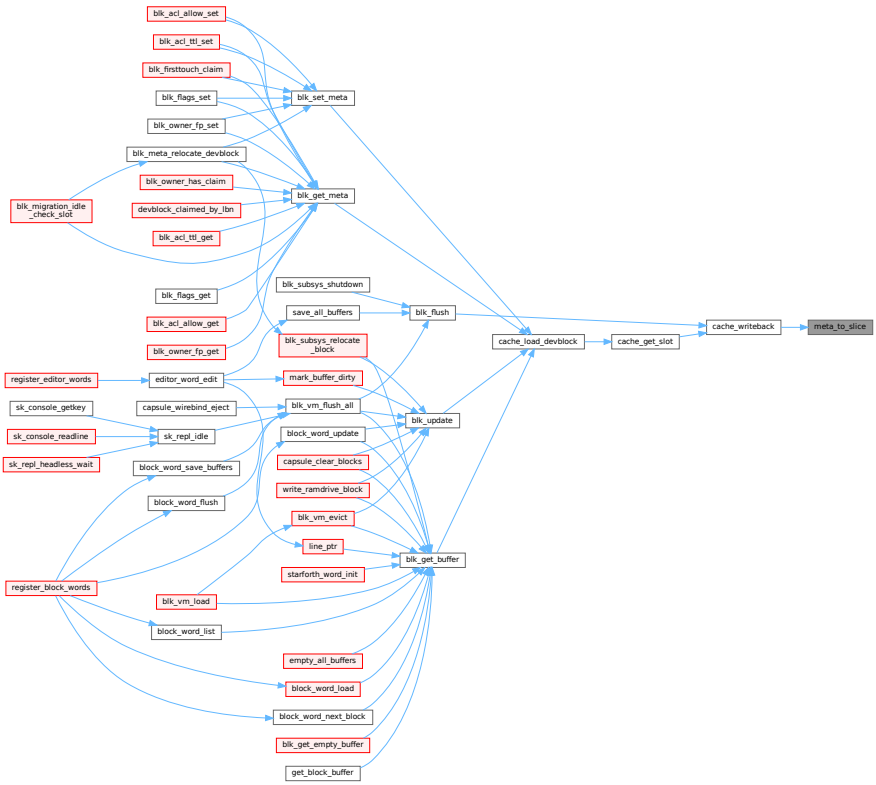
References memcpy(), memset(), META_PER_BLOCK, and minzu().

Referenced by cache_writeback().

Here is the call graph for this function:



Here is the caller graph for this function:

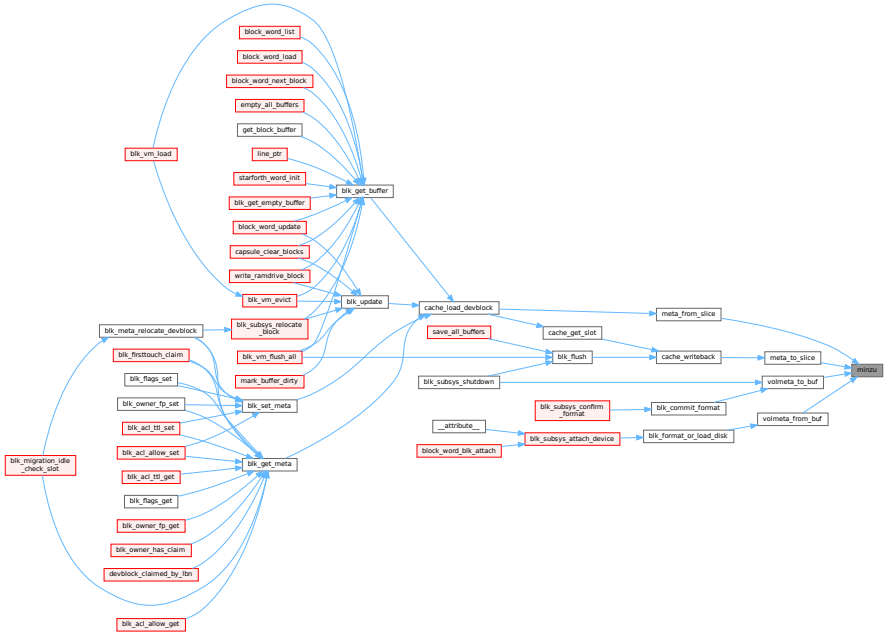


7.277.3.68 minzu()

```
static size_t minzu (
            size_t a,
            size_t b )  [inline], [static]
```

Referenced by meta_from_slice(), meta_to_slice(), volmeta_from_buf(), and volmeta_to_buf().

Here is the caller graph for this function:

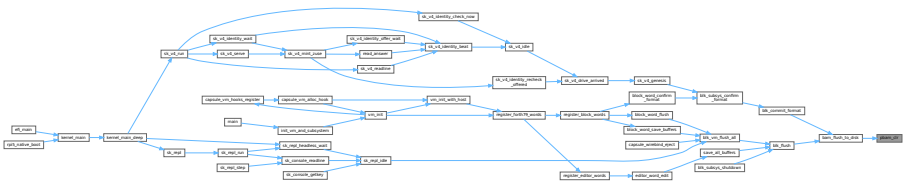


7.277.3.69 pbam_clr()

```
static void pbam_clr (
            uint8_t * bm,
            uint32_t i )  [inline], [static]
```

Referenced by bam_flush_to_disk().

Here is the caller graph for this function:

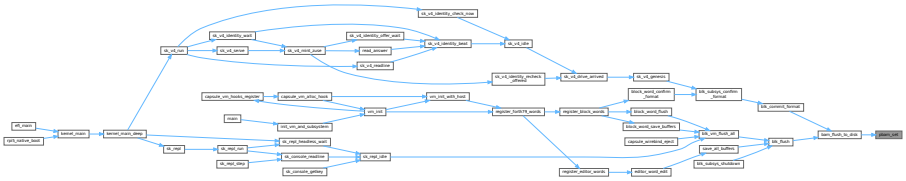


7.277.3.70 pbam_set()

```
static void pbam_set (
            uint8_t * bm,
            uint32_t i )  [inline], [static]
```

Referenced by bam_flush_to_disk().

Here is the caller graph for this function:



7.277.3.71 pbam_test()

```
static int pbam_test (
            const uint8_t * bm,
            uint32_t i )  [inline], [static]
```

Referenced by bam_sync_from_disk().

Here is the caller graph for this function:

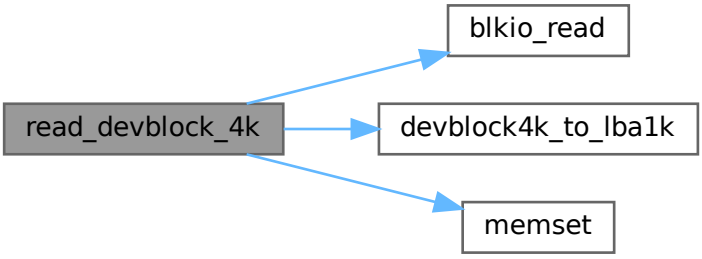
7.277.3.72 read_devblock_4k()

```
static int read_devblock_4k (
            blk_dev_slot_t * slot,
            uint32_t dev4k,
            uint8_t * buf4k )  [static]
```

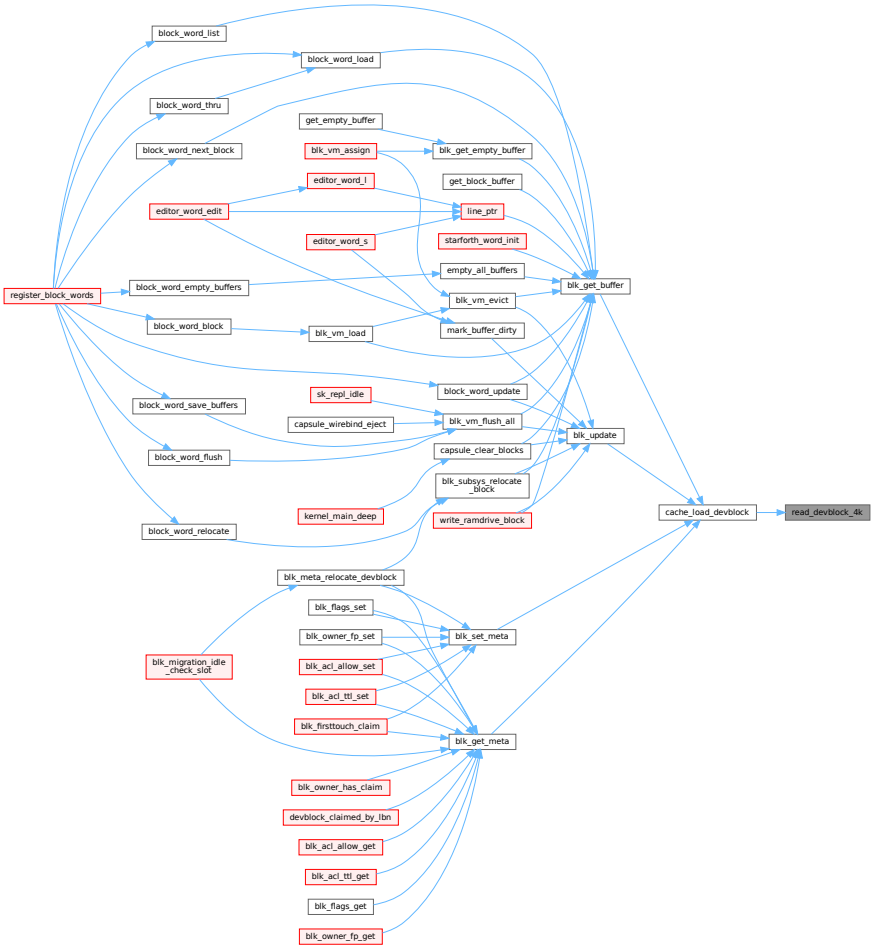
References base, BLK_OK, BLKIO_OK, blkio_read(), blk_dev_slot::dev, devblock4k_to_lba1k(), memset(), and blk_dev_slot::total_blkio_blocks_1k.

Referenced by cache_load_devblock().

Here is the call graph for this function:



Here is the caller graph for this function:



7.277.3.73 read_header_4k()

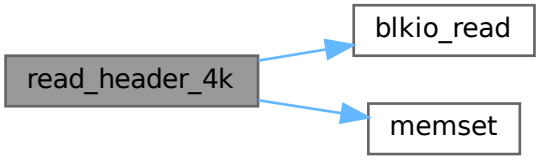
```
static int read_header_4k (
```

```
            blk_dev_slot_t * slot,
            uint8_t * buf4k )  [static]
```

References BLK_OK, BLKIO_OK, blkio_read(), blk_dev_slot::dev, and memset().

Referenced by blk_format_or_load_disk().

Here is the call graph for this function:



Here is the caller graph for this function:

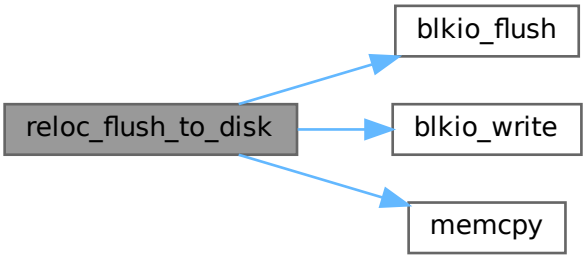
7.277.3.74 reloc_flush_to_disk()

```
static int reloc_flush_to_disk (
            blk_dev_slot_t * slot )  [static]
```

References BLK_EIO, BLK_OK, blkio_flush(), BLKIO_OK, blkio_write(), blk_dev_slot::dev, g, memcpy(), blk_volume_meta_t::reloc_devblocks, blk_volume_meta_t::reloc_start, and blk_dev_slot::vol_meta.

Referenced by blk_subsys_relocate_block().

Here is the call graph for this function:



Here is the caller graph for this function:

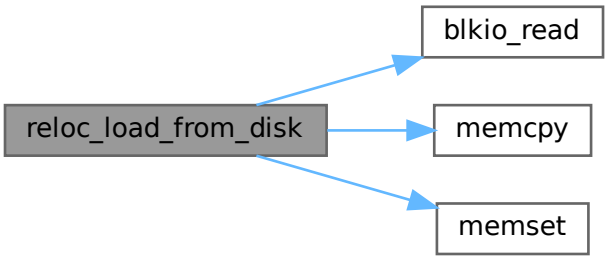
7.277.3.75 reloc_load_from_disk()

```
static int reloc_load_from_disk (
            blk_dev_slot_t * slot )  [static]
```

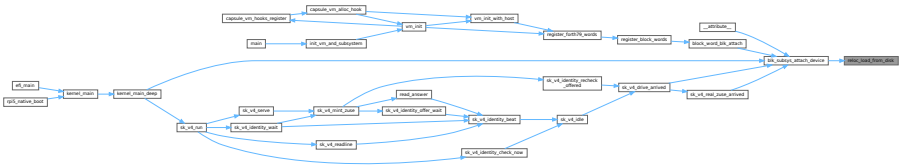
References BLK_EINVAL, BLK_OK, BLK_RELOC_MAX_ENTRIES, BLKIO_OK, blkio_read(), count, blk_dev_slot::dev, g, memcpy(), memset(), blk_volume_meta_t::reloc_devblocks, blk_volume_meta_t::reloc_start, and blk_dev_slot::vol_meta.

Referenced by blk_subsys_attach_device().

Here is the call graph for this function:



Here is the caller graph for this function:



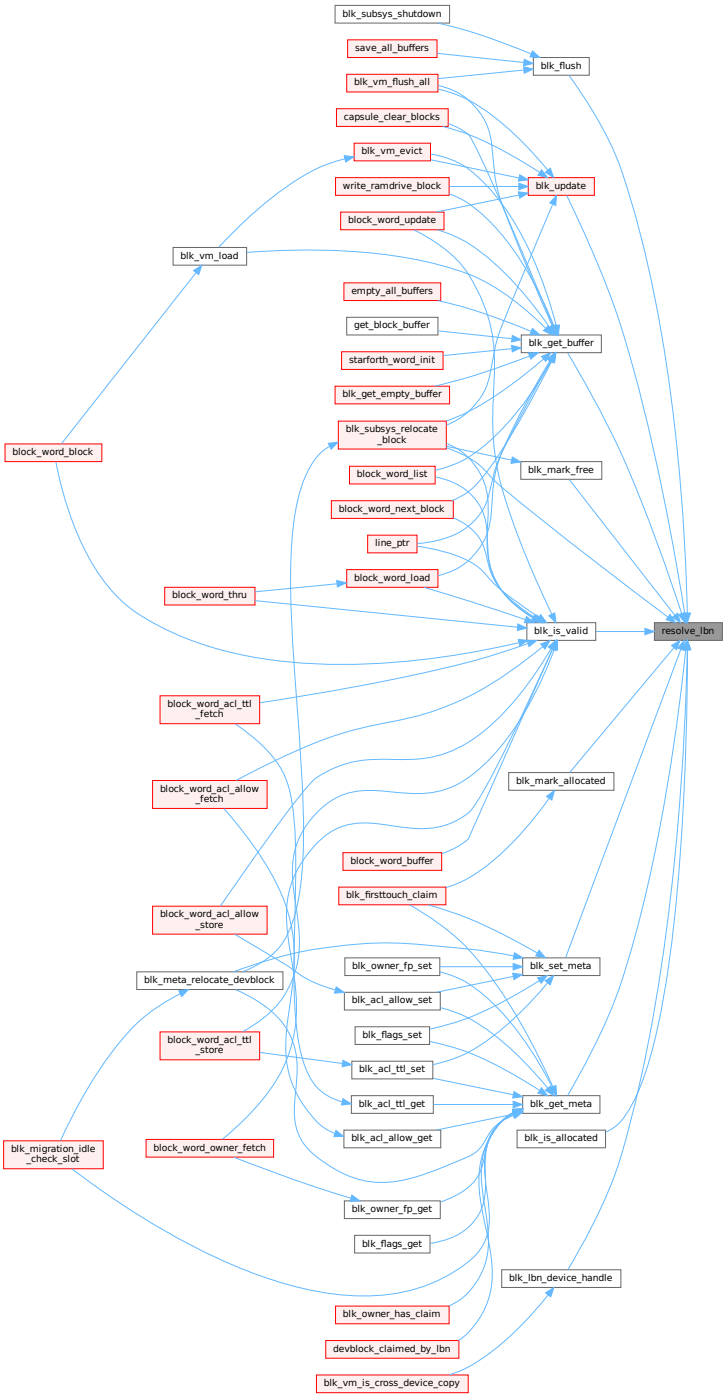
7.277.3.76 resolve_lbn()

```
static uint32_t resolve_lbn (
            uint32_t lbn )  [static]
```

References blk_reloc_entry_t::actual_lbn, and g.

Referenced by blk_flush(), blk_get_buffer(), blk_get_meta(), blk_is_allocated(), blk_is_valid(), blk_lbn_device_handle(), blk_mark_allocated(), blk_mark_free(), blk_set_meta(), blk_subsys_relocate_block(), and blk_update().

Here is the caller graph for this function:



7.277.3.77 slot_by_dev()

```
static blk_dev_slot_t * slot_by_dev (
            struct blkio_dev * dev )  [static]
```

References blk_dev_slot::dev, g, and blk_dev_slot::next.

Referenced by blk_get_device_free_blocks(), and blk_get_device_range().

Here is the caller graph for this function:

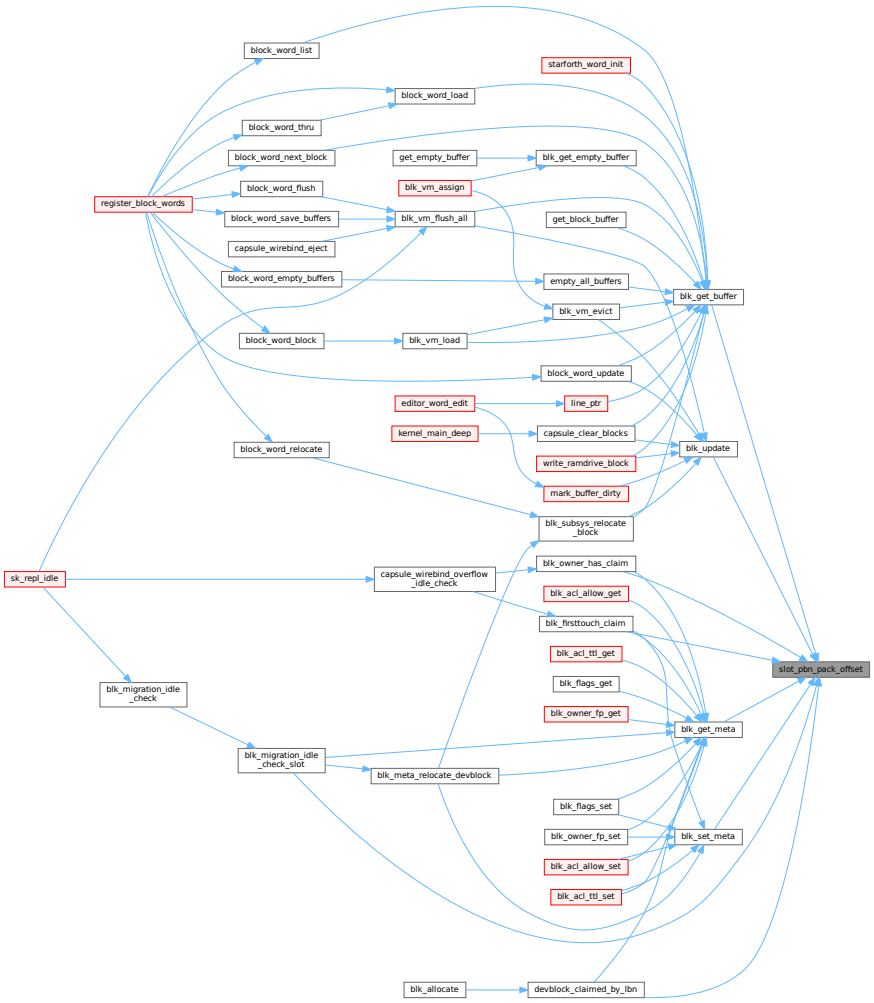
7.277.3.78 slot_pbn_pack_offset()

```
static uint32_t slot_pbn_pack_offset (
            uint32_t rel_pbn )  [inline], [static]
```

References BLK_PACK_RATIO.

Referenced by blk_firsttouch_claim(), blk_get_buffer(), blk_get_meta(), blk_migration_idle_check_slot(), blk_owner_has_claim(), blk_set_meta(), blk_update(), and devblock_claimed_by_lbn().

Here is the caller graph for this function:



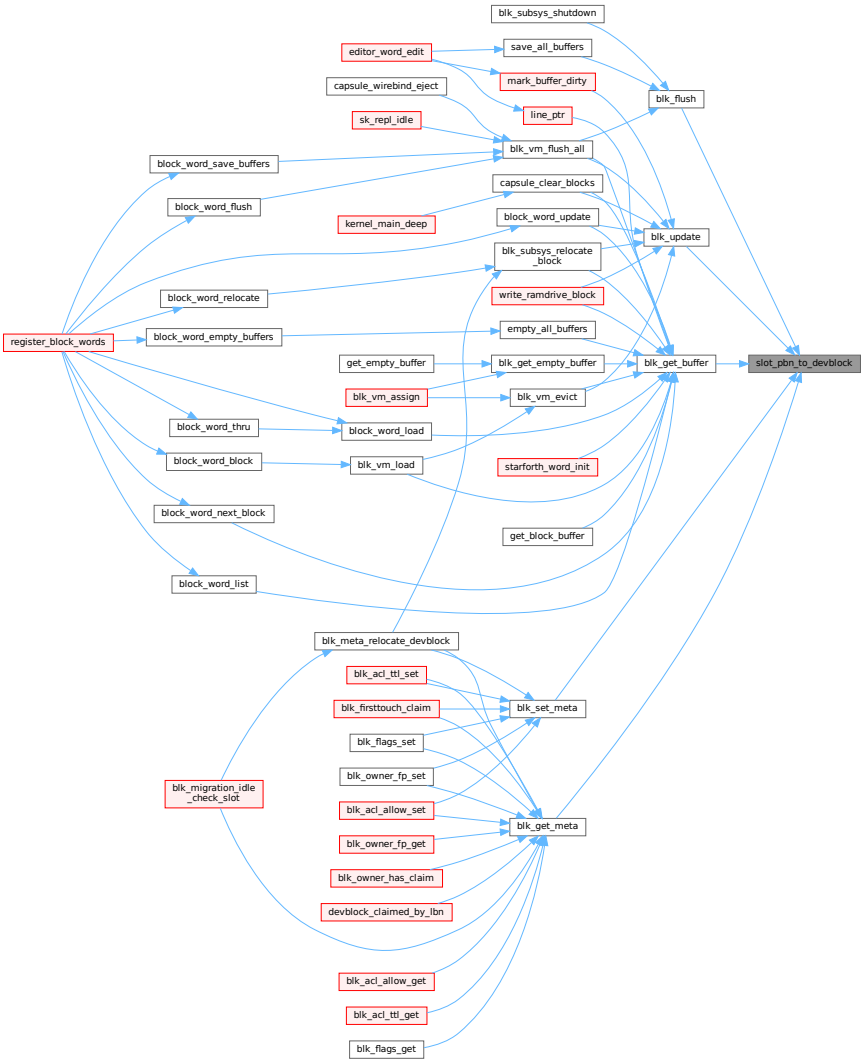
7.277.3.79 slot_pbn_to_devblock()

```
static uint32_t slot_pbn_to_devblock (
            uint32_t rel_pbn )  [inline], [static]
```

References BLK_PACK_RATIO.

Referenced by blk_flush(), blk_get_buffer(), blk_get_meta(), blk_set_meta(), and blk_update().

Here is the caller graph for this function:



7.277.3.80 udiv_floor()

```
static uint32_t udiv_floor (
            uint32_t a,
            uint32_t b )  [inline], [static]
```

Referenced by blk_compute_fresh_geometry(), and blk_format_or_load_disk().

Here is the caller graph for this function:

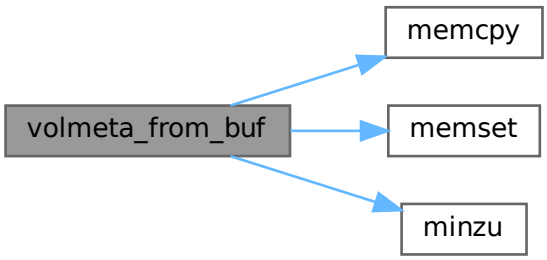
7.277.3.81 volmeta_from_buf()

```
static void volmeta_from_buf (
            blk_volume_meta_t * out,
            const uint8_t * buf )  [inline], [static]
```

References BLK_DEVICE_SECTOR, memcpy(), memset(), and minzu().

Referenced by blk_format_or_load_disk().

Here is the call graph for this function:



Here is the caller graph for this function:

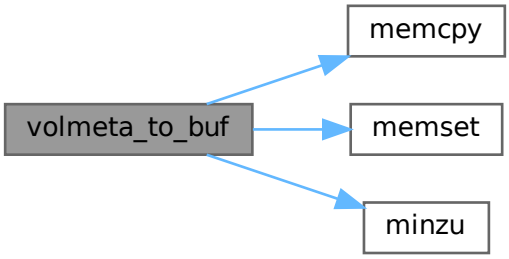
7.277.3.82 volmeta_to_buf()

```
static void volmeta_to_buf (
            const blk_volume_meta_t * in,
            uint8_t * buf )  [inline], [static]
```

References BLK_DEVICE_SECTOR, memcpy(), memset(), and minzu().

Referenced by blk_commit_format(), and blk_subsys_shutdown().

Here is the call graph for this function:



Here is the caller graph for this function:

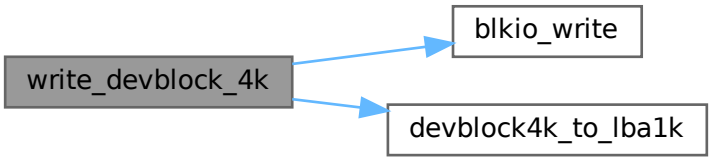
7.277.3.83 write_devblock_4k()

```
static int write_devblock_4k (
            blk_dev_slot_t * slot,
            uint32_t dev4k,
            const uint8_t * buf4k )  [static]
```

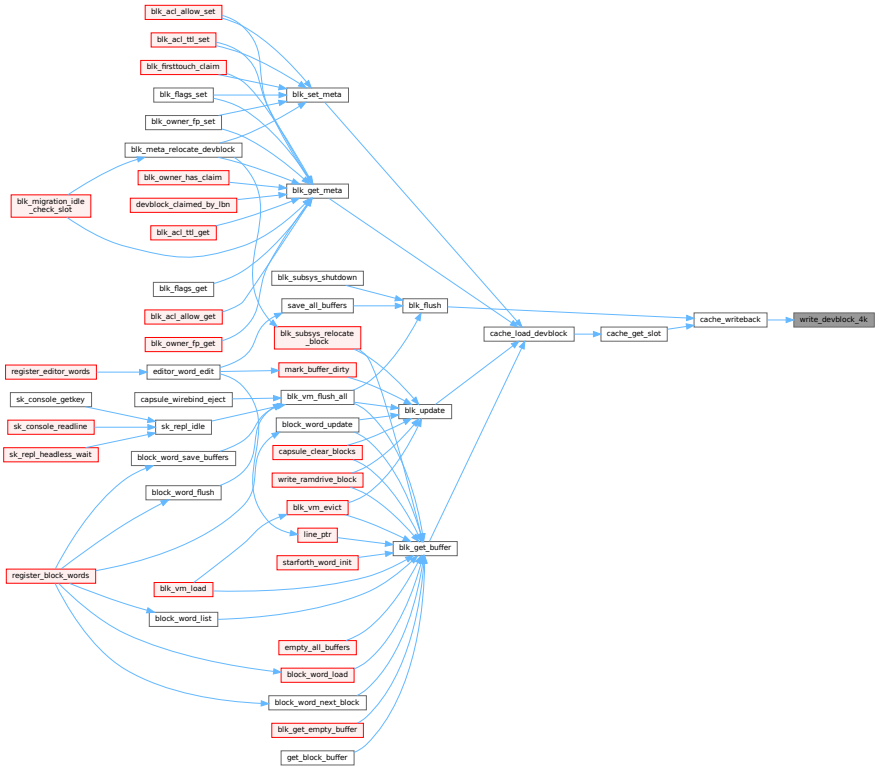
References base, BLK_EIO, BLK_OK, BLKIO_OK, blkio_write(), blk_dev_slot::dev, devblock4k_to_lba1k(), and blk_dev_slot::total_blkio_blocks_1k.

Referenced by cache_writeback().

Here is the call graph for this function:



Here is the caller graph for this function:



7.277.3.84 write_header_4k()

```
static int write_header_4k (
            blk_dev_slot_t * slot,
            const uint8_t * buf4k )  [static]
```

References BLK_EIO, BLK_OK, BLKIO_OK, blkio_write(), and blk_dev_slot::dev.

Referenced by blk_commit_format(), and blk_subsys_shutdown().

Here is the call graph for this function:

Here is the caller graph for this function:

7.277.4 Variable Documentation

7.277.4.1 crc64_inited

```
int crc64_inited = 0  [static]
```

Referenced by compute_crc64(), and crc64_init().

7.277.4.2 crc64_table

```
uint64_t crc64_table[256]  [static]
```

Referenced by compute_crc64(), and crc64_init().

7.277.4.3 dirty_ram

```
uint8_t dirty_ram[BLK_RAM_BLOCKS]
```

7.277.4.4 epoch

```
uint64_t epoch
```

Referenced by blk_vm_check_epoch().

7.277.4.5 [struct]

```
struct { ... } g  [static]
```

Referenced by ansi256_to_rgb(), apply_sgr(), blk_allocate(), blk_flush(), blk_get_buffer(), blk_get_meta(), blk_get_total_blocks(), blk_is_allocated(), blk_is_valid(), blk_mark_allocated(), blk_mark_free(), blk_migration_idle_check(), blk_set_meta(), blk_subsys_add_raw_device(), blk_subsys_attach_device(), blk_subsys_confirm_format(), blk_subsys_detach_device(), blk_subsys_epoch(), blk_subsys_init(), blk_subsys_relocate_block(), blk_subsys_shutdown(), blk_update(), chain_append(), draw_cursor_glyph(), first_disk_slot(), gpt_new_guid(), lbn_to_slot(), pack_pixel(), reloc_flush_to_disk(), reloc_load_from_disk(), resolve_lbn(), sha512_block(), sk_v4_drive_arrived(), and slot_by_dev().

7.277.4.6 g_format_bam_for

```
uint64_t g_format_bam_for  [static]
```

Referenced by blk_compute_fresh_geometry(), and blk_subsys_set_format_bam_for().

7.277.4.7 head

```
blk_dev_slot_t* head
```

Referenced by fc_rebuild().

7.277.4.8 initialized

```
int initialized
```

7.277.4.9 ram_base

```
uint8_t* ram_base
```

Referenced by blk_subsys_init(), blkio_factory_open(), and init_vm_and_subsystem().

7.277.4.10 ram_size

```
size_t ram_size
```

Referenced by blk_subsys_init(), emit_region_with_carveouts(), and init_vm_and_subsystem().

7.277.4.11 ram_user

```
uint32_t ram_user
```

7.277.4.12 reloc

```
blk_reloc_entry_t reloc[500u]
```

7.277.4.13 reloc_count

```
uint32_t reloc_count
```

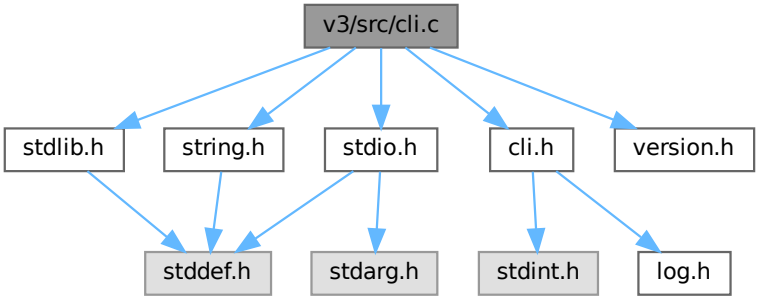
7.277.4.14 total_user_lbn

```
uint64_t total_user_lbn
```

7.278 v3/src/cli.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cli.h"
#include "version.h"
```

Include dependency graph for cli.c:



Functions

- static int parse_u32 (const char *s, uint32_t *out)
- void cli_print_usage (const char *program_name)

  Print usage/help information.
- CLIConfig cli_parse (int argc, char *argv[])

  Parse command-line arguments into CLIConfig.

7.278.1 Function Documentation

7.278.1.1 cli_parse()

```
CLIConfig cli_parse (
            int argc,
            char * argv[] )
```

Parse command-line arguments into CLIConfig.

Parameters

argc	Argument count
argv	Argument vector

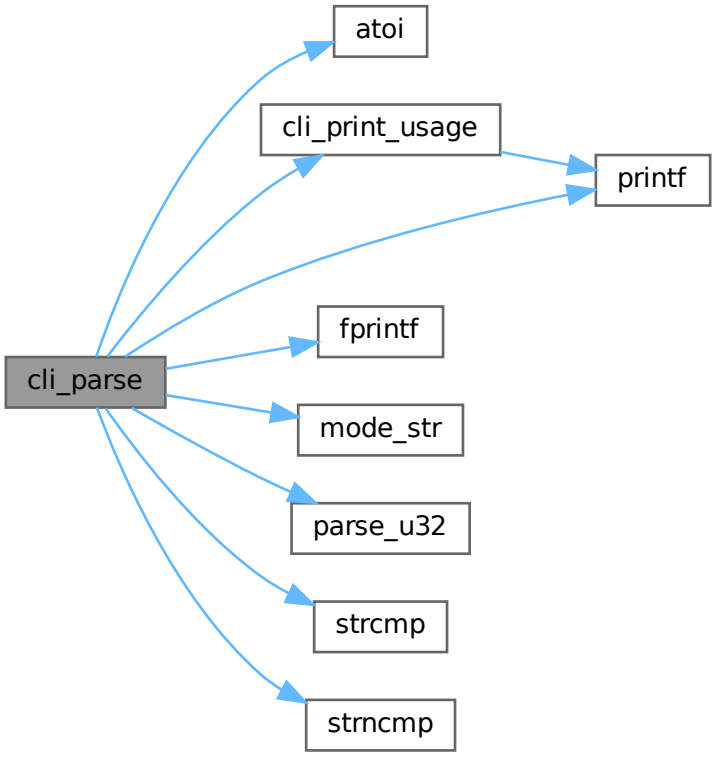
Returns

CLIConfig struct with parsed options, or exits on –version/–help

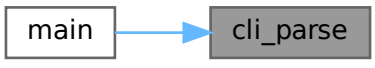
References atoi(), CLIConfig::benchmark, CLIConfig::benchmark_iterations, CLIConfig::break_me, cli_print_usage(), CLIConfig::disk_img_path, CLIConfig::doe_experiment, CLIConfig::fail_fast, fprintf(), HEARTBEAT_LOG_FULL, CLIConfig::heartbeat_log_mode, HEARTBEAT_LOG_OFF, HEARTBEAT_LOG_SUMMARY, LOG_DEBUG, LOG_ERROR, LOG_INFO, CLIConfig::log_level, CLIConfig::log_level_explicitly_set, LOG_NONE, LOG_TEST, LOG_WARN, mode_str(), parse_u32(), printf(), CLIConfig::ram_disk_mb, CLIConfig::run_tests, CLIConfig::run_tests_flag_observed, CLIConfig::script_mode, STARFORTH_VERSION_FULL, stderr, strcmp(), and strncmp().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.278.1.2 cli_print_usage()

```
void cli_print_usage (
            const char * program_name )
```

Print usage/help information.

Parameters

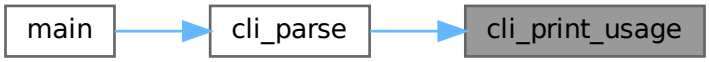
program_name	Name of the executable

References printf().

Referenced by cli_parse().

Here is the call graph for this function:

Here is the caller graph for this function:

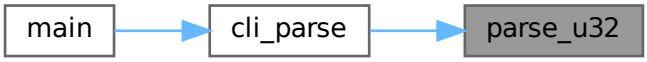


7.278.1.3 parse_u32()

```
static int parse_u32 (
            const char * s,
            uint32_t * out )  [static]
```

Referenced by cli_parse().

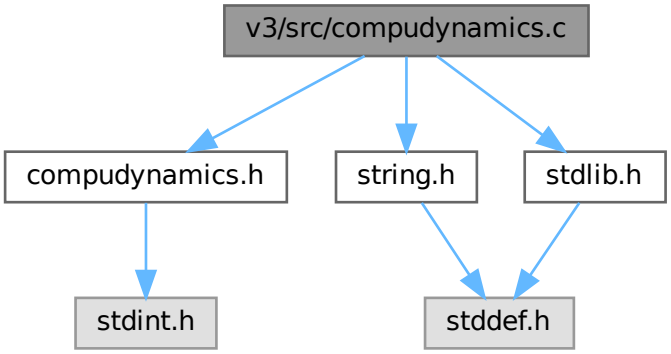
Here is the caller graph for this function:



7.279 v3/src/compudynamics.c File Reference

```
#include "compudynamics.h"
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for compudynamics.c:



Functions

- CDTuning cd_tuning_word (void)
- CDTuning cd_tuning_vm (void)
- static uint32_t cd_isqrt32 (uint32_t n)
- static uint64_t cd_isqrt64 (uint64_t n)
- static uint32_t cd_ln_approx_q16 (uint32_t n)
- CDRegimeMetrics cd_classify_ids (const uint32_t *ids, uint32_t count, uint32_t locality_lookback)

  Classify a recent entity-ID touch sequence into diversity/volatility/ locality, purely from execution counts – no wall-clock input anywhere.
- int cd_config_table_init (CDConfigTable *table, const CDTuning *tuning)

  Allocate a config table sized for tuning->num_configs / num_regimes.
- void cd_config_table_free (CDConfigTable *table)

  Free all memory owned by a config table.
- void cd_config_table_seed (CDConfigTable *table, const CDTuning *tuning, const uint32_t *initial_scores, uint8_t initial_config)

  Seed every (regime, config) score with initial_scores[config], and set the starting active config to initial_config.
- uint8_t cd_config_table_tick (CDConfigTable *table, const CDTuning *tuning, uint8_t regime, uint32_t anova_stability_q16, int use_joint_convergence, uint32_t current_window, uint32_t current_locality_q16)

  Score the currently-active config from this tick's outcome and UCB1-reselect this tick's config, for the given regime.
- void cd_config_table_force (CDConfigTable *table, uint8_t config_idx)

  Force the table onto an externally-chosen config (as if UCB1 had picked it).

7.279.1 Function Documentation

7.279.1.1 cd_classify_ids()

```
CDRegimeMetrics cd_classify_ids (
            const uint32_t * ids,
            uint32_t count,
            uint32_t locality_lookback )
```

Classify a recent entity-ID touch sequence into diversity/volatility/ locality, purely from execution counts – no wall-clock input anywhere.

Parameters

ids	Recent entity IDs, oldest first, chronological order
count	Number of valid entries in ids (clamped internally to CD_MAX_CLASSIFY_DEPTH)
locality_lookback	How many preceding touches count as "recent" for the locality measure (clamped to count)

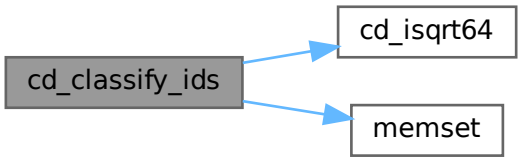
Returns

Classified regime metrics; all-zero if count == 0

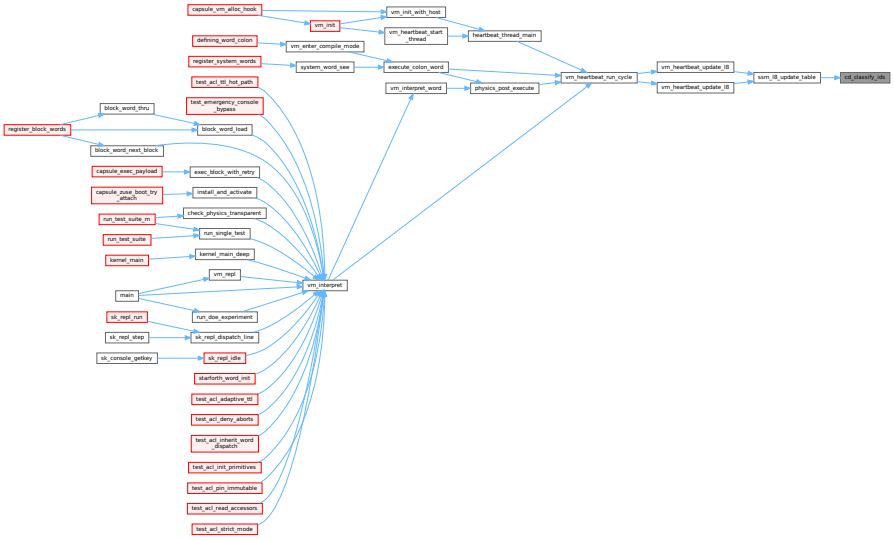
References cd_isqrt64(), CD_MAX_CLASSIFY_DEPTH, count, CDRegimeMetrics::diversity_q16, CDRegimeMetrics::locality_q16, memset(), and CDRegimeMetrics::volatility_q16.

Referenced by ssm_l8_update_table().

Here is the call graph for this function:



Here is the caller graph for this function:



7.279.1.2 cd_config_table_force()

```
void cd_config_table_force (
            CDConfigTable * table,
            uint8_t config_idx )
```

Force the table onto an externally-chosen config (as if UCB1 had picked it).

One-tick nudge, not a hold – see ssm_l8_force_config().

References CDConfigTable::current_config, CDConfigTable::entries, and CDConfigTable::num_configs.

Referenced by ssm_l8_force_config().

Here is the caller graph for this function:

7.279.1.3 cd_config_table_free()

```
void cd_config_table_free (
            CDConfigTable * table )
```

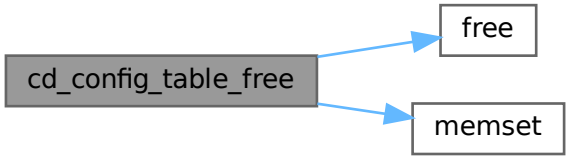
Free all memory owned by a config table.

Safe on a zeroed table.

References CDConfigTable::entries, free(), memset(), CDConfigTable::regime_scores, CDConfigTable::regime_ticks, and CDConfigTable::total_regime_ticks.

Referenced by cd_config_table_init(), and ssm_l8_free_table().

Here is the call graph for this function:



Here is the caller graph for this function:

7.279.1.4 cd_config_table_init()

```
int cd_config_table_init (
            CDConfigTable * table,
            const CDTuning * tuning )
```

Allocate a config table sized for tuning->num_configs / num_regimes.

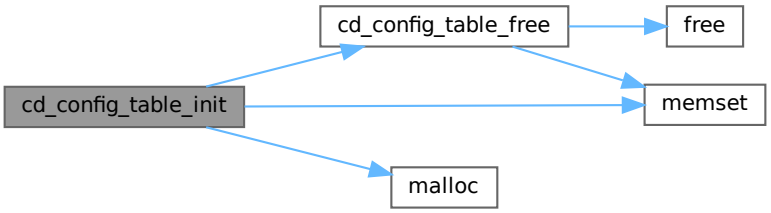
Returns

0 on success, -1 on malloc failure (table left zeroed, unusable)

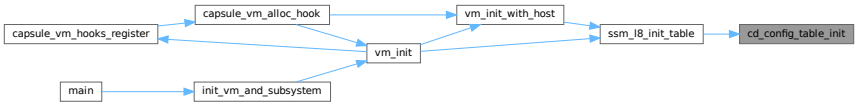
References cd_config_table_free(), CDConfigTable::entries, malloc(), memset(), CDTuning::num_configs, CDConfigTable::num_configs, CDTuning::num_regimes, CDConfigTable::num_regimes, CDConfigTable::regime_scores, CDConfigTable::regime_ticks, and CDConfigTable::total_regime_ticks.

Referenced by ssm_l8_init_table().

Here is the call graph for this function:



Here is the caller graph for this function:



7.279.1.5 cd_config_table_seed()

```
void cd_config_table_seed (
            CDConfigTable * table,
            const CDTuning * tuning,
            const uint32_t * initial_scores,
            uint8_t initial_config )
```

Seed every (regime, config) score with initial_scores[config], and set the starting active config to initial_config.

Parameters

initial_scores	Array of tuning->num_configs initial benefit scores

References CDConfigEntry::_pad, CDConfigEntry::benefit_score, CDConfigEntry::config_bits, CDConfigTable::current_config, CDConfigTable::current_regime, CDConfigTable::entries, CDConfigEntry::has_prev, CDConfigTable::num_configs, CDConfigTable::num_regimes, CDConfigEntry::prev_locality_q16, CDConfigEntry::prev_window, CDConfigTable::regime_scores, CDConfigTable::regime_ticks, CDConfigEntry::tick_count, and CDConfigTable::total_regime_ticks.

Referenced by ssm_l8_seed_from_doe().

Here is the caller graph for this function:

7.279.1.6 cd_config_table_tick()

```
uint8_t cd_config_table_tick (
            CDConfigTable * table,
            const CDTuning * tuning,
            uint8_t regime,
            uint32_t anova_stability_q16,
            int use_joint_convergence,
            uint32_t current_window,
            uint32_t current_locality_q16 )
```

Score the currently-active config from this tick's outcome and UCB1-reselect this tick's config, for the given regime.

Combined reward = joint_weight ∗ window/locality-convergence-stability

- anova_weight ∗ anova_stability_q16 Joint-convergence is skipped (falls back to ANOVA-only) on a config's first tick, or when use_joint_convergence is false (the caller's config bits don't include loops whose convergence this signal is meant to track – word level's L5/L6, see ssm_jacquard.c).

Parameters

table	Config table (must be initialised)
tuning	Tuning knobs matching this table's sizing
regime	Current regime index, < tuning->num_regimes
anova_stability_q16	Caller-supplied stability signal, Q16 [0,65536]
use_joint_convergence	Whether to include the window/locality signal
current_window	Caller's current window-width metric
current_locality_q16	Caller's current locality_q16 from cd_classify_ids

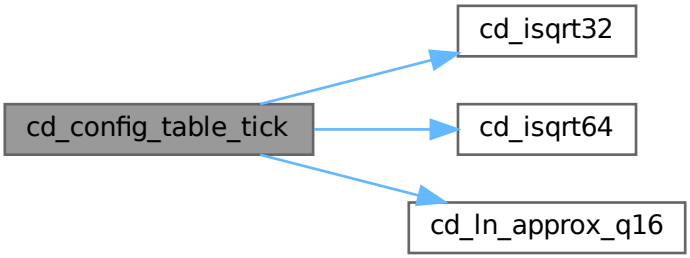
Returns

The newly-selected config's config_bits

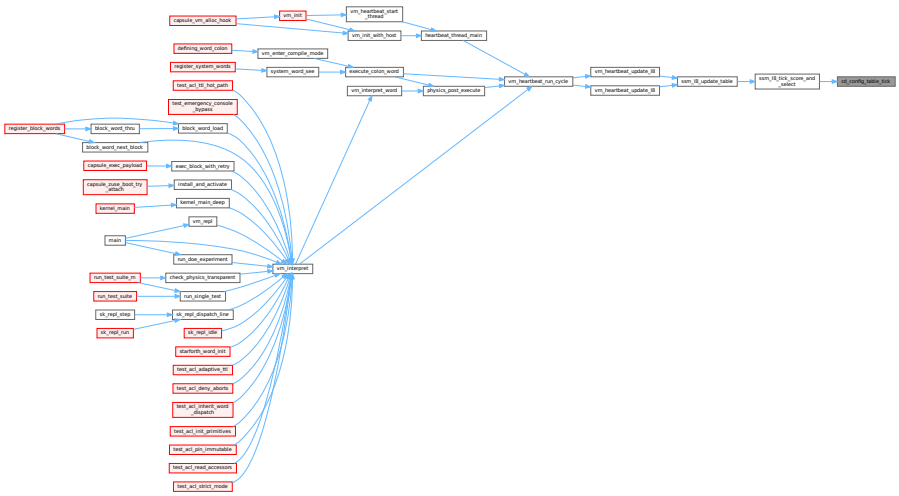
References CDTuning::anova_weight_q16, cd_isqrt32(), cd_isqrt64(), cd_ln_approx_q16(), CDConfigEntry::config_bits, CDConfigTable::current_config, CDConfigTable::current_regime, CDConfigTable::entries, entry, CDTuning::joint_weight_q16, CDConfigTable::num_configs, CDConfigTable::regime_scores, CDConfigTable::regime_ticks, CDTuning::reward_gain, CDTuning::score_decay_factor_q16, CDTuning::score_max, CDTuning::score_min, CDConfigTable::total_regime_ticks, and CDTuning::ucb_k.

Referenced by ssm_l8_tick_score_and_select().

Here is the call graph for this function:



Here is the caller graph for this function:

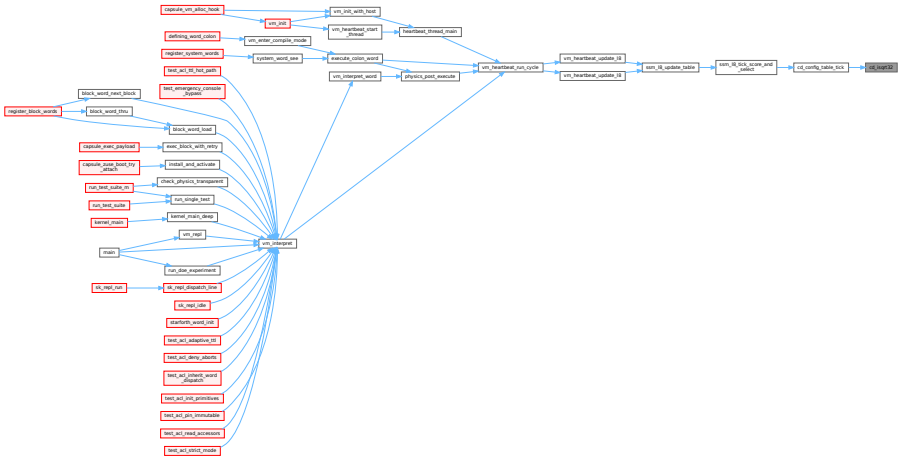


7.279.1.7 cd_isqrt32()

```
static uint32_t cd_isqrt32 (
            uint32_t n )  [static]
```

Referenced by cd_config_table_tick().

Here is the caller graph for this function:

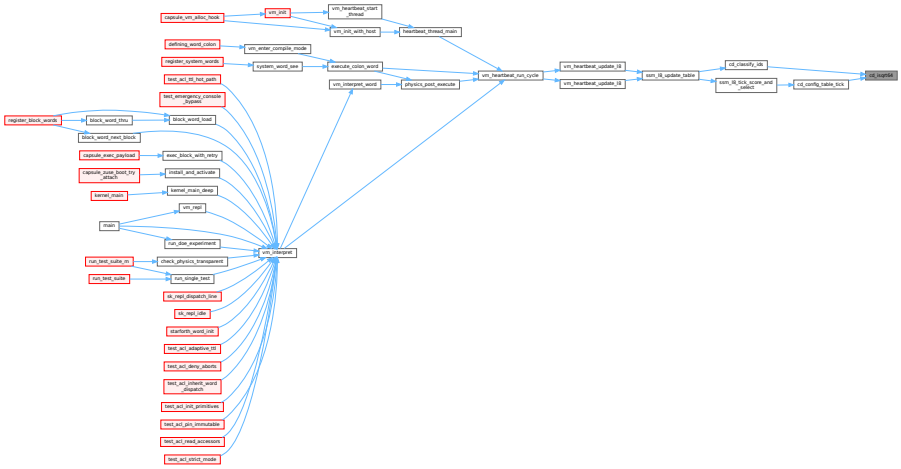


7.279.1.8 cd_isqrt64()

```
static uint64_t cd_isqrt64 (
            uint64_t n )  [static]
```

Referenced by cd_classify_ids(), and cd_config_table_tick().

Here is the caller graph for this function:

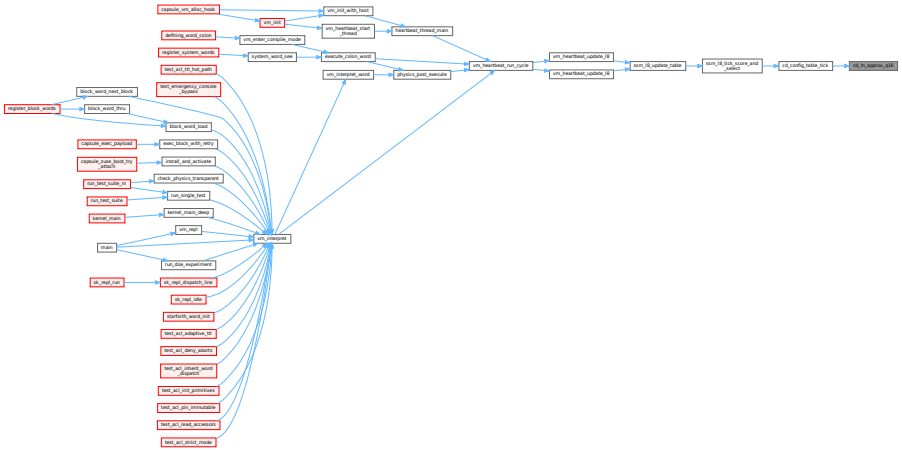


7.279.1.9 cd_ln_approx_q16()

```
static uint32_t cd_ln_approx_q16 (
            uint32_t n )  [static]
```

Referenced by cd_config_table_tick().

Here is the caller graph for this function:



7.279.1.10 cd_tuning_vm()

```
CDTuning cd_tuning_vm (
            void  )
```

References CDTuning::anova_weight_q16, CDTuning::diversity_high_threshold_q16, CDTuning::id_window_depth, CDTuning::joint_weight_q16, CDTuning::locality_high_threshold_q16, CDTuning::locality_lookback, CDTuning::num_configs, CDTuning::num_regimes, CDTuning::reward_gain, CDTuning::score_decay_factor_q16, CDTuning::score_max, CDTuning::score_min, CDTuning::ucb_k, and CDTuning::volatility_high_threshold_q16.

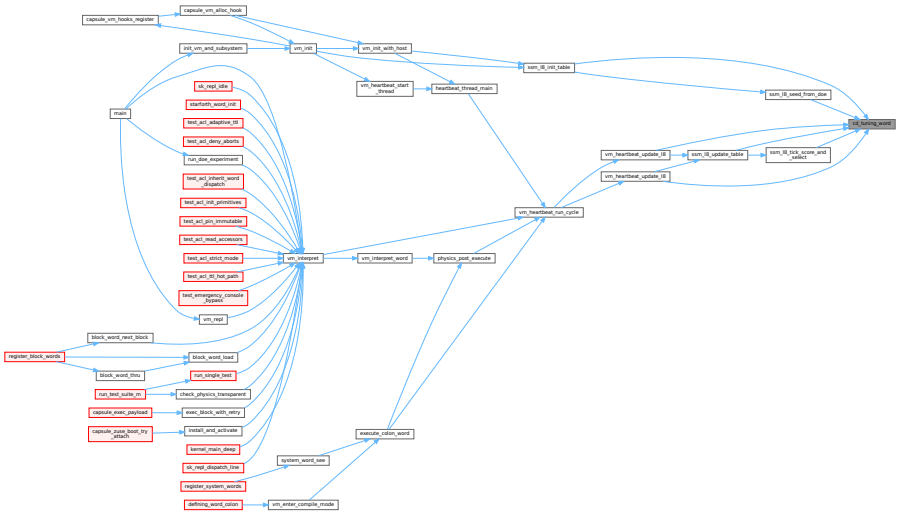
7.279.1.11 cd_tuning_word()

```
CDTuning cd_tuning_word (
            void  )
```

References CDTuning::anova_weight_q16, CDTuning::diversity_high_threshold_q16, CDTuning::id_window_depth, CDTuning::joint_weight_q16, CDTuning::locality_high_threshold_q16, CDTuning::locality_lookback, CDTuning::num_configs, CDTuning::num_regimes, CDTuning::reward_gain, CDTuning::score_decay_factor_q16, CDTuning::score_max, CDTuning::score_min, CDTuning::ucb_k, and CDTuning::volatility_high_threshold_q16.

Referenced by ssm_l8_init_table(), ssm_l8_seed_from_doe(), ssm_l8_tick_score_and_select(), ssm_l8_update_table(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

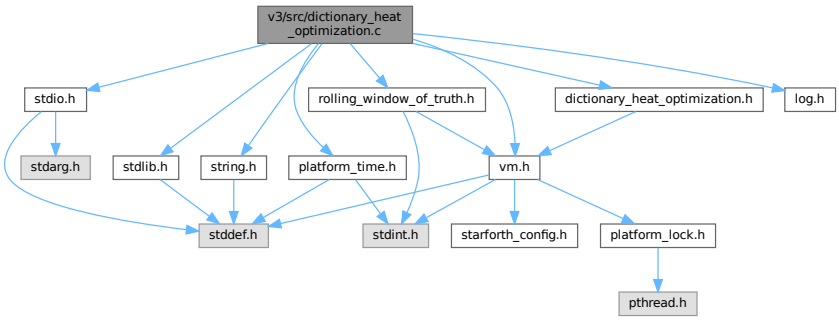
Here is the caller graph for this function:



7.280 v3/src/dictionary_heat_optimization.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "vm.h"
#include "rolling_window_of_truth.h"
#include "dictionary_heat_optimization.h"
#include "log.h"
#include "platform_time.h"
```

Include dependency graph for dictionary_heat_optimization.c:



Functions

- static int compare_heat_ascending (const void *a, const void *b)

  qsort comparator for `cell_t` execution-heat values in ascending order.
- void dict_update_heat_percentiles (VM *vm)

  Collect dictionary execution heats and compute the 25th, 50th, and 75th percentiles.

- DictEntry * dict_find_word_heat_aware (VM *vm, const char *name, size_t len)

  Three-pass heat-stratified dictionary lookup, hottest tier first.
- static int compare_dict_entry_heat_desc (const void *a, const void *b)

  qsort comparator for `DictEntry*` pointers in descending execution-heat order.
- void dict_reorganize_buckets_by_heat (VM *vm)

  Sort all 256 first-character hash buckets by descending execution heat.
- void dict_adaptive_optimization_pass (VM *vm)

  Throttled adaptive optimization pass — runs at most once per second.

7.280.1 Function Documentation

7.280.1.1 compare_dict_entry_heat_desc()

```
static int compare_dict_entry_heat_desc (
            const void * a,
            const void * b )  [static]
```

qsort comparator for `DictEntry*` pointers in descending execution-heat order.

Dereferences two `DictEntry**` pointers and compares their `execution_heat` fields, returning ?1 if the first entry is hotter, +1 if the second is hotter, or 0 on a tie. Used exclusively by `dict_reorganize_buckets_by_heat()` to sort each of the 256 first-character hash buckets so that hot words appear at lower array indices, reducing the average scan depth during the medium and cold passes of `dict_find_word_heat_aware()`.

Parameters

a	Pointer to a `DictEntry*` (first element).
b	Pointer to a `DictEntry*` (second element).

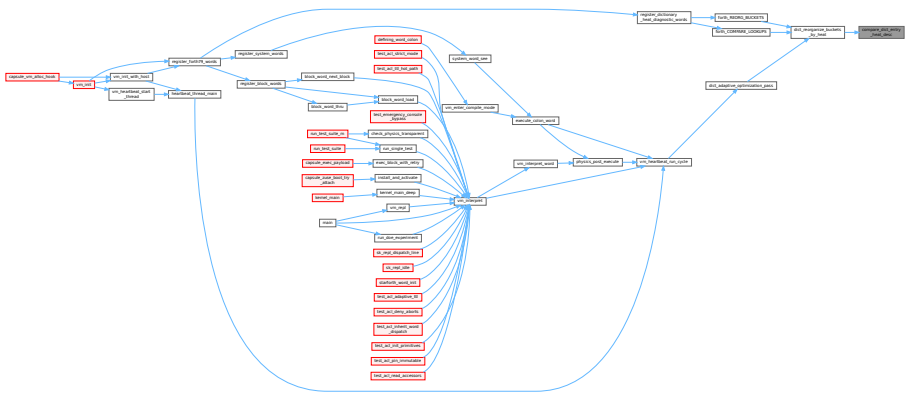
Returns

?1 if `a` is hotter; +1 if `b` is hotter; 0 if equal.

References DictEntry::execution_heat.

Referenced by dict_reorganize_buckets_by_heat().

Here is the caller graph for this function:



7.280.1.2 compare_heat_ascending()

```
static int compare_heat_ascending (
            const void * a,
            const void * b )  [static]
```

qsort comparator for `cell_t` execution-heat values in ascending order.

Compares two `cell_t` values passed as `const void*` pointers (the signature required by `qsort`). Returns ?1, 0, or +1 following C standard three-way comparison convention. Used exclusively by `dict_update_heat_percentiles()` to sort the scratch `heats[]` array before computing the 25th, 50th, and 75th percentile thresholds.

Parameters

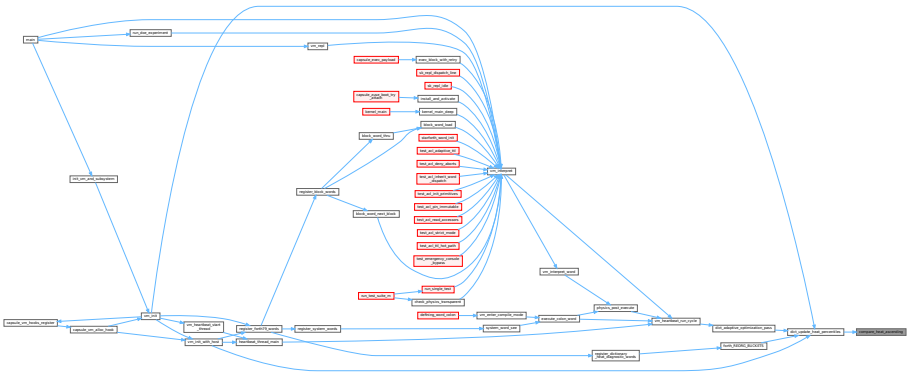
a	Pointer to the first `cell_t` value.
b	Pointer to the second `cell_t` value.

Returns

?1 if $*a < *b$; +1 if $*a > *b$; 0 if equal.

Referenced by dict_update_heat_percentiles().

Here is the caller graph for this function:



7.280.1.3 dict_adaptive_optimization_pass()

```
void dict_adaptive_optimization_pass (
            VM * vm )
```

Throttled adaptive optimization pass — runs at most once per second.

Periodically reoptimize dictionary based on latest heat patterns.

Called from the interpreter loop (or heartbeat) on every word dispatch; the 1 Hz gate (`now_ns - vm->last_↩ bucket_reorg_ns` < 1,000,000,000) ensures the three expensive sub-steps run at a rate that is imperceptible to FORTH execution while still keeping the dictionary layout fresh:

1. **Update heat percentiles** — calls `dict_update_heat_percentiles()` to re-sample `execution_heat` across all words and compute the 25th / 50th / 75th thresholds used by the stratified lookup.
2. **Reorganize buckets** — calls `dict_reorganize_buckets_by_heat()` to sort each of the 256 first-character hash buckets so that the hottest words sit at the lowest array indices, shortening the average scan depth during the hot-tier pass.
3. **Select lookup strategy** — queries the rolling window for the current pattern-diversity score via `rolling_window_measure_diversity()`:
   - Diversity $>$ 70 (high churn in which words are called) $\rightarrow$ strategy 1 (`heat-aware` lookup, `dict_find_word_heat_aware()`), where the heat stratification most pays off.
   - Diversity $\leq$ 70 (stable repetitive pattern) $\rightarrow$ strategy 0 (naive newest-first bucket scan), which is already nearly O(1) for programs that call the same small word set repeatedly. The chosen strategy is stored in `vm->lookup_strategy` and read by the interpreter's dictionary lookup dispatch.

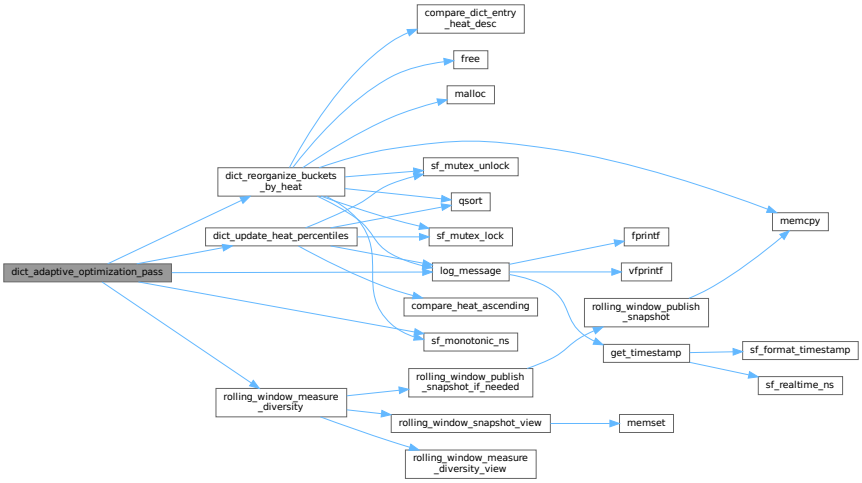
Parameters

vm	The VM to optimise. No-op if NULL or if less than 1 second has elapsed since the last pass (`vm->last_bucket_reorg_ns`).

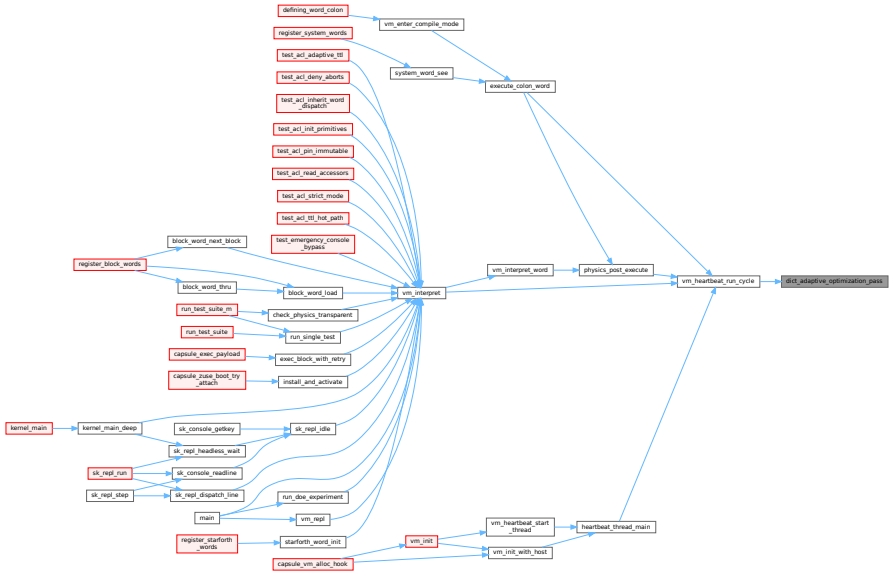
References dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), VM::last_bucket_reorg_ns, LOG_DEBUG, log_message(), VM::lookup_strategy, VM::rolling_window, rolling_window_measure_diversity(), and sf_monotonic_ns().

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.280.1.4 dict_find_word_heat_aware()

```
DictEntry * dict_find_word_heat_aware (
            VM * vm,
            const char * name,
            size_t len )
```

Three-pass heat-stratified dictionary lookup, hottest tier first.

Find word with heat-aware search strategy.

Searches the first-character bucket `sf_fc_list`[(unsigned char)name[0]] for a word whose `name` and `name_len` match `name` / `len`, scanning in three priority tiers ordered by execution heat:

1. **Hot pass** (heat $\geq$ `heat_threshold_75th`): words in the top 25% by execution frequency. Most FORTH programs spend the vast majority of ticks in a handful of hot primitives; checking this tier first yields a near-constant-time hit on steady-state workloads.
2. **Medium pass** (heat in [threshold_25th, threshold_75th)): the middle 50% of words. These are occasionally-called words that still appear often enough to warrant a dedicated pass before touching the long tail.
3. **Cold pass** (heat $<$ `heat_threshold_25th`): the rarely-executed bottom quartile. Words that have never been called also live here.

Within each pass, matching uses the same cheap-first filter chain as the standard lookup: `name_len` comparison $\rightarrow$ last-character byte check $\rightarrow$ full `memcmp`. The last-character pre-filter is skipped for single-character names.

Returns `NULL` only if the word is absent from the bucket; all three passes exhausted without a hit. Also returns `NULL` immediately if `vm`, `name`, or `len` are invalid.

Heat thresholds are updated lazily by `dict_adaptive_optimization_pass()` ($\approx$1 Hz); stale thresholds degrade to a full three-pass scan without affecting correctness.

Parameters

vm	The VM; provides `heat_threshold_*` fields and the bucket table.
name	Word name to search for (not NUL-terminated; length in `len`).
len	Length of `name` in bytes; must be ≥ 1.

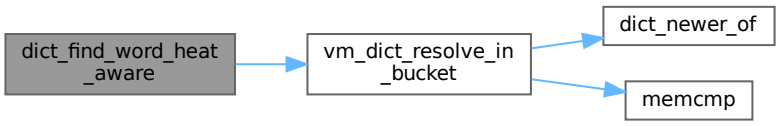
Returns

Pointer to the matching `DictEntry`, or `NULL` if not found.

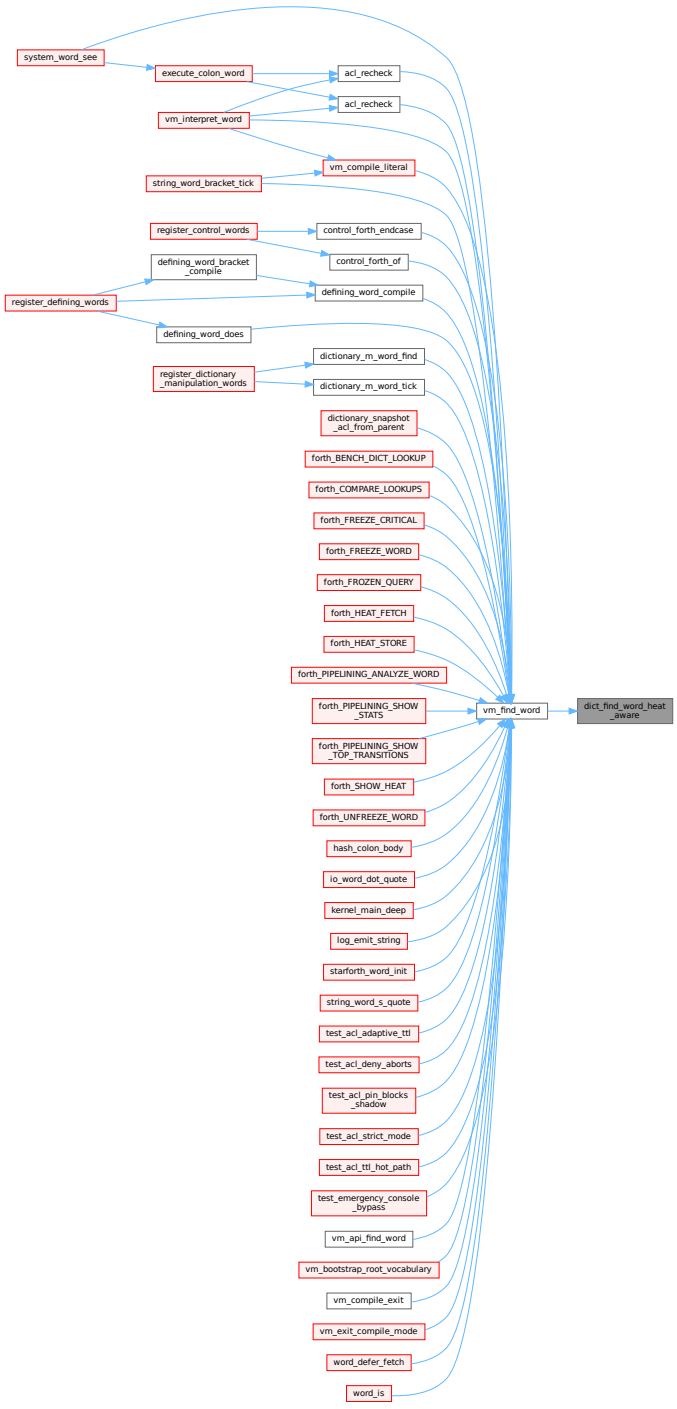
References name, sf_fc_count, sf_fc_list, and vm_dict_resolve_in_bucket().

Referenced by vm_find_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.280.1.5 dict_reorganize_buckets_by_heat()

```
void dict_reorganize_buckets_by_heat (
            VM * vm )
```

Sort all 256 first-character hash buckets by descending execution heat.

Reorganize dictionary buckets by inference-driven priorities.

Holds `vm->dict_lock` for the entire operation to prevent concurrent `DictEntry` frees (from word redefinition or GC) from invalidating the pointer array while it is being sorted. For each non-empty bucket in `sf_fc_list`[0..255]:

1. Reads `sf_fc_count`[bucket_id] and `sf_fc_cap`[bucket_id]; skips the bucket if `count > cap` (indicates a TOCTOU race — logs a warning).
2. Allocates a temporary copy of the bucket pointer array.
3. Re-verifies `count` has not changed (TOCTOU double-check); if it has, frees the copy and skips.
4. Sorts the copy with `qsort` + `compare_dict_entry_heat_desc` so that the hottest `DictEntry*` comes first.
5. Copies the sorted array back over the original bucket.
6. Frees the temporary copy.

After returning, `vm->last_bucket_reorg_ns` is updated to the completion timestamp so that `dict_adaptive_optimization_pass()` can enforce its 1 Hz throttle. The elapsed reorganization time is emitted at `LOG_DEBUG`.

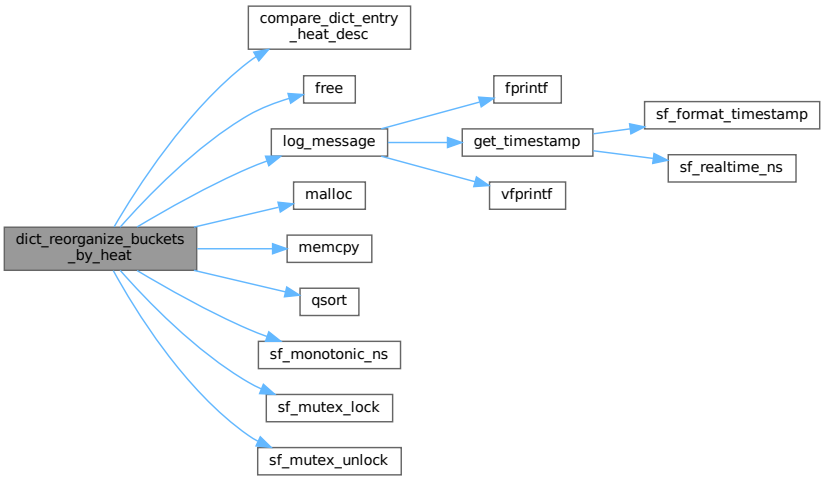
Parameters

vm	The VM whose first-character hash buckets are reorganized. No-op if NULL. `dict_lock` is held during the entire scan.

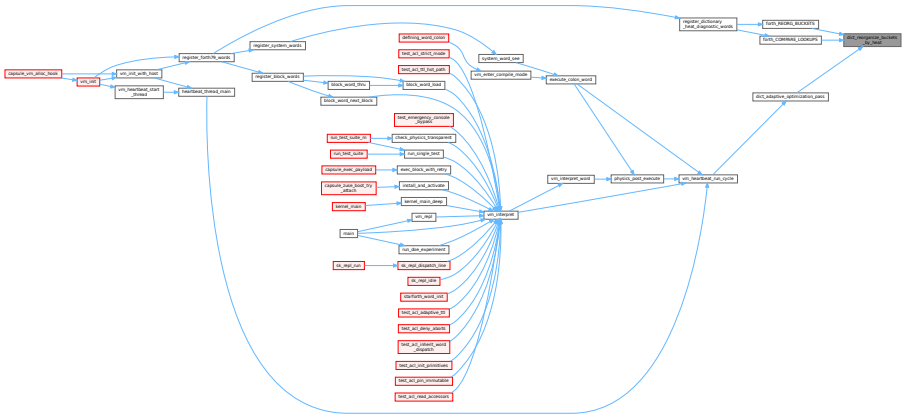
References compare_dict_entry_heat_desc(), count, VM::dict_lock, free(), VM::last_bucket_reorg_ns, LOG_DEBUG, log_message(), LOG_WARN, malloc(), memcpy(), qsort(), sf_fc_cap, sf_fc_count, sf_fc_list, sf_monotonic_ns(), sf_mutex_lock(), and sf_mutex_unlock().

Referenced by dict_adaptive_optimization_pass(), forth_COMPARE_LOOKUPS(), and forth_REORG_BUCKETS().

Here is the call graph for this function:



Here is the caller graph for this function:



7.280.1.6 dict_update_heat_percentiles()

```
void dict_update_heat_percentiles (
                VM * vm )
```

Collect dictionary execution heats and compute the 25th, 50th, and 75th percentiles.

Calculate heat percentile thresholds for all words in dictionary.

Traverses the full linked-list dictionary under `vm->dict_lock` to snapshot every word's `execution_heat` into a stack-allocated scratch array (up to `DICTIONARY_SIZE` entries). After releasing the lock it sorts the snapshot with `qsort` + `compare_heat_ascending` and computes percentile indices using integer arithmetic:

```
vm->heat_threshold_25th = heats[(count * 25) / 100];
vm->heat_threshold_50th = heats[(count * 50) / 100];
vm->heat_threshold_75th = heats[(count * 75) / 100];
```

If the dictionary contains no words all three thresholds are set to 0. The lock is held only during traversal; sorting is performed outside the critical section to minimise contention with the concurrent heartbeat decay sweep that also walks the word list.

The resulting thresholds are consumed by `dict_find_word_heat_aware()` (which stratifies its search by heat tier) and by `dict_adaptive_optimization_pass()` (which calls this function as its first step every ~1 Hz).

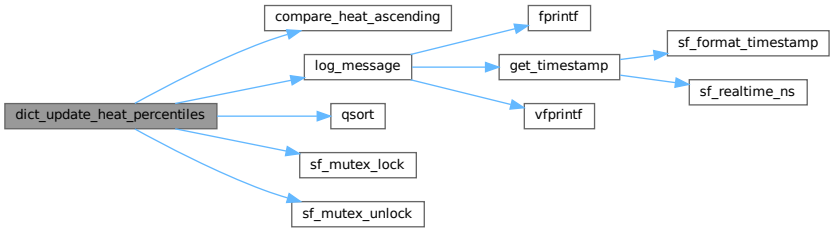
Parameters

vm	The VM whose dictionary is sampled and whose `heat_threshold_25th` / `heat_threshold_50th` / `heat_threshold_75th` fields are updated. No-op if NULL.

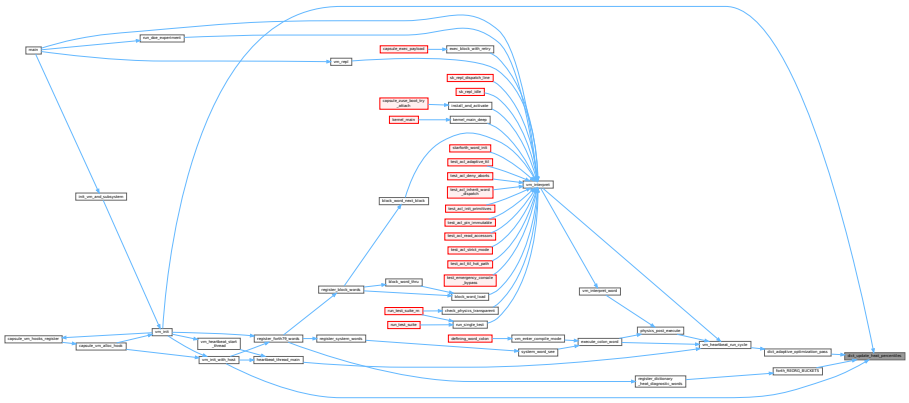
References compare_heat_ascending(), count, VM::dict_lock, DICTIONARY_SIZE, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::latest, LOG_DEBUG, log_message(), qsort(), sf_mutex_lock(), and sf_mutex_unlock().

Referenced by dict_adaptive_optimization_pass(), forth_REORG_BUCKETS(), vm_init(), and vm_init_with_host().

Here is the call graph for this function:



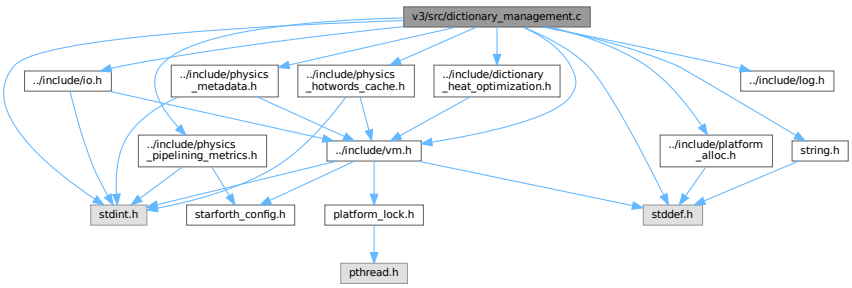
Here is the caller graph for this function:



7.281 v3/src/dictionary_management.c File Reference

```
#include "../include/vm.h"
#include "../include/io.h"
#include "../include/log.h"
#include "../include/physics_metadata.h"
#include "../include/physics_hotwords_cache.h"
#include "../include/physics_pipelining_metrics.h"
#include "../include/dictionary_heat_optimization.h"
#include "../include/platform_alloc.h"
#include <string.h>
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for dictionary_management.c:



Macros

- #define SF_FC_BUCKETS 256

Functions

- static uint32_t vm_dictionary_acquire_word_id (VM *vm)
- void vm_dictionary_track_entry (VM *vm, DictEntry *entry)
- void vm_dictionary_untrack_entry (VM *vm, DictEntry *entry)
- DictEntry * vm_dictionary_lookup_by_word_id (VM *vm, uint32_t word_id)
- int vm_dict_entry_ok (VM *vm, DictEntry *candidate)
- static void * sf_xrealloc (void *p, size_t nbytes)
- static void sf_fc_clear_all (void)
- static int sf_fc_reserve (size_t i, size_t need)
- static void sf_rebuild_fc_index (VM *vm)
- static void sf_try_fast_append (VM *vm)
- static DictEntry * dict_newer_of (VM *vm, DictEntry *a, DictEntry *b)

  Return whichever of two entries is the more recent definition.
- DictEntry * vm_dict_resolve_in_bucket (VM *vm, DictEntry **bucket, size_t n, const char *name, size_t len)
- DictEntry * vm_find_word (VM *vm, const char *name, size_t len)

  Finds a word in the dictionary by name.
- DictEntry * vm_create_word (VM *vm, const char *name, size_t len, word_func_t func)

  Creates a new word in the dictionary.
- DictEntry * vm_dictionary_find_by_func (VM *vm, word_func_t func)

  Finds a word by its function pointer.
- DictEntry * vm_dictionary_find_latest_by_func (VM *vm, word_func_t func)
- cell_t * vm_dictionary_get_data_field (DictEntry *entry)

  Gets pointer to a word's data field.
- void vm_hide_word (VM *vm)

  Marks the latest word as hidden.
- void vm_smudge_word (VM *vm)
- void vm_pin_execution_heat (VM *vm)
- void vm_unpin_execution_heat (VM *vm)
- void vm_dictionary_index_reset (void)

  Resets the dictionary index.

Variables

- DictEntry ** sf_fc_list [256]
- size_t sf_fc_count [256]
- size_t sf_fc_cap [256]
- static DictEntry * sf_cached_latest = NULL

7.281.1 Macro Definition Documentation

7.281.1.1 SF_FC_BUCKETS

```
#define SF_FC_BUCKETS 256
```

7.281.2 Function Documentation

7.281.2.1 dict_newer_of()

```
static DictEntry * dict_newer_of (
            VM * vm,
            DictEntry * a,
            DictEntry * b )  [static]
```

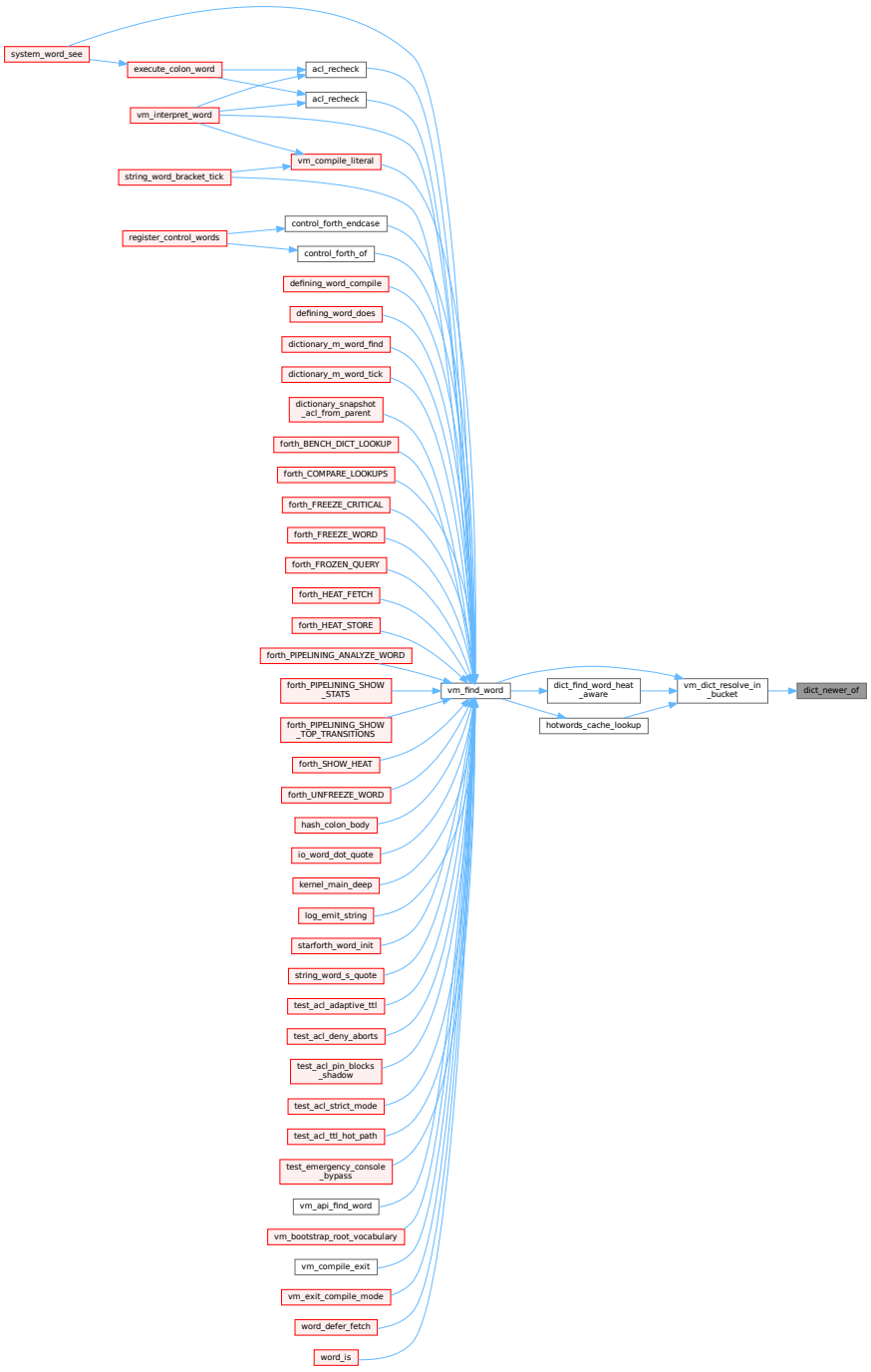
Return whichever of two entries is the more recent definition.

Walks the definition chain from `vm->latest`; the first of (`a`, `b`) encountered is the newer one. The link chain is the only reliable age order — word_id values are recycled and bucket positions are shuffled by `dict_reorganize_buckets_by_heat()`, so neither can arbitrate age. Only invoked when a bucket holds two visible entries with the same name (redefinition/shadowing), so the walk cost is off the common path.

References VM::latest.

Referenced by vm_dict_resolve_in_bucket().

Here is the caller graph for this function:



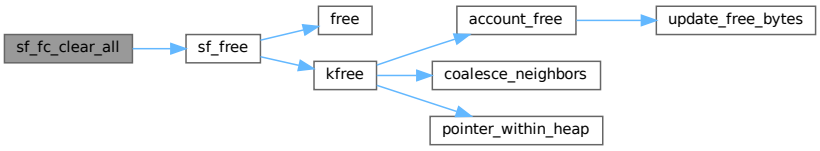
7.281.2.2 sf_fc_clear_all()

```
static void sf_fc_clear_all (
            void  )  [static]
```

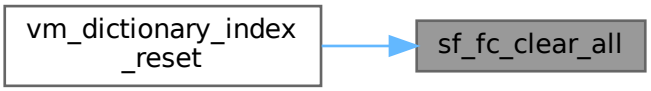
References SF_FC_BUCKETS, sf_fc_cap, sf_fc_count, sf_fc_list, and sf_free().

Referenced by vm_dictionary_index_reset().

Here is the call graph for this function:



Here is the caller graph for this function:



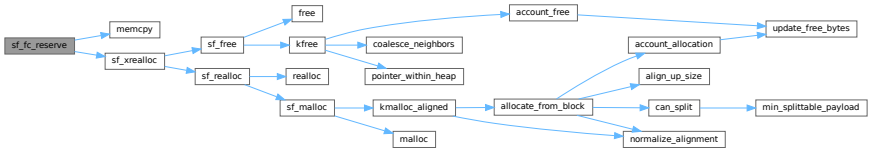
7.281.2.3 sf_fc_reserve()

```
static int sf_fc_reserve (
            size_t i,
            size_t need )  [static]
```

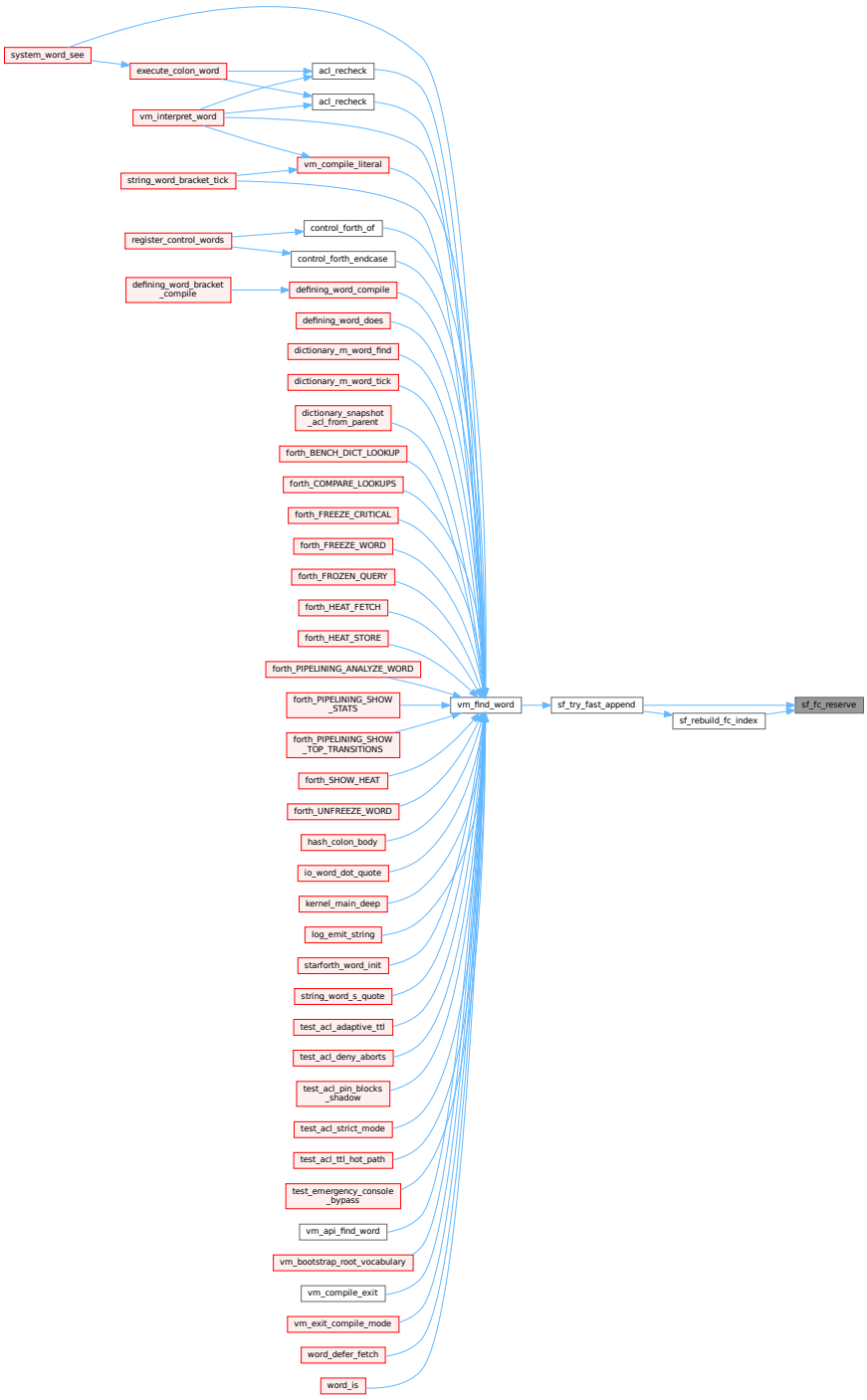
References memcpy(), sf_fc_cap, sf_fc_count, sf_fc_list, and sf_xrealloc().

Referenced by sf_rebuild_fc_index(), and sf_try_fast_append().

Here is the call graph for this function:



Here is the caller graph for this function:



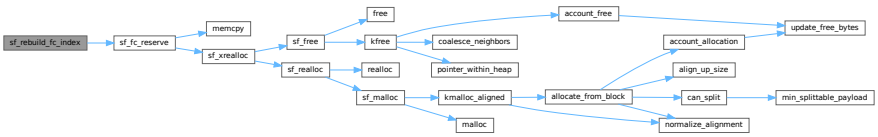
7.281.2.4 sf_rebuild_fc_index()

```
static void sf_rebuild_fc_index (
            VM * vm )  [static]
```

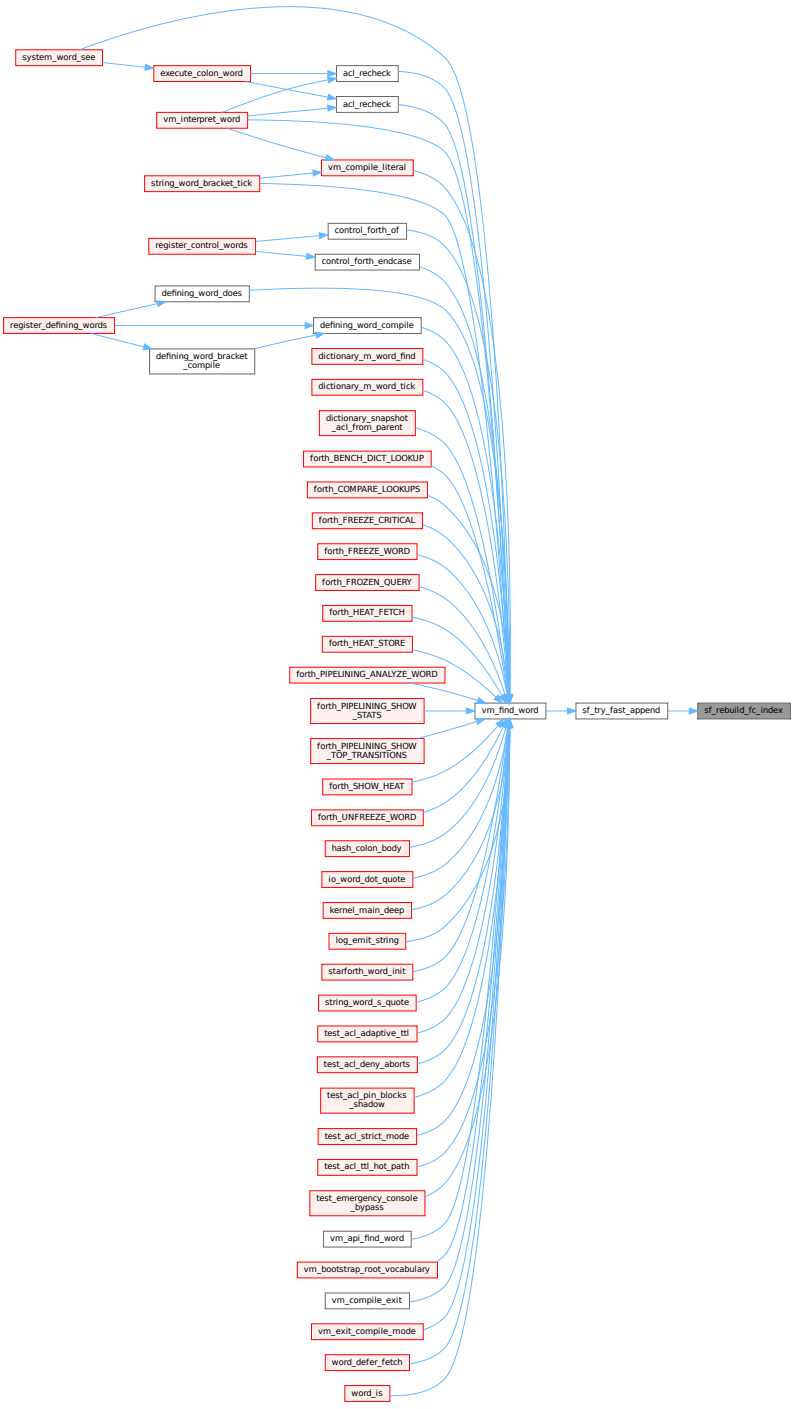
References VM::latest, DictEntry::name, sf_cached_latest, SF_FC_BUCKETS, sf_fc_count, sf_fc_list, and sf_fc_reserve().

Referenced by sf_try_fast_append().

Here is the call graph for this function:



Here is the caller graph for this function:



7.281.2.5 sf_try_fast_append()

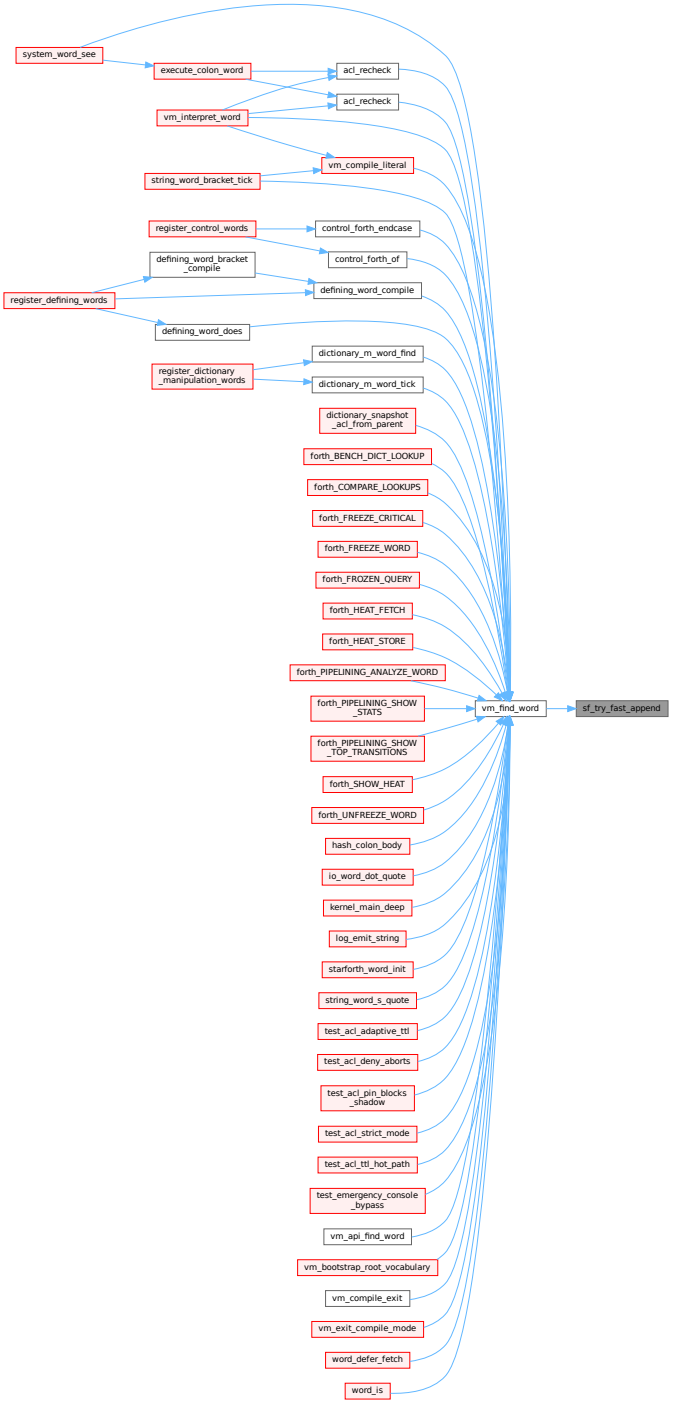
```
static void sf_try_fast_append (
            VM * vm )  [static]
```

References VM::latest, DictEntry::link, DictEntry::name, sf_cached_latest, sf_fc_count, sf_fc_list, sf_fc_reserve(), and sf_rebuild_fc_index().

Referenced by vm_find_word().

Here is the call graph for this function:

Here is the caller graph for this function:



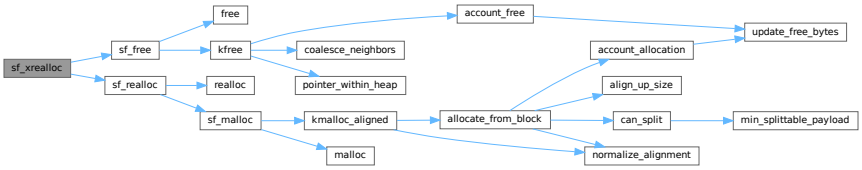
7.281.2.6 sf_xrealloc()

```
static void * sf_xrealloc (
            void * p,
            size_t nbytes )  [inline], [static]
```

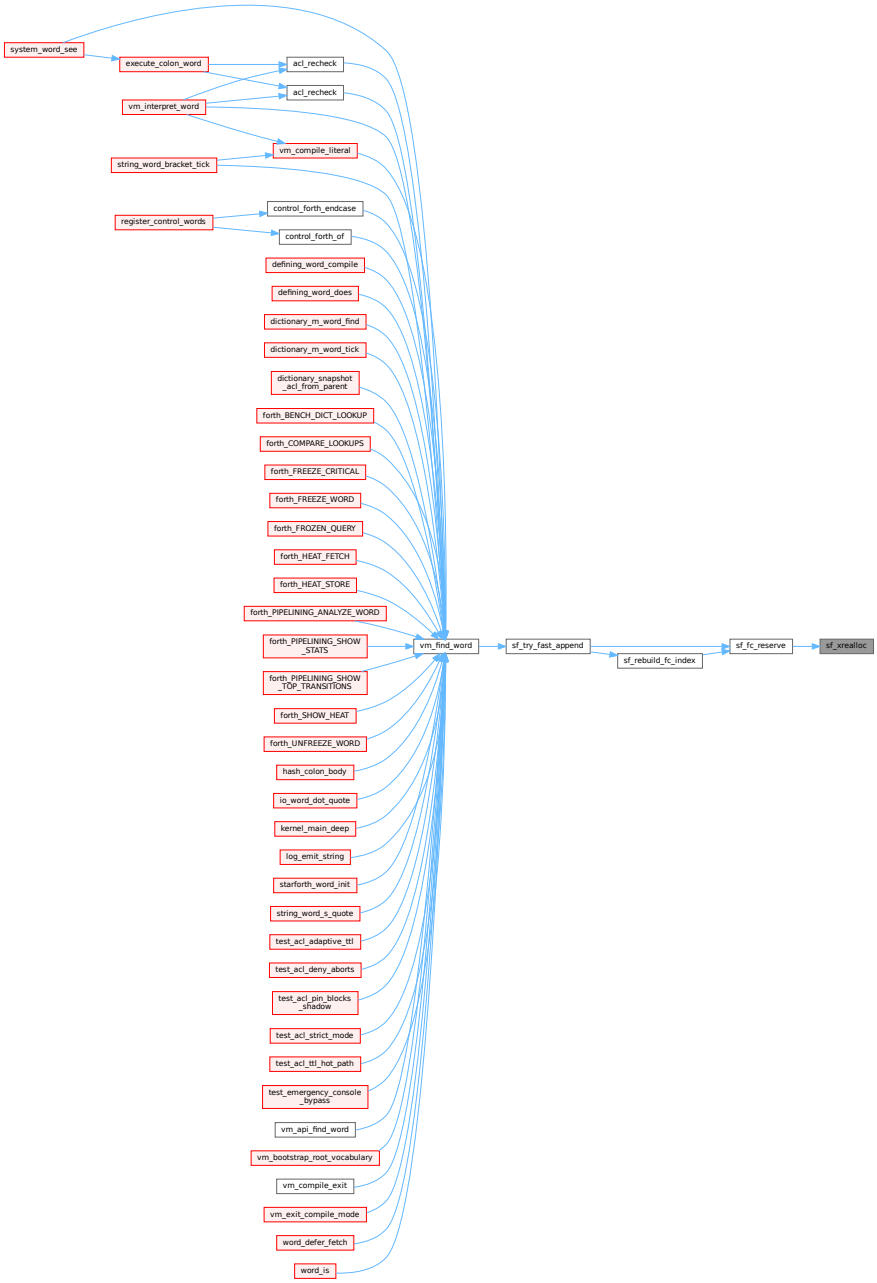
References sf_free(), and sf_realloc().

Referenced by sf_fc_reserve().

Here is the call graph for this function:



Here is the caller graph for this function:



7.281.2.7 vm_create_word()

```
DictEntry * vm_create_word (
            VM * vm,
            const char * name,
            size_t len,
            word_func_t func )
```

Creates a new word in the dictionary.

Allocates and initializes a new dictionary entry with the given name and function. The word is added to the dictionary but index update is deferred until next lookup.

Parameters

vm	The virtual machine context
name	Name for the new word
len	Length of the name string
func	Function pointer for the word's behavior

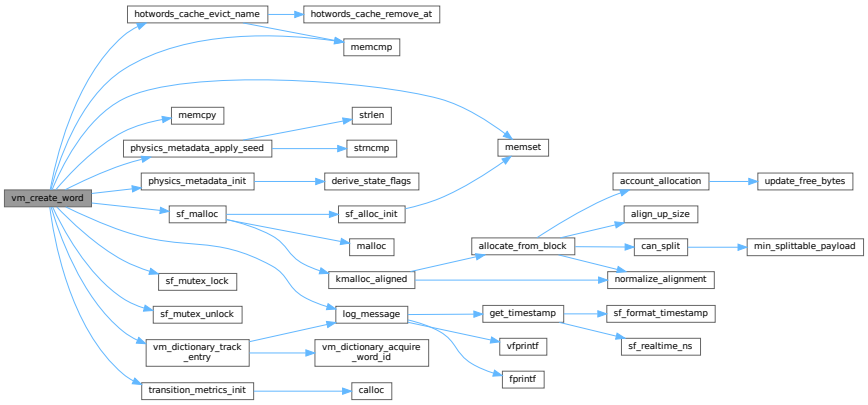
Returns

DictEntry* Pointer to new dictionary entry or NULL on failure

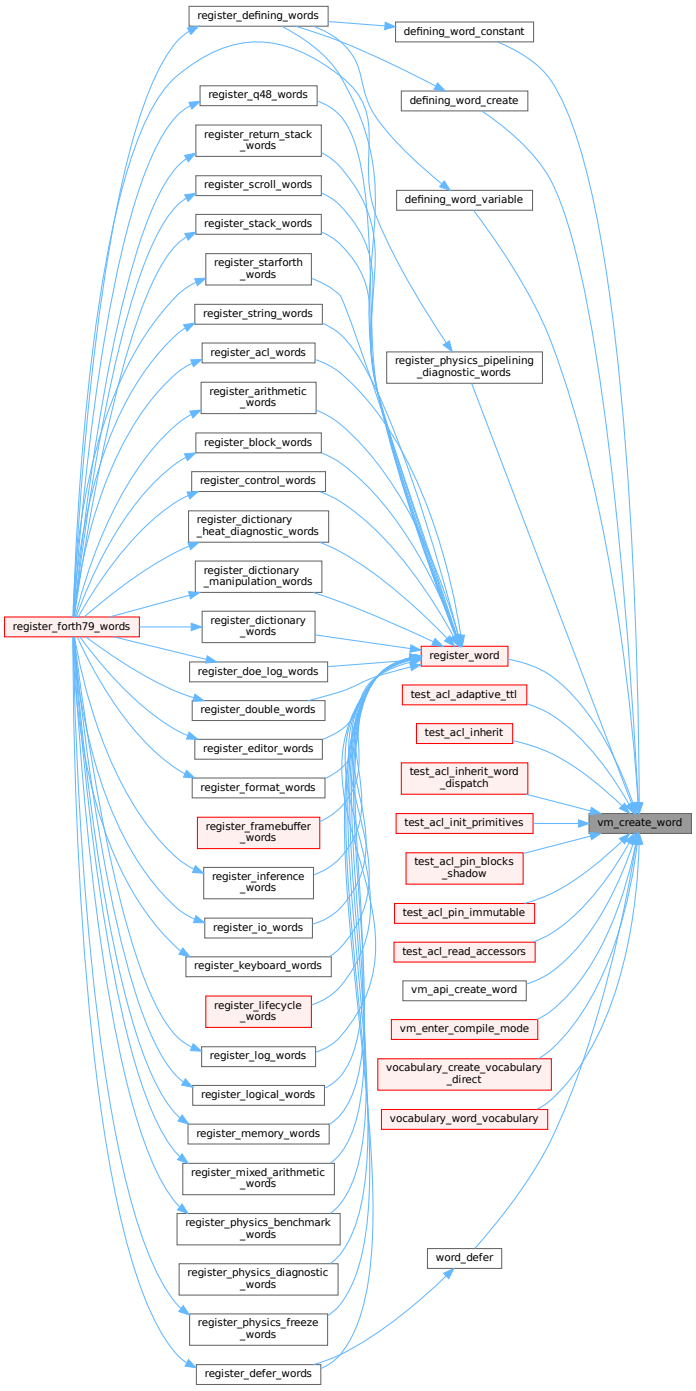
References ACL_MODE_TTL, ACL_USER_DEFAULT, base, VM::dict_lock, entry, VM::error, func, DictEntry::func, VM::hotwords_cache, hotwords_cache_evict_name(), VM::latest, LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, memcmp(), memcpy(), memset(), name, physics_metadata_apply_seed(), physics_metadata_init(), sf_malloc(), sf_mutex_lock(), sf_mutex_unlock(), transition_metrics_init(), vm_dictionary_track_entry(), WORD_ID_INVALID, and WORD_NAME_MAX.

Referenced by defining_word_constant(), defining_word_create(), defining_word_variable(), register_physics_pipelining_diagnostic_w register_word(), test_acl_adaptive_ttl(), test_acl_inherit(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_blocks_shadow(), test_acl_pin_immutable(), test_acl_read_accessors(), vm_api_create_word(), vm_enter_compile_mode(), vocabulary_create_vocabulary_direct(), vocabulary_word_vocabulary(), and word_defer().

Here is the call graph for this function:



Here is the caller graph for this function:



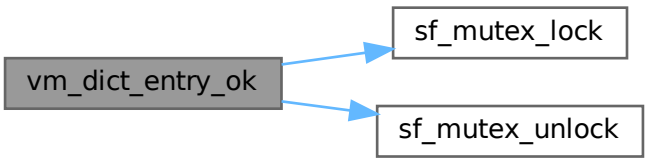
7.281.2.8 vm_dict_entry_ok()

```
int vm_dict_entry_ok (
            VM * vm,
            DictEntry * candidate )
```

References VM::dict_lock, VM::latest, sf_mutex_lock(), and sf_mutex_unlock().

Referenced by starforth_word_execution_heat_fetch(), starforth_word_execution_heat_store(), and system_word_execute().

Here is the call graph for this function:



Here is the caller graph for this function:

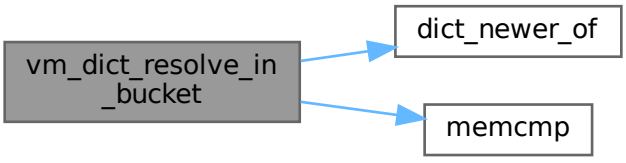
7.281.2.9 vm_dict_resolve_in_bucket()

```
DictEntry * vm_dict_resolve_in_bucket (
            VM * vm,
            DictEntry ** bucket,
            size_t n,
            const char * name,
            size_t len )
```

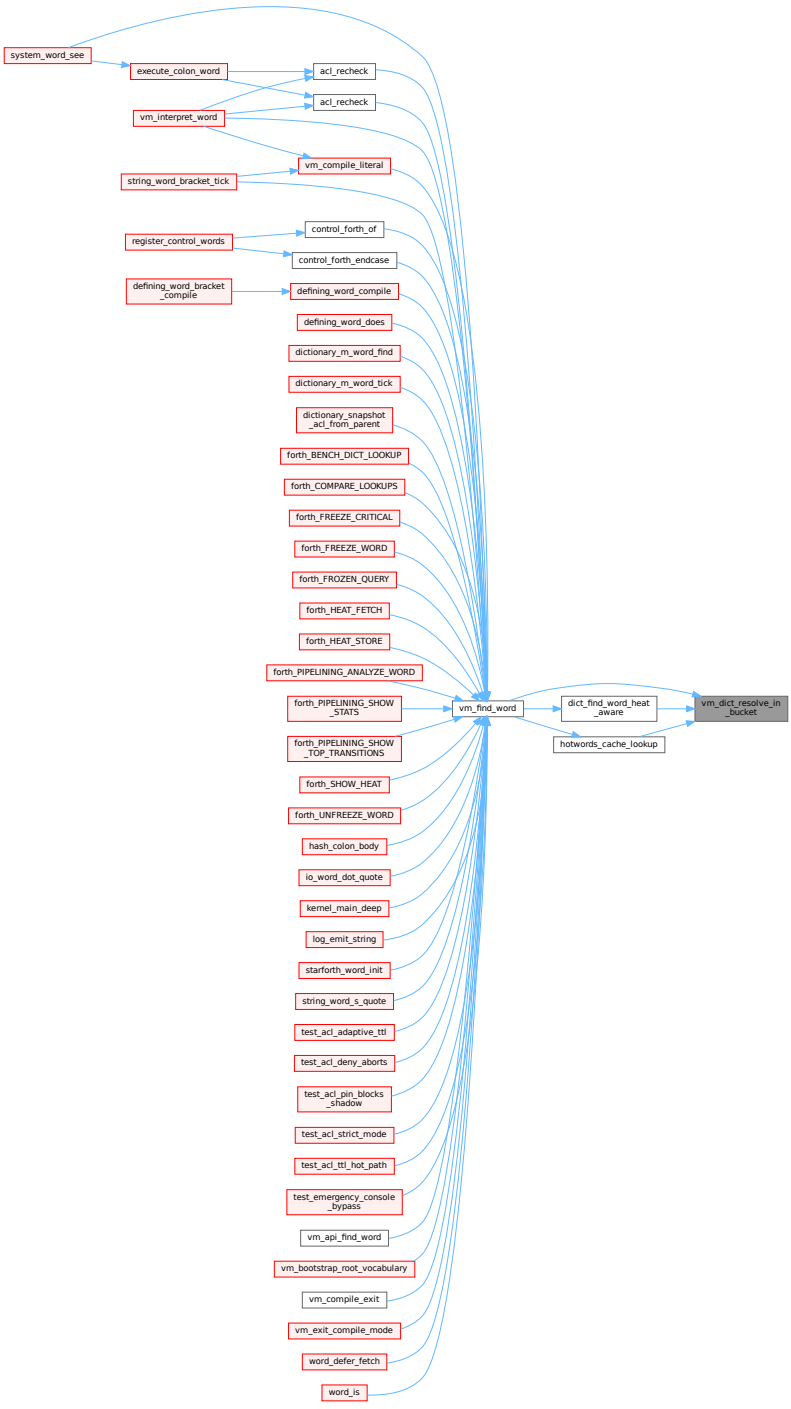
References dict_newer_of(), DictEntry::flags, memcmp(), name, DictEntry::name, DictEntry::name_len, UNLIKELY, WORD_HIDDEN, and WORD_SMUDGED.

Referenced by dict_find_word_heat_aware(), hotwords_cache_lookup(), and vm_find_word().

Here is the call graph for this function:



Here is the caller graph for this function:



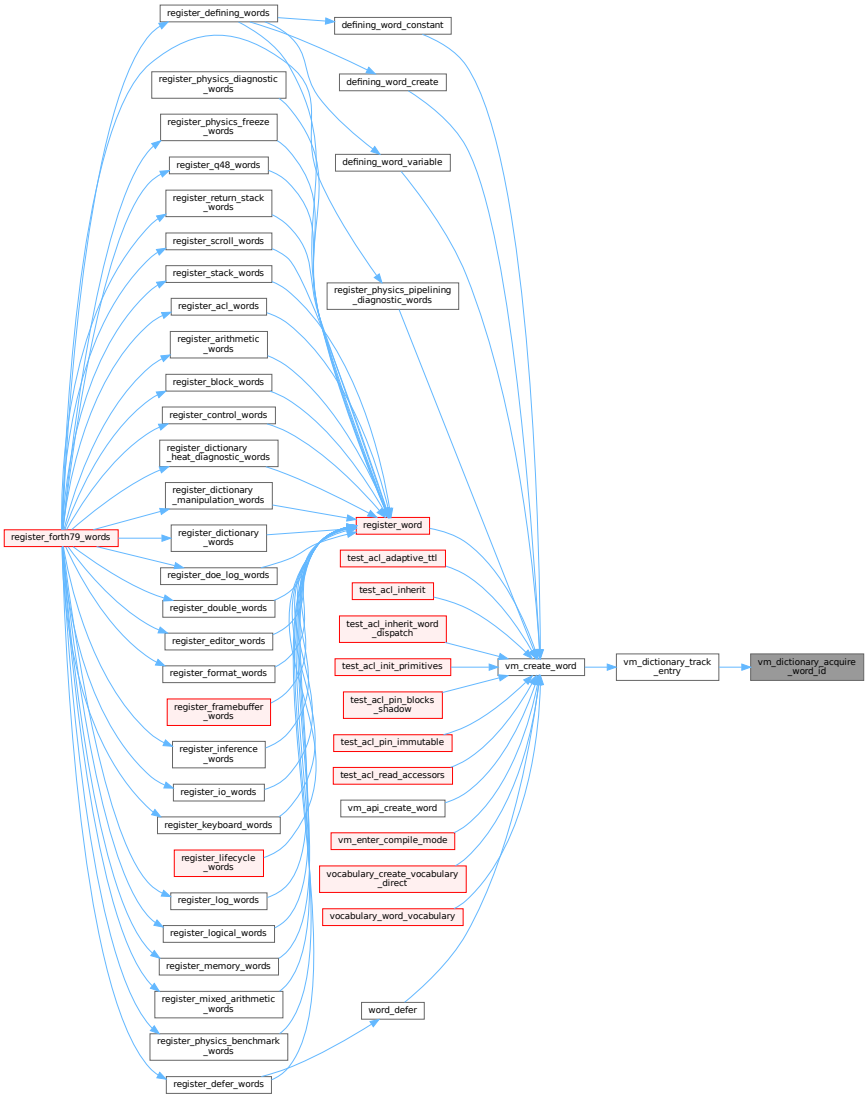
7.281.2.10 vm_dictionary_acquire_word_id()

```
static uint32_t vm_dictionary_acquire_word_id (
            VM * vm )  [static]
```

References DICTIONARY_SIZE, VM::next_word_id, VM::recycled_word_id_count, VM::recycled_word_ids, and WORD_ID_INVALID.

Referenced by vm_dictionary_track_entry().

Here is the caller graph for this function:



7.281.2.11 vm_dictionary_find_by_func()

```
DictEntry * vm_dictionary_find_by_func (
            VM * vm,
            word_func_t func )
```

Finds a word by its function pointer.

Performs a linear scan of the dictionary to find an entry with matching function.

Parameters

vm	The virtual machine context
func	Function pointer to search for

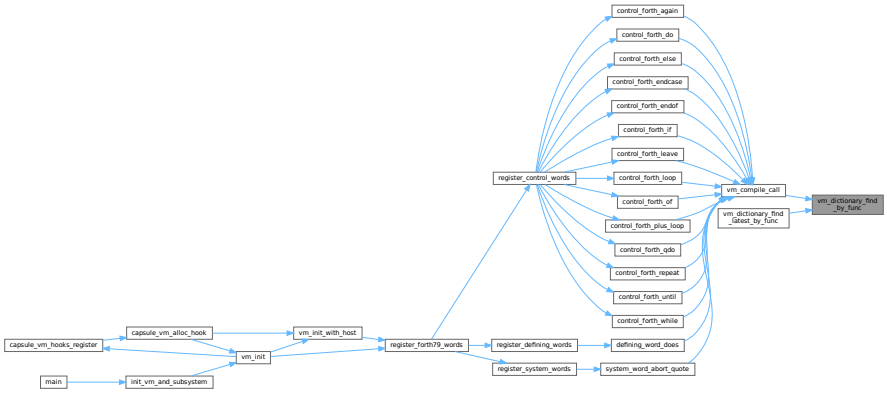
Returns

DictEntry* Pointer to found dictionary entry or NULL if not found

References func, and VM::latest.

Referenced by vm_compile_call(), and vm_dictionary_find_latest_by_func().

Here is the caller graph for this function:



7.281.2.12 vm_dictionary_find_latest_by_func()

```
DictEntry * vm_dictionary_find_latest_by_func (
            VM * vm,
            word_func_t func )
```

References func, and vm_dictionary_find_by_func().

Here is the call graph for this function:

7.281.2.13 vm_dictionary_get_data_field()

```
cell_t * vm_dictionary_get_data_field (
            DictEntry * entry )
```

Gets pointer to a word's data field.

Calculates and returns pointer to the aligned data field following the name field.

Parameters

entry	Dictionary entry to get data field from

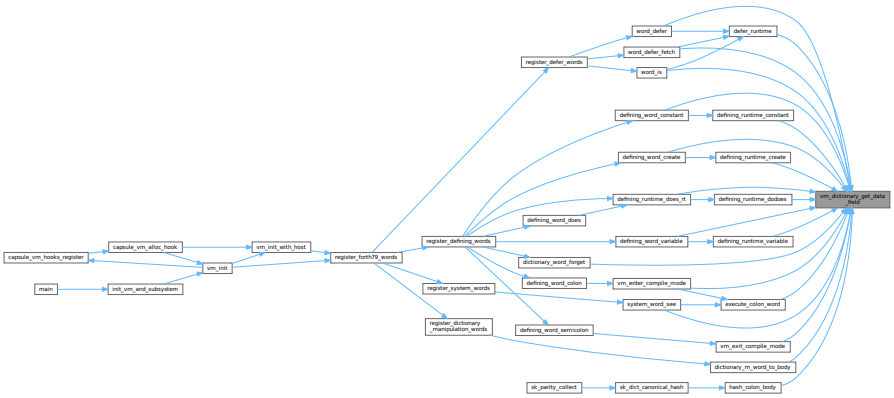
Returns

cell_t* Pointer to data field or NULL if entry is NULL

References base, entry, and name.

Referenced by defer_runtime(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_does_rt(), defining_runtime_variable(), defining_word_constant(), defining_word_create(), defining_word_variable(), dictionary_m_word_to_body(), dictionary_word_forget(), execute_colon_word(), hash_colon_body(), system_word_see(), vm_enter_compile_mode(), vm_exit_compile_mode(), word_defer(), word_defer_fetch(), and word_is().

Here is the caller graph for this function:



7.281.2.14 vm_dictionary_index_reset()

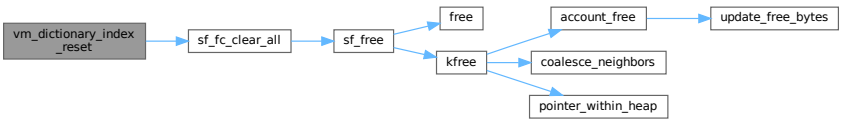
```
void vm_dictionary_index_reset (
            void  )
```

Resets the dictionary index.

Clears all index data and cached state. Used when switching vocabularies.

References sf_cached_latest, and sf_fc_clear_all().

Here is the call graph for this function:



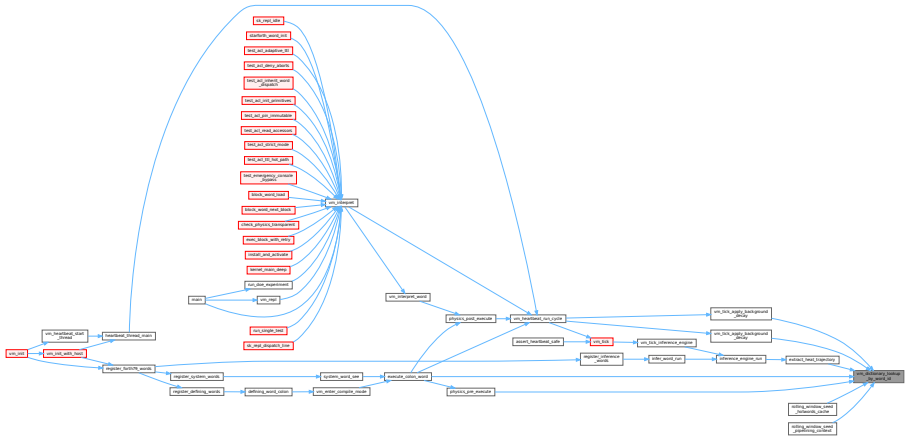
7.281.2.15 vm_dictionary_lookup_by_word_id()

```
DictEntry * vm_dictionary_lookup_by_word_id (
            VM * vm,
            uint32_t word_id )
```

References DICTIONARY_SIZE, word_id, and VM::word_id_map.

Referenced by execute_colon_word(), extract_heat_trajectory(), physics_pre_execute(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

Here is the caller graph for this function:



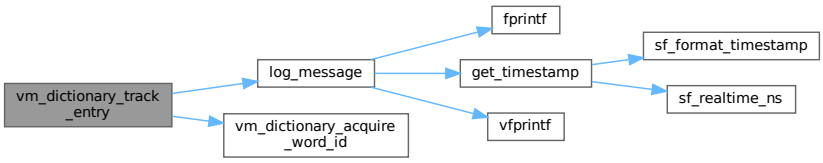
7.281.2.16 vm_dictionary_track_entry()

```
void vm_dictionary_track_entry (
            VM * vm,
            DictEntry * entry )
```

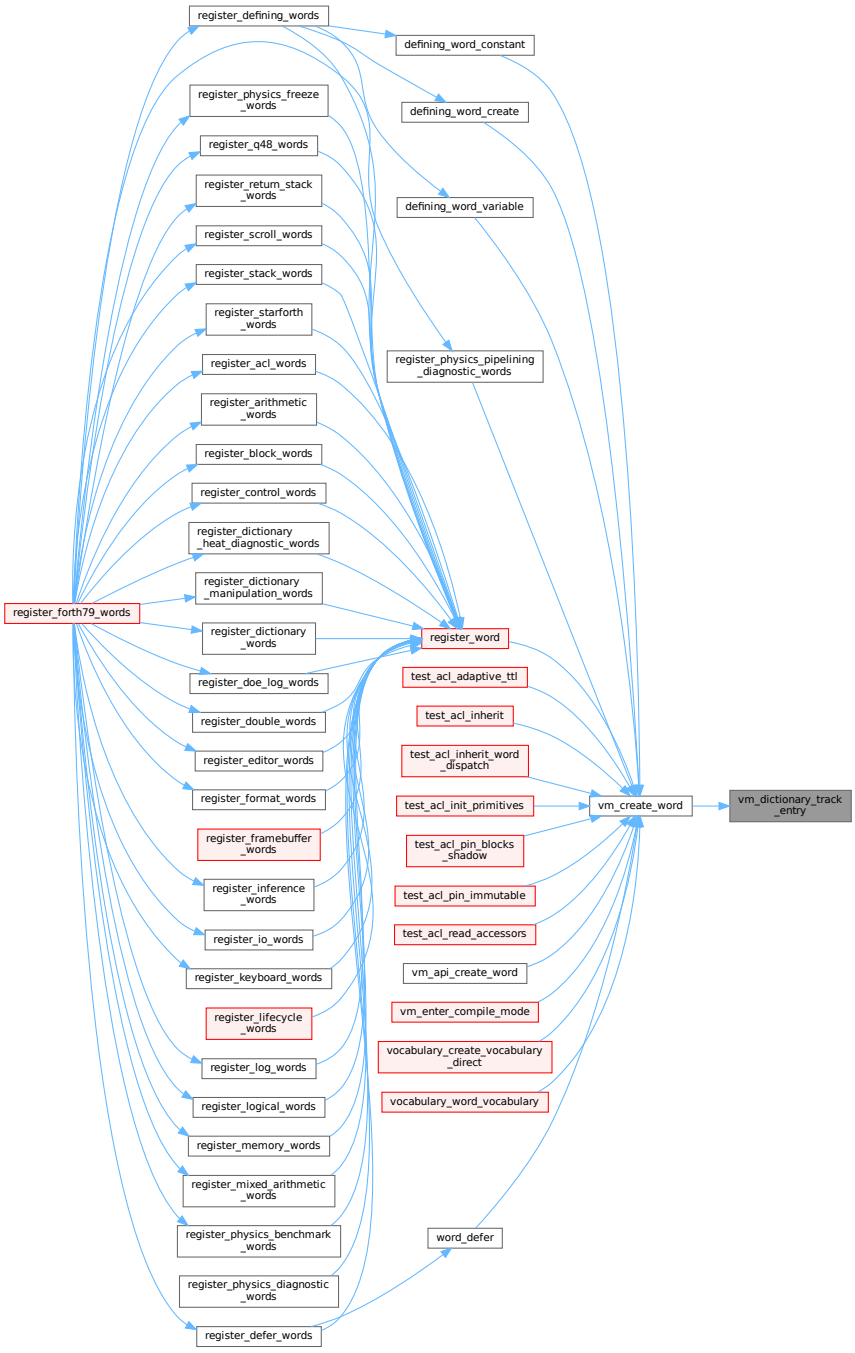
References DICTIONARY_SIZE, entry, log_message(), LOG_WARN, vm_dictionary_acquire_word_id(), word_id, WORD_ID_INVALID, and VM::word_id_map.

Referenced by vm_create_word().

Here is the call graph for this function:



Here is the caller graph for this function:



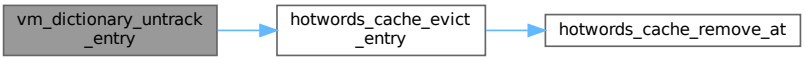
7.281.2.17 vm_dictionary_untrack_entry()

```
void vm_dictionary_untrack_entry (
            VM * vm,
            DictEntry * entry )
```

References DICTIONARY_SIZE, entry, VM::hotwords_cache, hotwords_cache_evict_entry(), VM::recycled_word_id_count, VM::recycled_word_ids, word_id, WORD_ID_INVALID, and VM::word_id_map.

Referenced by dictionary_word_forget().

Here is the call graph for this function:



Here is the caller graph for this function:

7.281.2.18 vm_find_word()

```
DictEntry * vm_find_word (
            VM * vm,
            const char * name,
            size_t len )
```

Finds a word in the dictionary by name.

Searches for a word in the dictionary using an optimized first-character index. The search is performed newest-first within the matching first-character bucket.

Parameters

vm	The virtual machine context
name	The name to search for
len	Length of the name string

Returns

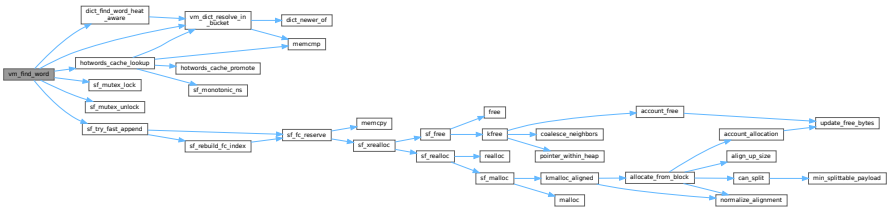
DictEntry* Pointer to found dictionary entry or NULL if not found

References dict_find_word_heat_aware(), VM::dict_lock, HeartbeatState::dictionary_lookups, VM::heartbeat, VM::hotwords_cache, hotwords_cache_lookup(), VM::latest, VM::lookup_strategy, name, sf_cached_latest, sf_fc_count, sf_fc_list, sf_mutex_lock(), sf_mutex_unlock(), sf_try_fast_append(), UNLIKELY, and vm_dict_resolve_in_bucket().

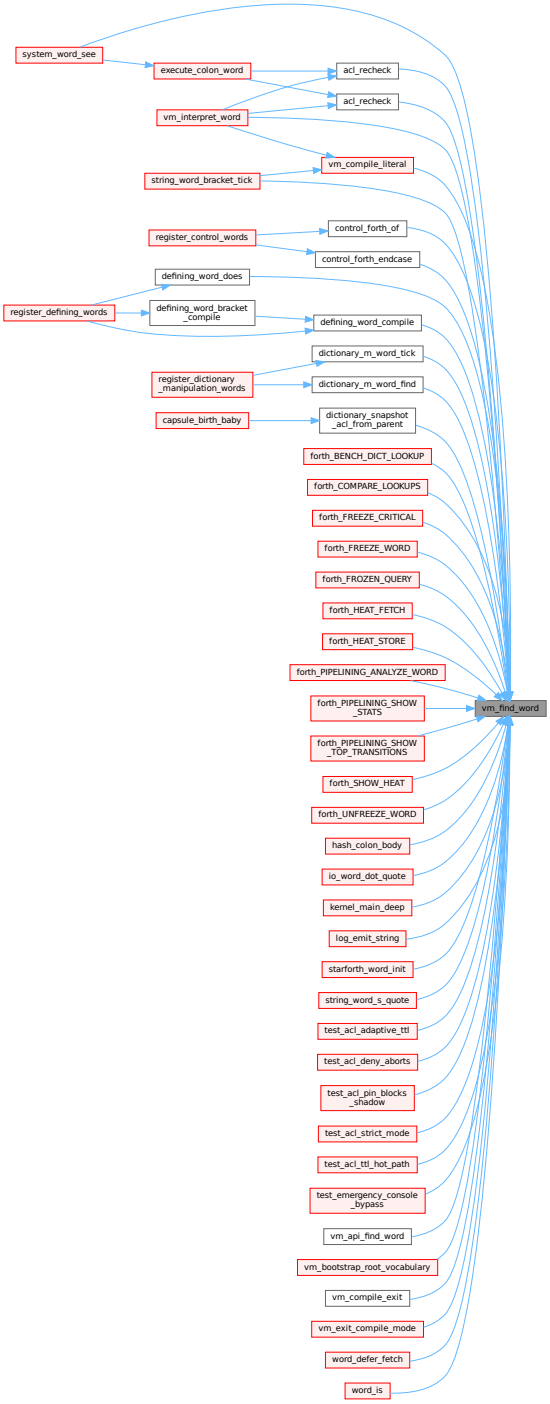
Referenced by acl_recheck(), acl_recheck(), control_forth_endcase(), control_forth_of(), defining_word_compile(), defining_word_does(), dictionary_m_word_find(), dictionary_m_word_tick(), dictionary_snapshot_acl_from_parent(), forth_BENCH_DICT_LOOKUP(), forth_COMPARE_LOOKUPS(), forth_FREEZE_CRITICAL(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_STORE(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), hash_colon_body(), io_word_dot_quote(), kernel_main_deep(), log_emit_string(),

starforth_word_init(), string_word_bracket_tick(), string_word_s_quote(), system_word_see(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_pin_blocks_shadow(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_byp vm_api_find_word(), vm_bootstrap_root_vocabulary(), vm_compile_exit(), vm_compile_literal(), vm_exit_compile_mode(), vm_interpret_word(), word_defer_fetch(), and word_is().

Here is the call graph for this function:



Here is the caller graph for this function:



7.281.2.19 vm_hide_word()

```
void vm_hide_word (
            VM * vm )
```

Marks the latest word as hidden.

Sets the WORD_HIDDEN flag on the most recently defined word.

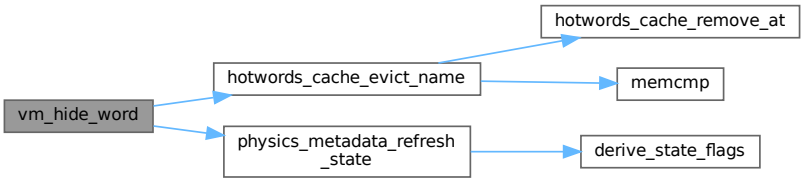
Parameters

vm	The virtual machine context

References DictEntry::flags, VM::hotwords_cache, hotwords_cache_evict_name(), VM::latest, DictEntry::name, DictEntry::name_len, physics_metadata_refresh_state(), and WORD_HIDDEN.

Referenced by vm_api_hide_word().

Here is the call graph for this function:



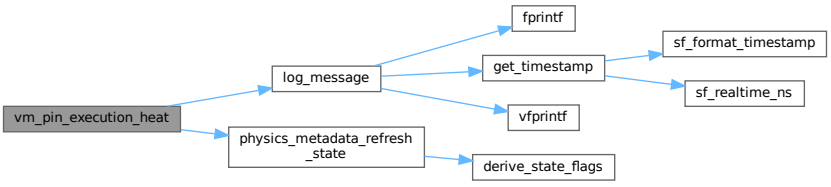
Here is the caller graph for this function:

7.281.2.20 vm_pin_execution_heat()

```
void vm_pin_execution_heat (
            VM * vm )
```

References VM::error, DictEntry::flags, VM::latest, LOG_ERROR, log_message(), physics_metadata_refresh_state(), and WORD_PINNED.

Here is the call graph for this function:



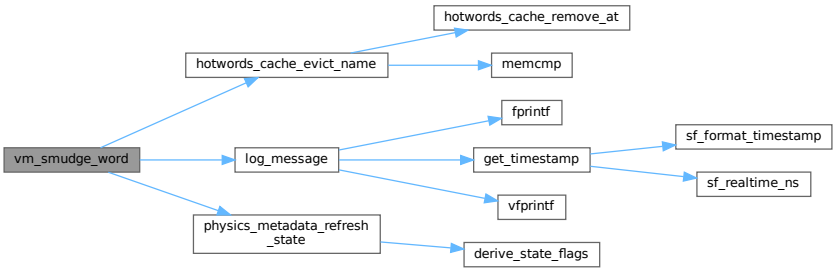
7.281.2.21 vm_smudge_word()

```
void vm_smudge_word (
            VM * vm )
```

References VM::error, DictEntry::flags, VM::hotwords_cache, hotwords_cache_evict_name(), VM::latest, LOG_ERROR, log_message(), DictEntry::name, DictEntry::name_len, physics_metadata_refresh_state(), and WORD_SMUDGED.

Referenced by vm_api_smudge_word().

Here is the call graph for this function:



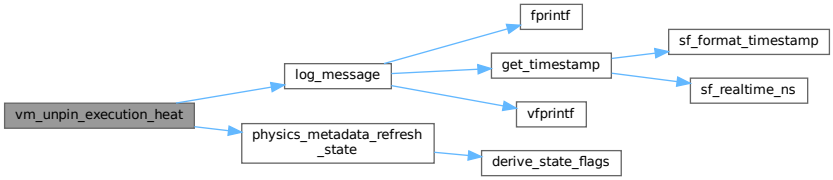
Here is the caller graph for this function:

7.281.2.22 vm_unpin_execution_heat()

```
void vm_unpin_execution_heat (
            VM * vm )
```

References VM::error, DictEntry::flags, VM::latest, LOG_ERROR, log_message(), and physics_metadata_refresh_state().

Here is the call graph for this function:



7.281.3 Variable Documentation

7.281.3.1 sf_cached_latest

```
DictEntry* sf_cached_latest = NULL  [static]
```

Referenced by sf_rebuild_fc_index(), sf_try_fast_append(), vm_dictionary_index_reset(), and vm_find_word().

7.281.3.2 sf_fc_cap

```
size_t sf_fc_cap[256]
```

Referenced by dict_reorganize_buckets_by_heat(), sf_fc_clear_all(), and sf_fc_reserve().

7.281.3.3 sf_fc_count

```
size_t sf_fc_count[256]
```

Referenced by dict_find_word_heat_aware(), dict_reorganize_buckets_by_heat(), sf_fc_clear_all(), sf_fc_reserve(), sf_rebuild_fc_index(), sf_try_fast_append(), and vm_find_word().

7.281.3.4 sf_fc_list

```
DictEntry** sf_fc_list[256]
```

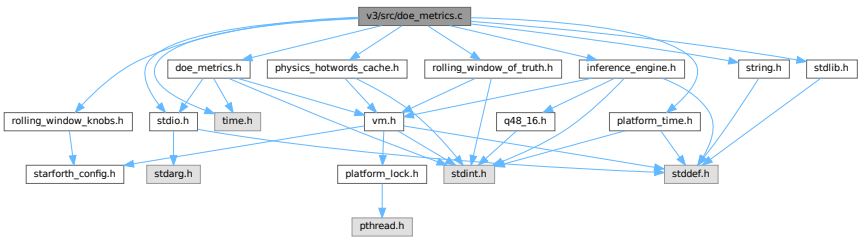
Referenced by dict_find_word_heat_aware(), dict_reorganize_buckets_by_heat(), sf_fc_clear_all(), sf_fc_reserve(), sf_rebuild_fc_index(), sf_try_fast_append(), and vm_find_word().

7.282 v3/src/doe_metrics.c File Reference

Design of Experiments metrics collection implementation.

```
#include "doe_metrics.h"
#include "physics_hotwords_cache.h"
#include "rolling_window_of_truth.h"
#include "rolling_window_knobs.h"
#include "inference_engine.h"
#include "platform_time.h"
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for doe_metrics.c:



Functions

- int32_t metrics_get_cpu_temp_c (void)

  Get current CPU temperature in Celsius.
- int32_t metrics_get_cpu_freq_mhz (void)

  Get current CPU frequency in MHz.
- void metrics_get_timestamp (char *buf, size_t bufsize)

  Get current timestamp as ISO 8601 string.
- static void extract_cache_metrics (const HotwordsCache *cache, DoeMetrics *metrics)

  Extract metrics from VM hotwords cache stats.
- DoeMetrics metrics_from_vm (VM *vm, uint64_t workload_duration_ns, int32_t cpu_temp_delta_c, int32_t cpu_freq_delta_mhz)

  Extract metrics from entire VM.
- void metrics_write_csv_header (FILE *out)

  Write CSV header.
- void metrics_write_csv_row (FILE *out, const DoeMetrics *metrics)

  Write CSV row - MUST match header column order exactly.
- void metrics_print_text (FILE *out, const DoeMetrics *metrics)

  Print metrics as human-readable text.
- static uint64_t isqrt (uint64_t n)

  Integer square root using Newton's method (pure C99, no math.h)
- void metrics_write_james_law_csv_row (FILE *out, const DoeMetrics *metrics, VM *vm)

  Write reduced James Law CSV row (20 columns for window scaling experiment)

Variables

- struct {
    uint64_t total_lookups
    uint64_t cache_hits
    uint64_t bucket_hits
  } physics_global_stats

7.282.1 Detailed Description

Design of Experiments metrics collection implementation.

7.282.2 Function Documentation

7.282.2.1 extract_cache_metrics()

```
static void extract_cache_metrics (
            const HotwordsCache * cache,
            DoeMetrics * metrics )  [static]
```

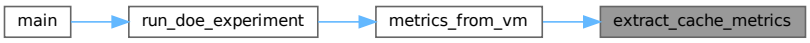
Extract metrics from VM hotwords cache stats.

References DoeMetrics::bucket_hit_percent, DoeMetrics::bucket_hits, HotwordsStats::bucket_hits, DoeMetrics::bucket_search_laten
HotwordsStats::bucket_search_samples, DoeMetrics::bucket_search_stddev_ns, HotwordsStats::bucket_search_total_ns_q48,
HotwordsStats::bucket_search_variance_sum_q48, DoeMetrics::cache_hit_latency_ns, DoeMetrics::cache_hit_percent,

HotwordsStats::cache_hit_samples, DoeMetrics::cache_hit_stddev_ns, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, DoeMetrics::cache_hits, HotwordsStats::cache_hits, HotwordsCache_s::stats, and HotwordsStats::total_lookups.

Referenced by metrics_from_vm().

Here is the caller graph for this function:



7.282.2.2 isqrt()

```
static uint64_t isqrt (
            uint64_t n )  [static]
```

Integer square root using Newton's method (pure C99, no math.h)

Referenced by metrics_write_james_law_csv_row().

Here is the caller graph for this function:

7.282.2.3 metrics_from_vm()

```
DoeMetrics metrics_from_vm (
            VM * vm,
            uint64_t workload_duration_ns,
            int32_t cpu_temp_delta_c,
            int32_t cpu_freq_delta_mhz )
```

Extract metrics from entire VM.

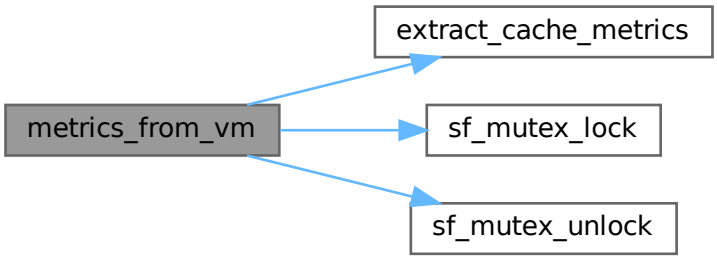
Extract complete metrics snapshot from VM.

References DoeMetrics::actual_window_size, DoeMetrics::adaptive_shrink_rate, DoeMetrics::avg_word_heat, DoeMetrics::cache_demotions, DoeMetrics::cache_promotions, DoeMetrics::ci_lower_95, DoeMetrics::ci_upper_95, DoeMetrics::context_accuracy_percent, DoeMetrics::context_correct, DoeMetrics::context_predictions_total, DoeMetrics::cpu_freq_delta_mhz_q48, DoeMetrics::cpu_temp_delta_c_q48, DECAY_MIN_INTERVAL, DoeMetrics::decay_min_inter

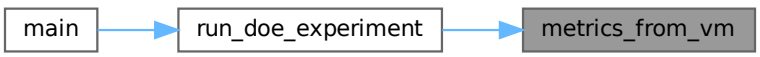
DECAY_RATE_PER_US_Q16, DoeMetrics::decay_rate_q16, DoeMetrics::decay_slope, VM::decay_slope_q48, VM::dict_lock, DoeMetrics::dictionary_lookups, HeartbeatState::dictionary_lookups, DoeMetrics::early_exit_count, HeartbeatState::early_exit_count, RollingWindowOfTruth::effective_window_size, DoeMetrics::enable_hotwords_cache, ENABLE_HOTWORDS_CACHE, DoeMetrics::enable_loop_1_heat_tracking, DoeMetrics::enable_loop_2_rolling_window, DoeMetrics::enable_loop_3_linear_decay, DoeMetrics::enable_loop_4_pipelining, DoeMetrics::enable_loop_5_window_inference, DoeMetrics::enable_loop_6_decay_inference, DoeMetrics::enable_loop_7_adaptive_heartrate, DoeMetrics::enable_pipelining, ENABLE_PIPELINING, HotwordsCache_s::enabled, HotwordsStats::evictions, extract_cache_metrics(), DoeMetrics::final_effective_v VM::heartbeat, DoeMetrics::heat_cache_demotion_threshold, DoeMetrics::hot_word_count, VM::hotwords_cache, HOTWORDS_EXECUTION_HEAT_THRESHOLD, DoeMetrics::inference_run_count, HeartbeatState::inference_run_count, VM::last_inference_outputs, VM::latest, DoeMetrics::memory_allocated_bytes, VM::pipeline_metrics, DoeMetrics::prefetch_accuracy DoeMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_attempts, DoeMetrics::prefetch_hits, PipelineGlobalMetrics::prefetch HotwordsStats::promotions, VM::rolling_window, DoeMetrics::rolling_window_size, ROLLING_WINDOW_SIZE, DoeMetrics::rolling_window_width, sf_mutex_lock(), sf_mutex_unlock(), DoeMetrics::speedup_vs_baseline, DoeMetrics::stale_word_count, DoeMetrics::stale_word_ratio, HotwordsCache_s::stats, DoeMetrics::tick_count, HeartbeatState::tick_count, DoeMetrics::tick_target_ns, HeartbeatState::tick_target_ns, DoeMetrics::total_executions, RollingWindowOfTruth::total_executions, DoeMetrics::total_heat, DoeMetrics::total_lookups, HotwordsStats::total_lookups, DoeMetrics::total_runtime_ms, VM::tuning_lock, DoeMetrics::vm_workload_duration_ns_q48, DoeMetrics::window_diversity_percent DoeMetrics::window_final_size_bytes, DoeMetrics::window_tuning_checks, PipelineGlobalMetrics::window_tuning_checks, DoeMetrics::window_variance_q48, InferenceOutputs::window_variance_q48, word_count, DoeMetrics::words_executed, and HeartbeatState::words_executed.

Referenced by run_doe_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:



7.282.2.4 metrics_get_cpu_freq_mhz()

```
int32_t metrics_get_cpu_freq_mhz (
            void  )
```

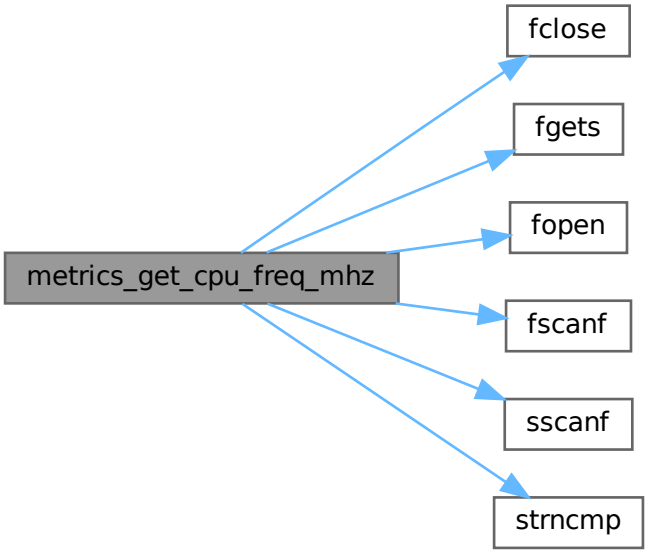
Get current CPU frequency in MHz.

Get current CPU frequency (MHz)

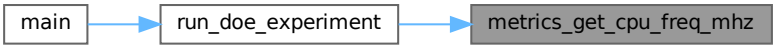
References fclose(), fgets(), fopen(), fscanf(), sscanf(), and strncmp().

Referenced by run_doe_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:



7.282.2.5 metrics_get_cpu_temp_c()

```
int32_t metrics_get_cpu_temp_c (
            void  )
```

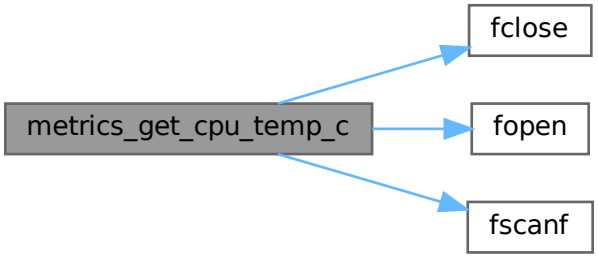
Get current CPU temperature in Celsius.

Get current CPU temperature (Celsius)

References fclose(), fopen(), and fscanf().

Referenced by run_doe_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:

7.282.2.6 metrics_get_timestamp()

```
void metrics_get_timestamp (
            char * buf,
            size_t bufsize )
```

Get current timestamp as ISO 8601 string.

Format: "YYYY-MM-DDTHH:MM:SS"

Parameters

buf	Buffer to write timestamp (minimum 32 bytes)
bufsize	Size of buffer

7.282.2.7 metrics_print_text()

```
void metrics_print_text (
```

```
            FILE * out,
            const DoeMetrics * metrics )
```

Print metrics as human-readable text.

Outputs metrics in formatted text for debugging/verification.

Parameters

out	Output file stream
metrics	Metrics structure to print

References DoeMetrics::bucket_hit_percent, DoeMetrics::bucket_hits, DoeMetrics::bucket_search_latency_ns, DoeMetrics::bucket_search_stddev_ns, DoeMetrics::cache_hit_latency_ns, DoeMetrics::cache_hit_percent, DoeMetrics::cache_hit_stddev_ns, DoeMetrics::cache_hits, DoeMetrics::context_accuracy_percent, DoeMetrics::context_correct, DoeMetrics::context_predictions_total, DoeMetrics::cpu_freq_delta_mhz_q48, DoeMetrics::cpu_temp_delta_c_q48, DoeMetrics::decay_slope, fprintf(), DoeMetrics::rolling_window_width, DoeMetrics::total_lookups, and DoeMetrics::vm_workload_dur

Here is the call graph for this function:

7.282.2.8 metrics_write_csv_header()

```
void metrics_write_csv_header (
            FILE * out )
```

Write CSV header.

Write CSV header row.

References fprintf().

Here is the call graph for this function:

7.282.2.9 metrics_write_csv_row()

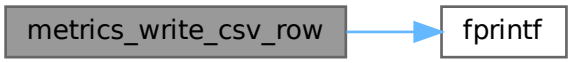
```
void metrics_write_csv_row (
            FILE * out,
            const DoeMetrics * metrics )
```

Write CSV row - MUST match header column order exactly.

Write metrics as CSV row.

References DoeMetrics::actual_window_size, DoeMetrics::adaptive_shrink_rate, DoeMetrics::avg_word_heat, DoeMetrics::bucket_hit_percent, DoeMetrics::bucket_hits, DoeMetrics::bucket_search_latency_ns, DoeMetrics::bucket_search_stdde DoeMetrics::cache_demotions, DoeMetrics::cache_hit_latency_ns, DoeMetrics::cache_hit_percent, DoeMetrics::cache_hit_stddev_n DoeMetrics::cache_hits, DoeMetrics::cache_promotions, DoeMetrics::ci_lower_95, DoeMetrics::ci_upper_95, DoeMetrics::context_accuracy_percent, DoeMetrics::context_correct, DoeMetrics::context_predictions_total, DoeMetrics::cpu_freq_delta_mhz_q48, DoeMetrics::cpu_temp_delta_c_q48, DoeMetrics::decay_min_interval_ns, DoeMetrics::decay_rate_q16, DoeMetrics::decay_slope, DoeMetrics::dictionary_lookups, DoeMetrics::early_exit_count, DoeMetrics::enable_hotwords_cache, DoeMetrics::enable_loop_1_heat_tracking, DoeMetrics::enable_loop_2_rolling_window, DoeMetrics::enable_loop_3_linear_decay, DoeMetrics::enable_loop_4_pipelining, DoeMetrics::enable_loop_5_window_inference, DoeMetrics::enable_loop_6_decay_inference, DoeMetrics::enable_loop_7_adaptive_heartrate, DoeMetrics::enable_pipelining, DoeMetrics::final_effective_window_size, fprintf(), DoeMetrics::heat_cache_demotion_threshold, DoeMetrics::hot_word_count, DoeMetrics::inference_run_count, DoeMetrics::memory_allocated_bytes, DoeMetrics::prefetch_accuracy_percent, DoeMetrics::prefetch_attempts, DoeMetrics::prefetch_hits, DoeMetrics::rolling_window_size, DoeMetrics::rolling_window_width, DoeMetrics::speedup_vs_baseline, DoeMetrics::stale_word_count, DoeMetrics::stale_word_ratio, DoeMetrics::tick_count, DoeMetrics::tick_target_ns, DoeMetrics::total_executions, DoeMetrics::total_heat, DoeMetrics::total_lookups, DoeMetrics::total_runtime_ms, DoeMetrics::vm_workload_duration_ns_q48, DoeMetrics::window_diversity_percent, DoeMetrics::window_final_size_bytes, DoeMetrics::window_tuning_checks, DoeMetrics::window_variance_q48, and DoeMetrics::words_executed.

Here is the call graph for this function:



7.282.2.10 metrics_write_james_law_csv_row()

```
void metrics_write_james_law_csv_row (
            FILE * out,
            const DoeMetrics * metrics,
            VM * vm )
```

Write reduced James Law CSV row (20 columns for window scaling experiment)

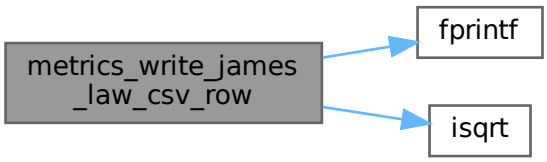
Write reduced James Law CSV row (20 columns)

This outputs only the subset of metrics needed for the James Law validation experiment. Format matches the header in run_window_sweep.sh (minus the 5 script-prepended columns).

Expected columns (20): workload_duration_ns, l8_mode_final, lambda_effective, K_statistic, K_deviation, entropy, cv, temporal_decay, stability_score, ns_per_word, total_runtime_ms, window_final, hot_word_count, avg_heat, bucket_mean_K, bucket_cv_K, bucket_window_var, bucket_heat_var, bucket_mode_transitions, bucket_collapse↩ _flag

References DoeMetrics::avg_word_heat, HeartbeatState::bucket_collapse_flag, HeartbeatState::bucket_mode_transitions, HeartbeatState::bucket_sum_heat_variance, HeartbeatState::bucket_sum_K, HeartbeatState::bucket_sum_K_squared, HeartbeatState::bucket_sum_window_variance, HeartbeatState::bucket_tick_count, DoeMetrics::cache_hit_latency_ns, DoeMetrics::cache_hit_stddev_ns, DoeMetrics::decay_slope, DoeMetrics::final_effective_window_size, fprintf(), VM::heartbeat, HEARTBEAT_TICK_BUFFER_SIZE, DoeMetrics::hot_word_count, isqrt(), HeartbeatTickSnapshot::l8_mode, DoeMetrics::rolling_window_size, DoeMetrics::rolling_window_width, DoeMetrics::stale_word_ratio, HeartbeatState::tick_buffer, HeartbeatState::tick_buffer_write_index, DoeMetrics::total_runtime_ms, DoeMetrics::vm_workload_duration_ns_q48, DoeMetrics::window_diversity_percent, and DoeMetrics::words_executed.

Here is the call graph for this function:



7.282.3 Variable Documentation

7.282.3.1 bucket_hits

```
uint64_t bucket_hits
```

7.282.3.2 cache_hits

```
uint64_t cache_hits
```

7.282.3.3 [struct]

```
struct { ... } physics_global_stats
```

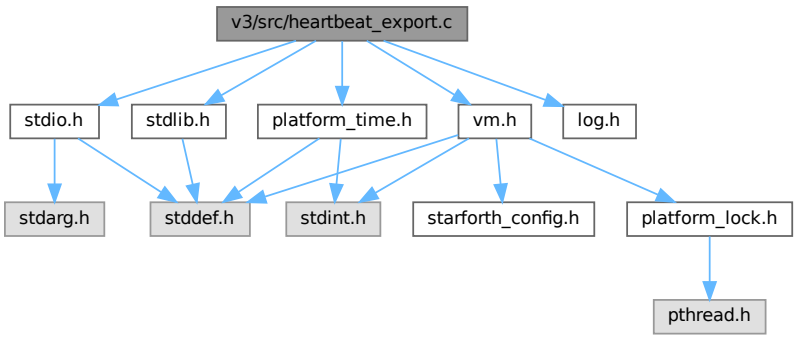
7.282.3.4 total_lookups

```
uint64_t total_lookups
```

7.283 v3/src/heartbeat_export.c File Reference

```
#include "vm.h"
#include "log.h"
#include "platform_time.h"
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for heartbeat_export.c:



Functions

- void heartbeat_capture_tick_snapshot (VM *vm, HeartbeatTickSnapshot *snapshot)

  Capture current VM metrics into a heartbeat tick snapshot.
- void heartbeat_emit_tick_row (VM *vm, HeartbeatTickSnapshot *snapshot)

  Emit a heartbeat tick snapshot as CSV row to stderr.

7.283.1 Function Documentation

7.283.1.1 heartbeat_capture_tick_snapshot()

```
void heartbeat_capture_tick_snapshot (
            VM * vm,
            HeartbeatTickSnapshot * snapshot )
```

Capture current VM metrics into a heartbeat tick snapshot.

Collects 11 metrics per tick for multivariate dynamics analysis. Called every heartbeat tick to record VM state.

Parameters

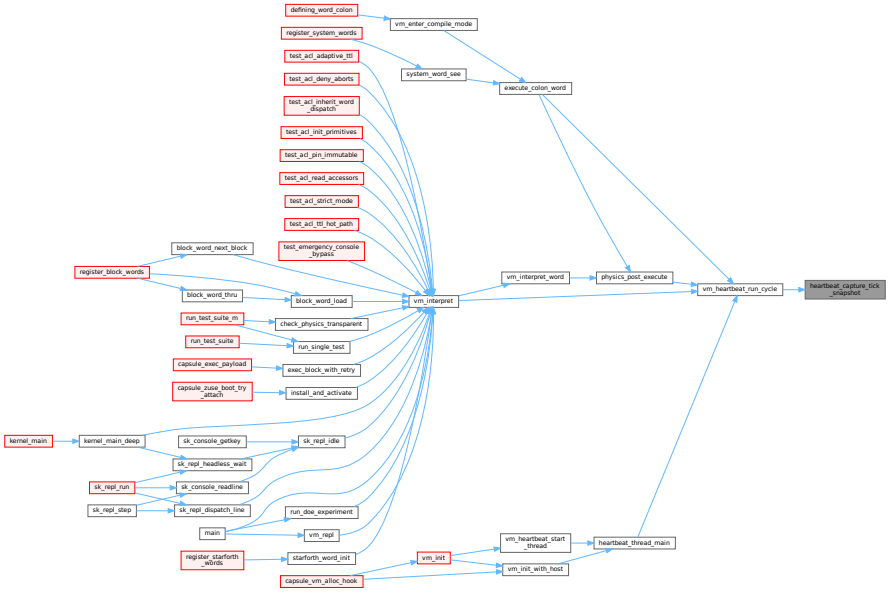
vm	Pointer to the VM instance
snapshot	Pointer to snapshot structure to populate

References HeartbeatTickSnapshot::actual_window_size, HeartbeatTickSnapshot::avg_word_heat, HeartbeatTickSnapshot::bucket_
HeartbeatTickSnapshot::cache_hits_delta, HeartbeatState::early_exit_count, RollingWindowOfTruth::effective_window_size,

HeartbeatTickSnapshot::elapsed_ns, entry, HeartbeatTickSnapshot::estimated_jitter_ns, VM::heartbeat, HEARTBEAT_TICK_NS, HeartbeatTickSnapshot::hot_word_count, VM::latest, HeartbeatTickSnapshot::predicted_label_hits, VM::rolling_window, HeartbeatState::tick_count, HeartbeatTickSnapshot::tick_interval_ns, HeartbeatTickSnapshot::tick_number, RollingWindowOfTruth::total_executions, HeartbeatTickSnapshot::window_width, word_count, HeartbeatTickSnapshot::word_executi and HeartbeatState::words_executed.

Referenced by vm_heartbeat_run_cycle().

Here is the caller graph for this function:



7.283.1.2 heartbeat_emit_tick_row()

```
void heartbeat_emit_tick_row (
            VM * vm,
            HeartbeatTickSnapshot * snapshot )
```

Emit a heartbeat tick snapshot as CSV row to stderr.

Outputs 11 comma-separated values (no header) for real-time streaming. Called every heartbeat tick to emit metrics immediately.

Format: tick_number,elapsed_ns,tick_interval_ns,cache_hits_delta, bucket_hits_delta,word_executions_↩ delta,hot_word_count, avg_word_heat,window_width,actual_window_size,predicted_label_hits,estimated_jitter_ns

Parameters

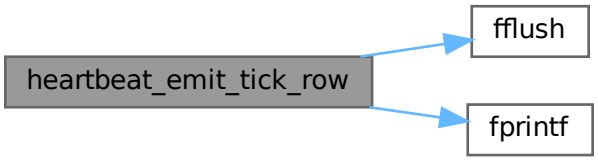
vm	Pointer to the VM instance (unused, for API consistency)
snapshot	Pointer to the snapshot to emit

References HeartbeatTickSnapshot::actual_window_size, HeartbeatTickSnapshot::avg_word_heat, HeartbeatTickSnapshot::bucket_ HeartbeatTickSnapshot::cache_hits_delta, HeartbeatTickSnapshot::elapsed_ns, HeartbeatTickSnapshot::estimated_jitter_ns, fflush(), fprintf(), HeartbeatTickSnapshot::hot_word_count, HeartbeatTickSnapshot::predicted_label_hits, stderr,

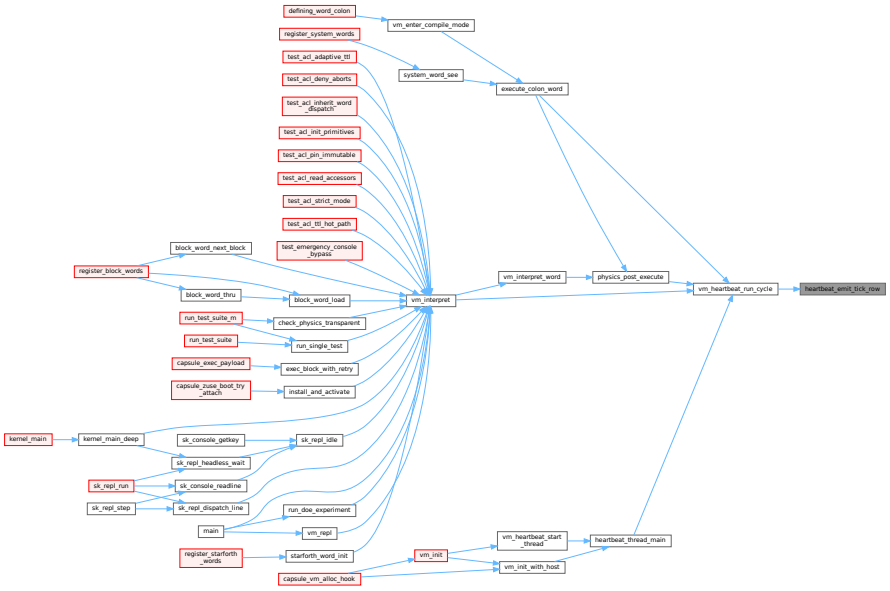
HeartbeatTickSnapshot::tick_interval_ns, HeartbeatTickSnapshot::tick_number, HeartbeatTickSnapshot::window_width, and HeartbeatTickSnapshot::word_executions_delta.

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:

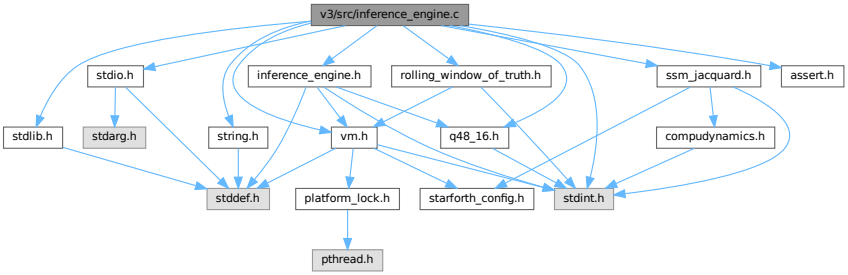


7.284 v3/src/inference_engine.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <stdint.h>
#include <assert.h>
#include "inference_engine.h"
#include "q48_16.h"
#include "vm.h"
#include "rolling_window_of_truth.h"
```

```
#include "ssm_jacquard.h"
```

Include dependency graph for inference_engine.c:



Functions

- static int has_variance_stabilized (q48_16_t current_variance, q48_16_t last_variance)

  ANOVA early-exit check: test whether variance has stabilised.
- static uint64_t * extract_heat_trajectory (RollingWindowOfTruth *window, VM *vm, uint64_t *out_length)

  Build a heat trajectory array from the rolling window of truth.
- q48_16_t compute_variance_q48 (const uint64_t *heat_data, uint64_t length)

  Compute the variance of a heat trajectory in Q48.16 fixed-point.
- static q48_16_t compute_median_q48 (const q48_16_t *data, uint32_t length)

  Compute the median of a Q48.16 array.
- static q48_16_t compute_mean_q48 (const q48_16_t *data, uint32_t length)

  Compute the arithmetic mean of a Q48.16 array.
- static q48_16_t compute_levene_statistic (const q48_16_t *chunk_variances, uint32_t num_chunks, uint32_t chunk_size)

  Compute Levene's W statistic for equality of variance.
- uint32_t find_variance_inflection (const uint64_t *heat_data, uint64_t trajectory_length, q48_16_t full_variance)

  Find the minimum window size where heat variance is statistically stable.
- uint64_t infer_decay_slope_q48 (const uint64_t *heat_data, uint64_t length)

  Infer the decay slope from a heat trajectory via log-linear regression.
- static uint64_t compute_fit_quality (const uint64_t *heat_data, uint64_t length, uint64_t slope_q48)

  Compute R^2-style fit quality for the inferred decay slope.
- void inference_engine_run (InferenceInputs *inputs, InferenceOutputs *outputs)

  Run adaptive inference engine on window data.
- const char * inference_outputs_to_string (const InferenceOutputs *outputs)

  Format inference outputs as a human-readable string.
- int inference_outputs_validate (const InferenceOutputs *outputs)

  Validate inference outputs for sanity.

7.284.1 Function Documentation

7.284.1.1 compute_fit_quality()

```
static uint64_t compute_fit_quality (
            const uint64_t * heat_data,
```

```
    uint64_t length,
    uint64_t slope_q48 )  [static]
```

Compute R^2-style fit quality for the inferred decay slope.

Currently returns a fixed placeholder of 0.8 in Q48.16 (52429). A full residual-sum-of-squares implementation is planned but deferred — the placeholder is sufficient for the DoE convergence signal.

Parameters

heat_data	Array of heat values (unused in current implementation)
length	Array length (unused in current implementation)
slope_q48	Inferred slope in Q48.16 (unused in current implementation)

Returns

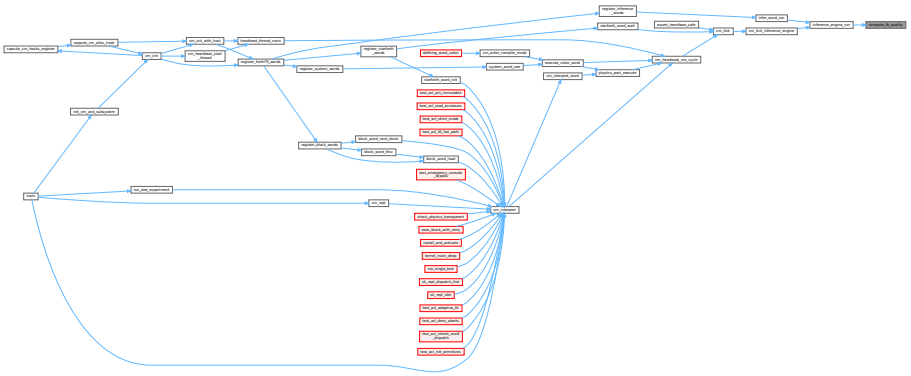
Fit quality in Q48.16; currently always 0.8 (52429)

References q48_from_u64().

Referenced by inference_engine_run().

Here is the call graph for this function:

Here is the caller graph for this function:



7.284.1.2 compute_levene_statistic()

```
static q48_16_t compute_levene_statistic (
            const q48_16_t * chunk_variances,
            uint32_t num_chunks,
            uint32_t chunk_size )  [static]
```

Compute Levene's W statistic for equality of variance.

Tests the null hypothesis H?: all chunk variances are equal. Uses the median-based variant for robustness: z_i = |var_i - median_var|, then W = (K-1)∗N∗Σ(z_i - z_bar)^2 / (K∗N ∗ var(z)). Returns 0 for fewer than 2 chunks. Compare result to LEVENE_CRITICAL_VALUE_Q48 (≈ 6.5 at α=0.05): W $>$ critical means variances differ significantly.

Parameters

chunk_variances	Array of K per-chunk variance values (Q48.16)
num_chunks	Number of chunks K (must be >= 2)
chunk_size	Number of samples N per chunk

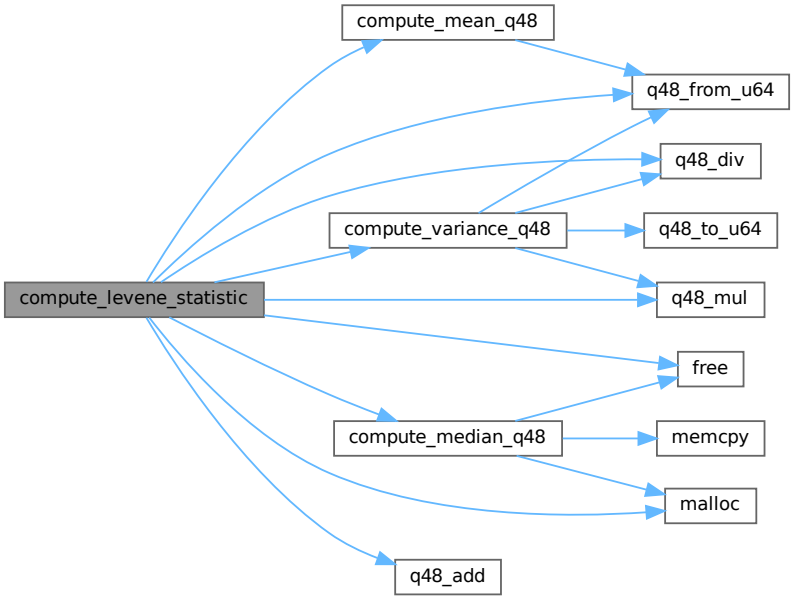
Returns

Levene's W statistic in Q48.16 format

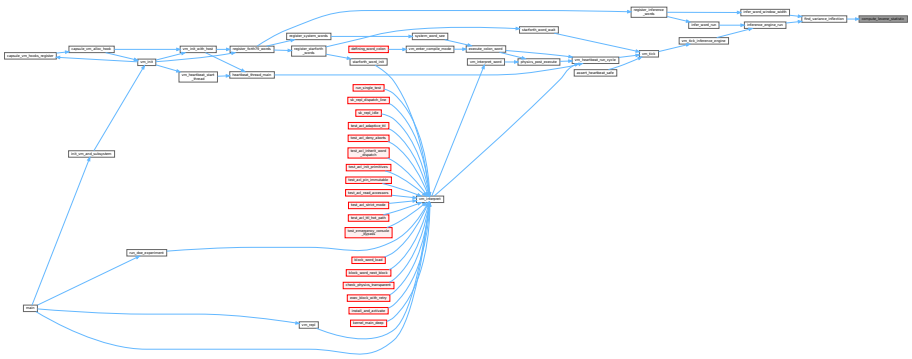
References compute_mean_q48(), compute_median_q48(), compute_variance_q48(), free(), malloc(), q48_add(), q48_div(), q48_from_u64(), and q48_mul().

Referenced by find_variance_inflection().

Here is the call graph for this function:



Here is the caller graph for this function:



7.284.1.3 compute_mean_q48()

```
static q48_16_t compute_mean_q48 (
            const q48_16_t * data,
            uint32_t length )  [static]
```

Compute the arithmetic mean of a Q48.16 array.

Sums the integer parts of each element (>> 16) and returns the result converted back to Q48.16. Returns 0 for an empty array.

Parameters

data	Array of Q48.16 values
length	Number of elements

Returns

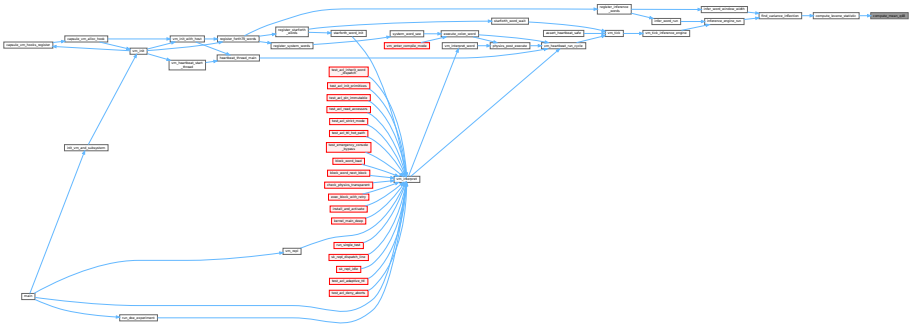
Arithmetic mean in Q48.16 format

References q48_from_u64().

Referenced by compute_levene_statistic().

Here is the call graph for this function:

Here is the caller graph for this function:



7.284.1.4 compute_median_q48()

```
static q48_16_t compute_median_q48 (
            const q48_16_t * data,
            uint32_t length )  [static]
```

Compute the median of a Q48.16 array.

Makes a temporary copy, sorts it with bubble sort (acceptable for the small chunk counts used in Levene's test, typically K <= 16), and returns the middle element. Returns 0 for an empty array.

Parameters

data	Array of Q48.16 values
length	Number of elements

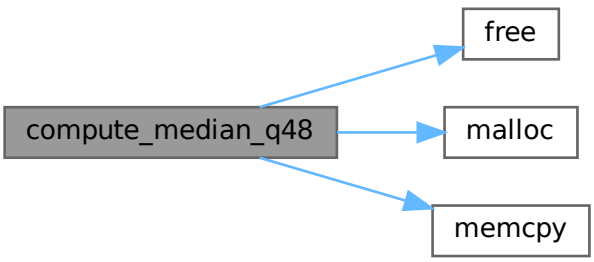
Returns

Median value in Q48.16 format

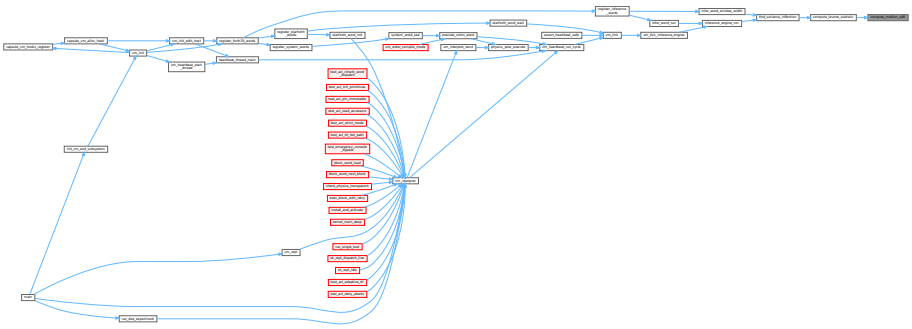
References free(), malloc(), and memcpy().

Referenced by compute_levene_statistic().

Here is the call graph for this function:



Here is the caller graph for this function:



7.284.1.5 compute_variance_q48()

```
q48_16_t compute_variance_q48 (
            const uint64_t * heat_data,
            uint64_t length )
```

Compute the variance of a heat trajectory in Q48.16 fixed-point.

Computes the population variance: Σ(heat[i] - mean)^2 / N. All intermediate calculations use Q48.16 arithmetic. Returns 0 for an empty array.

Parameters

heat_data	Array of execution heat values (raw uint64_t counts)
length	Number of elements in `heat_data`

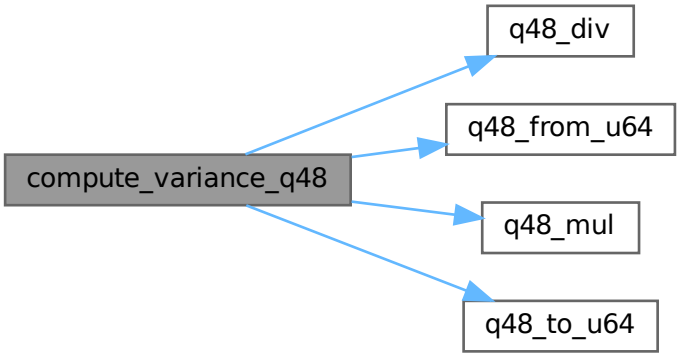
Returns

Population variance in Q48.16 format

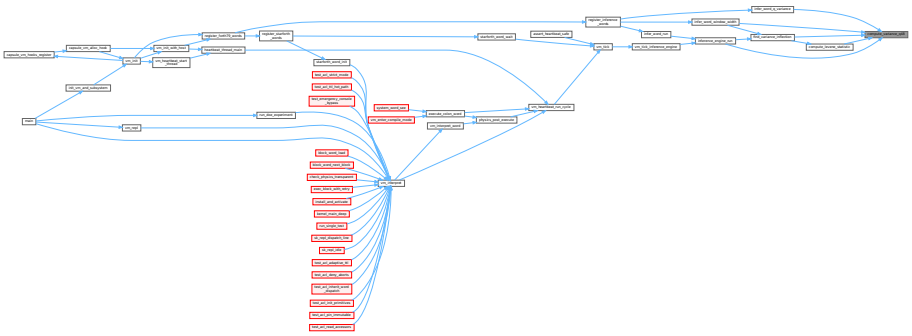
References q48_div(), q48_from_u64(), q48_mul(), and q48_to_u64().

Referenced by compute_levene_statistic(), find_variance_inflection(), infer_word_q_variance(), infer_word_window_width(), and inference_engine_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.284.1.6 extract_heat_trajectory()

```
static uint64_t * extract_heat_trajectory (
            RollingWindowOfTruth * window,
            VM * vm,
            uint64_t * out_length )  [static]
```

Build a heat trajectory array from the rolling window of truth.

Linearises the rolling window via `rolling_window_export_execution_history()`, then translates each word-id slot to its current `execution_heat` by looking up the entry under `vm->dict_lock`. The result is a freshly allocated array of `*out_length` uint64_t values in chronological order. The caller is responsible for freeing the returned pointer. Returns NULL (and sets `*out_length` = 0) on allocation failure or empty window.

Parameters

window	Rolling window to export from
vm	VM whose dictionary is consulted for heat values
out_length	Output: number of elements in the returned array

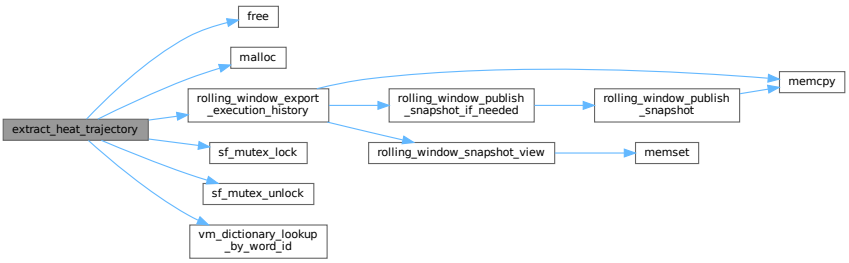
Returns

Heap-allocated trajectory array, or NULL on failure

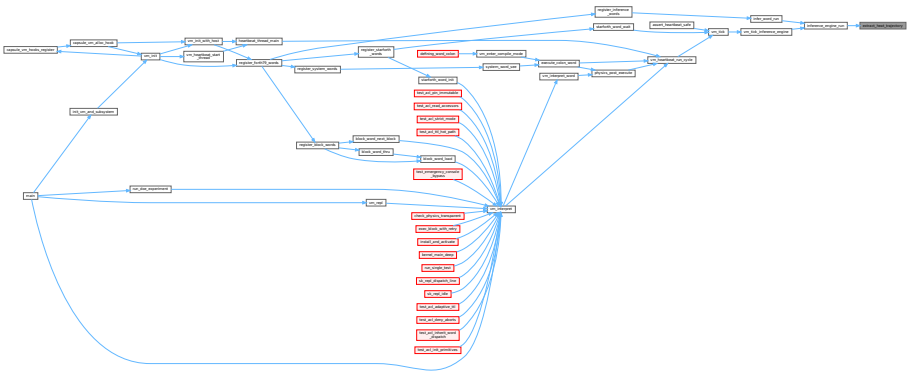
References VM::dict_lock, DICTIONARY_SIZE, RollingWindowOfTruth::effective_window_size, entry, free(), RollingWindowOfTruth::is_warm, malloc(), rolling_window_export_execution_history(), ROLLING_WINDOW_SIZE, sf_mutex_lock(), sf_mutex_unlock(), vm_dictionary_lookup_by_word_id(), and word_id.

Referenced by inference_engine_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.284.1.7 find_variance_inflection()

```
uint32_t find_variance_inflection (
            const uint64_t * heat_data,
            uint64_t trajectory_length,
            q48_16_t full_variance )
```

Find the minimum window size where heat variance is statistically stable.

Implements Loop #5 (window width inference). Scans candidate window sizes from ADAPTIVE_MIN_WINDOW↩ _SIZE to the full trajectory in steps of 64, divides the trajectory into disjoint chunks of that size, and applies Levene's test. Returns the first size where W <= 6.5 (α=0.05 critical value). Falls back to the full trajectory length if no size passes. full_variance is unused (retained for API compatibility with callers that pre-compute it).

Parameters

heat_data	Heat trajectory array
trajectory_length	Number of elements in `heat_data`
full_variance	Unused; retained for API compatibility

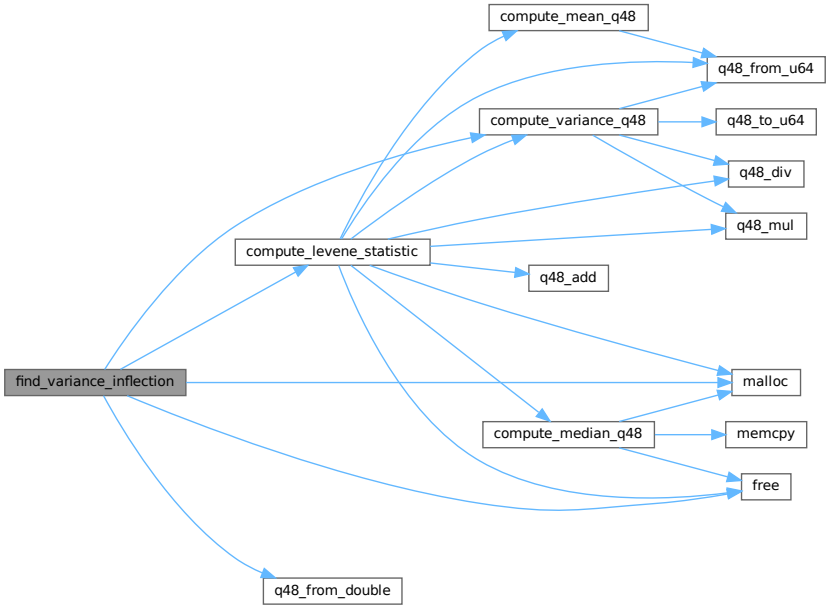
Returns

Minimum sufficient window width in [ADAPTIVE_MIN_WINDOW_SIZE, ROLLING_WINDOW_SIZE]

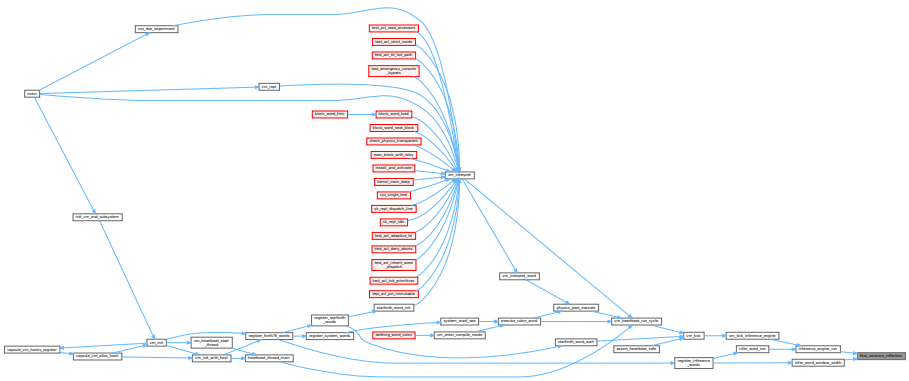
References ADAPTIVE_MIN_WINDOW_SIZE, compute_levene_statistic(), compute_variance_q48(), free(), malloc(), q48_from_double(), and ROLLING_WINDOW_SIZE.

Referenced by infer_word_window_width(), and inference_engine_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.284.1.8 has_variance_stabilized()

```
static int has_variance_stabilized (
            q48_16_t current_variance,
            q48_16_t last_variance )  [static]
```

ANOVA early-exit check: test whether variance has stabilised.

Computes |current - last| / last. If the relative change is <= 5% (threshold = 3276 in Q48.16), the variance is considered stable and full inference can be skipped (~100 cycles vs ~5–10k for a full run). Always returns 0 (unstable) on the first call when `last_variance` == 0.

Parameters

current_variance	Variance from the current snapshot (Q48.16)
last_variance	Variance from the previous inference run (Q48.16)

Returns

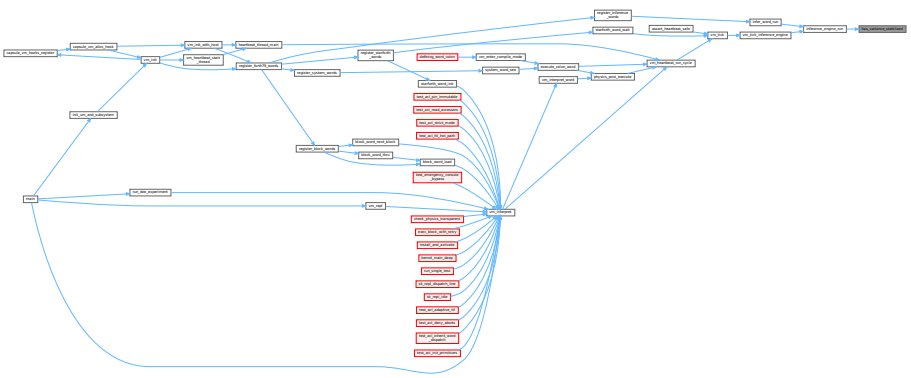
1 if variance is stable (skip inference), 0 if full run needed

References q48_div().

Referenced by inference_engine_run().

Here is the call graph for this function:

Here is the caller graph for this function:



7.284.1.9 infer_decay_slope_q48()

```
uint64_t infer_decay_slope_q48 (
            const uint64_t * heat_data,
            uint64_t length )
```

Infer the decay slope from a heat trajectory via log-linear regression.

Implements Loop #6 (decay slope inference). Fits the model ln(heat[t]) = ln(h?) - slope∗t using the closed-form OLS solution: slope = (n∗Σ(t·ln(h)) - Σt·Σln(h)) / (n·Σt^2 - $(\Sigma t)^2$). The sign of the numerator is taken as absolute value since decay rate is always positive. Zero heat values are skipped. Returns 0 if fewer than 2 samples are available.

Parameters

heat_data	Array of execution heat values (raw counts)
length	Number of elements in `heat_data`

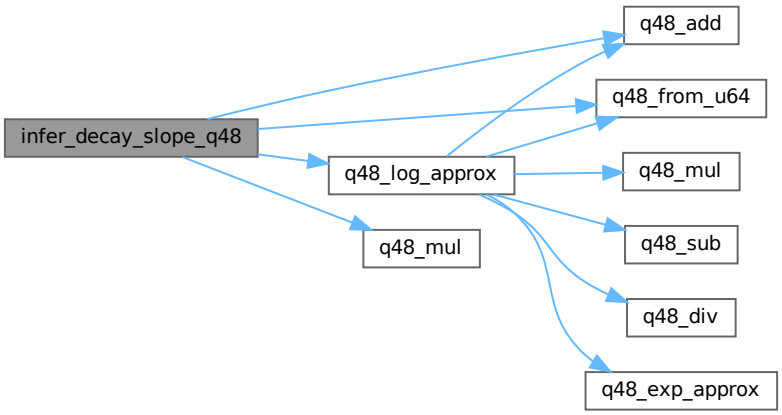
Returns

Inferred decay slope in Q48.16 format (0 = insufficient data)

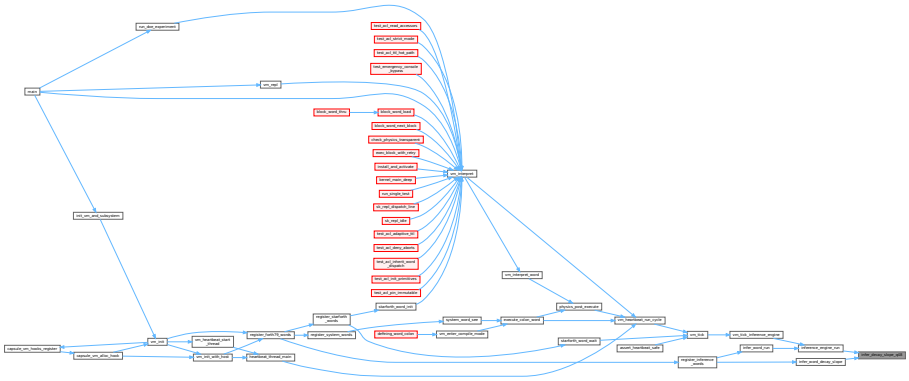
References q48_add(), q48_from_u64(), q48_log_approx(), and q48_mul().

Referenced by infer_word_decay_slope(), and inference_engine_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.284.1.10 inference_engine_run()

```
void inference_engine_run (
            InferenceInputs * inputs,
            InferenceOutputs * outputs )
```

Run adaptive inference engine on window data.

High-level flow:

1. ANOVA Check: Is variance delta $<$ 5%?
   - YES: Return cached outputs immediately (cost: $\sim$100 cycles)
   - NO: Continue to full inference
2. Extract trajectory: Fresh execution_heat snapshot from dictionary
3. Window Width Inference:
   - Compute variance across entire window
   - Detect inflection point (where d(variance)/d(size) $\rightarrow$ 0)
   - Return optimal window size
4. Decay Slope Inference:
   - Transform trajectory to log space (ln(heat[t]))
   - Fit linear regression: ln(heat) = a - slope$*$t
   - Extract slope coefficient in Q48.16
5. Diagnostics:
   - Compute fit quality (R^2 or residual)
   - Store variance metric for next early-exit check

Parameters

inputs	Snapshot of window + metrics (caller-provided)
outputs	Tuning parameters + diagnostics (caller-provided, will be updated)

Returns

void (results_run_01_2025_12_08 in outputs struct)

Preconditions:

- inputs != NULL
- outputs != NULL
- inputs->window != NULL
- inputs->window->is_warm == 1 (window has enough data)
- inputs->trajectory_length > 1

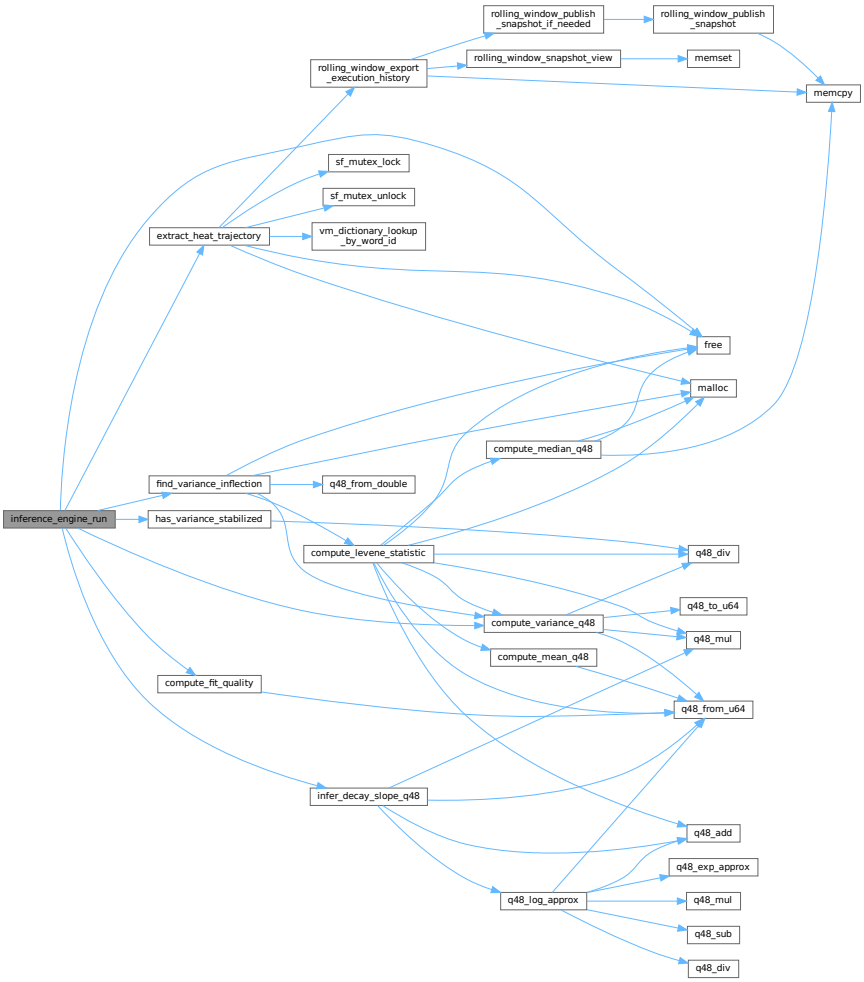
Side Effects:

- Modifies outputs struct (caller must preserve if caching)
- Allocates temporary trajectory buffer (freed on return)

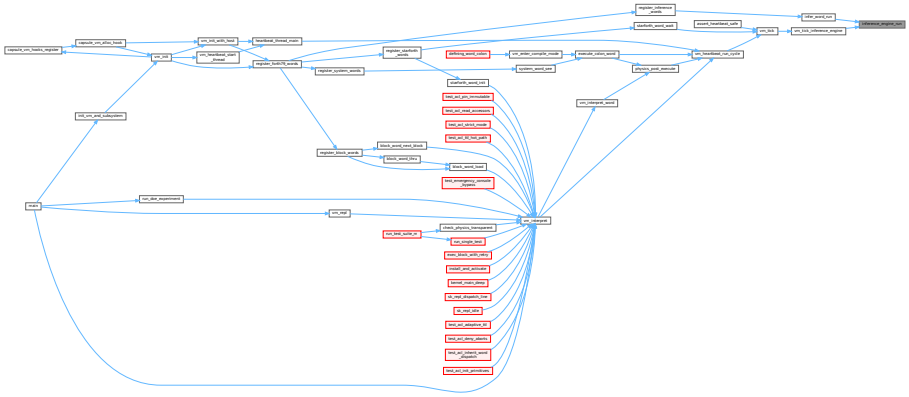
References InferenceOutputs::adaptive_decay_slope, InferenceOutputs::adaptive_window_width, compute_fit_quality(), compute_variance_q48(), InferenceOutputs::early_exited, extract_heat_trajectory(), find_variance_inflection(), free(), has_variance_stabilized(), infer_decay_slope_q48(), ssm_config_t::L5_window_inference, ssm_config_t::L6_decay_inference, InferenceOutputs::slope_fit_quality_q48, VM::ssm_config, InferenceInputs::vm, InferenceInputs::window, and InferenceOutputs::window_variance_q48.

Referenced by infer_word_run(), and vm_tick_inference_engine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.284.1.11 inference_outputs_to_string()

```
const char * inference_outputs_to_string (
```

```
            const InferenceOutputs * outputs )
```

Format inference outputs as a human-readable string.

Writes into a static 256-byte buffer — not thread-safe, for diagnostics only. Format: "window=N var=↩ F slope=F quality=F (cached|full)".

Parameters

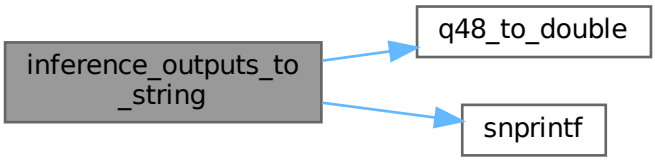
outputs	Inference outputs to format (NULL produces "(null)")

Returns

Pointer to a static string buffer

References InferenceOutputs::adaptive_decay_slope, InferenceOutputs::adaptive_window_width, InferenceOutputs::early_exited, q48_to_double(), InferenceOutputs::slope_fit_quality_q48, snprintf(), and InferenceOutputs::window_variance_q48.

Here is the call graph for this function:



7.284.1.12 inference_outputs_validate()

```
int inference_outputs_validate (
            const InferenceOutputs * outputs )
```

Validate inference outputs for sanity.

Checks that adaptive_window_width is in [ADAPTIVE_MIN_WINDOW_SIZE, ROLLING_WINDOW_↩ SIZE], that adaptive_decay_slope is non-zero and <= 100.0 in Q48.16, and that slope_fit_↩ quality_q48 is in [0.0, 1.0].

Parameters

outputs	Inference outputs to validate (NULL returns 0)

Returns

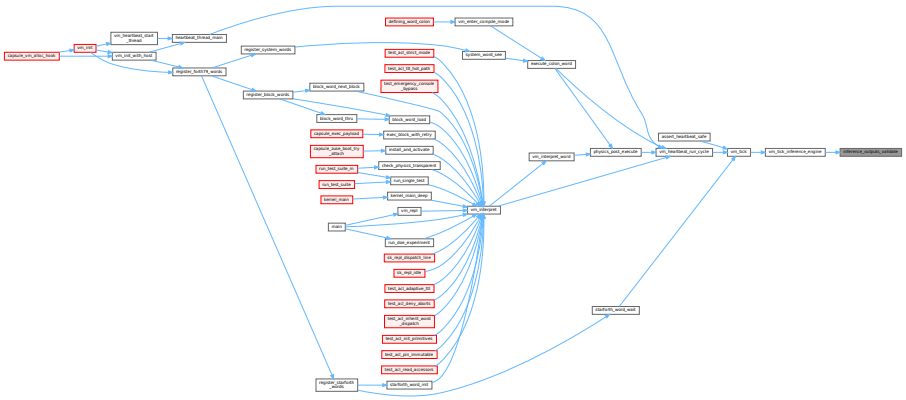
1 if all fields are within acceptable ranges, 0 otherwise

References InferenceOutputs::adaptive_decay_slope, ADAPTIVE_MIN_WINDOW_SIZE, InferenceOutputs::adaptive_window_width, q48_from_u64(), ROLLING_WINDOW_SIZE, and InferenceOutputs::slope_fit_quality_q48.

Referenced by vm_tick_inference_engine().

Here is the call graph for this function:

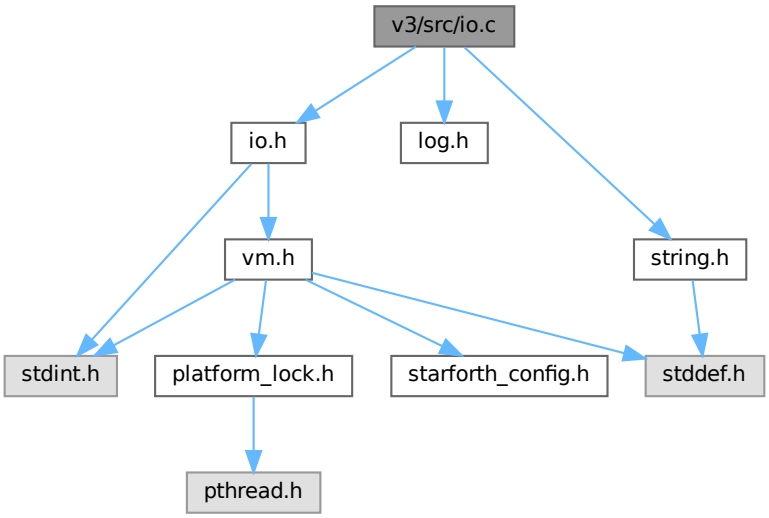
Here is the caller graph for this function:



7.285 v3/src/io.c File Reference

```
#include "io.h"
#include "log.h"
#include <string.h>
```

Include dependency graph for io.c:



Functions

- void io_init (VM *vm)

  Initialize the IO subsystem.
- int io_write_block (VM *vm, int block_num, const unsigned char *buffer)

  Write a block to VM memory.

7.285.1 Function Documentation

7.285.1.1 io_init()

```
void io_init (
            VM * vm )
```

Initialize the IO subsystem.

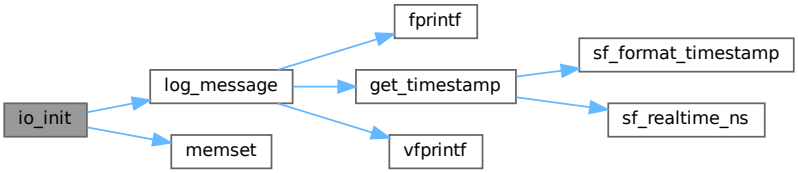
Parameters

vm	Pointer to VM instance

Initializes the block device view over the unified VM memory. Zeros out the entire VM memory space.

References BLOCK_SIZE, LOG_ERROR, LOG_INFO, log_message(), MAX_BLOCKS, VM::memory, memset(), and VM_MEMORY_SIZE.

Here is the call graph for this function:



7.285.1.2 io_write_block()

```
int io_write_block (
            VM * vm,
            int block_num,
            const unsigned char * buffer )
```

Write a block to VM memory.

Parameters

vm	Pointer to VM instance
block_num	Block number to write (0 to MAX_BLOCKS-1)
buffer	Source buffer containing block data

Returns

0 on success, -1 on error

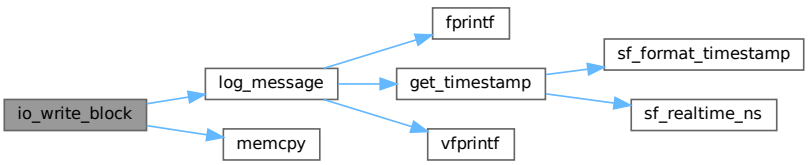
Writes a BLOCK_SIZE chunk of data from the provided buffer into the VM's memory at the specified block number offset. The block number must be within valid range. The entire block is written atomically.

Note

The buffer must contain at least BLOCK_SIZE bytes of valid data

References BLOCK_SIZE, LOG_DEBUG, LOG_ERROR, log_message(), MAX_BLOCKS, memcpy(), and VM::memory.

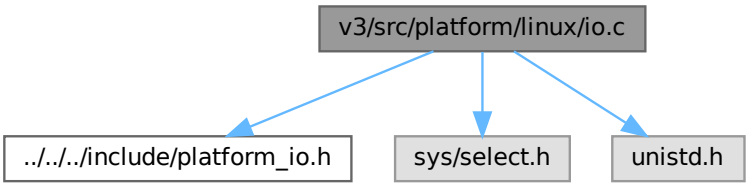
Here is the call graph for this function:



7.286 v3/src/platform/linux/io.c File Reference

```
#include "../../../include/platform_io.h"
#include <sys/select.h>
#include <unistd.h>
```

Include dependency graph for io.c:



Functions

- int sf_terminal_ready (void)

  Non-blocking check for whether a key/character is available to read.

7.286.1 Function Documentation

7.286.1.1 sf_terminal_ready()

```
int sf_terminal_ready (
            void  )
```

Non-blocking check for whether a key/character is available to read.

Must not block and must not consume the byte if one is found (a following KEY/getchar() must still return that same byte).

Returns

1 if input is ready, 0 otherwise

References timeval::tv_sec, and timeval::tv_usec.

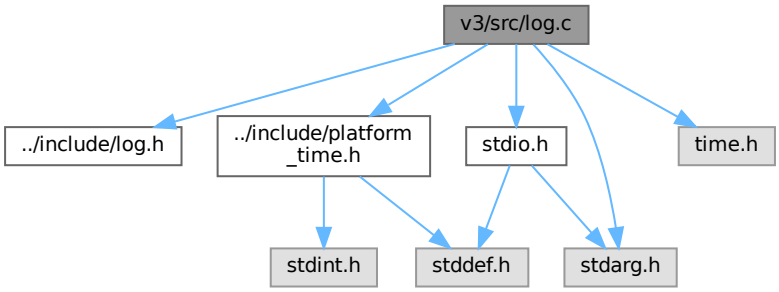
Referenced by io_word_question_terminal().

Here is the caller graph for this function:

7.287 v3/src/log.c File Reference

```
#include "../include/log.h"
#include "../include/platform_time.h"
#include <stdio.h>
#include <stdarg.h>
#include <time.h>
```

Include dependency graph for log.c:



Functions

- void log_set_level (LogLevel level)

  Set the global logging level.
- LogLevel log_get_level (void)

  Gets the current global logging level.
- static void get_timestamp (char ∗buffer, size_t size)

  Helper function to generate timestamp string.
- void log_message (LogLevel level, const char ∗fmt,...)

  Log a formatted message at the specified level.
- void log_test_result (const char ∗word_name, TestResult result)

  Log a test result with colored output.

Variables

- static LogLevel current_level = LOG_INFO
- static const char ∗ level_names [ ]
- static const char ∗ level_colors [ ]
- static const char ∗ test_pass_color = "\x1b[32m"
- static const char ∗ test_fail_color = "\x1b[31m"
- static const char ∗ test_skip_color = "\x1b[33m"
- static const char ∗ color_reset = "\x1b[0m"

7.287.1 Function Documentation

7.287.1.1 get_timestamp()

```
static void get_timestamp (
            char * buffer,
            size_t size )  [static]
```

Helper function to generate timestamp string.

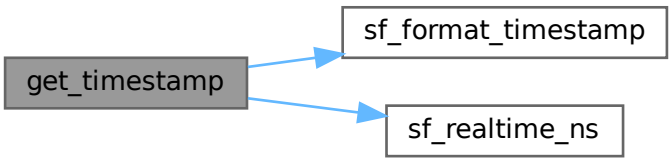
Parameters

buffer	Output buffer for the timestamp string
size	Size of the output buffer

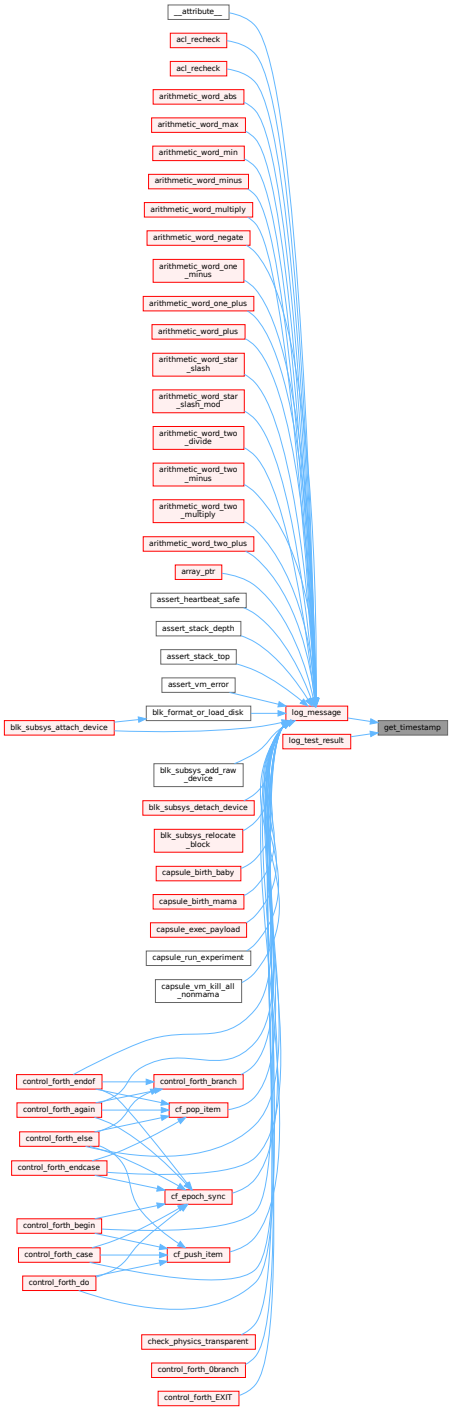
References sf_format_timestamp(), and sf_realtime_ns().

Referenced by log_message(), and log_test_result().

Here is the call graph for this function:



Here is the caller graph for this function:



7.287.1.2 log_get_level()

```
LogLevel log_get_level (
            void  )
```

Gets the current global logging level.

Get the current logging level.

Returns

LogLevel Current logging level

References current_level.

Referenced by log_word_get_level().

Here is the caller graph for this function:

7.287.1.3 log_message()

```
void log_message (
            LogLevel level,
            const char * fmt,
             ...  )
```

Log a formatted message at the specified level.

Parameters

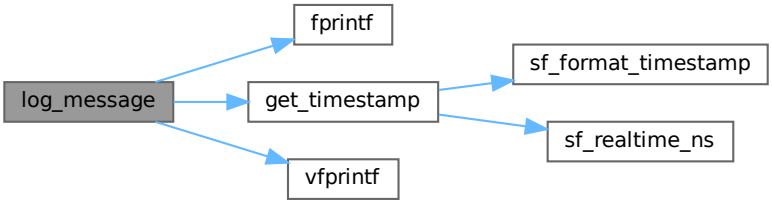
level	The logging level for this message
fmt	Printf-style format string
...	Variable arguments for format string

References color_reset, current_level, fprintf(), get_timestamp(), level, level_colors, level_names, LOG_NONE, stderr, and vfprintf().

Referenced by __attribute__(), acl_recheck(), acl_recheck(), arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), array_ptr(), assert_heartbeat_safe(), assert_stack_depth(), assert_stack_top(), assert_vm_error(), blk_format_or_load_disk(), blk_subsys_add_raw_device(), blk_subsys_attach_device(), blk_subsys_detach_device(), blk_subsys_relocate_block(), capsule_birth_baby(), capsule_birth_mama(), capsule_exec_payload(), capsule_run_experiment(), capsule_vm_kill_all_nonmama(), cf_epoch_sync(), cf_pop_item(), cf_push_item(), check_physics_transparent(), control_forth_0branch(), control_forth_again(), control_forth_begin(), control_forth_branch(), control_forth_case(), control_forth_do(), control_forth_else(), control_forth_endcase(), control_forth_endof(), control_forth_EXIT(), control_forth_I(), control_forth_if(), control_forth_J(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_runtime_do(), control_forth_runtime_leave(), control_forth_runtime_loop(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), control_forth_then(), control_forth_UNLOOP(), control_forth_until(), control_forth_while(), defer_runtime(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_does_rt(), defining_runtime_lit(), defining_runtime_variable(), defining_word_colon(), defining_word_compile(), defining_word_constant(), defining_word_create(), defining_word_does(), defining_word_immediate(), defining_word_left_bracket(), defining_word_literal(), defining_word_right_bracket(), defining_word_variable(), dict_adaptive_optimization_pass(), dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), dictionary_m_word_tick(), dictionary_m_word_to_body(), dictionary_word_fence(), dictionary_word_forget(),

dump_and_die(), editor_word_edit(), editor_word_l(), editor_word_s(), editor_word_show(), emit_cell(), enable_benchmark_mode(), execute_colon_word(), fail_and_exit(), fbw_plot(), format_word_hold(), forth_acl_allow_store(), forth_acl_inherit(), forth_acl_mode_store(), forth_acl_ttl_store(), forth_paren(), hosted_lifecycle_birth(), hosted_lifecycle_kill(), hosted_lifecycle_pause(), hosted_lifecycle_resume(), hosted_lifecycle_use(), infer_word_run(), init_blkio(), init_report(), init_vocabulary_system(), io_init(), io_word_type(), io_write_block(), log_do_emit(), log_emit_string(), log_str_emit(), log_word_set_level(), logical_word_and(), logical_word_equals(), logical_word_false(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_true(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), main(), memory_word_cells(), mixed_math_word_mod(), pop_xt(), print_module_summary(), register_arithmetic_words(), register_forth79_words(), register_word(), return_stack_word_r_fetch(), return_stack_word_r_from(), return_stack_word_to_r(), rolling_window_run_adaptive_pass(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), run_acl_words_tests(), run_all_tests(), run_arithmetic_words_tests(), run_benchmark_mode(), run_block_words_tests(), run_break_me_tests(), run_control_words_tests(), run_defining_words_tests(), run_dictionary_manipulation_words_tests(), run_dictionary_words_tests(), run_doe_experiment(), run_double_words_tests(), run_format_words_tests(), run_inference_words_tests(), run_integration_tests(), run_io_words_tests(), run_logical_words_tests(), run_mama_forth_words_tests(), run_memory_words_tests(), run_mixed_arithmetic_words_tests(), run_module_tests(), run_physics_freeze_words_tests(), run_q48_words_tests(), run_return_stack_words_tests(), run_single_test(), run_stack_words_tests(), run_starforth_words_tests(), run_stress_tests(), run_string_words_tests(), run_system_words_tests(), run_test_suite(), run_test_suite_m(), run_vocabulary_words_tests(), run_word_tests(), sk_repl_dispatch_line(), sk_repl_idle(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), stack_word_depth(), stack_word_drop(), stack_word_dup(), stack_word_minus_rot(), stack_word_over(), stack_word_pick(), stack_word_question_dup(), stack_word_roll(), stack_word_rot(), stack_word_swap(), starforth_word_execution_heat_fetch(), starforth_word_execution_heat_store(), starforth_word_init(), starforth_word_paren_dash(), starforth_word_random(), starforth_word_seed(), starforth_word_top_words(), starforth_word_wait(), starforth_word_zuse_pubkey_fetch(), string_word_blank(), string_word_bracket_tick(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_count(), string_word_expect(), string_word_minus_trailing(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), system_word_reboot(), test_acl_adaptive_ttl(), test_acl_strict_mode(), vblk_io(), verify_mint(), vm_align(), vm_allot(), vm_assert_interpreter_enabled(), vm_bootstrap_root_vocabulary(), vm_bootstrap_scr(), vm_compile_call(), vm_compile_exit(), vm_compile_literal(), vm_compile_word(), vm_create_word(), vm_dictionary_track_entry(), vm_enter_compile_mode(), vm_exit_compile_mode(), vm_get_block_addr(), vm_heartbeat_start_thread(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_init(), vm_init_with_host(), vm_interpret_word(), vm_load_cell(), vm_load_u8(), vm_make_immediate(), vm_pin_execution_heat(), vm_pop(), vm_push(), vm_repl(), vm_smudge_word(), vm_store_cell(), vm_store_u8(), vm_tick_inference_engine(), vm_tick_slope_validator(), vm_tick_window_tuner(), vm_unpin_execution_heat(), vocabulary_create_vocabulary_direct(), vocabulary_select_runtime(), vocabulary_word_definitions(), vocabulary_word_forth(), vocabulary_word_vocabulary(), vt100_enable_ttf(), vt100_init(), word_defer(), word_defer_fetch(), word_divide(), word_divmod(), word_is(), xhci_bot_read_data_in(), xhci_bot_receive_csw(), xhci_bot_send_read10(), xhci_bot_send_read_capacity10(), xhci_bot_send_test_unit_ready(), xhci_bot_send_write10(), xhci_bot_write_data_out(), xhci_bringup(), xhci_cmd_address_device(), xhci_cmd_configure_endpoint(), xhci_cmd_configure_hid_endpoint(), xhci_cmd_disable_slot(), xhci_cmd_enable_slot(), xhci_cmd_reset_endpoint(), xhci_cmd_set_tr_dequeue_pointer(), xhci_drain_pending_event(), xhci_ep0_bot_mass_storage_reset(), xhci_ep0_clear_endpoint_ha
xhci_ep0_get_config_descriptor(), xhci_ep0_get_device_descriptor(), xhci_ep0_hid_set_protocol(), xhci_ep0_set_configuration(), xhci_find_and_map(), xhci_handle_bulk_stall(), xhci_handle_port_connected(), xhci_log_hex32(), xhci_poll_one(), xhci_queue_pending_event(), xhci_stall_fail(), and xhci_take_from_bios().

Here is the call graph for this function:



7.287.1.4 log_set_level()

```
void log_set_level (
            LogLevel level )
```

Set the global logging level.

Parameters

level	The new logging level to set

Note

Default level is LOG_INFO

References current_level, and level.

Referenced by log_word_set_level(), and main().

Here is the caller graph for this function:

7.287.1.5 log_test_result()

```
void log_test_result (
            const char * word_name,
            TestResult result )
```

Log a test result with colored output.

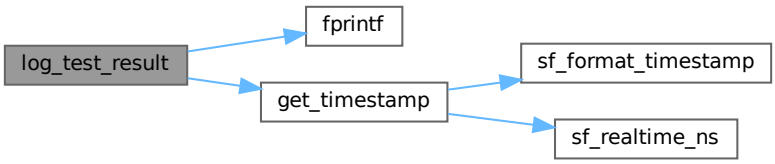
Parameters

word_name	Name of the word being tested
result	Result of the test execution

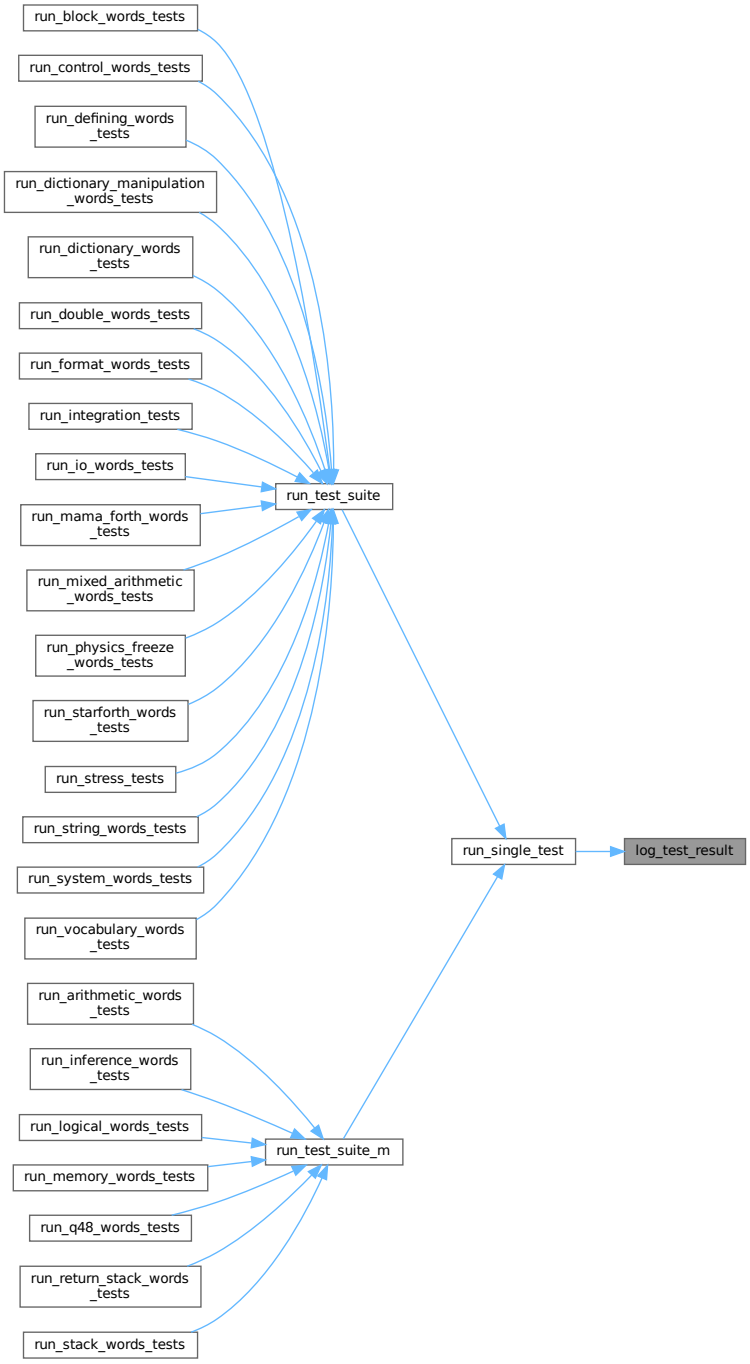
References color_reset, current_level, fprintf(), get_timestamp(), LOG_NONE, LOG_TEST, stderr, TEST_FAIL, test_fail_color, TEST_PASS, test_pass_color, TEST_SKIP, and test_skip_color.

Referenced by run_single_test().

Here is the call graph for this function:



Here is the caller graph for this function:



7.287.2 Variable Documentation

7.287.2.1 color_reset

```
const char* color_reset = "\x1b[0m"  [static]
```

Referenced by log_message(), and log_test_result().

7.287.2.2 current_level

```
LogLevel current_level = LOG_INFO  [static]
```

Referenced by log_get_level(), log_message(), log_set_level(), and log_test_result().

7.287.2.3 level_colors

```
const char* level_colors[]  [static]
```

Initial value:

```
= {
    "\x1b[31m",
    "\x1b[33m",
    "\x1b[32m",
    "\x1b[35m",
    "\x1b[34m"
}
```

Referenced by log_message().

7.287.2.4 level_names

```
const char* level_names[]  [static]
```

Initial value:

```
= {
    "ERROR",
    "WARN",
    "INFO",
    "TEST",
    "DEBUG"
}
```

Referenced by log_message().

7.287.2.5 test_fail_color

```
const char* test_fail_color = "\x1b[31m"  [static]
```

Referenced by log_test_result().

7.287.2.6 test_pass_color

```
const char* test_pass_color = "\x1b[32m"  [static]
```

Referenced by log_test_result().

7.287.2.7 test_skip_color

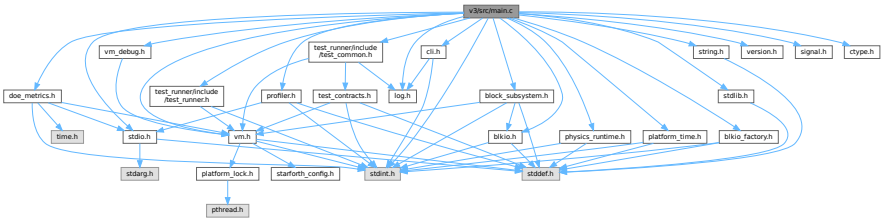
```
const char* test_skip_color = "\x1b[33m"  [static]
```

Referenced by log_test_result().

7.288 v3/src/main.c File Reference

```
#include "vm.h"
#include "vm_debug.h"
#include "log.h"
#include "profiler.h"
#include "physics_runtime.h"
#include "platform_time.h"
#include "version.h"
#include "cli.h"
#include "block_subsystem.h"
#include "test_runner/include/test_runner.h"
#include "test_runner/include/test_common.h"
#include "doe_metrics.h"
#include "blkio.h"
#include "blkio_factory.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <ctype.h>
#include <stdint.h>
```

Include dependency graph for main.c:



Macros

- #define STARFORTH_STATE_BYTES 8192u

Functions

- __attribute__ ((weak))
- static void cleanup_blkio (void)
- void cleanup_and_exit (void)

  Performs cleanup operations before program termination.
- void signal_handler (int sig)
- void print_usage (const char *program_name)

  Prints program usage information.
- static int init_blkio (const CLIConfig *config)

  Initialize block I/O from CLI configuration.
- static int init_vm_and_subsystem (VM *vm)

  Initialize VM core and block subsystem.
- static void run_benchmark_mode (VM *vm, const CLIConfig *config)

  Run benchmark mode and exit.

- static void run_optional_tests (VM *vm, const CLIConfig *config)

  Run optional test modes after system initialization.
- static void run_doe_experiment (VM *vm)

  Run Design of Experiments experiment.
- int main (int argc, char *argv[])

  Main entry point for StarForth.

Variables

- static int g_blkio_opened = 0
- static uint8_t * g_ram_base = NULL
- static size_t g_ram_bytes = 0
- static int g_ram_is_mmap = 0

7.288.1 Macro Definition Documentation

7.288.1.1 STARFORTH_STATE_BYTES

```
#define STARFORTH_STATE_BYTES 8192u
```

7.288.2 Function Documentation

7.288.2.1 __attribute__()

```
__attribute__ (
             (weak)  )
```

7.288.2.2 cleanup_and_exit()

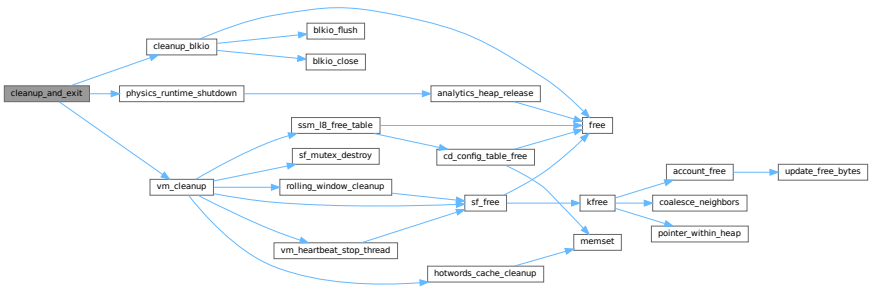
```
void cleanup_and_exit (
             void  )
```

Performs cleanup operations before program termination.

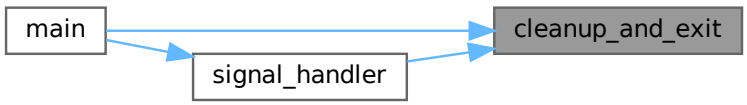
References cleanup_blkio(), physics_runtime_shutdown(), and vm_cleanup().

Referenced by main(), and signal_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



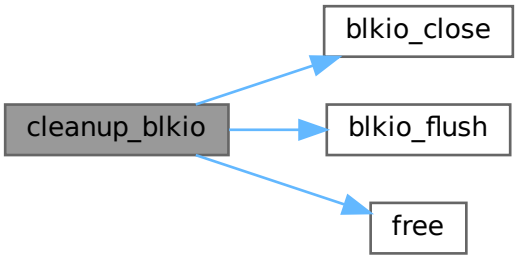
7.288.2.3 cleanup_blkio()

```
static void cleanup_blkio (
            void  )  [static]
```

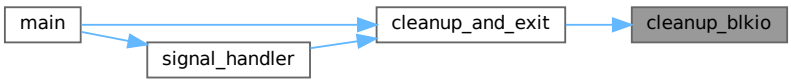
References blkio_close(), blkio_flush(), free(), g_blkio_opened, g_ram_base, g_ram_bytes, and g_ram_is_mmap.

Referenced by cleanup_and_exit().

Here is the call graph for this function:



Here is the caller graph for this function:



7.288.2.4 init_blkio()

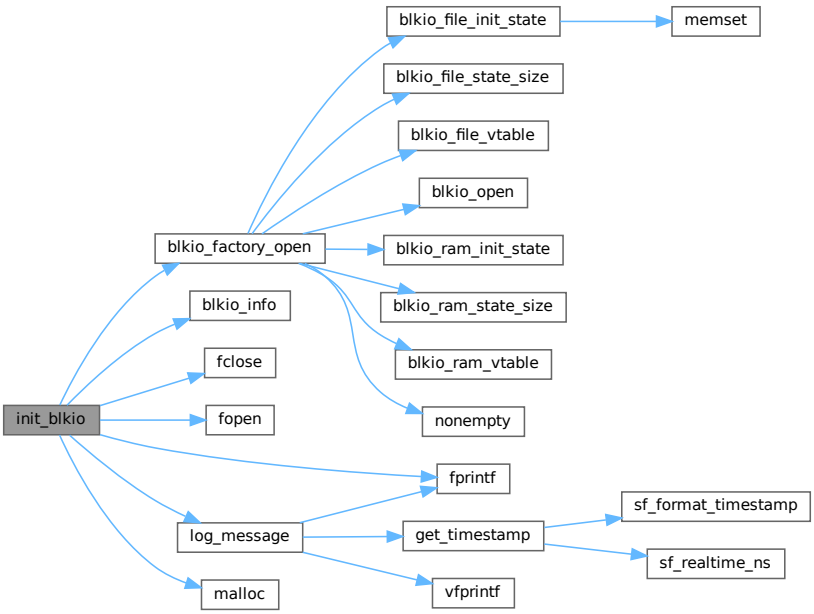
```
static int init_blkio (
            const CLIConfig * config )  [static]
```

Initialize block I/O from CLI configuration.

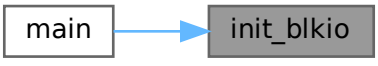
References blkio_factory_open(), BLKIO_FORTH_BLOCK_SIZE, blkio_info(), BLKIO_OK, CLIConfig::disk_img_path, fclose(), fopen(), blkio_info_t::forth_block_size, fprintf(), g_blkio_opened, g_ram_base, g_ram_bytes, g_ram_is_mmap, LOG_INFO, log_message(), LOG_WARN, malloc(), blkio_info_t::phys_size_bytes, CLIConfig::ram_disk_mb, blkio_info_t::read_only, STARFORTH_STATE_BYTES, stderr, and blkio_info_t::total_blocks.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.288.2.5 init_vm_and_subsystem()

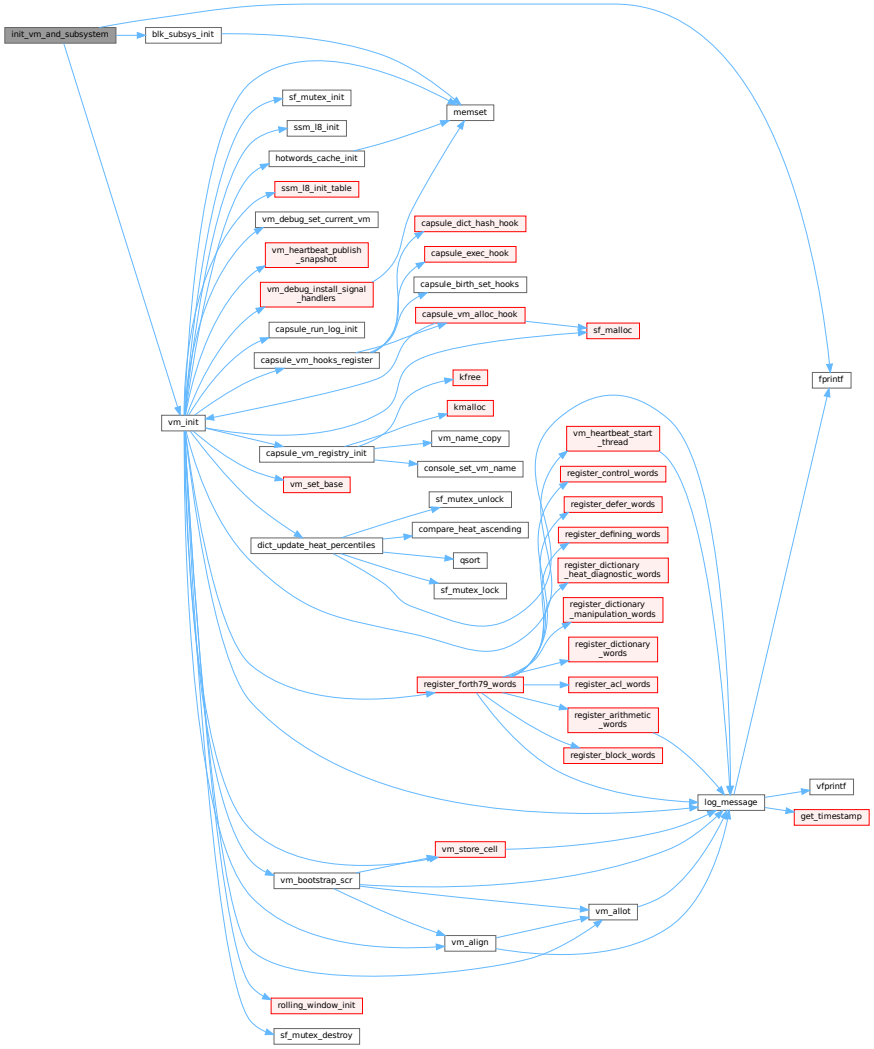
```
static int init_vm_and_subsystem (
            VM * vm )  [static]
```

Initialize VM core and block subsystem.

References BLK_FORTH_SIZE, BLK_RAM_BLOCKS, blk_subsys_init(), VM::error, fprintf(), g_blkio_opened, ram_base, ram_size, stderr, stdout, and vm_init().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

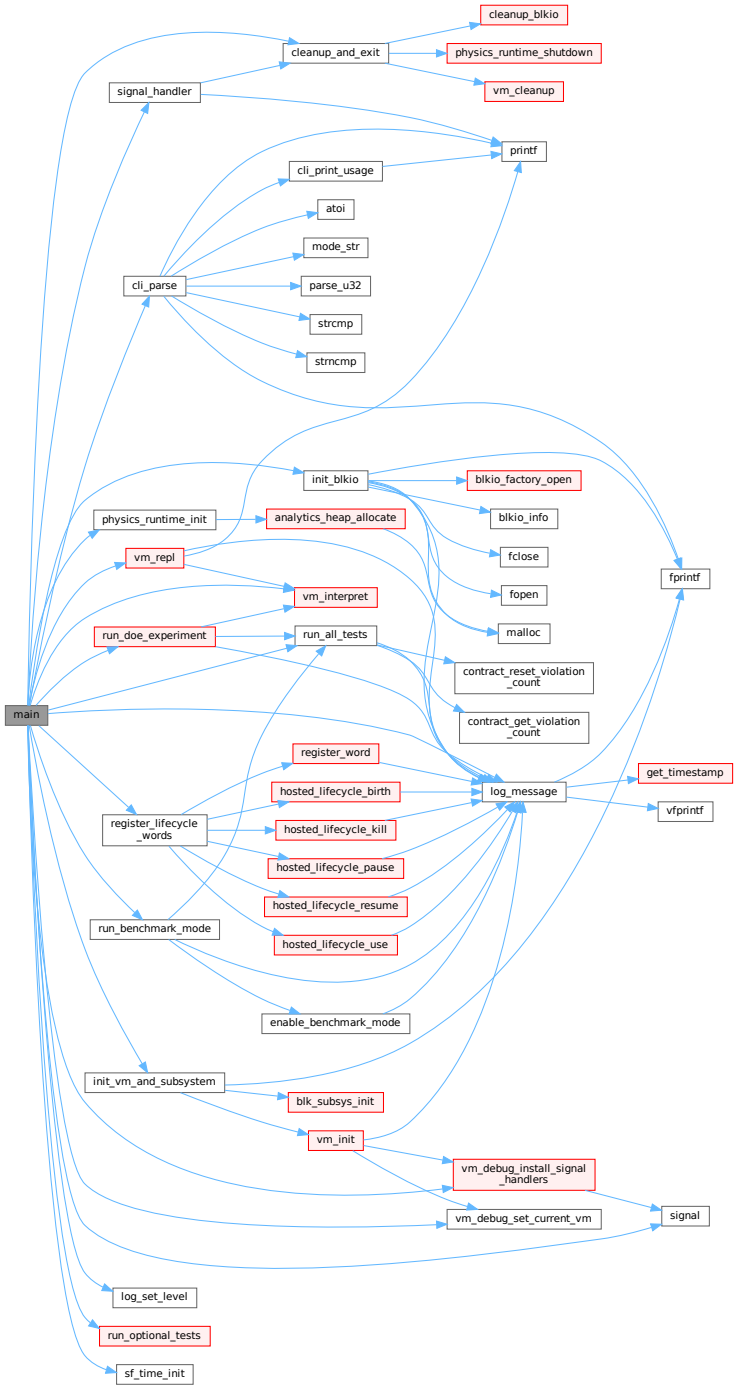
7.288.2.6 main()

```
int main (
            int argc,
            char * argv[] )
```

Main entry point for StarForth.

References CLIConfig::benchmark, CLIConfig::break_me, cleanup_and_exit(), cli_parse(), VM::dict_fence_here, VM::dict_fence_latest, CLIConfig::doe_experiment, VM::error, CLIConfig::fail_fast, fail_fast, VM::here, init_blkio(), init_vm_and_subsystem(), VM::latest, LOG_ERROR, LOG_INFO, CLIConfig::log_level, CLIConfig::log_level_explicitly_set, log_message(), LOG_NONE, log_set_level(), LOG_TEST, LOG_WARN, physics_runtime_init(), register_lifecycle_words(), run_all_tests(), run_benchmark_mode(), run_doe_experiment(), run_optional_tests(), CLIConfig::run_tests_flag_observed, CLIConfig::script_mode, sf_time_init(), SIGINT, signal(), signal_handler(), SIGTERM, vm_debug_install_signal_handlers(), vm_debug_set_current_vm(), vm_interpret(), and vm_repl().

Here is the call graph for this function:



7.288.2.7 print_usage()

```
void print_usage (
            const char * program_name )
```

Prints program usage information.

Parameters

program_name	Name of the executable

References printf().

Here is the call graph for this function:

7.288.2.8 run_benchmark_mode()

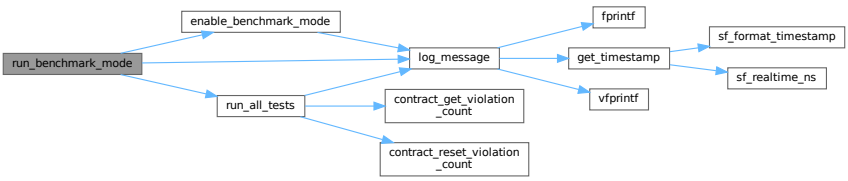
```
static void run_benchmark_mode (
            VM * vm,
            const CLIConfig * config )  [static]
```

Run benchmark mode and exit.

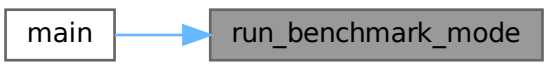
References CLIConfig::benchmark_iterations, enable_benchmark_mode(), LOG_INFO, log_message(), and run_all_tests().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



7.288.2.9 run_doe_experiment()

```
static void run_doe_experiment (
            VM * vm )  [static]
```

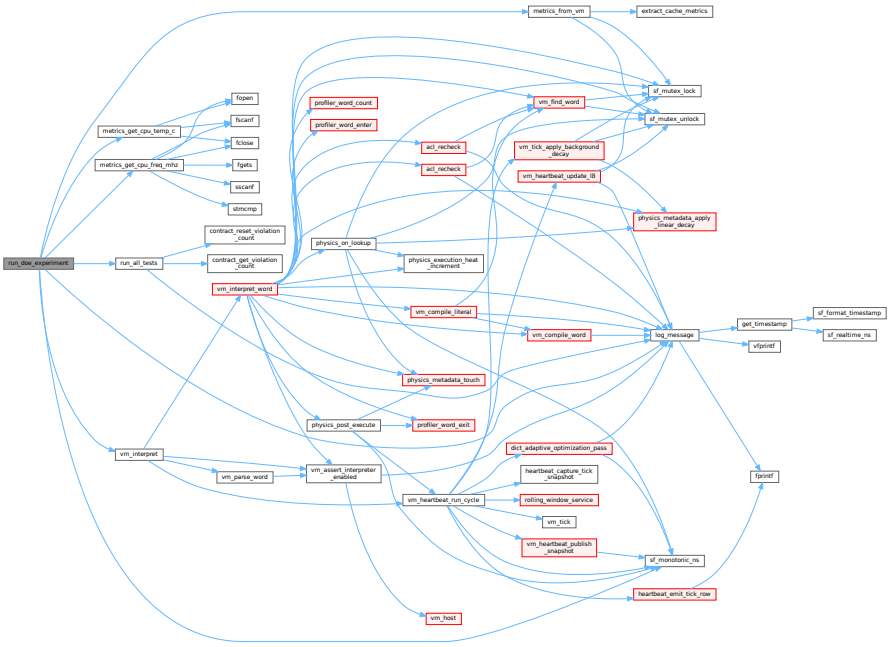
Run Design of Experiments experiment.

Executes the full test harness N times, collecting metrics after each run. Writes results to CSV file for statistical analysis.

References RollingWindowOfTruth::effective_window_size, RollingWindowOfTruth::is_warm, LOG_INFO, log_message(), metrics_from_vm(), metrics_get_cpu_freq_mhz(), metrics_get_cpu_temp_c(), VM::rolling_window, run_all_tests(), sf_monotonic_ns(), RollingWindowOfTruth::total_executions, and vm_interpret().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

7.288.2.10 run_optional_tests()

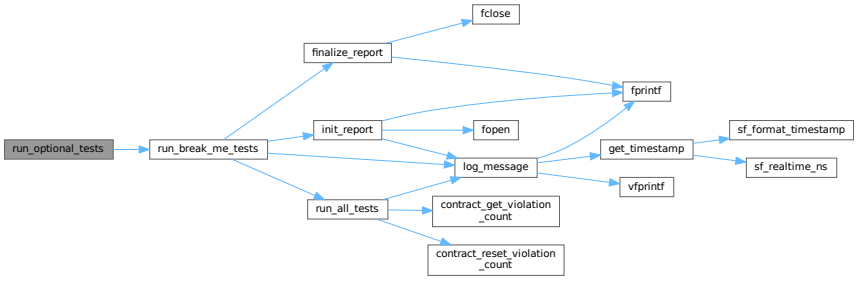
```
static void run_optional_tests (
            VM * vm,
            const CLIConfig * config )  [static]
```

Run optional test modes after system initialization.

References CLIConfig::break_me, and run_break_me_tests().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

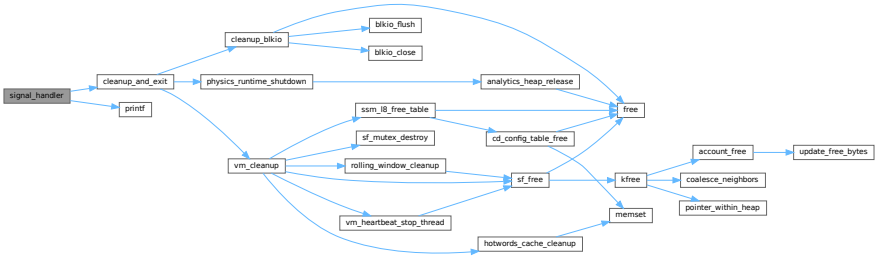
7.288.2.11 signal_handler()

```
void signal_handler (
            int sig )
```

References cleanup_and_exit(), printf(), SIGINT, and SIGTERM.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

7.288.3 Variable Documentation

7.288.3.1 g_blkio_opened

```
int g_blkio_opened = 0  [static]
```

Referenced by cleanup_blkio(), init_blkio(), and init_vm_and_subsystem().

7.288.3.2 g_ram_base

```
uint8_t* g_ram_base = NULL  [static]
```

Referenced by cleanup_blkio(), and init_blkio().

7.288.3.3 g_ram_bytes

```
size_t g_ram_bytes = 0  [static]
```

Referenced by cleanup_blkio(), and init_blkio().

7.288.3.4 g_ram_is_mmap

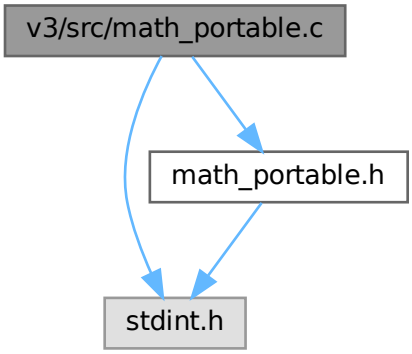
```
int g_ram_is_mmap = 0  [static]
```

Referenced by cleanup_blkio(), and init_blkio().

7.289 v3/src/math_portable.c File Reference

```
#include <stdint.h>
#include "math_portable.h"
```

Include dependency graph for math_portable.c:



Functions

- static int64_t exp_q48 (int64_t x)

  Compute exp(x) via Taylor series in Q48.16 fixed-point.
- int64_t sqrt_q48 (int64_t x)

  Compute sqrt(x) via Newton-Raphson in Q48.16 fixed-point.
- int64_t erf_q48 (int64_t x)

  Compute erf(x) via Abramowitz & Stegun 7.1.26 in Q48.16 fixed-point.

7.289.1 Function Documentation

7.289.1.1 erf_q48()

```
int64_t erf_q48 (
            int64_t x )
```

Compute erf(x) via Abramowitz & Stegun 7.1.26 in Q48.16 fixed-point.

Error function in Q48.16 fixed-point Input: x in Q48.16 format (dimensionless Z-score) Output: erf(x) in Q48.↩ 16 format (range: -Q48_SCALE to +Q48_SCALE) Uses Abramowitz & Stegun approximation (7.1.26) Maximum error: 1.5e-7 relative to floating-point erf Properties:

Implements the rational approximation from Abramowitz & Stegun, formula 7.1.26:
$\mathrm{erf}(x) \approx \mathrm{sign}(x) \times \sqrt{1 - \exp(-x^2 \times (4/\pi + a \cdot x^2) / (1 + a \cdot x^2))}$

where $a \approx 0.147$ (in Q48.16: 9633). The maximum error of the true mathematical approximation is $|\varepsilon| < 1.5 \times 10^{-}$?; the fixed-point encoding introduces additional rounding of approximately 1 ULP in Q48.16.

All arithmetic is performed in 64-bit integer Q48.16 format — no floating-point operations. The computation sequence:

1. Extract sign; work with $|x|$.
2. Compute x^2 with 8-bit prescaling to prevent overflow: `x_sq` = (x >> 8) * (x >> 8).
3. Evaluate the A&S numerator ($4/\pi + a \cdot x^2$) and denominator ($1 + a \cdot x^2$).
4. Compute `ratio` = numerator / denominator (shift to preserve Q48.16 scale).
5. Compute `exp_arg` = ?x^2 × ratio and evaluate `exp_q48(exp_arg)`.
6. Compute `base` = 1 ? exp_val, clamp to [0, 1].
7. Return `sign` × sqrt_q48(base).

Q48.16 constants used:

- `a_q48` = 9633 (0.147 × 65536)
- `four_over_pi_q48` = 83328 ($4/\pi$ × 65536 ≈ 1.2732 × 65536)

Consumed by `hotwords_speedup_estimate()` to compute P(speedup > 1.1×) and P(speedup > 2.0×) via the normal CDF identity:

```
P(Z ≤ z) = 0.5 × (1 + erf(z / √2))
```

where $\sqrt{2}$ in Q48.16 = 92681.

Parameters

x	Input value in Q48.16 format (dimensionless Z-score; 1.0 = 65536). May be negative; sign determines the sign of the result.

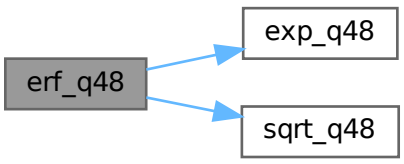
Returns

erf(x) in Q48.16 format, in the range [?65536, +65536] corresponding to (?1.0, +1.0).

References base, exp_q48(), and sqrt_q48().

Referenced by hotwords_speedup_estimate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.289.1.2 exp_q48()

```
static int64_t exp_q48 (
            int64_t x )  [static]
```

Compute exp(x) via Taylor series in Q48.16 fixed-point.

Evaluates the classic Taylor expansion:
exp(x) = 1 + x + $x^2/2!$ + $x^3/3!$ + ... (up to 40 terms)

entirely in 64-bit integer arithmetic using the Q48.16 representation (1.0 = 65536 = 2^16).

The function is intentionally scoped to the range the physics engine actually uses — the argument to `erf_q48()` is $-x^2$ times a ratio near 1, yielding values in roughly the [-20, 0] interval. Outside this range the implementation provides coarse-grained clamping:

- x == 0 → returns `Q48` (1.0 exactly)
- x > 20.0 → saturates at `Q48` << 8 (approx e^20, avoids overflow)
- x < -100.0 → underflows to 0 (result is negligible)

The term loop exits early on the first term smaller than 1 ULP (the series has converged) and guards against intermediate overflow by clamping `result` at `INT64_MAX-1`.

Note

This is a file-local (`static`) helper; callers are `erf_q48()` only.

Parameters

x	Input value in Q48.16 format (1.0 = 65536). Intended range: approximately [-100×65536, +20×65536].

Returns

exp(x) in Q48.16 format; 0 on underflow; saturated at `INT64_MAX-1` on overflow.

Referenced by erf_q48().

Here is the caller graph for this function:

7.289.1.3 sqrt_q48()

```
int64_t sqrt_q48 (
            int64_t x )
```

Compute sqrt(x) via Newton-Raphson in Q48.16 fixed-point.

Square root in Q48.16 fixed-point Input: x in Q48.16 format Output: sqrt(x) in Q48.16 format Uses Newton-↩ Raphson: x_{n+1} = (x_n + a/x_n) / 2 Converges in ∼10-15 iterations Returns 0 for negative inputs (guard against domain error)

Applies the Newton-Raphson recurrence:
`r_{n+1} = (r_n^2 + x) / (2 × r_n)`

for up to 20 iterations, with all arithmetic scaled to the Q48.16 representation (1.0 = 65536). Intermediate products use 8-bit right shifts on both operands before squaring to prevent 64-bit overflow:
`guess_squared = (guess » 8) * (guess » 8)`

This sacrifices 16 bits of precision in the squared term, which is acceptable because the division by `2*guess` restores the correct order of magnitude. Convergence is declared when successive iterates differ by at most 1 in Q48.16 units (sub-nanosecond for timing uses).

Special cases handled without iteration:

- x ≤ 0 → returns 0 (no imaginary results)
- x == 65536 (1.0) → returns 65536 exactly

Initial guess: x/2 for x > 1.0, else 1.0. Values above 1.0 converge from above; values in (0, 1.0) converge from the 1.0 initial guess.

Parameters

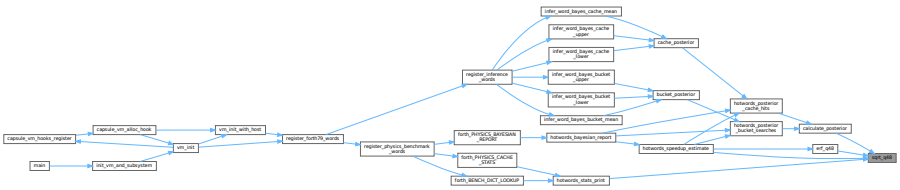
x	Non-negative value in Q48.16 format (1.0 = 65536). Negative values are treated as zero.

Returns

sqrt(x) in Q48.16 format; 0 for x ≤ 0.

Referenced by calculate_posterior(), erf_q48(), hotwords_speedup_estimate(), and hotwords_stats_print().

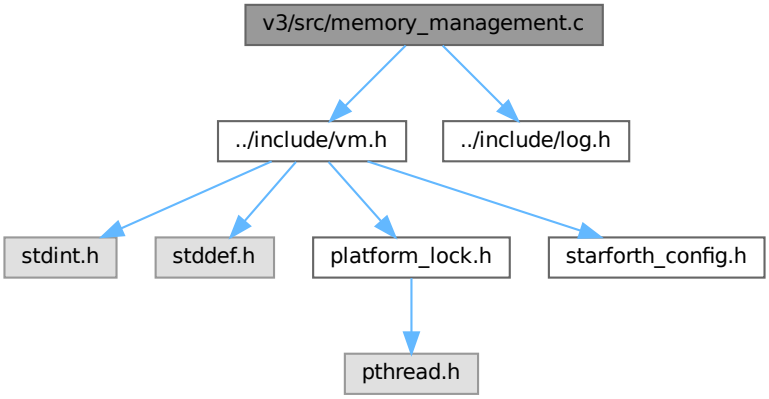
Here is the caller graph for this function:



7.290 v3/src/memory_management.c File Reference

```
#include "../include/vm.h"
#include "../include/log.h"
```

Include dependency graph for memory_management.c:



Functions

- void * vm_allot (VM *vm, size_t bytes)

  Allocates memory from the VM's dictionary space.
- void vm_align (VM *vm)

  Aligns the VM's dictionary pointer (here) to the cell boundary.
- void * vm_get_block_addr (VM *vm, int block_num)

  Gets the memory address for a specified block number.
- int vm_addr_to_block (VM *vm, void *addr)

  Converts a memory address to its corresponding block number.

7.290.1 Function Documentation

7.290.1.1 vm_addr_to_block()

```
int vm_addr_to_block (
            VM * vm,
            void * addr )
```

Converts a memory address to its corresponding block number.

Parameters

vm	Pointer to the VM instance
addr	Memory address to convert

Returns

Block number (0 to MAX_BLOCKS-1) or -1 if address is outside VM memory

Note

Used for determining which block a memory address belongs to

References BLOCK_SIZE, VM::memory, and VM_MEMORY_SIZE.

7.290.1.2 vm_align()

```
void vm_align (
            VM * vm )
```

Aligns the VM's dictionary pointer (here) to the cell boundary.

Parameters

vm	Pointer to the VM instance

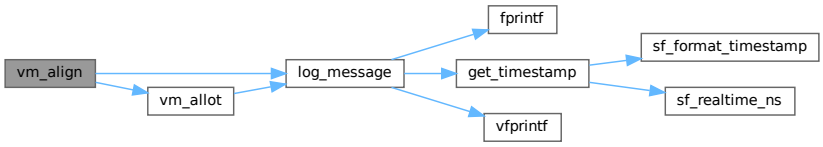
Note

Adds padding bytes if necessary to ensure proper alignment for cell_t type

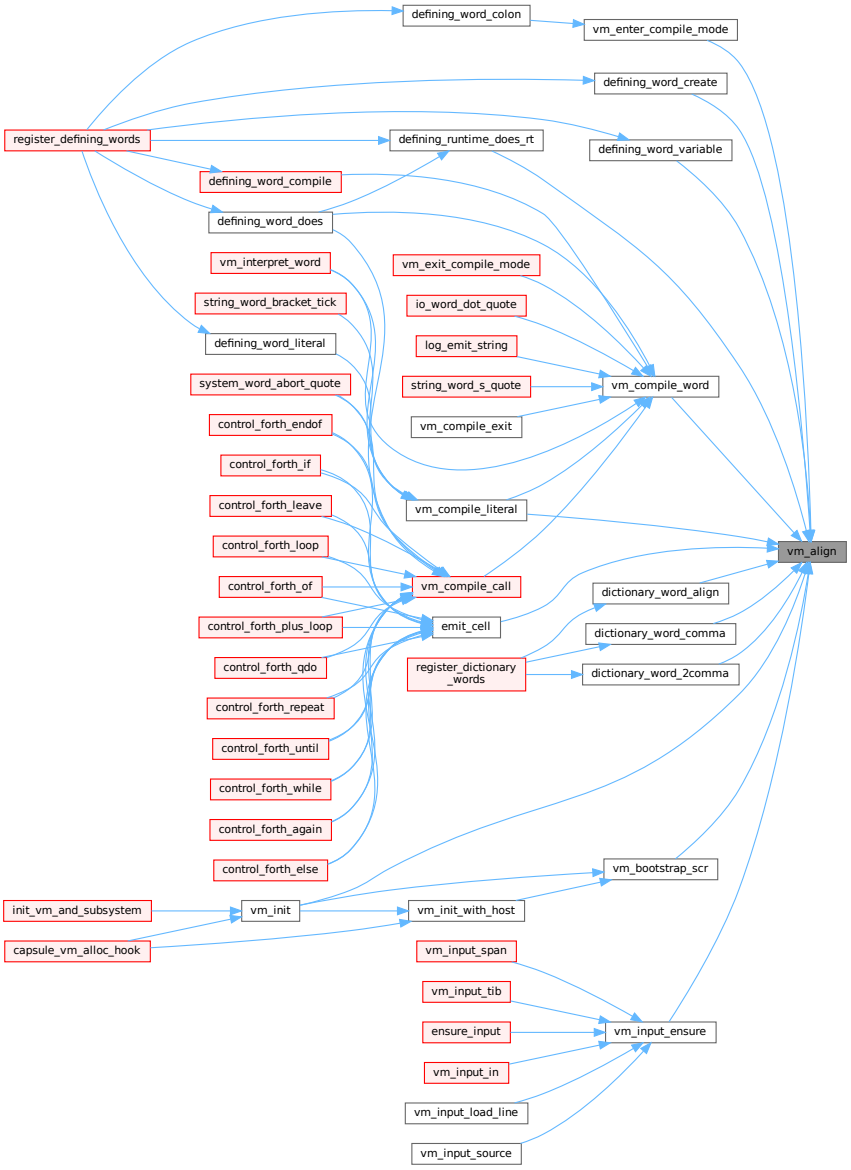
References bytes, VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), and vm_allot().

Referenced by defining_runtime_does_rt(), defining_word_create(), defining_word_variable(), dictionary_word_2comma(), dictionary_word_align(), dictionary_word_comma(), emit_cell(), vm_bootstrap_scr(), vm_compile_literal(), vm_compile_word(), vm_enter_compile_mode(), vm_init(), and vm_input_ensure().

Here is the call graph for this function:



Here is the caller graph for this function:



7.290.1.3 vm_allot()

```
void * vm_allot (
            VM * vm,
            size_t bytes )
```

Allocates memory from the VM's dictionary space.

Parameters

vm	Pointer to the VM instance
bytes	Number of bytes to allocate

Returns

Pointer to the allocated memory or NULL if allocation fails

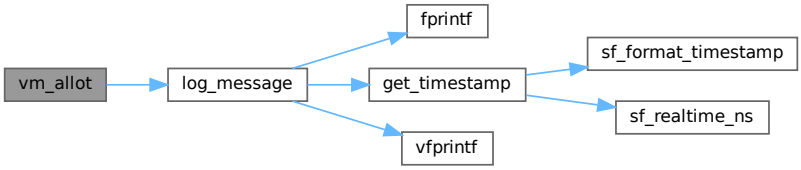
Note

The allocation is done from the VM's dictionary space which is limited to DICTIONARY_MEMORY_SIZE

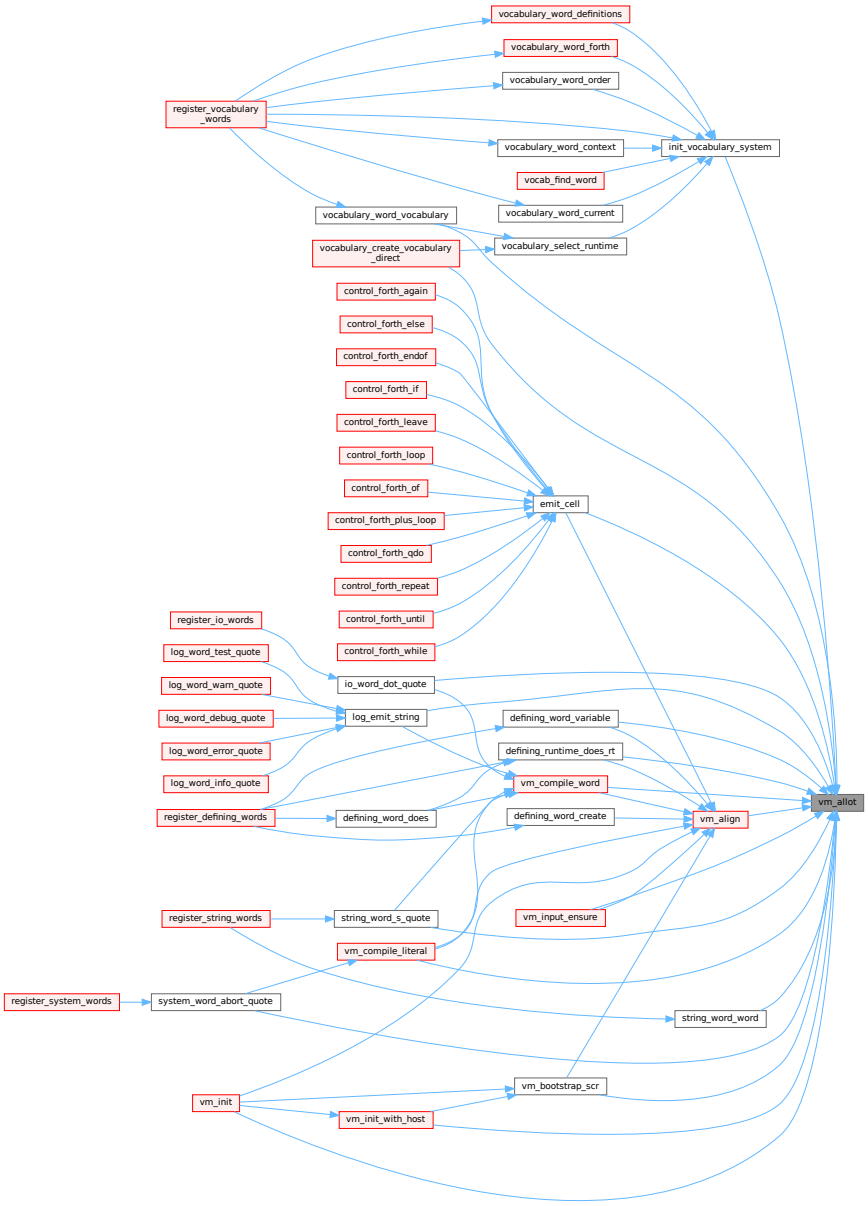
References BLOCK_SIZE, bytes, DICTIONARY_MEMORY_SIZE, VM::error, VM::here, LOG_DEBUG, LOG_ERROR, log_message(), and VM::memory.

Referenced by defining_runtime_does_rt(), defining_word_variable(), emit_cell(), init_vocabulary_system(), io_word_dot_quote(), log_emit_string(), string_word_s_quote(), string_word_word(), system_word_abort_quote(), vm_align(), vm_bootstrap_scr(), vm_compile_literal(), vm_compile_word(), vm_init(), vm_init_with_host(), vm_input_ensure(), vocabulary_create_vocabulary_direct(), and vocabulary_word_vocabulary().

Here is the call graph for this function:



Here is the caller graph for this function:



7.290.1.4 vm_get_block_addr()

```
void * vm_get_block_addr (
            VM * vm,
            int block_num )
```

Gets the memory address for a specified block number.

Parameters

vm	Pointer to the VM instance
block_num	Block number to get the address for

Returns

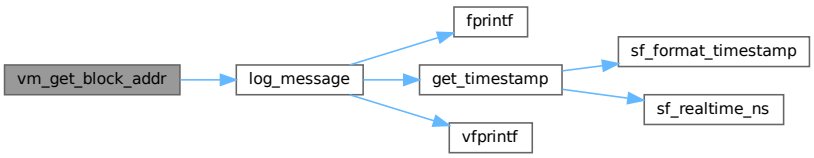
Pointer to the start of the block or NULL if block number is invalid

Note

Blocks are fixed-size memory regions of BLOCK_SIZE bytes

References BLOCK_SIZE, LOG_ERROR, log_message(), MAX_BLOCKS, and VM::memory.

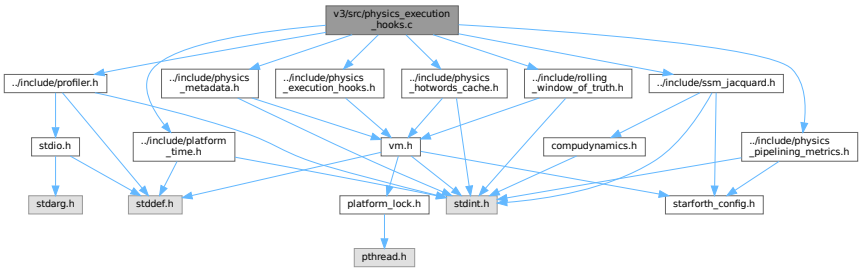
Here is the call graph for this function:



7.291 v3/src/physics_execution_hooks.c File Reference

```
#include "../include/physics_execution_hooks.h"
#include "../include/platform_time.h"
#include "../include/profiler.h"
#include "../include/physics_metadata.h"
#include "../include/physics_hotwords_cache.h"
#include "../include/physics_pipelining_metrics.h"
#include "../include/rolling_window_of_truth.h"
#include "../include/ssm_jacquard.h"
```

Include dependency graph for physics_execution_hooks.c:



Functions

- void physics_pre_execute (VM *vm, DictEntry *word, DictEntry *prev)

  Pre-execution physics hook — called before every word dispatch.
- void physics_post_execute (VM *vm, DictEntry *word)

  Post-execution physics hook — called after every word dispatch.
- void physics_on_lookup (VM *vm, DictEntry *entry, DictEntry *canon)

  Physics hook for the outer interpreter word-lookup path.

7.291.1 Function Documentation

7.291.1.1 physics_on_lookup()

```
void physics_on_lookup (
            VM * vm,
            DictEntry * entry,
            DictEntry * canon )
```

Physics hook for the outer interpreter word-lookup path.

Called when the outer interpreter successfully finds a word by name, before executing it. Applies linear decay and increments heat on both the found entry and the canonical entry (if different), and updates `last_active_ns` / `last_decay_ns` on both. Acquires `vm->dict_lock` for the entire operation to ensure thread safety.

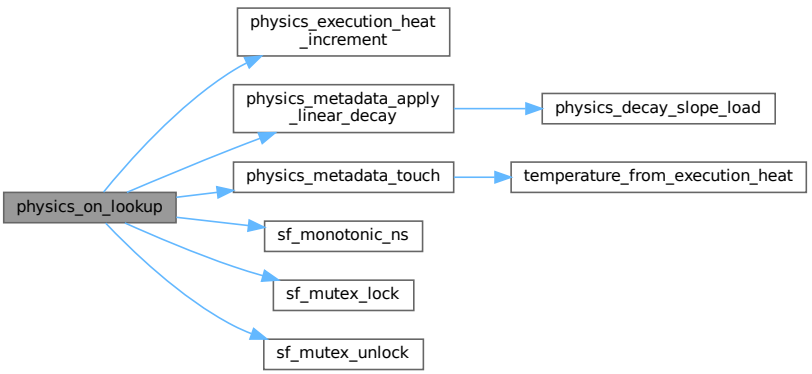
Parameters

vm	Pointer to the VM structure
entry	DictEntry found by name search
canon	Canonical DictEntry (may equal `entry` or be the smudge target)

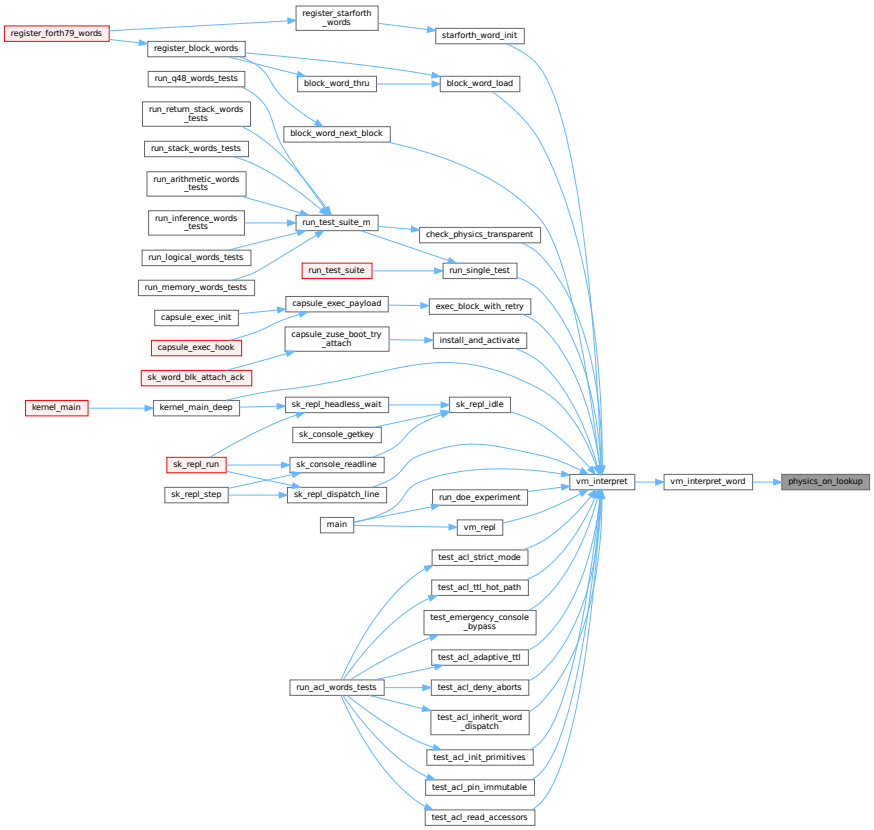
References VM::dict_lock, entry, DictEntry::execution_heat, VM::heartbeat, DictPhysics::last_active_ns, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, DictEntry::physics, physics_execution_heat_increment(), physics_metadata_apply_linear_decay(), physics_metadata_touch(), sf_monotonic_ns(), sf_mutex_lock(), sf_mutex_unlock(), and HeartbeatState::tick_count.

Referenced by vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.291.1.2 physics_post_execute()

```
void physics_post_execute (
            VM * vm,
            DictEntry * word )
```

Post-execution physics hook — called after every word dispatch.

Consolidates all physics work that must happen after a word completes:

- Updates physics metadata (`temperature_q8`, `last_active_ns`) via `physics_metadata_touch()`
- Notifies the profiler (`profiler_word_exit()`)
- Increments `vm->heartbeat.words_executed`
- When running without a background heartbeat thread, checks whether enough words have been executed to trigger a heartbeat cycle (`vm_heartbeat_run_cycle()`)

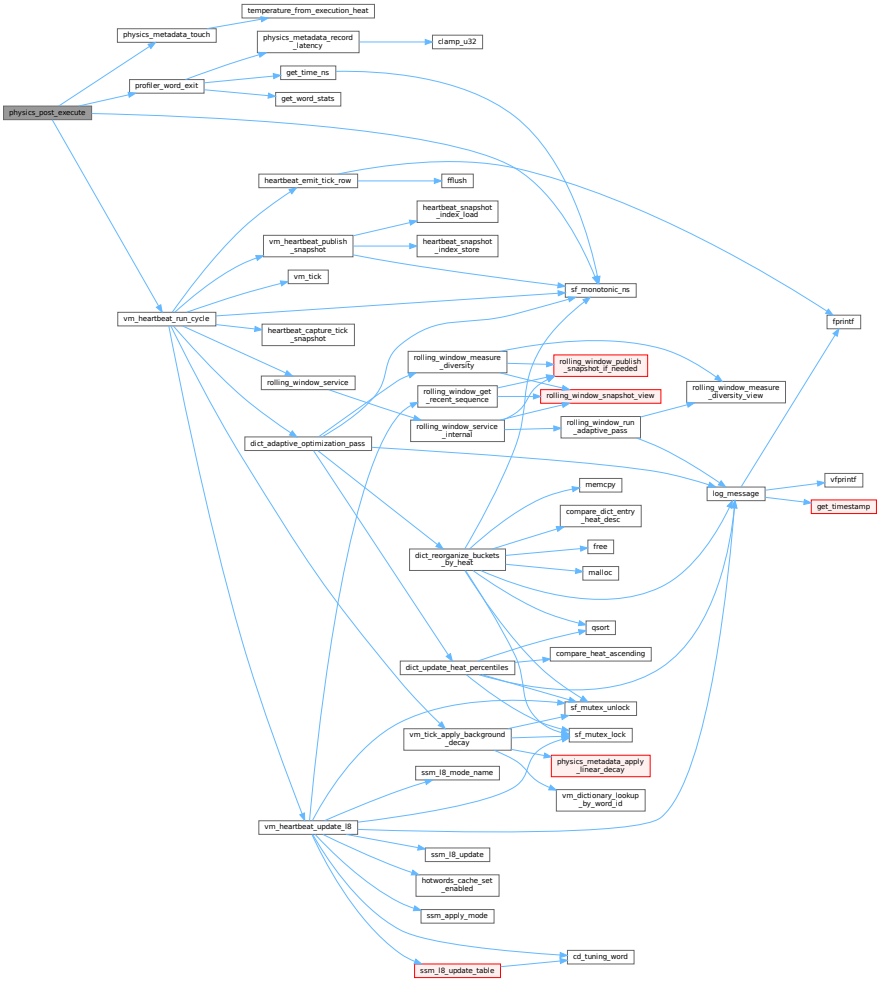
Parameters

vm	Pointer to the VM structure
word	DictEntry that just finished executing

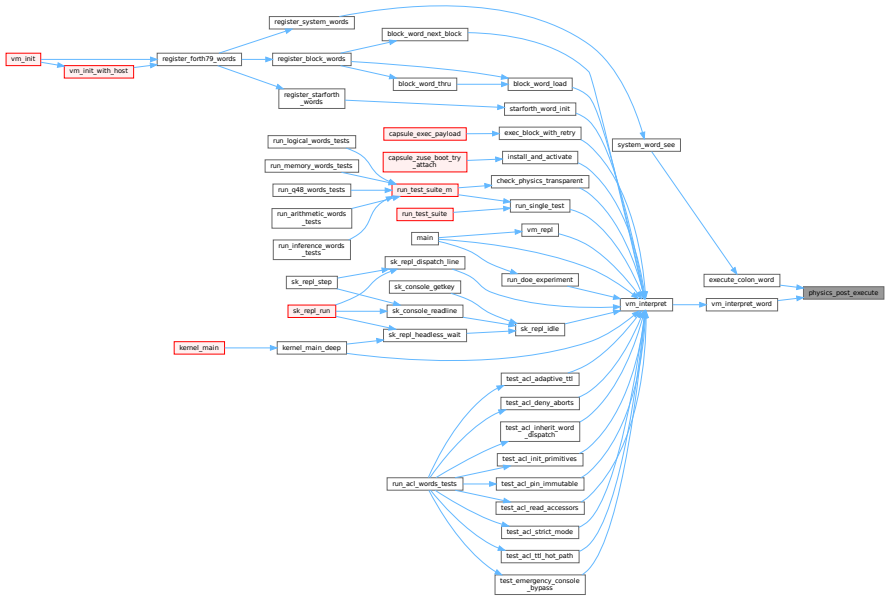
References HeartbeatState::check_counter, DictEntry::execution_heat, VM::heartbeat, HEARTBEAT_CHECK_FREQUENCY, physics_metadata_touch(), profiler_word_exit(), sf_monotonic_ns(), vm_heartbeat_run_cycle(), HeartbeatState::words_executed, and HeartbeatState::worker.

Referenced by execute_colon_word(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.291.1.3 physics_pre_execute()

```
void physics_pre_execute (
            VM * vm,
            DictEntry * word,
            DictEntry * prev )
```

Pre-execution physics hook — called before every word dispatch.

Consolidates all physics work that must happen before a word runs:

- Loop #3: applies linear decay based on elapsed time since last execution
- Loop #1: increments `execution_heat` for the word
- Loop #2: records the word-id into the rolling window of truth
- Loop #4: detects a pipelining prefetch hit (if `prev`'s speculation matched `word`), records the transition from `prev` to `word`, updates the probability cache, and issues a speculative hotwords cache promotion for the most likely next word

Parameters

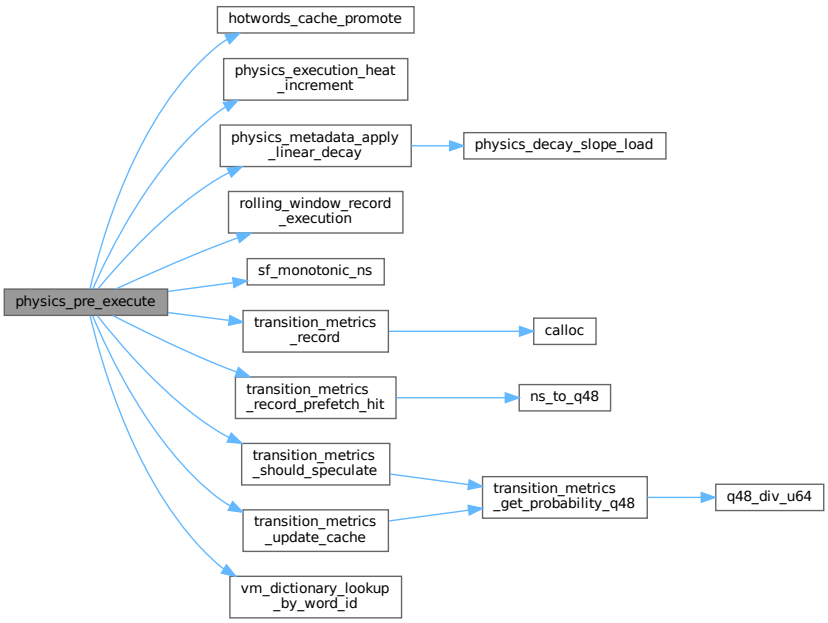
vm	Pointer to the VM structure
word	DictEntry being executed
prev	DictEntry executed immediately before `word` (may be NULL)

References DICTIONARY_SIZE, ENABLE_HOTWORDS_CACHE, ENABLE_PIPELINING, HotwordsCache_s::enabled, DictEntry::execution_heat, VM::heartbeat, VM::hotwords_cache, hotwords_cache_promote(), HOTWORDS_EXECUTION_HEAT_TH

ssm_config_t::L4_pipelining, DictPhysics::last_active_ns, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, WordTransitionMetrics::most_likely_next_word_id, DictEntry::physics, physics_execution_heat_increment(), physics_metadata_apply_linear_decay(), VM::pipeline_metrics, WordTransitionMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::rolling_window, rolling_window_record_executio sf_monotonic_ns(), VM::ssm_config, HeartbeatState::tick_count, DictEntry::transition_metrics, transition_metrics_record(), transition_metrics_record_prefetch_hit(), transition_metrics_should_speculate(), transition_metrics_update_cache(), vm_dictionary_lookup_by_word_id(), word_id, and DictEntry::word_id.

Referenced by execute_colon_word().

Here is the call graph for this function:



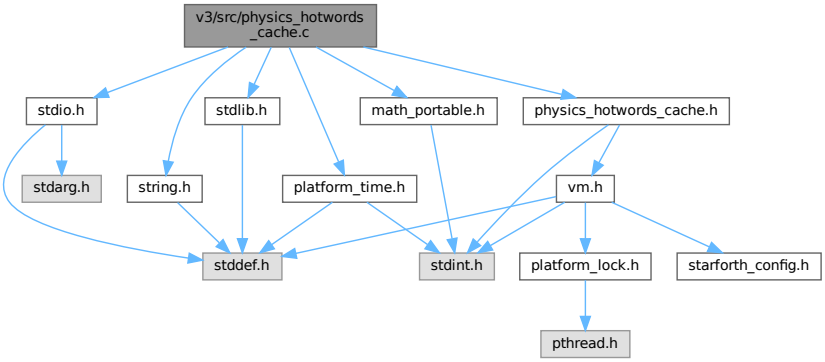
Here is the caller graph for this function:

7.292 v3/src/physics_hotwords_cache.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "physics_hotwords_cache.h"
#include "platform_time.h"
```

```
#include "math_portable.h"
```

Include dependency graph for physics_hotwords_cache.c:



Functions

- static double q48_to_ns_double (int64_t q48)

  Convert a Q48.16 fixed-point value to a `double` nanosecond count.
- void hotwords_cache_init (HotwordsCache *cache)

  Initialise a `HotwordsCache` to its empty, enabled state.
- void hotwords_cache_cleanup (HotwordsCache *cache)

  Zero all fields in a `HotwordsCache`, releasing no heap memory.
- DictEntry * hotwords_cache_lookup (VM *vm, HotwordsCache *cache, DictEntry **bucket, size_t bucket↩_count, const char *name, size_t len)

  Look up a FORTH word using the hot-words cache, falling back to the bucket.
- static void hotwords_cache_remove_at (HotwordsCache *cache, uint32_t i)
- void hotwords_cache_evict_entry (HotwordsCache *cache, const DictEntry *word)

  Evict every cache slot holding this exact entry.
- void hotwords_cache_evict_name (HotwordsCache *cache, const char *name, size_t len)

  Evict every cache slot whose entry matches this name.
- void hotwords_cache_promote (HotwordsCache *cache, DictEntry *word)

  Promote a `DictEntry` into the hot-words cache, evicting the oldest entry if full.
- void hotwords_bucket_reorder (DictEntry **bucket, size_t count)

  Sort a hash bucket in-place by descending execution heat (bubble sort).
- void hotwords_stats_reset (HotwordsStats *stats)

  Zero all fields in a `HotwordsStats` structure.
- void hotwords_stats_print (const HotwordsStats *stats)

  Print a formatted hot-words cache statistics report to `stdout`.
- void hotwords_cache_set_enabled (HotwordsCache *cache, int enabled)

  Enable or disable the hot-words cache at runtime.
- static BayesianLatencyPosterior calculate_posterior (int64_t total_ns_q48, int64_t variance_sum_q48, int64_t min_ns, int64_t max_ns, uint64_t sample_count)

  Compute a Bayesian latency posterior for one measurement distribution.
- BayesianLatencyPosterior hotwords_posterior_cache_hits (const HotwordsStats *stats)

  Return the Bayesian latency posterior for cache-hit observations.
- BayesianLatencyPosterior hotwords_posterior_bucket_searches (const HotwordsStats *stats)

  Return the Bayesian latency posterior for bucket-search observations.

- struct SpeedupEstimate hotwords_speedup_estimate (const HotwordsStats *stats)

  Compute a speedup estimate with credible intervals using the delta method.
- void hotwords_bayesian_report (const HotwordsStats *stats_with_cache, const HotwordsStats *stats_without_cache)

  Print a full Bayesian cache performance report comparing two experimental conditions.

Variables

- HotwordsCache g_hotwords_cache = {0}

7.292.1 Function Documentation

7.292.1.1 calculate_posterior()

```
static BayesianLatencyPosterior calculate_posterior (
            int64_t total_ns_q48,
            int64_t variance_sum_q48,
            int64_t min_ns,
            int64_t max_ns,
            uint64_t sample_count )  [static]
```

Compute a Bayesian latency posterior for one measurement distribution.

Given the accumulated Q48.16 statistics for a set of `sample_count` timing observations (either cache-hit or bucket-search latencies), computes the following posterior quantities entirely in Q48.16 integer arithmetic:

- **Mean** (`mean_ns_q48`): `total_ns_q48` / sample_count.
- **Variance** (`stddev_ns_q48`): $E[X^2]$? $E[X]^2$ where $E[X^2]$ is the per-sample average of `variance_sum_q48` and $E[X]^2$ = $mean^2$ scaled to Q48.16 via $>>$ 16. Variance is clamped to 0 to guard numerical rounding.
- **Standard deviation** (`stddev_ns_q48`): `sqrt_q48(variance)`.
- **Median** (`median_ns_q48`): (min_ns + max_ns) / 2 — a quick midpoint estimate rather than a true median, consistent across all call sites.
- **Standard error** (`se_q48`): `stddev` / $\sqrt{\cdot}$N, computed by converting `sample_count` to Q48.16 (`sample_count` $<<$ 16) and calling `sqrt_q48()`, then dividing `stddev` $<<$ 16 by the result.
- **95% credible interval**: `mean` $\pm$ Z?? $\times$ SE where `Z??` = 128431 (1.96 $\times$ 65536).
- **99% credible interval**: `mean` $\pm$ Z?? $\times$ SE where `Z??` = 168888 (2.576 $\times$ 65536). Lower credible bounds are clamped at 0 (latency is non-negative).

Returns a zeroed `BayesianLatencyPosterior` if `sample_count` is 0.

Parameters

total_ns_q48	Sum of all latency observations in Q48.16.
variance_sum_q48	Sum of squared latency observations in Q48.16 (i.e., Σ(x?^{2} $<<$ 16)).
min_ns	Minimum observed latency in integer nanoseconds.
max_ns	Maximum observed latency in integer nanoseconds.
sample_count	Number of observations; 0 $\rightarrow$ returns zeroed struct.

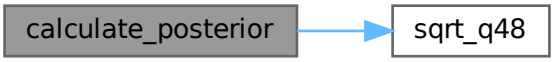
Returns

`BayesianLatencyPosterior` with all fields in Q48.16.

References BayesianLatencyPosterior::credible_lower_95, BayesianLatencyPosterior::credible_lower_99, BayesianLatencyPosterior:
BayesianLatencyPosterior::credible_upper_99, BayesianLatencyPosterior::mean_ns_q48, BayesianLatencyPosterior::median_ns_q4
BayesianLatencyPosterior::sample_count, sqrt_q48(), and BayesianLatencyPosterior::stddev_ns_q48.

Referenced by hotwords_posterior_bucket_searches(), and hotwords_posterior_cache_hits().

Here is the call graph for this function:



Here is the caller graph for this function:

7.292.1.2 hotwords_bayesian_report()

```
void hotwords_bayesian_report (
            const HotwordsStats * stats_with_cache,
            const HotwordsStats * stats_without_cache )
```

Print a full Bayesian cache performance report comparing two experimental conditions.

Generate detailed Bayesian inference report Compares cache vs bucket performance with confidence measures.

Emits a structured Unicode report to `stdout` comparing the latency distributions of two measurement sets — one collected with the hot-words cache enabled, the other collected without (baseline). The report is divided into five sections:

1. **WITH CACHE ENABLED** — posterior mean, stddev, and 95%/99% credible intervals for cache-hits and bucket-searches separately, via `hotwords_posterior_cache_hits()` and `hotwords_posterior_bucket_sea`
2. **WITHOUT CACHE (Baseline)** — posterior mean, stddev, and 95% credible interval for all lookups (treated as bucket searches). If no bucket data is available in `stats_without_cache`, the bucket-search mean from the with-cache run is used as an approximation.
3. **SPEEDUP ESTIMATE (Bucket vs Cache)** — `hotwords_speedup_estimate()` point estimate, 95% credible interval, P($>1.1\times$), and P($>2.0\times$).

4. **CACHE EFFECTIVENESS** — total lookups, cache-hit rate, bucket-hit rate, and miss rate as percentages of total.
5. **OVERALL PERFORMANCE** — weighted-average latency with cache vs. baseline all-bucket latency, overall speedup, and absolute time saved per lookup. The weighted average is: (N_cache×avg_cache + N_↩ bucket×avg_bucket) / (N_cache + N_bucket).

All latency values are displayed in nanoseconds (floating-point via `q48_to_ns_double()`) for readability; underlying accumulators are Q48.16 integer throughout.

No-op if either `stats_with_cache` or `stats_without_cache` is NULL.

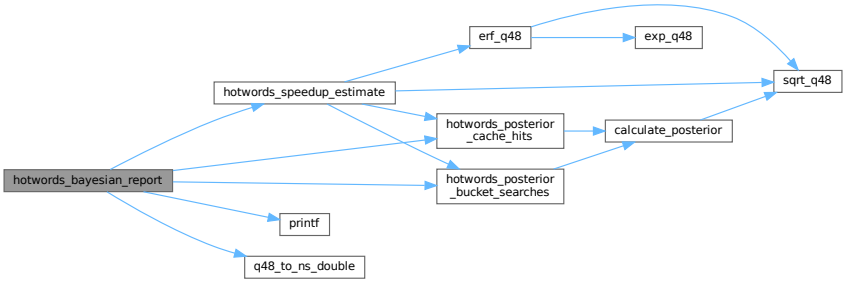
Parameters

stats_with_cache	Statistics from a run with cache enabled.
stats_without_cache	Statistics from a baseline run without cache.

References HotwordsStats::bucket_hits, HotwordsStats::cache_hits, BayesianLatencyPosterior::credible_lower_95, SpeedupEstimate::credible_lower_95_q48, BayesianLatencyPosterior::credible_lower_99, BayesianLatencyPosterior::credible_upper SpeedupEstimate::credible_upper_95_q48, BayesianLatencyPosterior::credible_upper_99, hotwords_posterior_bucket_searches(), hotwords_posterior_cache_hits(), hotwords_speedup_estimate(), BayesianLatencyPosterior::mean_ns_q48, HotwordsStats::misses, printf(), SpeedupEstimate::probability_gt_10pct_q48, SpeedupEstimate::probability_gt_double_q48, q48_to_ns_double(), BayesianLatencyPosterior::sample_count, SpeedupEstimate::speedup_factor_q48, and BayesianLatencyPosterior::stddev_ns_q48.

Referenced by forth_PHYSICS_BAYESIAN_REPORT().

Here is the call graph for this function:



Here is the caller graph for this function:

7.292.1.3 hotwords_bucket_reorder()

```
void hotwords_bucket_reorder (
            DictEntry ** bucket,
            size_t count )
```

Sort a hash bucket in-place by descending execution heat (bubble sort).

Reorder a bucket by execution heat (move hot words forward) Called when execution heat delta exceeds threshold.

Performs a classic in-place bubble sort over the `bucket` array so that words with higher `execution_heat` appear at lower array indices. The hot-words cache and heat-stratified lookup both benefit from having the hottest words at the front of each bucket: the first-character hash buckets are linear arrays, and placing hot words up front reduces the scan depth on Stage 1 cache misses that fall through to Stage 2 bucket search.

Bubble sort is chosen deliberately: buckets are small (typically single digits to low dozens of words per first-character class), already nearly sorted after the previous reorganization pass, and the cost of a $O(n^2)$ inner loop on $n \leq 30$ is negligible compared to the outer `qsort` on all 256 buckets in `dict_reorganize_buckets_by_heat()`. For the individual per-bucket call path (e.g. after a single promotion), bubble sort avoids the `malloc` overhead that `qsort's` temporary storage would require.

No-op for `count` < 2.

Parameters

bucket	Array of `DictEntry*` pointers to sort in-place. NULL-check is the caller's responsibility.
count	Number of valid entries in `bucket`.

References count, and execution_heat.

7.292.1.4 hotwords_cache_cleanup()

```
void hotwords_cache_cleanup (
            HotwordsCache * cache )
```

Zero all fields in a `HotwordsCache`, releasing no heap memory.

Cleanup hot-words cache.

Calls `memset(cache, 0, sizeof(*cache))` to scrub every field including the inline `cache[]` pointer array and all statistics. The cache holds no separately allocated memory (the `DictEntry` pointers inside `cache[]` are owned by the dictionary, not the cache), so no `free()` calls are needed.

Intended for use at VM teardown or when the caller wants to reset the cache to a known-zero state between benchmark runs. After this call the cache is in an unusable state (`enabled` = 0); call `hotwords_cache_init()` to reinitialise it.

Parameters

cache	Cache to zero; no-op if NULL.

References memset().

Referenced by vm_cleanup().

Here is the call graph for this function:

Here is the caller graph for this function:

7.292.1.5 hotwords_cache_evict_entry()

```
void hotwords_cache_evict_entry (
            HotwordsCache * cache,
            const DictEntry * word )
```

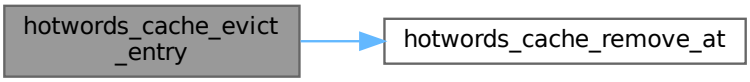
Evict every cache slot holding this exact entry.

Must be called before an entry is freed (FORGET) — a stale pointer in the ring is a use-after-free.

References HotwordsCache_s::cache, HotwordsCache_s::cache_count, and hotwords_cache_remove_at().

Referenced by vm_dictionary_untrack_entry().

Here is the call graph for this function:



Here is the caller graph for this function:

7.292.1.6 hotwords_cache_evict_name()

```
void hotwords_cache_evict_name (
            HotwordsCache * cache,
            const char * name,
            size_t len )
```

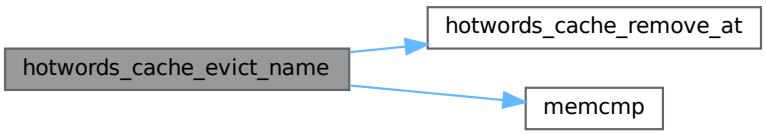
Evict every cache slot whose entry matches this name.

Called whenever name resolution may change: word creation (shadowing), smudge/unsmudge, hide. The next lookup re-resolves through the bucket.

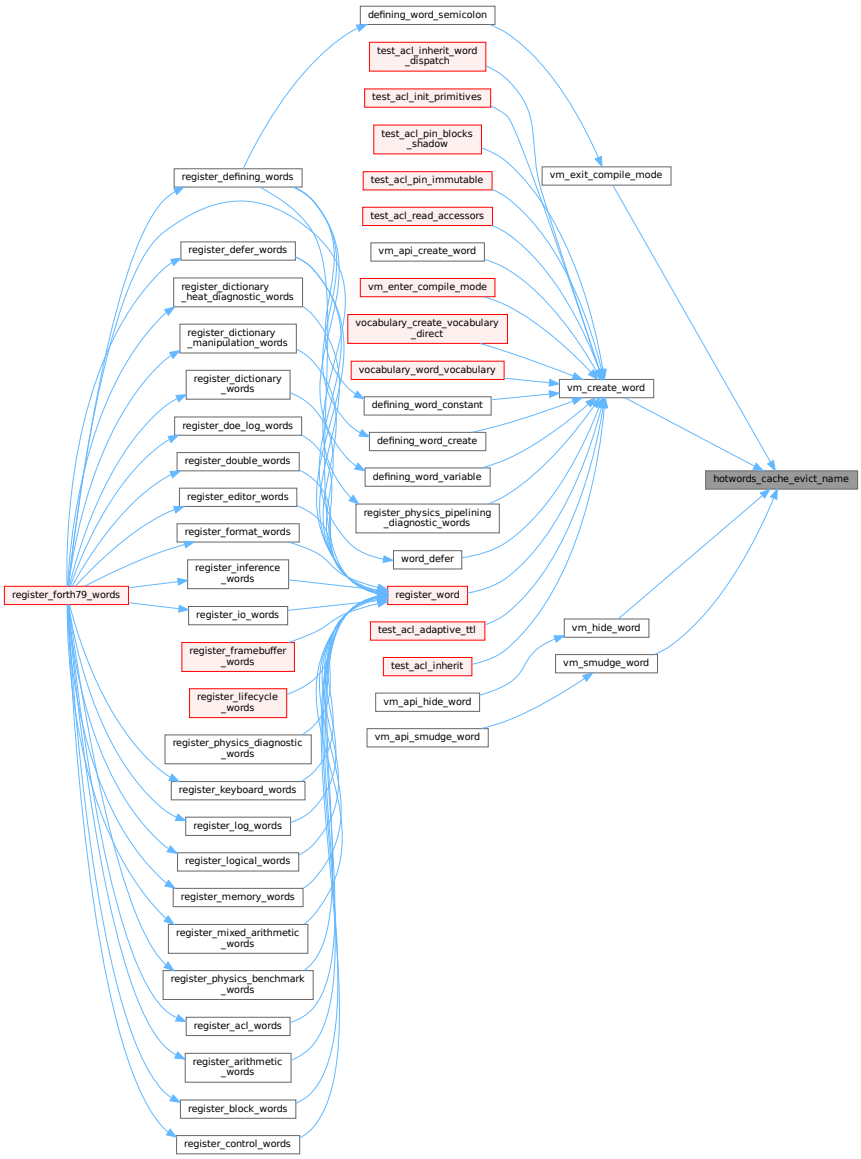
References HotwordsCache_s::cache, HotwordsCache_s::cache_count, hotwords_cache_remove_at(), memcmp(), name, DictEntry::name, and DictEntry::name_len.

Referenced by vm_create_word(), vm_exit_compile_mode(), vm_hide_word(), and vm_smudge_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.292.1.7 hotwords_cache_init()

```
void hotwords_cache_init (
            HotwordsCache * cache )
```

Initialise a `HotwordsCache` to its empty, enabled state.

Initialize hot-words cache Should be called during VM initialization.

Zeroes all fields via `memset` then explicitly sets:

- `cache->enabled` = `ENABLE_HOTWORDS_CACHE` (build-time compile flag; 0 disables the cache globally without a code change).

- `cache->cache_count` = 0, `cache->lru_index` = 0 — empty round-robin eviction ring.
- All counter fields in `cache->stats` to 0.
- `stats.min_cache_hit_ns` = `INT64_MAX`, `stats.max_cache_hit_ns` = `INT64_MIN` (sentinel values; updated to real observations on the first cache hit).
- `stats.min_bucket_search_ns` = `INT64_MAX`, `stats.max_bucket_search_ns` = `INT64_MIN` (same sentinel convention for bucket searches).
- All Q48.16 variance accumulators to 0.

Safe to call multiple times (idempotent via `memset`). Must be called before the first `hotwords_cache_lookup()` invocation; failure to do so leaves the sentinel min/max fields uninitialised, producing nonsense in the Bayesian posterior report.

Parameters

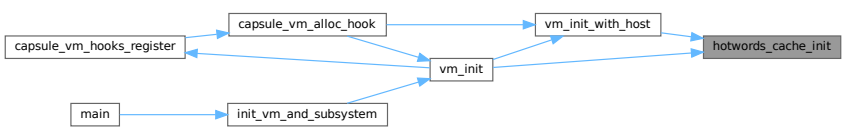
cache	Cache to initialise; no-op if NULL.

References HotwordsStats::bucket_hits, HotwordsStats::bucket_reorders, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsCache_s::cache_count, HotwordsStats::cache_hit_samples, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, HotwordsStats::cache_hits, ENABLE_HOTWORDS_CACHE, HotwordsCache_s::enabled, HotwordsStats::evictions, HotwordsCache_s::lru_index, HotwordsStats::max_bucket_search_ns, HotwordsStats::max_cache_hit_ns, memset(), HotwordsStats::min_bucket_search_ns, HotwordsStats::min_cache_hit_ns, HotwordsStats::misses, HotwordsStats::promotions, HotwordsCache_s::stats, and HotwordsStats::total_lookups.

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:

Here is the caller graph for this function:



7.292.1.8 hotwords_cache_lookup()

```
DictEntry * hotwords_cache_lookup (
            VM * vm,
            HotwordsCache * cache,
            DictEntry ** bucket,
            size_t bucket_count,
            const char * name,
            size_t len )
```

Look up a FORTH word using the hot-words cache, falling back to the bucket.

Lookup word in cache first, then fall back to bucket search Returns the found word, or NULL if not found.

The primary dictionary fast-path. Records a start timestamp via `sf_monotonic_ns()` and then executes a two-stage search:

Stage 1 — Cache check (O(HOTWORDS_CACHE_SIZE)): Scans the `cache->cache[]` ring for an entry whose `name_len` and name match `name/len`, using the cheap-first filter (`name_len` → last-character → `memcmp`). On a hit the elapsed nanoseconds are converted to Q48.16 and accumulated into `stats.cache_hit_total_ns_q48`; the squared value is accumulated into `stats.cache_hit_variance_sum_q48` for Bayesian posterior computation by `calculate_posterior()`. Min/max latency sentinels are also updated.

Stage 2 — Bucket fallback (O(bucket_count)): Scans `bucket[]` in reverse-insertion order (newest definition wins). On a hit the elapsed time is recorded in the `bucket_search_*` accumulators with the same Q48.16 and variance treatment as Stage 1. If the found word's `execution_heat` exceeds `HOTWORDS_EXECUTION_HEAT_THRESHOLD` it is promoted into the cache via `hotwords_cache_promote()`.

Disabled path: If `cache` is NULL or `cache->enabled` is 0, the function skips Stage 1 and performs a bare reverse-linear scan of `bucket[]` returning the result immediately (no statistics are recorded).

A miss increments `stats.misses` and returns NULL.

Parameters

cache	Hot-words cache; may be NULL or disabled (falls back to bucket scan without statistics).
bucket	First-character hash bucket — array of `DictEntry*` pointers, searched in reverse order.
bucket_count	Number of valid pointers in `bucket`.
name	Word name to find (not NUL-terminated).
len	Length of `name` in bytes.

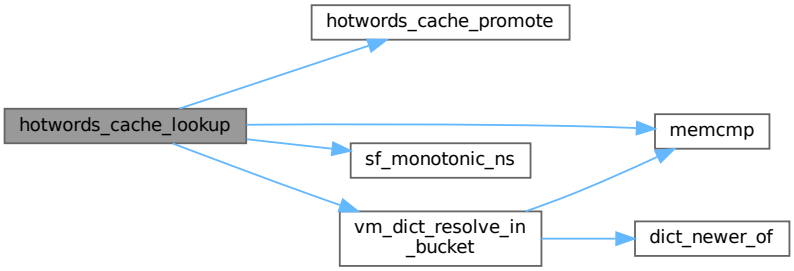
Returns

Matching `DictEntry*`, or NULL if not found.

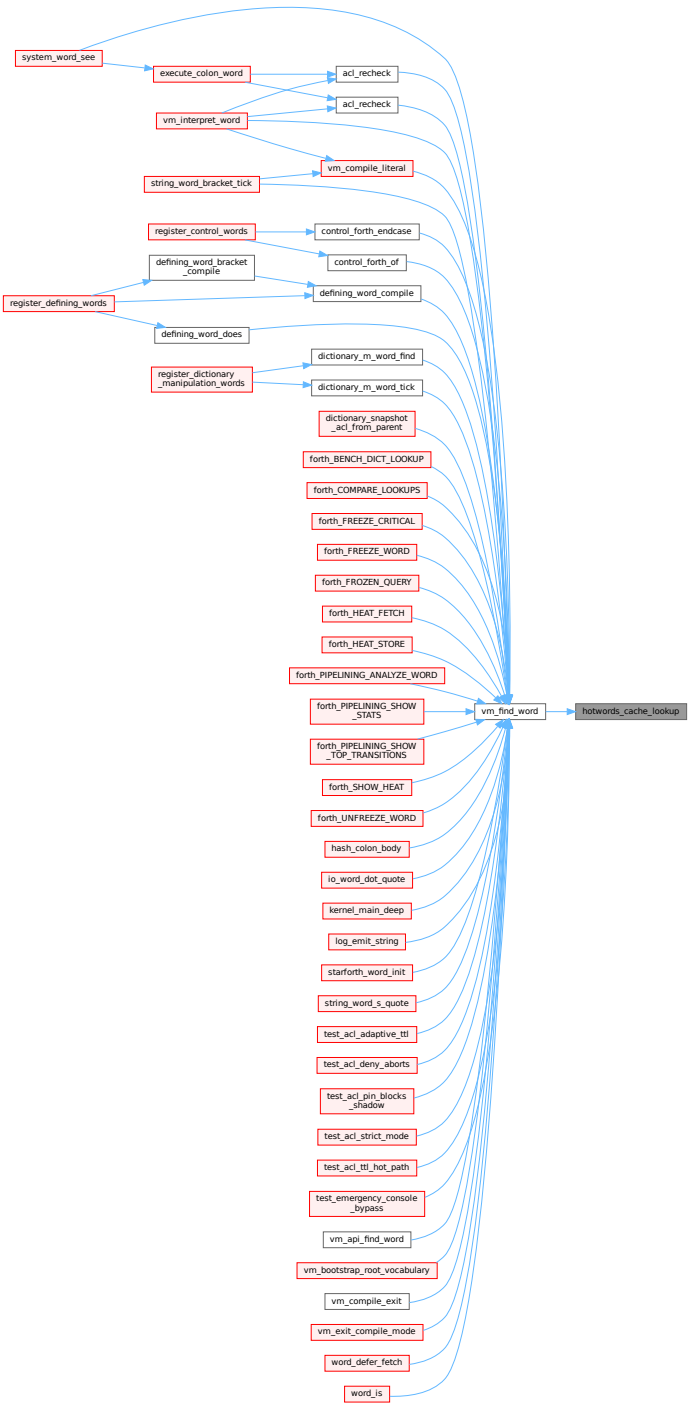
References HotwordsStats::bucket_hits, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsCache_s::cache, HotwordsCache_s::cache_count, HotwordsStats::cache_hit_samples, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, HotwordsStats::cache_hits, HotwordsCache_s::enabled, DictEntry::execution_heat, DictEntry::flags, hotwords_cache_promote(), HOTWORDS_EXECUTION_HEAT_THRESHOLD, HotwordsStats::max_bucket_search_ns, HotwordsStats::max_cache_hit_ns, memcmp(), HotwordsStats::min_bucket_search_ns, HotwordsStats::min_cache_hit_ns, HotwordsStats::misses, name, DictEntry::name, DictEntry::name_len, sf_monotonic_ns(), HotwordsCache_s::stats, HotwordsStats::total_lookups, vm_dict_resolve_in_bucket(), WORD_HIDDEN, and WORD_SMUDGED.

Referenced by vm_find_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.292.1.9 hotwords_cache_promote()

```
void hotwords_cache_promote (
            HotwordsCache * cache,
            DictEntry * word )
```

Promote a DictEntry into the hot-words cache, evicting the oldest entry if full.

Promote a word to the hot-words cache Called when a word's execution heat exceeds threshold.

Two paths depending on whether the cache has a free slot:

- **Slot available** (cache->cache_count < HOTWORDS_CACHE_SIZE): Appends word at cache->cache[cache↩ _count++] after verifying it is not already present (linear scan). Increments stats.promotions.
- **Cache full** (cache->cache_count >= HOTWORDS_CACHE_SIZE): Overwrites cache->cache[cache->lru_index] with word (round-robin least-recently-used eviction) and advances lru_index modulo HOTWORDS_CACHE_SIZE. Increments stats.evictions. The evicted entry is not freed — it remains alive in the dictionary; only its slot in the cache ring is surrendered.

Called from hotwords_cache_lookup() whenever a bucket search finds a word hot enough to warrant caching (heat > HOTWORDS_EXECUTION_HEAT_THRESHOLD).

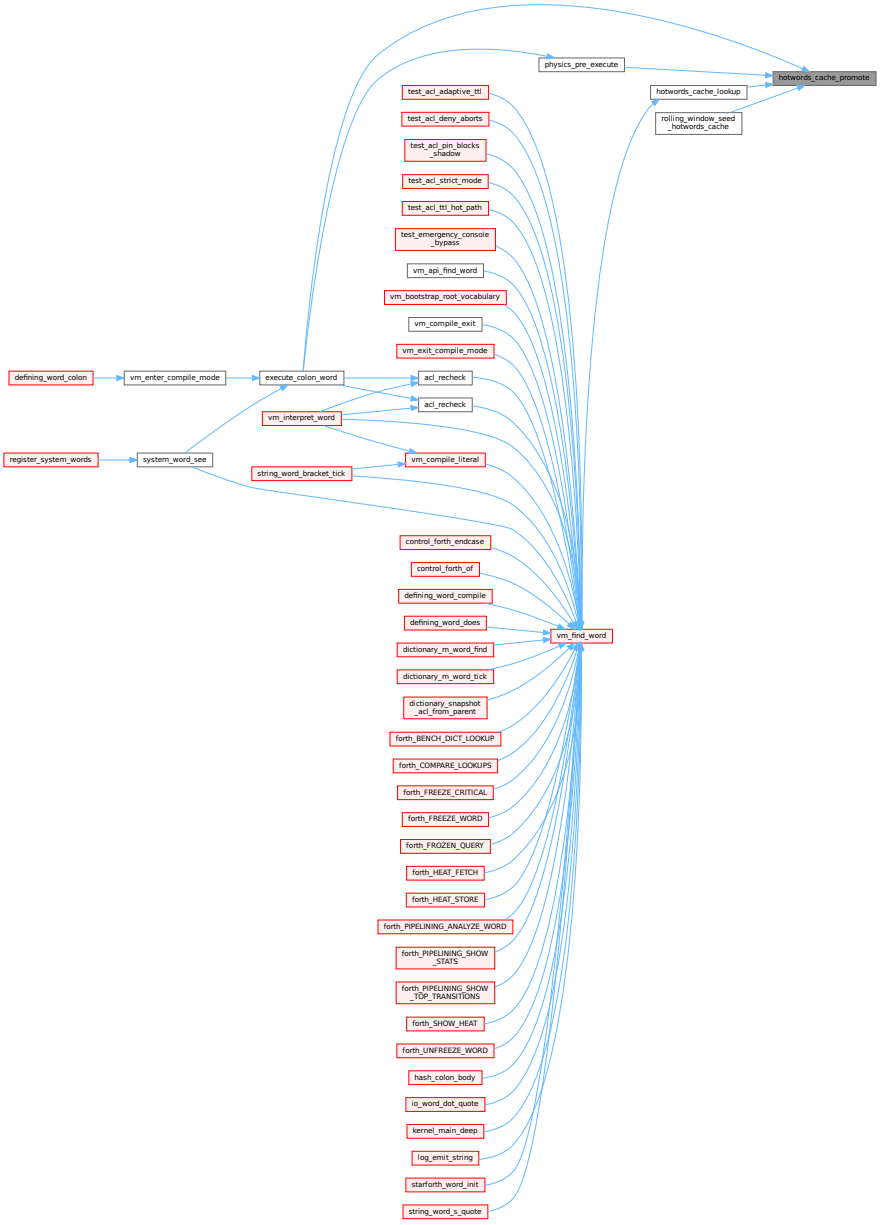
Parameters

cache	The cache to promote into; no-op if NULL.
word	The DictEntry to promote; no-op if NULL. Ownership is not transferred — the cache holds a borrowed pointer into the dictionary, which owns the entry.

References HotwordsCache_s::cache, HotwordsCache_s::cache_count, HotwordsStats::evictions, HOTWORDS_CACHE_SIZE, HotwordsCache_s::lru_index, HotwordsStats::promotions, and HotwordsCache_s::stats.

Referenced by execute_colon_word(), hotwords_cache_lookup(), physics_pre_execute(), and rolling_window_seed_hotwords_cache(

Here is the caller graph for this function:



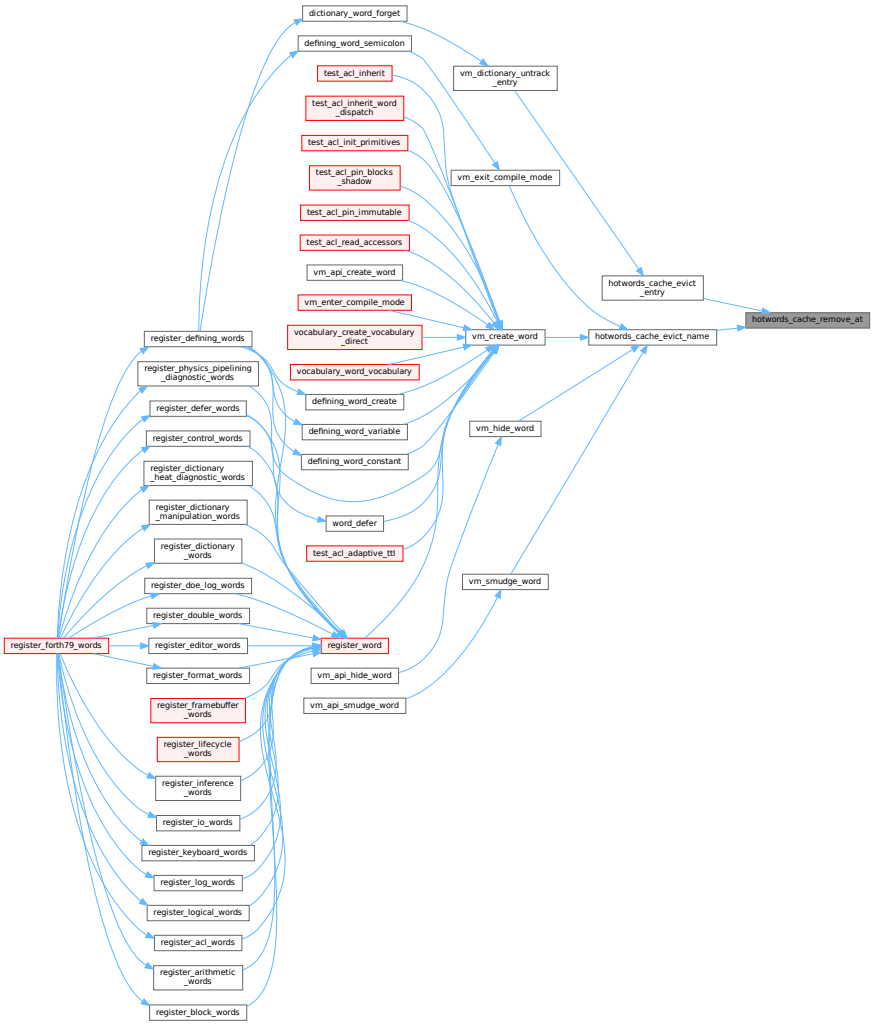
7.292.1.10 hotwords_cache_remove_at()

```
static void hotwords_cache_remove_at (
            HotwordsCache * cache,
            uint32_t i )  [static]
```

References HotwordsCache_s::cache, HotwordsCache_s::cache_count, and HotwordsCache_s::lru_index.

Referenced by hotwords_cache_evict_entry(), and hotwords_cache_evict_name().

Here is the caller graph for this function:



7.292.1.11 hotwords_cache_set_enabled()

```
void hotwords_cache_set_enabled (
            HotwordsCache * cache,
            int enabled )
```

Enable or disable the hot-words cache at runtime.

Enable/disable cache at runtime (for A/B testing)

Sets `cache->enabled` to `enabled` (non-zero = active, 0 = bypassed). When disabled, `hotwords_cache_lookup()` skips Stage 1 entirely and falls through directly to a bare bucket scan, recording no statistics. This allows benchmark harnesses to compare cache-on vs. cache-off behaviour without rebuilding the binary.

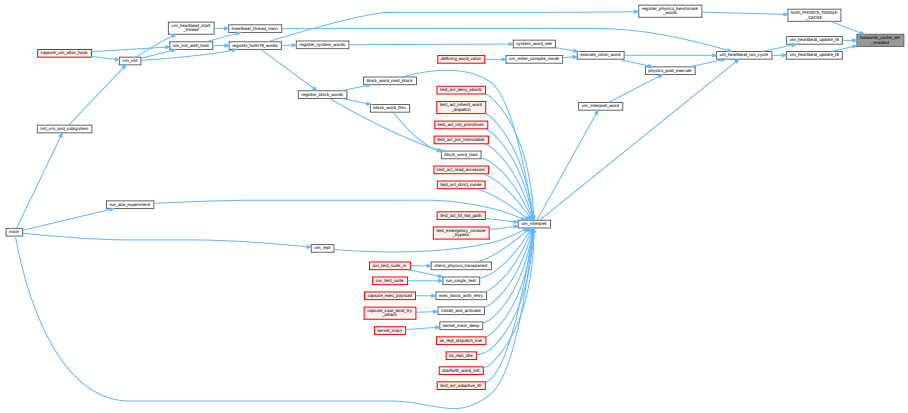
Parameters

cache	Cache to configure; no-op if NULL.
enabled	Non-zero to enable the cache; 0 to disable.

References HotwordsCache_s::enabled, and enabled.

Referenced by forth_PHYSICS_TOGGLE_CACHE(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the caller graph for this function:



7.292.1.12 hotwords_posterior_bucket_searches()

```
BayesianLatencyPosterior hotwords_posterior_bucket_searches (
            const HotwordsStats * stats )
```

Return the Bayesian latency posterior for bucket-search observations.

Calculate posterior distribution for bucket searches Uses accumulated samples to compute credible intervals.

Forwards the bucket-search accumulators from `stats` to `calculate_posterior()`: `bucket_search_total_ns_q48`, `bucket_search_variance_sum_q48`, `min_bucket_search_ns`, `max_bucket_search_ns`, and `bucket_search_samples`. Returns a zeroed `BayesianLatencyPosterior` if `stats` is NULL or no bucket-search samples have been recorded.

The returned posterior's mean and credible-interval fields feed `hotwords_speedup_estimate()` (as the slower distribution) and the baseline section of `hotwords_bayesian_report()`.

Parameters

stats	`HotwordsStats` containing bucket-search timing accumulators.

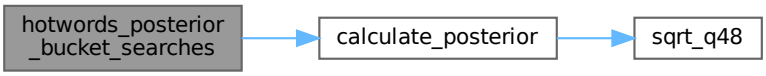
Returns

Posterior in Q48.16; all fields zero if no samples exist.

References HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_v
calculate_posterior(), HotwordsStats::max_bucket_search_ns, and HotwordsStats::min_bucket_search_ns.

Referenced by bucket_posterior(), hotwords_bayesian_report(), and hotwords_speedup_estimate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.292.1.13 hotwords_posterior_cache_hits()

```
BayesianLatencyPosterior hotwords_posterior_cache_hits (
            const HotwordsStats * stats )
```

Return the Bayesian latency posterior for cache-hit observations.

Calculate posterior distribution for cache hits Uses accumulated samples to compute credible intervals.

Forwards the cache-hit accumulators from `stats` to `calculate_posterior()`: `cache_hit_total_ns_q48`, `cache_hit_variance_sum_q48`, `min_cache_hit_ns`, `max_cache_hit_ns`, and `cache_hit_samples`. Returns a zeroed `BayesianLatencyPosterior` if `stats` is NULL or no cache-hit samples have been recorded yet.

The returned posterior's `mean_ns_q48`, `stddev_ns_q48`, and 95%/99% credible-interval fields are consumed by `hotwords_speedup_estimate()` and `hotwords_bayesian_report()` to compare cache-hit latency against bucket-search latency.

Parameters

stats	`HotwordsStats` containing cache-hit timing accumulators.

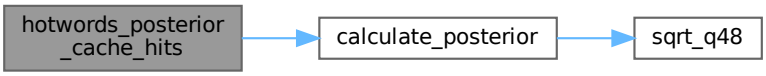
Returns

Posterior in Q48.16; all fields zero if no samples exist.

References HotwordsStats::cache_hit_samples, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, calculate_posterior(), HotwordsStats::max_cache_hit_ns, and HotwordsStats::min_cache_hit_ns.

Referenced by cache_posterior(), hotwords_bayesian_report(), and hotwords_speedup_estimate().

Here is the call graph for this function:



Here is the caller graph for this function:

7.292.1.14 hotwords_speedup_estimate()

```
struct SpeedupEstimate hotwords_speedup_estimate (
            const HotwordsStats * stats )
```

Compute a speedup estimate with credible intervals using the delta method.

Calculate speedup estimate with uncertainty.

Derives a dimensionless speedup ratio S = bucket_mean / cache_mean in Q48.16 fixed-point and propagates uncertainty via the delta method for ratios:
SE(S) ≈ S × sqrt((SE_bucket / bucket_mean)^2 + (SE_cache / cache_mean)^2)

All arithmetic is Q48.16 integer throughout; the only deviation is the intermediate `sqrt()` call which also runs in Q48.16 via `sqrt_q48()`.

Computed fields in the returned `SpeedupEstimate` (all Q48.16):

- `speedup_factor_q48` — point estimate of S.
- `credible_lower_95_q48` / `credible_upper_95_q48` — 95% credible interval: S ± 1.96 × SE(S) (Z?? = 128431 in Q48.16). Lower bound clamped at 0.
- `probability_gt_10pct_q48` — P(speedup > 1.1×), computed as `0.5` × (1 + erf(z / √2)) where z = (1.1 ? S) / SE(S) is negated to give the right-tail probability and √2 = 92681 in Q48.16.
- `probability_gt_double_q48` — P(speedup > 2.0×), same formula with target = 2.0 × 65536 = 131072.

Returns a default `SpeedupEstimate` with `speedup_factor_q48` = 65536 (1.0×) if either distribution has zero samples or if `stats` is NULL.

Parameters

stats	`HotwordsStats` with both cache-hit and bucket-search samples.

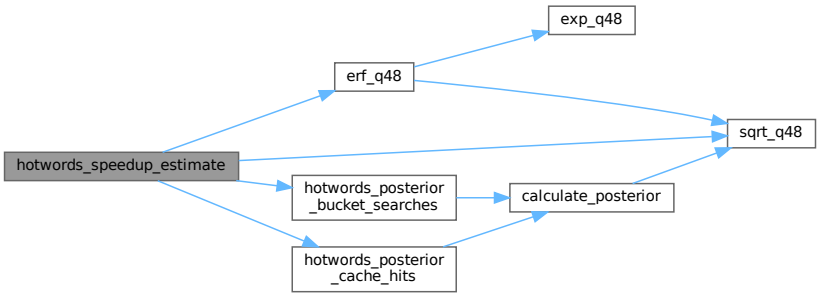
Returns

`SpeedupEstimate` with all fields in Q48.16 fixed-point.

References SpeedupEstimate::credible_lower_95_q48, SpeedupEstimate::credible_upper_95_q48, erf_q48(), hotwords_posterior_bucket_searches(), hotwords_posterior_cache_hits(), BayesianLatencyPosterior::mean_ns_q48, SpeedupEstimate::probability_gt_10pct_q48, SpeedupEstimate::probability_gt_double_q48, SpeedupEstimate::speedup_factor_q48, sqrt_q48(), and BayesianLatencyPosterior::stddev_ns_q48.

Referenced by hotwords_bayesian_report().

Here is the call graph for this function:



Here is the caller graph for this function:

7.292.1.15 hotwords_stats_print()

```
void hotwords_stats_print (
            const HotwordsStats * stats )
```

Print a formatted hot-words cache statistics report to `stdout`.

Print statistics to stdout.

Emits a Unicode box-drawing table covering three sections:

LOOKUPS — total, cache hits, bucket hits, misses with percentages.

LATENCY STATISTICS — for each of cache-hits and bucket-searches (when their sample counts are non-zero):

- Min / avg / max in nanoseconds, derived from the Q48.16 accumulators by right-shifting 16 bits: `avg_ns` = `total_ns_q48` / samples >> 16.
- Standard deviation computed from the Welford-style variance accumulator: `var` = $E[X^2]$? $E[X]^2$, then `stddev` = sqrt_q48(var), converted to nanoseconds via `q48_to_ns_double()` for display only.
- Speedup ratio (bucket avg / cache avg) when both sample sets are present.

CACHE MANAGEMENT — promotions, evictions, and bucket reorder counts.

If `stats` is NULL or the total lookup count is zero, prints a brief "No lookups performed." notice instead of the full table. All floating-point usage in this function is display-only (via `q48_to_ns_double()`); the underlying accumulators remain in Q48.16 integer format.

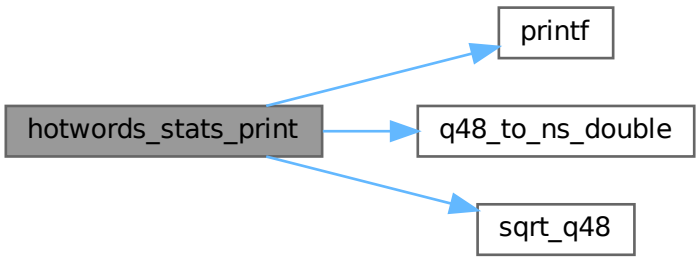
Parameters

stats	Statistics to report; no-op if NULL.

References HotwordsStats::bucket_hits, HotwordsStats::bucket_reorders, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsStats::cache_hit_samples, HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, HotwordsStats::cache_hits, HotwordsStats::evictions, HotwordsStats::max_bucket_search_ns, HotwordsStats::max_cache_hit_ns, HotwordsStats::min_bucket_s, HotwordsStats::min_cache_hit_ns, HotwordsStats::misses, printf(), HotwordsStats::promotions, q48_to_ns_double(), and sqrt_q48().

Referenced by forth_BENCH_DICT_LOOKUP(), and forth_PHYSICS_CACHE_STATS().

Here is the call graph for this function:



Here is the caller graph for this function:

7.292.1.16 hotwords_stats_reset()

```
void hotwords_stats_reset (
            HotwordsStats * stats )
```

Zero all fields in a `HotwordsStats` structure.

Reset statistics (for before/after comparison)

Resets every counter (`total_lookups`, `cache_hits`, `bucket_hits`, `misses`, `evictions`, `promotions`, `bucket_reorders`) to zero, resets all Q48.16 accumulator fields (`cache_hit_total_ns_q48`, `bucket_search_total_ns_q48`, `cache_hit_variance_sum_q48`, `bucket_search_variance_sum_q48`) to zero, resets sample counts to zero, and restores the min/max latency sentinels:

- `min_*_ns` = `INT64_MAX`
- `max_*_ns` = `INT64_MIN`

Intended for DoE benchmark runs that need a clean statistical baseline between treatment cells. The parent `HotwordsCache` also holds the `cache`[] pointer array and LRU state; those are not touched — only the `HotwordsStats` sub-struct is cleared.

Parameters

stats	Statistics structure to reset; no-op if NULL.

References HotwordsStats::bucket_hits, HotwordsStats::bucket_reorders, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsStats::cache_hit_samples HotwordsStats::cache_hit_total_ns_q48, HotwordsStats::cache_hit_variance_sum_q48, HotwordsStats::cache_hits, HotwordsStats::evictions, HotwordsStats::max_bucket_search_ns, HotwordsStats::max_cache_hit_ns, HotwordsStats::min_bucket_s HotwordsStats::min_cache_hit_ns, HotwordsStats::misses, HotwordsStats::promotions, and HotwordsStats::total_lookups.

Referenced by forth_BENCH_DICT_LOOKUP(), and forth_PHYSICS_RESET_STATS().

Here is the caller graph for this function:

7.292.1.17 q48_to_ns_double()

```
static double q48_to_ns_double (
            int64_t q48 )  [inline], [static]
```

Convert a Q48.16 fixed-point value to a `double` nanosecond count.

Divides `q48` by 65536.0 to recover the real-valued nanosecond count. This is the only floating-point operation in the module and is intentionally confined to display paths — `hotwords_stats_print()` and `hotwords_bayesian_report()` — so that all internal accumulation and statistical computation remains in Q48.16 integer arithmetic.

Parameters

q48	Value in Q48.16 format (1.0 ns = 65536).

Returns

The same quantity as a `double` in nanoseconds.

Referenced by hotwords_bayesian_report(), and hotwords_stats_print().

Here is the caller graph for this function:

7.292.2 Variable Documentation

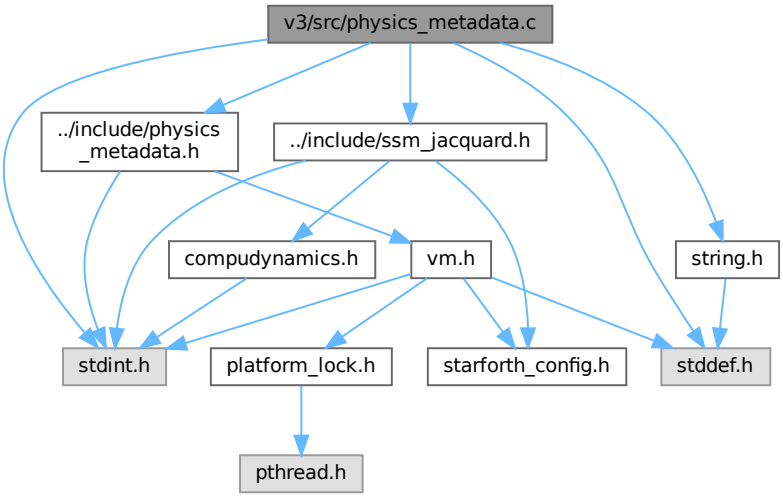
7.292.2.1 g_hotwords_cache

```
HotwordsCache g_hotwords_cache = {0}
```

7.293 v3/src/physics_metadata.c File Reference

```
#include "../include/physics_metadata.h"
#include "../include/ssm_jacquard.h"
#include <stddef.h>
#include <stdint.h>
#include <string.h>
```

Include dependency graph for physics_metadata.c:



Data Structures

- struct physics_seed_definition

Typedefs

- typedef struct physics_seed_definition physics_seed_definition_t

Functions

- static uint32_t clamp_u32 (uint64_t value)

  Clamp a uint64_t value to the uint32_t range.
- static uint16_t temperature_from_execution_heat (cell_t execution_heat, uint16_t prior_q8)

  Derive a smoothed Q8 temperature from raw execution heat.
- static uint8_t derive_state_flags (const DictEntry *entry)

  Derive the physics state flags bitmask from a DictEntry's word flags.
- void physics_metadata_init (DictEntry *entry, uint32_t header_bytes)

  Initialise physics metadata for a newly registered DictEntry.
- void physics_metadata_set_mass (DictEntry *entry, uint32_t total_bytes)

  Update the mass (byte footprint) of a DictEntry's physics metadata.
- void physics_metadata_touch (DictEntry *entry, cell_t execution_heat, uint64_t now_ns)

  Update physics metadata after a word execution completes.
- void physics_metadata_refresh_state (DictEntry *entry)

  Re-derive and update `state_flags` for a DictEntry.
- void physics_metadata_record_latency (DictEntry *entry, uint64_t sample_ns)

  Record an execution latency sample into the per-word EMA.
- void physics_metadata_apply_seed (DictEntry *entry)

  Apply DoE-derived prior physics values to a named DictEntry.
- void physics_metadata_apply_linear_decay (DictEntry *entry, uint64_t elapsed_ticks, VM *vm)

  Apply Loop #3 linear heat decay to a DictEntry.

Variables

- static const physics_seed_definition_t physics_seed_table [ ]

7.293.1 Typedef Documentation

7.293.1.1 physics_seed_definition_t

```
typedef struct physics_seed_definition physics_seed_definition_t
```

7.293.2 Function Documentation

7.293.2.1 clamp_u32()

```
static uint32_t clamp_u32 (
            uint64_t value )  [static]
```

Clamp a uint64_t value to the uint32_t range.

Parameters

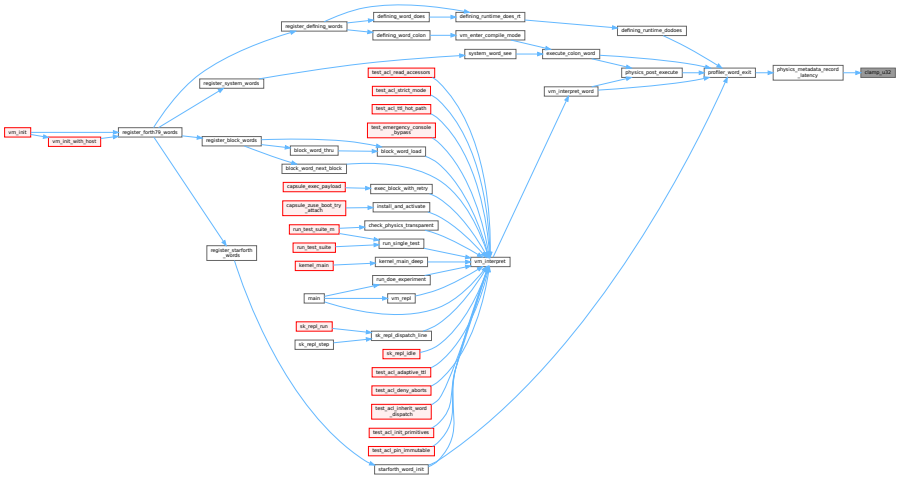
value	Input value to clamp

Returns

UINT32_MAX if value > UINT32_MAX, else (uint32_t)value

Referenced by physics_metadata_record_latency().

Here is the caller graph for this function:



7.293.2.2 derive_state_flags()

```
static uint8_t derive_state_flags (
            const DictEntry * entry )  [static]
```

Derive the physics state flags bitmask from a DictEntry's word flags.

Maps WORD_IMMEDIATE → PHYSICS_STATE_IMMEDIATE, WORD_PINNED → PHYSICS_STATE_PINNED, WORD_HIDDEN → PHYSICS_STATE_HIDDEN, and WORD_COMPILED → PHYSICS_STATE_COMPILED.

Parameters

entry	DictEntry to inspect (NULL returns 0)

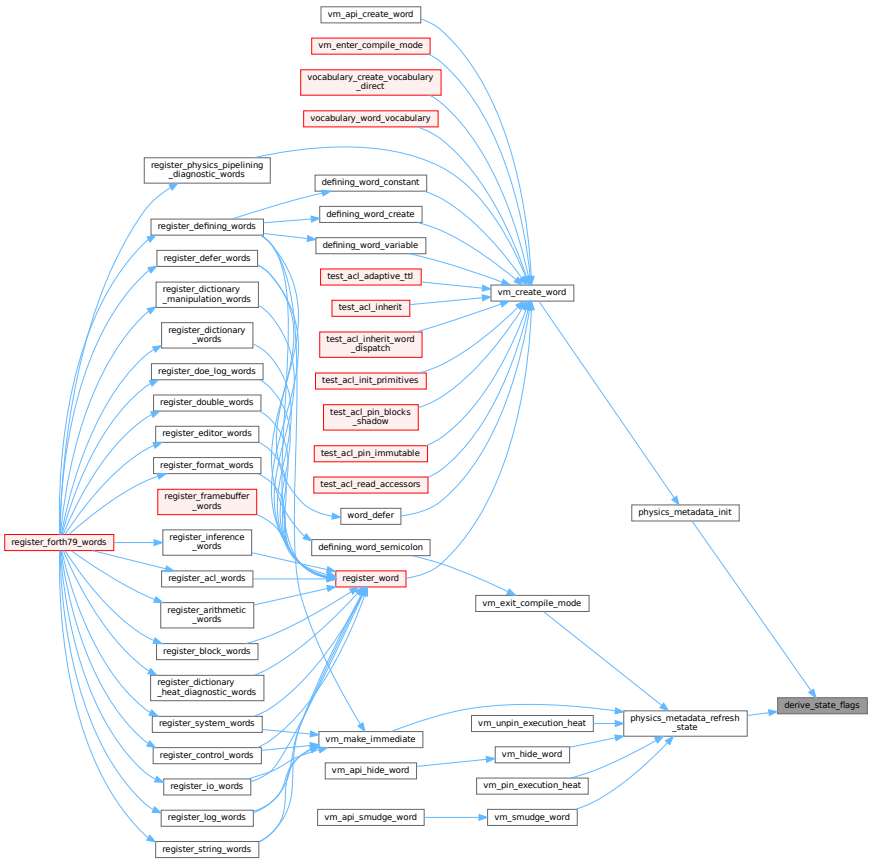
Returns

Bitmask of PHYSICS_STATE_* flags

References entry, flags, PHYSICS_STATE_COMPILED, PHYSICS_STATE_HIDDEN, PHYSICS_STATE_IMMEDIATE, PHYSICS_STATE_PINNED, WORD_COMPILED, WORD_HIDDEN, WORD_IMMEDIATE, and WORD_PINNED.

Referenced by physics_metadata_init(), and physics_metadata_refresh_state().

Here is the caller graph for this function:



7.293.2.3 physics_metadata_apply_linear_decay()

```
void physics_metadata_apply_linear_decay (
            DictEntry * entry,
            uint64_t elapsed_ticks,
            VM * vm )
```

Apply Loop #3 linear heat decay to a DictEntry.

Implements the thermodynamic dissipation model: H(t) = max(0, H? - d∗t), where t is measured in heartbeat ticks (`vm->heartbeat.tick_count`), never in wall-clock time. The decay amount is computed as (elapsed_ticks ∗ slope_q48) >> 16 using the Q48.16 adaptive decay slope stored in the VM's heartbeat state. Uses a GCC CAS loop for atomic heat update when __GNUC__ is defined, falling back to a plain subtraction otherwise.

Ticks, not nanoseconds: this entry's `execution_heat` feeds directly into `dict_hash` (see `capsule_dict_hash_hook()`), so any wall-clock dependency here makes dictionary state – and therefore dict_hash – architecture-variable under QEMU/TCG's differing real emulation speed, or whenever real elapsed time between two touches of the same entry varies for reasons unrelated to execution (e.g. a blocking I/O wait between them). `vm->heartbeat.tick_count` only advances from `vm_tick()`, called synchronously from the interpreter's own dispatch path – a purely execution-count-driven clock, immune to how long any intervening real-time work (including blocking I/O) actually took.

No-op when:

- `WORD_FROZEN` flag is set on the entry
- `elapsed_ticks` == 0 (no heartbeat tick has fired since last decay)
- `vm->ssm_config` has L3 disabled by the Jacquard mode selector

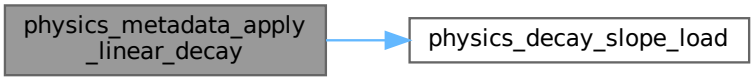
Parameters

entry	DictEntry whose heat is decayed
elapsed_ticks	Heartbeat ticks elapsed since last decay application
vm	VM providing the adaptive decay slope and L3 gate flag

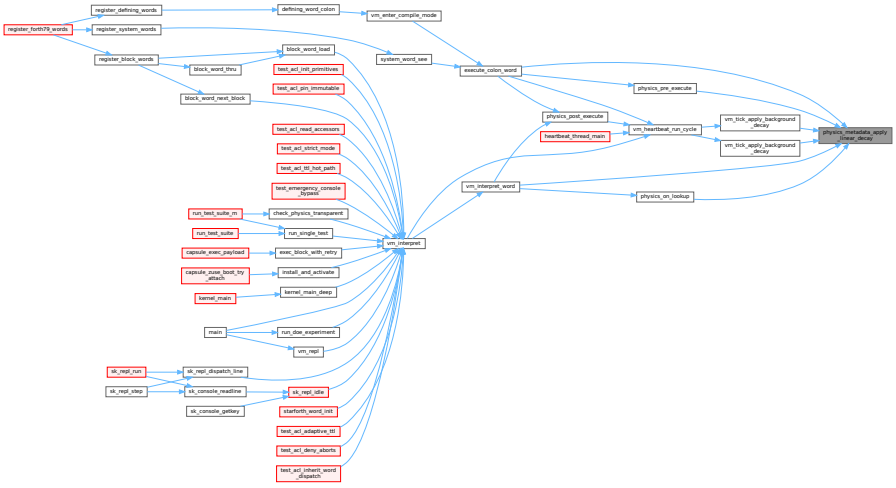
References entry, ssm_config_t::L3_linear_decay, physics_decay_slope_load(), VM::ssm_config, and WORD_FROZEN.

Referenced by execute_colon_word(), physics_on_lookup(), physics_pre_execute(), vm_interpret_word(), vm_tick_apply_background_decay(), and vm_tick_apply_background_decay().

Here is the call graph for this function:



Here is the caller graph for this function:



7.293.2.4 physics_metadata_apply_seed()

```
void physics_metadata_apply_seed (
            DictEntry * entry )
```

Apply DoE-derived prior physics values to a named DictEntry.

Searches the static `physics_seed_table` for an entry matching `entry->name`. If found, seeds `temperature_q8`, `avg_latency_ns`, `acl_hint`, and `pubsub_mask` with values derived from the 2^7 factorial DoE analysis. Only non-zero seed fields overwrite the current value. Called from `register_word()` for built-in primitives.

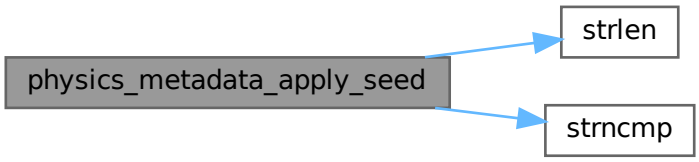
Parameters

entry	DictEntry to seed (no-op if NULL or no matching table entry)

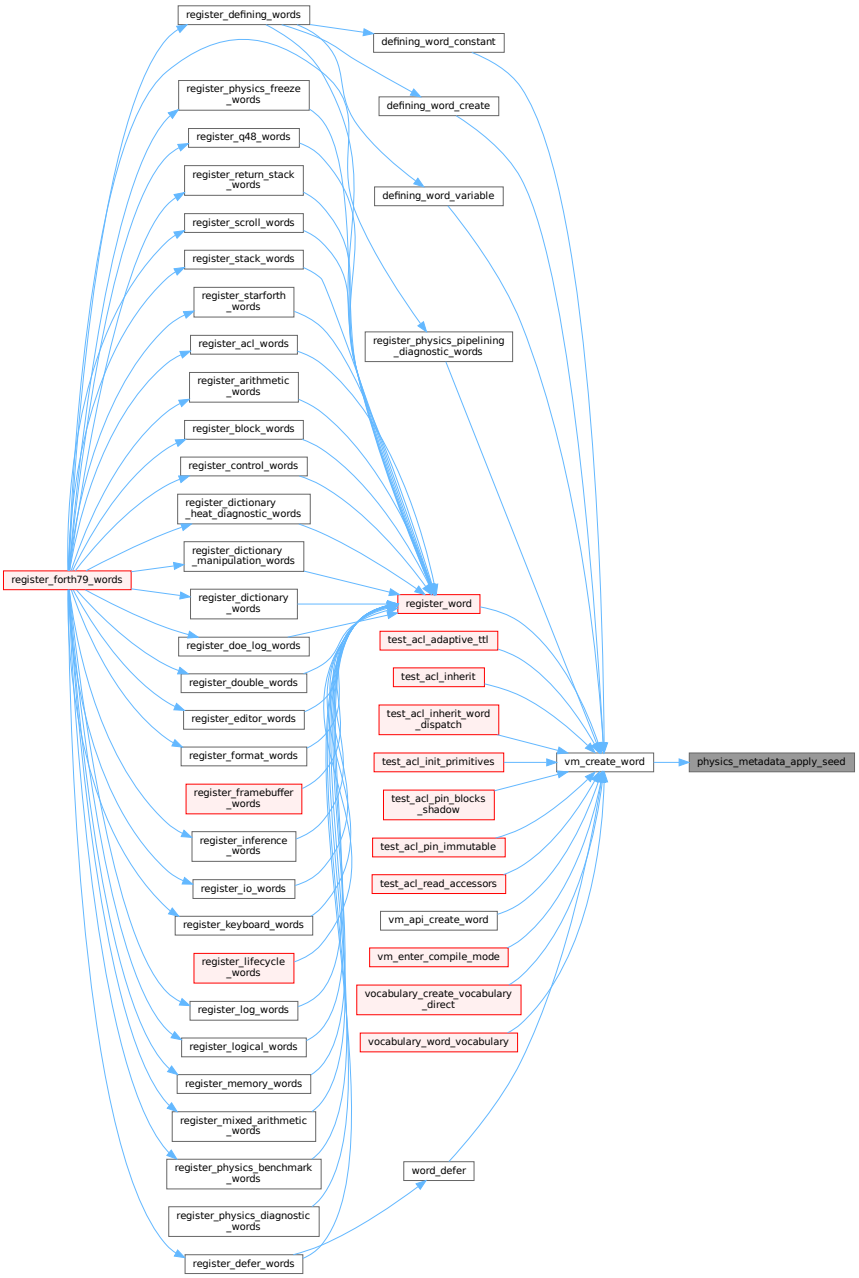
References entry, physics_seed_definition::name, physics_seed_table, strlen(), strncmp(), and physics_seed_definition::temperature_

Referenced by vm_create_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.293.2.5 physics_metadata_init()

```
void physics_metadata_init (
            DictEntry * entry,
            uint32_t header_bytes )
```

Initialise physics metadata for a newly registered DictEntry.

Zeroes all `DictPhysics` fields, sets `mass_bytes` to `header_bytes` (the byte size of the entry header), and derives initial `state_flags` from the entry's word flags. Called from `register_word()` immediately after the entry is linked into the dictionary.

Parameters

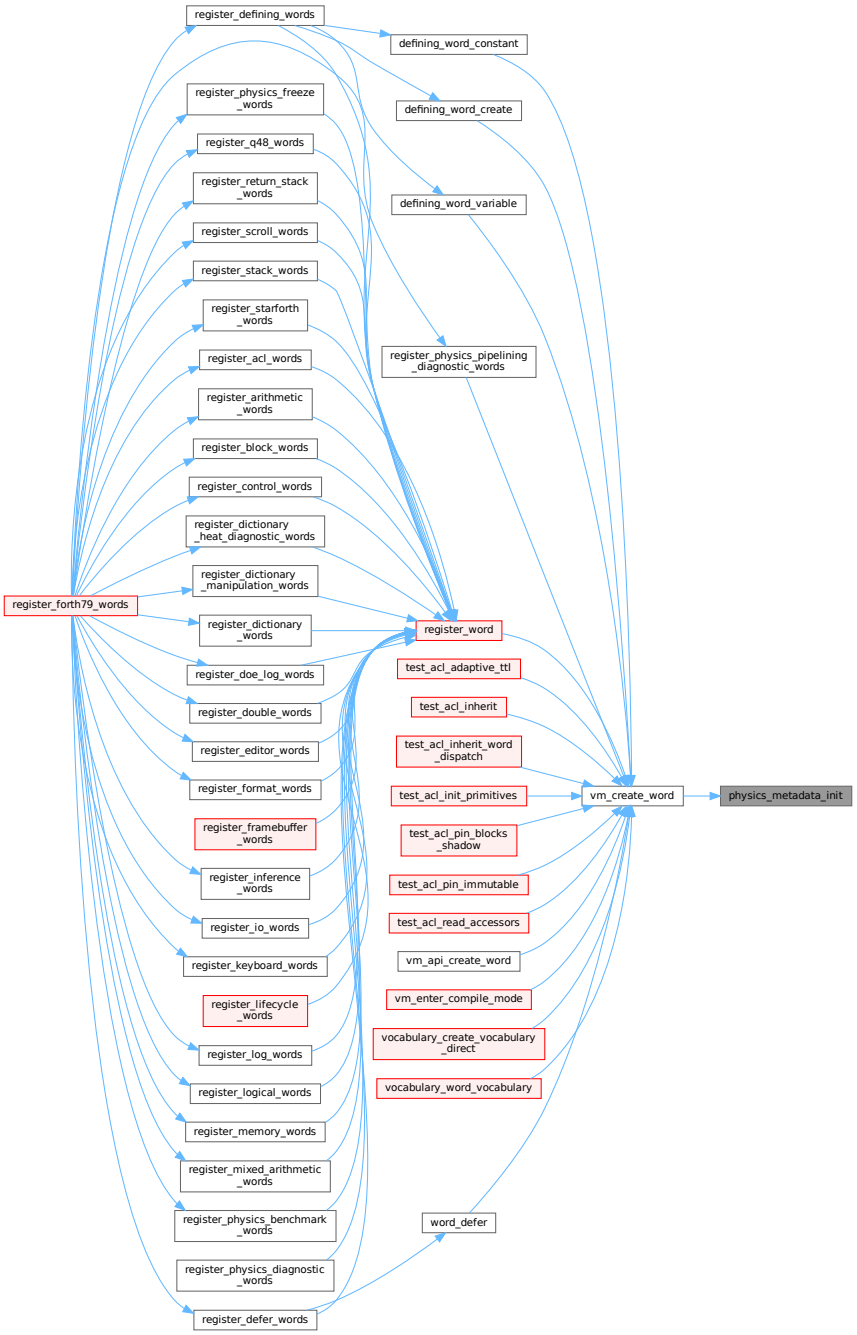
entry	DictEntry to initialise
header_bytes	Byte size of the entry's header region (for mass)

References derive_state_flags(), and entry.

Referenced by vm_create_word().

Here is the call graph for this function:

Here is the caller graph for this function:



7.293.2.6 physics_metadata_record_latency()

```
void physics_metadata_record_latency (
            DictEntry * entry,
            uint64_t sample_ns )
```

Record an execution latency sample into the per-word EMA.

Clamps `sample_ns` to uint32_t, then blends it into `avg_latency_ns` with a 1/4 EMA: new = (3∗prior + sample) / 4. Seeds `avg_latency_ns` directly from the first sample.

Parameters

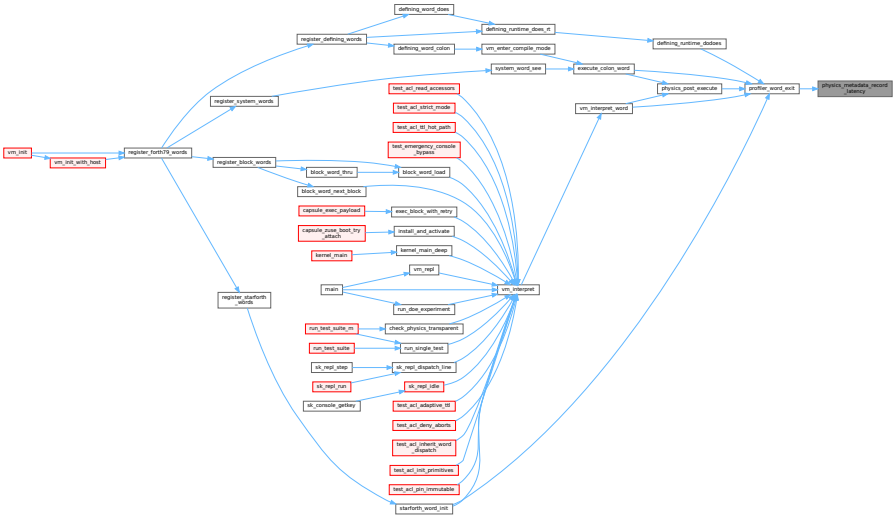
entry	DictEntry whose latency EMA is updated
sample_ns	Measured execution duration in nanoseconds

References clamp_u32(), and entry.

Referenced by profiler_word_exit().

Here is the call graph for this function:

Here is the caller graph for this function:



7.293.2.7 physics_metadata_refresh_state()

```
void physics_metadata_refresh_state (
            DictEntry * entry )
```

Re-derive and update `state_flags` for a DictEntry.

Called after word flags change (e.g. IMMEDIATE is set during compilation) to keep the physics state flags in sync with the word flags.

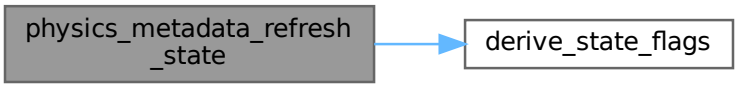
Parameters

entry	DictEntry whose state flags are to be refreshed

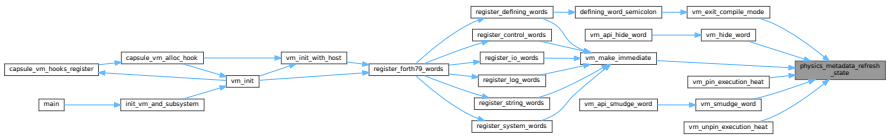
References derive_state_flags(), and entry.

Referenced by vm_exit_compile_mode(), vm_hide_word(), vm_make_immediate(), vm_pin_execution_heat(), vm_smudge_word(), and vm_unpin_execution_heat().

Here is the call graph for this function:



Here is the caller graph for this function:



7.293.2.8 physics_metadata_set_mass()

```
void physics_metadata_set_mass (
            DictEntry * entry,
            uint32_t total_bytes )
```

Update the mass (byte footprint) of a DictEntry's physics metadata.

Called after a colon definition or CREATE body is fully compiled so the mass reflects the total compiled size, not just the header.

Parameters

entry	DictEntry to update
total_bytes	Total compiled byte size (header + body)

References entry.

Referenced by vm_exit_compile_mode().

Here is the caller graph for this function:

7.293.2.9 physics_metadata_touch()

```
void physics_metadata_touch (
            DictEntry * entry,
            cell_t execution_heat,
            uint64_t now_ns )
```

Update physics metadata after a word execution completes.

Derives and stores a new smoothed `temperature_q8` from `execution_heat`, and updates both `last_active_ns` and `last_decay_ns` to `now_ns`. Called by `physics_post_execute()` on the post-execution path.

Parameters

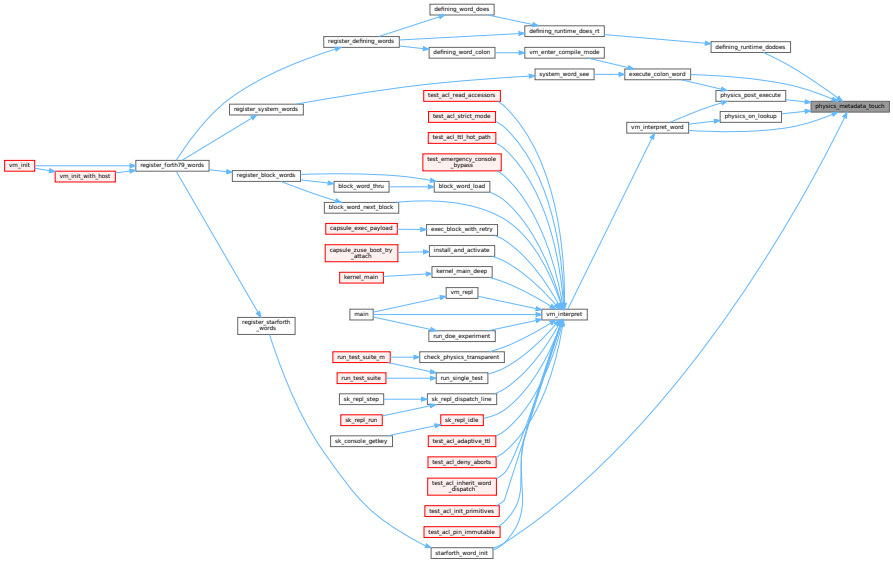
entry	DictEntry that was just executed
execution_heat	Current raw execution heat counter
now_ns	Current monotonic timestamp in nanoseconds

References entry, execution_heat, and temperature_from_execution_heat().

Referenced by defining_runtime_dodoes(), execute_colon_word(), physics_on_lookup(), physics_post_execute(), starforth_word_init(), and vm_interpret_word().

Here is the call graph for this function:

Here is the caller graph for this function:



7.293.2.10 temperature_from_execution_heat()

```
static uint16_t temperature_from_execution_heat (
            cell_t execution_heat,
            uint16_t prior_q8 )  [static]
```

Derive a smoothed Q8 temperature from raw execution heat.

Converts `execution_heat` to Q8 by left-shifting 8 bits, clamps to UINT16_MAX, then applies a 1/4 EMA blend with `prior_q8` to suppress jitter: result = (3∗prior + target) / 4. Returns 0 for non-positive heat.

Parameters

execution_heat	Raw execution heat counter for the word
prior_q8	Previous smoothed temperature in Q8 format

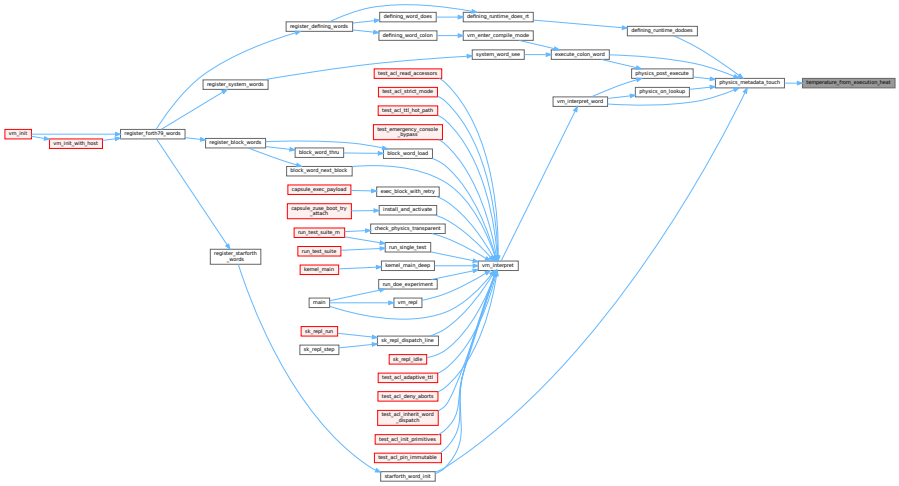
Returns

Updated smoothed temperature in Q8 format

References execution_heat.

Referenced by physics_metadata_touch().

Here is the caller graph for this function:



7.293.3 Variable Documentation

7.293.3.1 physics_seed_table

```
const physics_seed_definition_t physics_seed_table[]  [static]
```

Initial value:

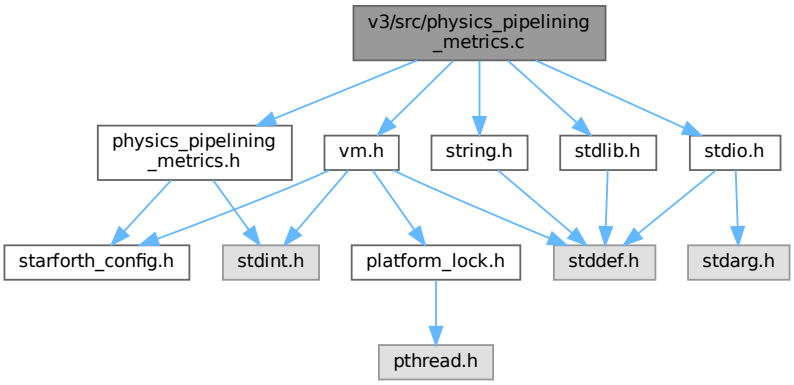
```
= {
    {"IF", 0x0100u, 400u, 0, 0},
    {"ELSE", 0x0100u, 400u, 0, 0},
    {"THEN", 0x0100u, 350u, 0, 0},
    {"BEGIN", 0x0120u, 450u, 0, 0},
    {"WHILE", 0x0120u, 450u, 0, 0},
    {"REPEAT", 0x0120u, 450u, 0, 0},
    {"DO", 0x0180u, 500u, 0, 0},
    {"LOOP", 0x0180u, 500u, 0, 0},
    {"+LOOP", 0x0200u, 520u, 0, 0},
    {"LEAVE", 0x0200u, 520u, 0, 0},
    {".", 0x0080u, 300u, 0, 0},
    {"EMIT", 0x0180u, 800u, 0, 0x02u},
    {"TYPE", 0x0180u, 900u, 0, 0x02u},
    {"BLOCK", 0x0200u, 950u, 0, 0x04u},
    {"BUFFER", 0x01C0u, 900u, 0, 0x04u},
    {"SAVE-BUFFERS", 0x0260u, 1400u, 0, 0x04u},
    {"FLUSH", 0x0240u, 1200u, 0, 0x04u},
    {"LOAD", 0x0220u, 1100u, 0, 0x04u},
    {"LIST", 0x0200u, 1000u, 0, 0x04u},
    {"SAVE-SYSTEM", 0x0280u, 2000u, 0x80u, 0x08u},
    {NULL, 0, 0, 0, 0}
}
```

Referenced by physics_metadata_apply_seed().

7.294 v3/src/physics_pipelining_metrics.c File Reference

```
#include "physics_pipelining_metrics.h"
#include "vm.h"
#include <stdlib.h>
#include <string.h>
```

`#include <stdio.h>`
Include dependency graph for physics_pipelining_metrics.c:



Functions

- static int64_t ns_to_q48 (int64_t ns)

  Convert a nanosecond count to Q48.16 fixed-point.
- static int64_t q48_to_ns (int64_t q48)

  Convert a Q48.16 fixed-point value to integer nanoseconds (truncated).
- static int64_t q48_div_u64 (int64_t a_q48, uint64_t b)

  Divide a Q48.16 value by a `uint64_t`, returning Q48.16.
- int transition_metrics_init (WordTransitionMetrics *metrics)

  Initialise a `WordTransitionMetrics` structure to its empty, ready-to-record state.
- int transition_metrics_record (WordTransitionMetrics *metrics, uint32_t next_word_id, uint32_t dict_size)

  Record one word-to-word transition in the transition heat table.
- int64_t transition_metrics_get_probability_q48 (const WordTransitionMetrics *metrics, uint32_t target_word_id)

  Return the transition probability to `target_word_id` in Q48.16 format.
- void transition_metrics_update_cache (WordTransitionMetrics *metrics, uint32_t dict_size)

  Scan the transition heat table and cache the highest-probability next word.
- void transition_metrics_record_prefetch_hit (WordTransitionMetrics *metrics, int64_t latency_saved_ns)

  Record a successful speculative prefetch and the latency it saved.
- void transition_metrics_record_prefetch_miss (WordTransitionMetrics *metrics, int64_t recovery_cost_ns)

  Record a failed speculative prefetch and its recovery overhead.
- int transition_metrics_should_speculate (const WordTransitionMetrics *metrics, uint32_t target_word_id)

  Decide whether to speculatively prefetch `target_word_id`.
- char * transition_metrics_stats_string (const WordTransitionMetrics *metrics)

  Allocate and return a human-readable transition metrics summary string.
- void transition_metrics_reset (WordTransitionMetrics *metrics)

  Reset a `WordTransitionMetrics` to empty without freeing the context window.
- void transition_metrics_cleanup (WordTransitionMetrics *metrics)

  Release all heap memory held by a `WordTransitionMetrics`.
- int transition_metrics_update_context_window (WordTransitionMetrics *metrics, uint32_t word_id)

  Slide the context window forward by one word, appending `word_id`.

- int transition_metrics_record_context (WordTransitionMetrics *metrics, const uint32_t *context, uint32_t window_size, uint32_t next_word_id, uint32_t dict_size)

  Record a context-conditioned transition (Phase 1 collection stub).
- char * transition_metrics_context_accuracy_string (const WordTransitionMetrics *metrics)

  Allocate and return a context-window status report string (Phase 1 stub).
- uint32_t transition_metrics_binary_chop_suggest_window (const WordTransitionMetrics *metrics, uint32_t current_window, double accuracy_at_current)

  Suggest a new context window size using binary-chop search (Phase 1 stub).

7.294.1 Function Documentation

7.294.1.1 ns_to_q48()

```
static int64_t ns_to_q48 (
            int64_t ns )  [inline], [static]
```

Convert a nanosecond count to Q48.16 fixed-point.

Shifts `ns` left by 16 bits to produce a Q48.16 value where the integer part represents nanoseconds (1.0 ns = 65536 in Q48.16). Used to convert raw platform nanosecond timestamps into the common Q48.16 format before accumulating into `prefetch_latency_saved_q48` and `misprediction_cost_q48`.

Parameters

ns	Integer nanosecond value to convert.

Returns

`ns` in Q48.16 format (`ns` × 65536).

Referenced by transition_metrics_record_prefetch_hit(), and transition_metrics_record_prefetch_miss().

Here is the caller graph for this function:

7.294.1.2 q48_div_u64()

```
static int64_t q48_div_u64 (
            int64_t a_q48,
            uint64_t b )  [inline], [static]
```

Divide a Q48.16 value by a `uint64_t`, returning Q48.16.

Computes `a_q48` / `b` using integer division. The result preserves the Q48.16 scaling of `a_q48` because `b` is dimensionless. Guards against divide-by-zero by returning 0 when `b` == 0. Used by `transition_metrics_get_probability_q48()` to compute the per-word transition probability:

```
P(next = w) = (count_q48 « 16) / total_transitions
```

where `count_q48` = (uint64_t)count << 16 and `total_transitions` is the denominator `b`.

Parameters

a_q48	Numerator in Q48.16 format.
b	Denominator as a plain `uint64_t` (dimensionless count).

Returns

`a_q48` / `b` in Q48.16; 0 if `b` == 0.

Referenced by transition_metrics_get_probability_q48().

Here is the caller graph for this function:

7.294.1.3 q48_to_ns()

```
static int64_t q48_to_ns (
            int64_t q48 )  [inline], [static]
```

Convert a Q48.16 fixed-point value to integer nanoseconds (truncated).

Shifts `q48` right by 16 bits, discarding the fractional part. The result is the integer nanosecond count represented by the Q48.16 value. Used in `transition_metrics_stats_string()` to convert the accumulated `prefetch_latency_saved_q48` and `misprediction_cost_q48` back to human-readable nanosecond integers for the stats report string.

Parameters

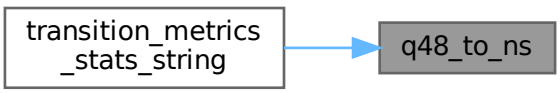
q48	Value in Q48.16 format.

Returns

Integer nanosecond equivalent (truncated, not rounded).

Referenced by transition_metrics_stats_string().

Here is the caller graph for this function:



7.294.1.4 transition_metrics_binary_chop_suggest_window()

```
uint32_t transition_metrics_binary_chop_suggest_window (
            const WordTransitionMetrics * metrics,
            uint32_t current_window,
            double accuracy_at_current )
```

Suggest a new context window size using binary-chop search (Phase 1 stub).

In Phase 2 this function will implement a binary-chop search over window sizes to converge on the width that maximises context-prediction accuracy:

- Start at window = 2.
- If accuracy improves when doubling, try 4; if it declines, try 1.
- Continue halving/doubling until the accuracy delta falls below a threshold.

Phase 1 stub behaviour: Doubles the current window each call up to 8, then wraps back to 1:

- `current_window` == 1 → suggests 2
- `current_window` == 2 → suggests 4
- `current_window` == 4 → suggests 8
- `current_window` ≥ 8 → suggests 1 (cycle back)
- Any other value → returns `current_window` unchanged.

The `accuracy_at_current` parameter (prediction accuracy [0.0, 1.0] at the current window size) is accepted for API stability with the Phase 2 caller but is not used in the stub.

Parameters

metrics	The metrics structure (unused in Phase 1 stub; reserved for Phase 2 accuracy history).
current_window	Currently active context window size.
accuracy_at_current	Prediction accuracy at `current_window` (unused in Phase 1 stub; range [0.0, 1.0]).

Returns

Suggested new window size.

7.294.1.5 transition_metrics_cleanup()

```
void transition_metrics_cleanup (
            WordTransitionMetrics * metrics )
```

Release all heap memory held by a `WordTransitionMetrics`.

Frees and NULLs:

- `metrics->transition_heat` — the per-word transition count array.
- `metrics->context_window` — the sliding context-window array.
- `metrics->context_transitions` — the Phase 2 sparse hash table (if allocated).

Does NOT zero the struct fields beyond NULLing the freed pointers. The caller must not use `metrics` after this call without a subsequent `transition_metrics_init()`. Contrast with `transition_metrics_reset()`, which keeps the context window allocated for reuse.

No-op if `metrics` is NULL.

Parameters

metrics	The metrics structure to clean up; no-op if NULL.

References WordTransitionMetrics::context_transitions, WordTransitionMetrics::context_window, free(), and WordTransitionMetrics::transition_heat.

Referenced by dictionary_word_forget().

Here is the call graph for this function:

Here is the caller graph for this function:

7.294.1.6 transition_metrics_context_accuracy_string()

```
char * transition_metrics_context_accuracy_string (
            const WordTransitionMetrics * metrics )
```

Allocate and return a context-window status report string (Phase 1 stub).

Allocates a 1024-byte buffer and fills it with a brief Phase 1 collection status message:

- `actual_window_size` — the active window width (0 if allocation failed).

- `total_context_transitions` — the number of context transitions recorded.
- A "[Phase 1 Collection - Phase 2 Analysis Pending]" status line.

Phase 2 will replace this stub with a real accuracy report showing per-window-size prediction accuracy derived from the sparse context-pattern hash table. The accuracy data will feed `transition_metrics_binary_chop_suggest_windo` to converge on the optimal context window width.

The caller is responsible for calling `free()` on the returned pointer. Returns NULL if `metrics` is NULL or `malloc()` fails.

Parameters

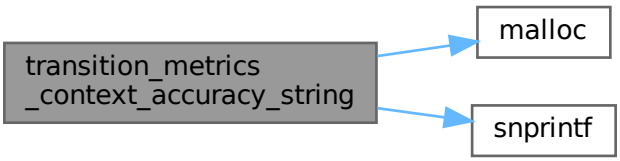
metrics	The metrics structure to report on.

Returns

Heap-allocated NUL-terminated string; caller must `free()`. NULL on allocation failure or NULL `metrics`.

References WordTransitionMetrics::actual_window_size, malloc(), snprintf(), and WordTransitionMetrics::total_context_transitions.

Here is the call graph for this function:



7.294.1.7 transition_metrics_get_probability_q48()

```
int64_t transition_metrics_get_probability_q48 (
            const WordTransitionMetrics * metrics,
            uint32_t target_word_id )
```

Return the transition probability to `target_word_id` in Q48.16 format.

Computes P(next = `target_word_id` | current word) as a Q48.16 fixed-point ratio:
`P = (transition_heat[target_word_id] « 16) / total_transitions`

using `q48_div_u64()` to guard against divide-by-zero.

Returns 0 if:

- `metrics` is NULL.

- `metrics->total_transitions` == 0 (no transitions recorded yet).
- `metrics->transition_heat` is NULL (lazy allocation has not occurred).

The caller is responsible for ensuring `target_word_id` is within the bounds of the allocated `transition_↩ heat` array; this function does not re-check bounds after verifying the NULL guard above (bounds were enforced during `transition_metrics_record()`).

Consumed by `transition_metrics_update_cache()` (scans all words to find the maximum) and `transition_metrics_should_speculate()` (checks whether a specific word's probability exceeds `SPECULATION_THRESHOLD_Q48`).

Parameters

metrics	The metrics structure to query.
target_word↩ _id	Zero-based word ID whose transition probability is requested; must be within the dict_size used at record time.

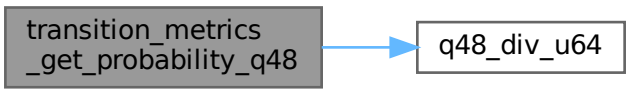
Returns

P(next = `target_word_id`) in Q48.16; 0 if unavailable.

References count, q48_div_u64(), WordTransitionMetrics::total_transitions, and WordTransitionMetrics::transition_heat.

Referenced by transition_metrics_should_speculate(), and transition_metrics_update_cache().

Here is the call graph for this function:



Here is the caller graph for this function:

7.294.1.8 transition_metrics_init()

```
int transition_metrics_init (
            WordTransitionMetrics * metrics )
```

Initialise a `WordTransitionMetrics` structure to its empty, ready-to-record state.

Sets all counter and accumulator fields to zero and performs the following allocations:

- `metrics->transition_heat` is left NULL — it is lazily allocated by the first call to `transition_metrics_record(` to avoid the `dict_size` parameter being required at construction time.
- If `TRANSITION_WINDOW_SIZE` > 0, allocates `metrics->context_window` as a zero-filled array of `TRANSITION_WINDOW_SIZE` `uint32_t` word IDs and sets `metrics->actual_window_size` accordingly. If the allocation fails, `actual_window_size` is set to 0 and context tracking degrades gracefully (all context functions become no-ops).
- `metrics->context_transitions` is left NULL — it is lazily allocated when Phase 2 context-pattern analysis is implemented.

Returns 0 on success or ?1 if `metrics` is NULL.

Parameters

metrics	The `WordTransitionMetrics` to initialise; must not be NULL.

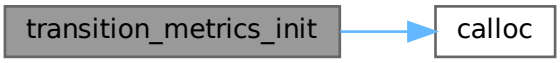
Returns

0 on success; ?1 if `metrics` is NULL.

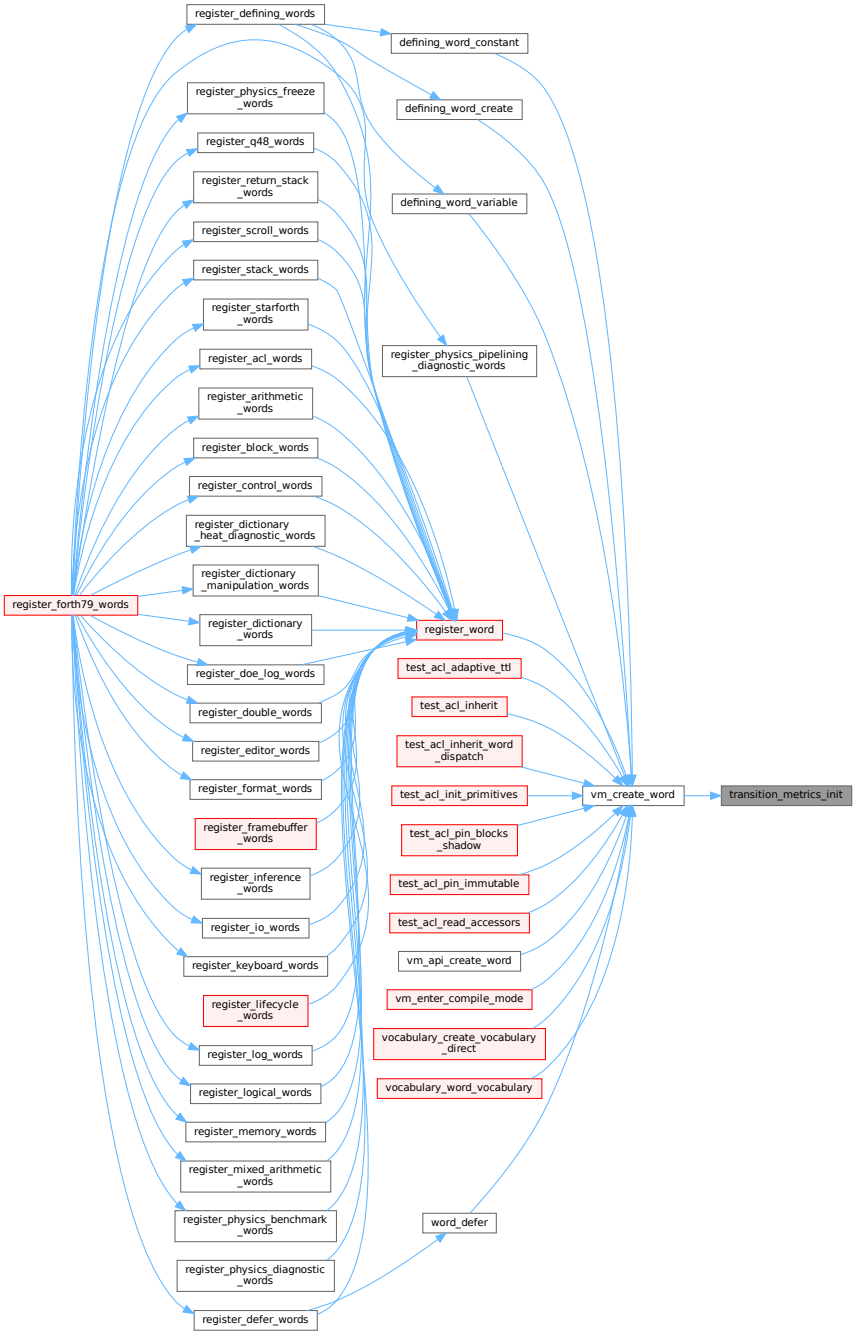
References WordTransitionMetrics::actual_window_size, calloc(), WordTransitionMetrics::context_transitions, WordTransitionMetrics::context_window, WordTransitionMetrics::context_window_pos, WordTransitionMetrics::max_transition_probab WordTransitionMetrics::misprediction_cost_q48, WordTransitionMetrics::most_likely_next_word_id, WordTransitionMetrics::prefetch_a WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_latency_saved_q48, WordTransitionMetrics::prefetch_misses, WordTransitionMetrics::total_context_transitions, WordTransitionMetrics::total_transitions, WordTransitionMetrics::transition_heat, and TRANSITION_WINDOW_SIZE.

Referenced by vm_create_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.294.1.9 transition_metrics_record()

```
int transition_metrics_record (
            WordTransitionMetrics * metrics,
            uint32_t next_word_id,
            uint32_t dict_size )
```

Record one word-to-word transition in the transition heat table.

Called from the interpreter loop each time a word executes and there is a known previous word. Increments `metrics->transition_heat`[next_word_id] by one and `metrics->total_transitions` by one.

Lazy allocation: The `transition_heat` array is not allocated at `transition_metrics_init()` time; on the first call when `metrics->transition_heat` == NULL, it is allocated as a zero-filled array of `dict_size` `uint64_t` counters. If the allocation fails ?1 is returned and the transition is silently dropped.

If `next_word_id` $\geq$ `dict_size` the call is rejected with ?1 to prevent an out-of-bounds write into `transition_heat` (which was allocated with size `dict_size`).

The accumulated `transition_heat`[] table is consumed by `transition_metrics_get_probability_q48()` and `transition_metrics_update_cache()` to determine the most-likely next word for speculative prefetch.

Parameters

metrics	The metrics structure to update.
next_word_id	Zero-based word ID of the word that followed the current word; must be $<$ `dict_size`.
dict_size	Current dictionary size, used both as the bounds check for `next_word_id` and as the allocation size for the lazy-allocated `transition_heat` array.

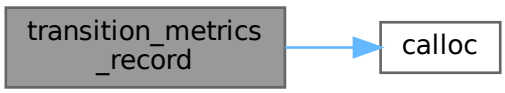
Returns

0 on success; ?1 on NULL `metrics`, out-of-bounds `next_word_id`, or allocation failure.

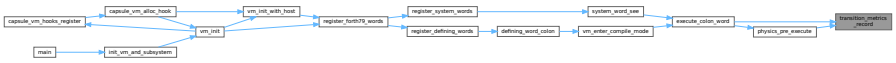
References calloc(), WordTransitionMetrics::total_transitions, and WordTransitionMetrics::transition_heat.

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the call graph for this function:



Here is the caller graph for this function:



7.294.1.10 transition_metrics_record_context()

```
int transition_metrics_record_context (
            WordTransitionMetrics * metrics,
            const uint32_t * context,
            uint32_t window_size,
            uint32_t next_word_id,
            uint32_t dict_size )
```

Record a context-conditioned transition (Phase 1 collection stub).

Accepts a `context` window of `window_size` preceding word IDs and the `next_word_id` that followed. In Phase 1 only the total count is tracked: `metrics->total_context_transitions` is incremented by one, providing a baseline count that Phase 2 analysis can use to size its hash tables.

Phase 2 TODO: Build a sparse hash table keyed on the context pattern (context[0], ..., context[window_size-1]) to record per-context next-word distributions. The Phase 2 analysis will feed `transition_metrics_binary_chop_suggest_w` with accuracy data to tune the optimal window size.

Returns ?1 (no-op) if `metrics` or `context` are NULL, `window_size` is 0, or `next_word_id` ≥ `dict_size`.

Parameters

metrics	The metrics structure to update.
context	Array of `window_size` word IDs preceding `next_word_id`.
window_size	Number of entries in `context`; must be > 0.
next_word_id	Word ID that followed the context sequence.
dict_size	Current dictionary size; used for bounds checking `next_word_id`.

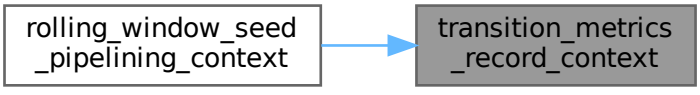
Returns

0 on success; ?1 on invalid arguments.

References WordTransitionMetrics::total_context_transitions.

Referenced by rolling_window_seed_pipelining_context().

Here is the caller graph for this function:



7.294.1.11 transition_metrics_record_prefetch_hit()

```
void transition_metrics_record_prefetch_hit (
            WordTransitionMetrics * metrics,
            int64_t latency_saved_ns )
```

Record a successful speculative prefetch and the latency it saved.

Increments `metrics->prefetch_hits` by one and accumulates `latency_saved_ns` (converted to Q48.16 via `ns_to_q48()`) into `metrics->prefetch_latency_saved_q48`. The accumulated total is reported by `transition_metrics_stats_string()` as the aggregate latency benefit of correct speculation.

Called by the speculative-execution path when a prefetched word matches the word that was actually executed next.

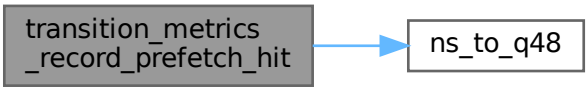
Parameters

metrics	The metrics structure to update; no-op if NULL.
latency_saved_ns	The number of nanoseconds saved by avoiding a full dictionary lookup on this particular hit.

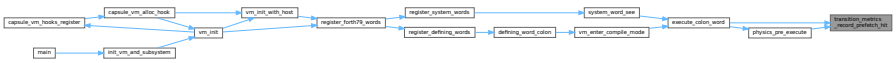
References ns_to_q48(), WordTransitionMetrics::prefetch_hits, and WordTransitionMetrics::prefetch_latency_saved_q48.

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the call graph for this function:



Here is the caller graph for this function:



7.294.1.12 transition_metrics_record_prefetch_miss()

```
void transition_metrics_record_prefetch_miss (
            WordTransitionMetrics * metrics,
            int64_t recovery_cost_ns )
```

Record a failed speculative prefetch and its recovery overhead.

Increments `metrics->prefetch_misses` by one and accumulates `recovery_cost_ns` (converted to Q48.16 via `ns_to_q48()`) into `metrics->misprediction_cost_q48`. The accumulated total is the aggregate penalty paid for incorrect speculation and is reported by `transition_metrics_stats_string()` as "Misprediction Cost".

The net benefit of the pipelining subsystem is:
```
net_ns = prefetch_latency_saved_q48 - misprediction_cost_q48
```

A positive net indicates that speculation is profitable overall.

Called by the speculative-execution path when the prefetched word does not match the word that was actually executed next (branch misprediction analogue).

Parameters

metrics	The metrics structure to update; no-op if NULL.
recovery_cost_ns	The number of nanoseconds spent unwinding the incorrect speculation (cache flush, state restore).

References WordTransitionMetrics::misprediction_cost_q48, ns_to_q48(), and WordTransitionMetrics::prefetch_misses.

Here is the call graph for this function:

7.294.1.13 transition_metrics_reset()

```
void transition_metrics_reset (
            WordTransitionMetrics * metrics )
```

Reset a `WordTransitionMetrics` to empty without freeing the context window.

Frees and NULLs `metrics->transition_heat` (so the next `transition_metrics_record()` will re-allocate it), then zeroes all counter and accumulator fields. Also zeroes the context-window array (if allocated) and resets `context_window_pos` to 0, and frees and NULLs `context_transitions`. `actual_window_size` and the `context_window` pointer itself are preserved — the window stays allocated and ready for reuse.

Intended for DoE benchmark runs that need a clean statistical baseline between treatment cells without the overhead of full teardown and reinitialisation. Cheaper than `transition_metrics_cleanup()` + `transition_metrics_init()` because it avoids reallocating the context window.

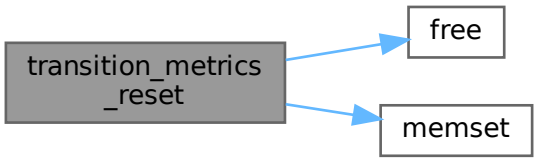
Parameters

metrics	The metrics structure to reset; no-op if NULL.

References WordTransitionMetrics::actual_window_size, WordTransitionMetrics::context_transitions, WordTransitionMetrics::context_
WordTransitionMetrics::context_window_pos, free(), WordTransitionMetrics::max_transition_probability_q48,
memset(), WordTransitionMetrics::misprediction_cost_q48, WordTransitionMetrics::most_likely_next_word_id,
WordTransitionMetrics::prefetch_attempts, WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_latency_saved_q48
WordTransitionMetrics::prefetch_misses, WordTransitionMetrics::total_context_transitions, WordTransitionMetrics::total_transitions,
and WordTransitionMetrics::transition_heat.

Referenced by forth_PIPELINING_RESET_ALL().

Here is the call graph for this function:



Here is the caller graph for this function:

7.294.1.14 transition_metrics_should_speculate()

```
int transition_metrics_should_speculate (
            const WordTransitionMetrics * metrics,
            uint32_t target_word_id )
```

Decide whether to speculatively prefetch `target_word_id`.

Applies a two-gate policy before authorising speculation:

1. **Minimum sample gate:** `metrics->total_transitions` must be $\geq$ `MIN_SAMPLES_FOR_SPECULATION`. Below this threshold the transition probabilities are statistically unreliable — speculation would be indistinguishable from guessing.
2. **Probability threshold gate:** the Q48.16 probability returned by `transition_metrics_get_probability_q48`(metrics, `target_word_id`) must be $\geq$ `SPECULATION_THRESHOLD_Q48`. Words with very low conditional probability of following the current word are not worth prefetching.

Both gates must pass for the function to return 1. A Phase 4 ROI check (comparing expected latency savings against expected misprediction cost) is planned but not yet implemented; a TODO comment marks the extension point.

Returns 0 if `metrics` or `transition_heat` are NULL, or if either gate fails.

Parameters

metrics	The metrics structure to consult.
target_word↩ _id	Zero-based ID of the candidate next word.

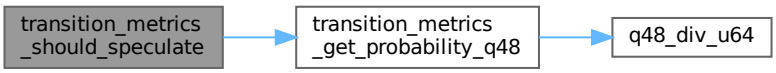
Returns

1 if speculation is recommended; 0 otherwise.

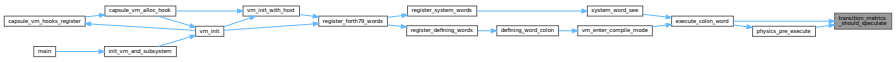
References MIN_SAMPLES_FOR_SPECULATION, SPECULATION_THRESHOLD_Q48, WordTransitionMetrics::total_transitions, WordTransitionMetrics::transition_heat, and transition_metrics_get_probability_q48().

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the call graph for this function:



Here is the caller graph for this function:



7.294.1.15 transition_metrics_stats_string()

```
char * transition_metrics_stats_string (
            const WordTransitionMetrics * metrics )
```

Allocate and return a human-readable transition metrics summary string.

Allocates a 2048-byte buffer and fills it with a multi-line report:

- Total transitions recorded.
- Prefetch attempts, hits (with hit-rate %), and misses.
- Aggregate latency saved by correct speculation (from `prefetch_latency_saved_q48`, converted to integer ns via `q48_to_ns()`).
- Aggregate misprediction cost (from `misprediction_cost_q48`).
- Net benefit (saved ? cost) in nanoseconds.
- The most-likely next word ID and its raw Q48.16 probability.

The caller is responsible for calling `free()` on the returned pointer. Returns NULL if `metrics` is NULL or `malloc()` fails.

Parameters

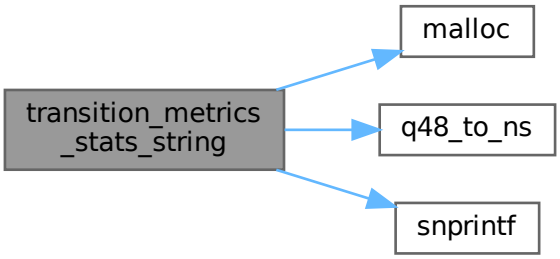
metrics	The metrics structure to summarise.

Returns

Heap-allocated NUL-terminated stats string; caller must `free()`. NULL on allocation failure or NULL `metrics`.

References malloc(), WordTransitionMetrics::max_transition_probability_q48, WordTransitionMetrics::misprediction_cost_q48, WordTransitionMetrics::most_likely_next_word_id, WordTransitionMetrics::prefetch_attempts, WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_latency_saved_q48, WordTransitionMetrics::prefetch_misses, q48_to_ns(), snprintf(), and WordTransitionMetrics::total_transitions.

Here is the call graph for this function:



7.294.1.16 transition_metrics_update_cache()

```
void transition_metrics_update_cache (
            WordTransitionMetrics * metrics,
            uint32_t dict_size )
```

Scan the transition heat table and cache the highest-probability next word.

Iterates over all `dict_size` entries in `metrics->transition_heat`[], calling `transition_metrics_get_probabil` for each non-zero entry and tracking the maximum. After the scan:

- `metrics->max_transition_probability_q48` holds the peak probability (Q48.16).
- `metrics->most_likely_next_word_id` holds the word ID of that peak.

These two cached fields allow `transition_metrics_should_speculate()` to make a prefetch decision in O(1) without re-scanning the full table on every word dispatch.

If `metrics`, `transition_heat`, or `total_transitions` are not valid, both fields are reset to 0 and the function returns immediately.

Called after each `transition_metrics_record()` in the high-frequency interpreter path, so the implementation must be fast. The full `dict_size` scan runs in O(dict_size) and skips zero-count entries early.

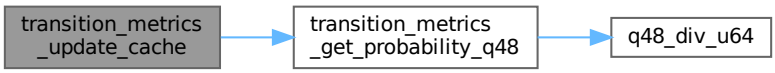
Parameters

metrics	The metrics structure to update.
dict_size	Number of entries in `metrics->transition_heat[]`; must match the size used at allocation time.

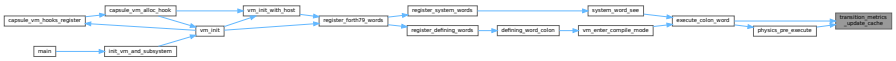
References WordTransitionMetrics::max_transition_probability_q48, WordTransitionMetrics::most_likely_next_word_id, WordTransitionMetrics::total_transitions, WordTransitionMetrics::transition_heat, and transition_metrics_get_probability_q48().

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the call graph for this function:



Here is the caller graph for this function:



7.294.1.17 transition_metrics_update_context_window()

```
int transition_metrics_update_context_window (
            WordTransitionMetrics * metrics,
            uint32_t word_id )
```

Slide the context window forward by one word, appending `word_id`.

Implements a fixed-length sliding window over the recent word-ID history. Each call shifts all existing entries one position to the left (dropping the oldest), then writes `word_id` into the last slot:

```
context_window[0..n-2] = context_window[1..n-1]
context_window[n-1] = word_id
```

This produces a FIFO queue of the last `actual_window_size` word IDs that can be passed to `transition_metrics_record_context()` for context-sensitive transition tracking (Phase 2).

Returns ?1 and is a no-op if:

- `metrics` is NULL.
- `context_window` is NULL (allocation at init failed — degraded mode).
- `actual_window_size` == 0.

Parameters

metrics	The metrics structure whose context window is updated.
word↩ _id	The word ID just executed; appended to the right of the window.

Returns

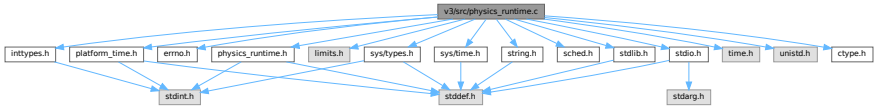
0 on success; ?1 if the context window is unavailable.

References WordTransitionMetrics::actual_window_size, WordTransitionMetrics::context_window, and word_id.

7.295 v3/src/physics_runtime.c File Reference

```
#include "physics_runtime.h"
#include "platform_time.h"
#include <errno.h>
#include <inttypes.h>
#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sched.h>
#include <sys/time.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
#include <ctype.h>
```

Include dependency graph for physics_runtime.c:



Data Structures

- struct physics_runtime_state

Macros

- #define PATH_MAX 4096

Typedefs

- typedef struct physics_runtime_state physics_runtime_state_t

Functions

- static size_t align_up (size_t value, size_t alignment)

  Round `value` up to the next multiple of `alignment`.
- static uint32_t parse_avg_milli (const char *line, const char *tag)

  Parse a floating-point average field from a Linux PSI file line.
- static int read_psi_file (const char *path, uint32_t *some10, uint32_t *some60, uint32_t *some300, uint32_t *full10, uint32_t *full60, uint32_t *full300)

  Read a Linux Pressure Stall Information (PSI) file.
- static int read_proc_stat_totals (uint64_t *total, uint64_t *idle)

  Read aggregate CPU jiffy totals from `/proc/stat`.
- static int resolve_cgroup_path (char *out, size_t out_sz)

  Resolve the cgroup v2 filesystem path for the current process.
- static uint64_t read_cgroup_cpu_usage_us (const char *cgpath)

  Read the cumulative CPU usage in microseconds from a cgroup's `cpu.stat`.
- static uint64_t read_cgroup_memory_current (const char *cgpath)

  Read current memory usage in bytes from a cgroup's `memory.current`.
- static int analytics_heap_allocate (size_t requested_bytes)

  Allocate and initialise the HOLA analytics heap.
- static void analytics_heap_release (void)

  Free the analytics heap and reset all global heap state.
- static size_t analytics_ring_free (const physics_analytics_header_t *hdr)

  Compute free bytes available in the analytics ring buffer.
- int physics_runtime_init (size_t analytics_heap_bytes)

  Initialise the physics runtime and analytics heap.
- void physics_runtime_shutdown (void)

  Shut down the physics runtime and release the analytics heap.
- const physics_analytics_heap_t * physics_analytics_heap_info (void)

  Return a pointer to the live analytics heap descriptor.
- int physics_analytics_publish_event (uint32_t channel, const void *payload, uint16_t payload_bytes)

  Publish an analytics event into the ring buffer.
- static int snapshot_posix (physics_host_snapshot_t *out)

  Collect a host system snapshot using POSIX / Linux interfaces.
- int physics_host_snapshot (physics_host_snapshot_t *out)

  Capture a host environment snapshot and publish it to the analytics ring.

Variables

- static physics_runtime_state_t g_runtime = {0}

7.295.1 Macro Definition Documentation

7.295.1.1 PATH_MAX

```
#define PATH_MAX 4096
```

7.295.2 Typedef Documentation

7.295.2.1 physics_runtime_state_t

```
typedef struct physics_runtime_state physics_runtime_state_t
```

7.295.3 Function Documentation

7.295.3.1 align_up()

```
static size_t align_up (
            size_t value,
            size_t alignment )  [static]
```

Round `value` up to the next multiple of `alignment`.

`alignment` must be a power of two. Used internally to satisfy cache-line and struct alignment requirements in the analytics heap.

Parameters

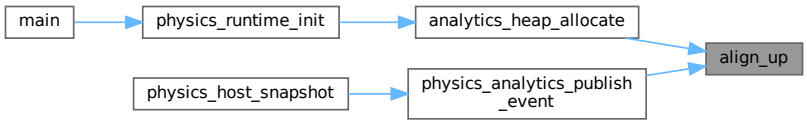
value	Value to align
alignment	Alignment boundary (must be power of two)

Returns

Smallest multiple of `alignment` that is >= `value`

Referenced by analytics_heap_allocate(), and physics_analytics_publish_event().

Here is the caller graph for this function:



7.295.3.2 analytics_heap_allocate()

```
static int analytics_heap_allocate (
            size_t requested_bytes )  [static]
```

Allocate and initialise the HOLA analytics heap.

Allocates a contiguous block of at least `requested_bytes` (or `PHYSICS_ANALYTICS_DEFAULT_HEAP↩_BYTES` if 0), enforcing a minimum that accommodates the header plus 512 KB ring buffer. Partitions the block into header (cache-line aligned), ring (60%), summary (30%), and scratch regions, then populates the `physics↩_analytics_header_t` with magic, version, and offsets. Safe to call repeatedly — returns 0 immediately if already ready.

Parameters

requested_bytes	Desired heap size in bytes (0 = use default)

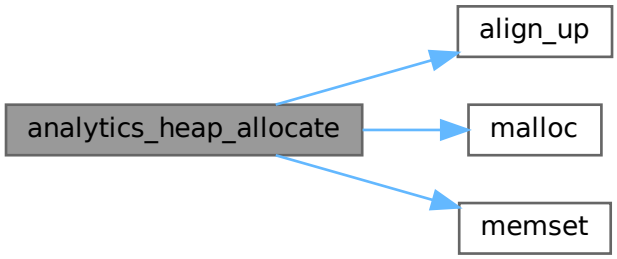
Returns

0 on success, -1 if allocation failed

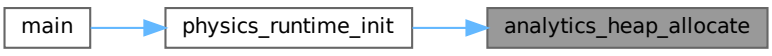
References align_up(), physics_runtime_state::backend_seq, base, physics_analytics_heap::base, bytes, physics_analytics_heap::bytes, physics_analytics_header::consume_seq, physics_analytics_header::dropped_events, physics_analytics_header::flags, g_runtime, physics_analytics_heap::header, physics_runtime_state::heap, physics_analytics_header::heap_bytes, physics_runtime_state::heap_ready, HOLA_PROTOCOL_MAJOR, HOLA_PROTOCOL_MINOR, HOLA_SHARED_MAGIC, physics_analytics_header::magic, malloc(), memset(), PHYSICS_ANALYTICS_DEFAULT_HEAP_BYTES, physics_analytics_header::produce_seq, physics_analytics_header::read_offset, physics_analytics_heap::ring, physics_analytics_header::ring_bytes, physics_analytics_heap::ring_bytes, physics_analytics_header::, physics_analytics_heap::scratch, physics_analytics_header::scratch_bytes, physics_analytics_heap::scratch_bytes, physics_analytics_header::scratch_offset, physics_analytics_heap::summary, physics_analytics_header::summary_bytes, physics_analytics_heap::summary_bytes, physics_analytics_header::summary_offset, physics_analytics_header::version_major, physics_analytics_header::version_minor, and physics_analytics_header::write_offset.

Referenced by physics_runtime_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.295.3.3 analytics_heap_release()

```
static void analytics_heap_release (
            void  )  [static]
```

Free the analytics heap and reset all global heap state.

Safe to call when the heap is not ready. After this call, `g_runtime.heap_ready` is 0 and the heap pointer is NULL.

References physics_runtime_state::backend_seq, physics_analytics_heap::base, free(), g_runtime, physics_runtime_state::heap, and physics_runtime_state::heap_ready.

Referenced by physics_runtime_shutdown().

Here is the call graph for this function:

Here is the caller graph for this function:

7.295.3.4 analytics_ring_free()

```
static size_t analytics_ring_free (
            const physics_analytics_header_t * hdr )  [static]
```

Compute free bytes available in the analytics ring buffer.

Assumes single-producer, single-consumer access with no concurrent writers. Clamps out-of-range `write_offset` and `read_offset` before computing.

Parameters

hdr	Pointer to the analytics heap header

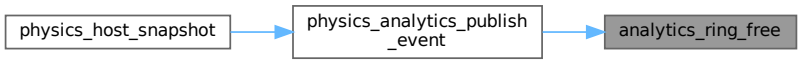
Returns

Number of free bytes available for a new event write

References physics_analytics_header::read_offset, physics_analytics_header::ring_bytes, and physics_analytics_header::write_offse

Referenced by physics_analytics_publish_event().

Here is the caller graph for this function:



7.295.3.5 parse_avg_milli()

```
static uint32_t parse_avg_milli (
            const char * line,
            const char * tag )  [static]
```

Parse a floating-point average field from a Linux PSI file line.

Searches `line` for `tag` (e.g. `"avg10="`), reads the following floating-point value, and returns it scaled to millipercentage (×1000). Returns 0 on parse failure or negative value.

Parameters

line	Raw text line from a `/proc/pressure/` PSI file (cpu, memory, or io)
tag	Field prefix to locate (e.g. `"avg10="`)

Returns

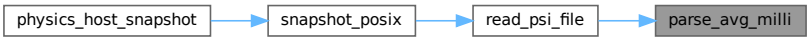
Scaled value in milli-units, or 0 on failure

References errno, and strlen().

Referenced by read_psi_file().

Here is the call graph for this function:

Here is the caller graph for this function:



7.295.3.6 physics_analytics_heap_info()

```
const physics_analytics_heap_t * physics_analytics_heap_info (
            void  )
```

Return a pointer to the live analytics heap descriptor.

Retrieve the analytics heap descriptor.

Returns

Pointer to the heap descriptor, or NULL if not yet initialised

References g_runtime, physics_runtime_state::heap, and physics_runtime_state::heap_ready.

7.295.3.7 physics_analytics_publish_event()

```
int physics_analytics_publish_event (
            uint32_t channel,
            const void * payload,
            uint16_t payload_bytes )
```

Publish an analytics event into the ring buffer.

Writes a `physics_analytics_event_header_t` followed by the payload (8-byte padded) into the ring at the current write offset. If insufficient space is available, increments `dropped_events` and returns -1. The event is timestamped with `sf_monotonic_ns()` at write time.

Parameters

channel	Event channel identifier (e.g. `PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT`)
payload	Event payload bytes (must not be NULL)
payload_bytes	Length of payload in bytes

Returns

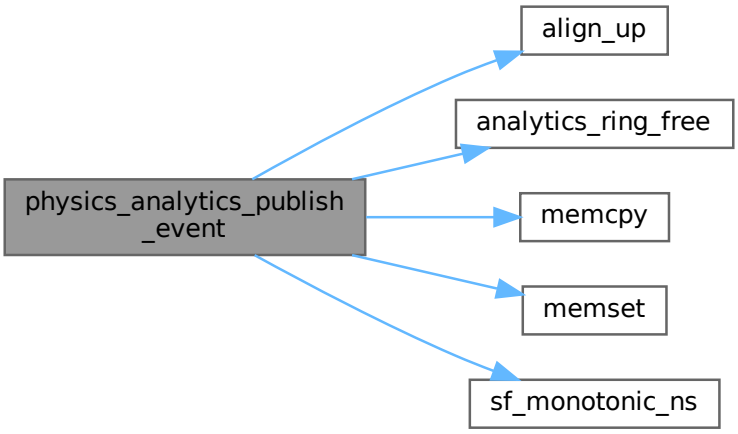
0 on success, -1 if the heap is not ready, payload is NULL, or ring is full

References align_up(), analytics_ring_free(), physics_analytics_event_header::channel, physics_analytics_header::dropped_events, g_runtime, physics_analytics_heap::header, physics_runtime_state::heap, physics_runtime_state::heap_ready, memcpy(), memset(), physics_analytics_event_header::payload_bytes, physics_analytics_header::produce_seq,

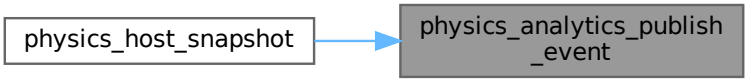
physics_analytics_event_header::reserved, ring, physics_analytics_heap::ring, physics_analytics_header::ring_bytes, sf_monotonic_ns(), physics_analytics_event_header::timestamp_ns, and physics_analytics_header::write_offset.

Referenced by physics_host_snapshot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.295.3.8 physics_host_snapshot()

```
int physics_host_snapshot (
            physics_host_snapshot_t * out )
```

Capture a host environment snapshot and publish it to the analytics ring.

Uses the POSIX backend. On success also publishes the snapshot as a PHYSICS_ANALYTICS_CHANNEL_↩ HOST_SNAPSHOT event to the analytics ring buffer (if the heap is initialised).

Parameters

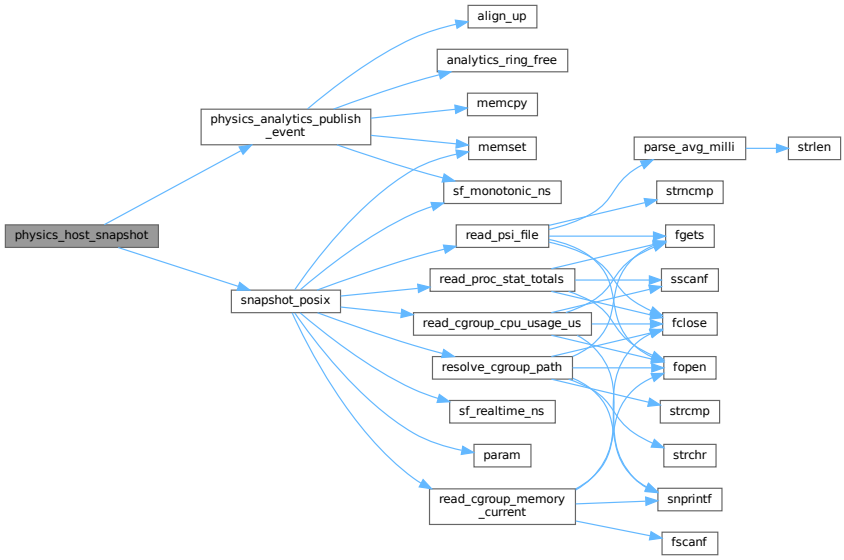
out	Output snapshot struct to populate (must not be NULL)

Returns

0 on success, -1 if `out` is NULL or the backend fails

References g_runtime, physics_runtime_state::heap_ready, PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT, physics_analytics_publish_event(), and snapshot_posix().

Here is the call graph for this function:



7.295.3.9 physics_runtime_init()

```
int physics_runtime_init (
            size_t analytics_heap_bytes )
```

Initialise the physics runtime and analytics heap.

Must be called before any other physics_runtime_∗ function. Idempotent — safe to call multiple times; subsequent calls return 0 immediately without re-allocating.

Parameters

analytics_heap_bytes	Desired analytics heap size (0 = default)

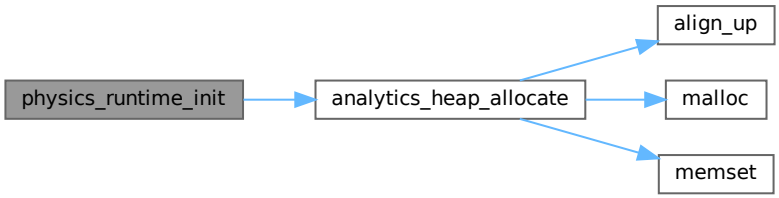
Returns

0 on success, -1 on allocation failure

References analytics_heap_allocate().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

7.295.3.10 physics_runtime_shutdown()

```
void physics_runtime_shutdown (
            void  )
```

Shut down the physics runtime and release the analytics heap.

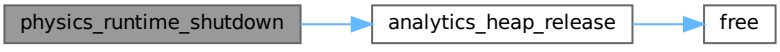
Tear down physics runtime services and release heap storage.

Safe to call when the runtime was never initialised. After this call the analytics heap is gone and no events can be published until `physics_runtime_init()` is called again.

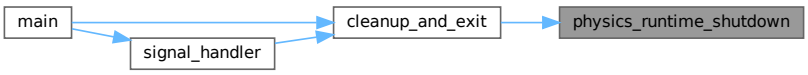
References analytics_heap_release().

Referenced by cleanup_and_exit().

Here is the call graph for this function:



Here is the caller graph for this function:



7.295.3.11 read_cgroup_cpu_usage_us()

```
static uint64_t read_cgroup_cpu_usage_us (
            const char * cgpath )  [static]
```

Read the cumulative CPU usage in microseconds from a cgroup's `cpu.stat`.

Opens <cgpath>/cpu.stat and extracts the `usage_usec` field. Returns 0 if the file cannot be opened or the field is absent.

Parameters

cgpath	Absolute path to the cgroup directory

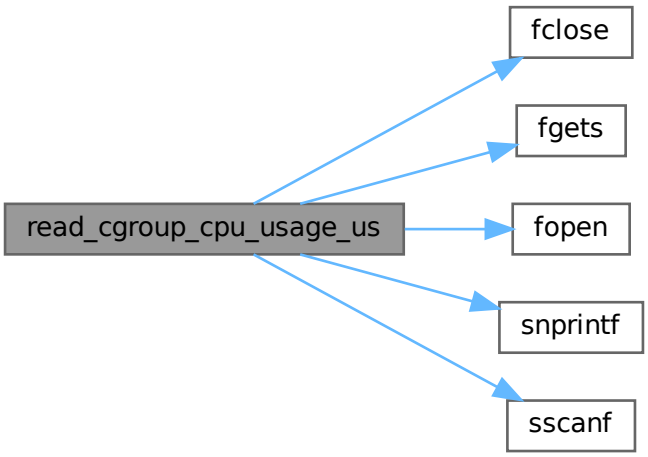
Returns

Cumulative CPU usage in microseconds, or 0 on failure

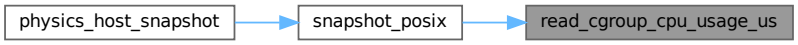
References fclose(), fgets(), fopen(), PATH_MAX, snprintf(), and sscanf().

Referenced by snapshot_posix().

Here is the call graph for this function:



Here is the caller graph for this function:



7.295.3.12 read_cgroup_memory_current()

```
static uint64_t read_cgroup_memory_current (
            const char * cgpath )  [static]
```

Read current memory usage in bytes from a cgroup's `memory.current`.

Opens <cgpath>/memory.current and reads the single integer value. Returns 0 if the file cannot be opened or parsed.

Parameters

cgpath	Absolute path to the cgroup directory

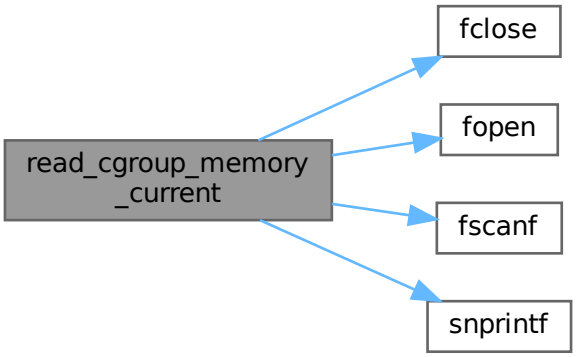
Returns

Current memory usage in bytes, or 0 on failure

References fclose(), fopen(), fscanf(), PATH_MAX, and snprintf().

Referenced by snapshot_posix().

Here is the call graph for this function:



Here is the caller graph for this function:

7.295.3.13 read_proc_stat_totals()

```
static int read_proc_stat_totals (
            uint64_t * total,
            uint64_t * idle )  [static]
```

Read aggregate CPU jiffy totals from `/proc/stat`.

Parses the first `cpu` line from `/proc/stat` and sums user, nice, system, idle, iowait, irq, softirq, and steal columns into `*total`. The idle estimate is `idle_j` + `iowait`. Either output pointer may be NULL.

Parameters

total	Output: total jiffy count across all categories
idle	Output: idle + iowait jiffies (effective idle)

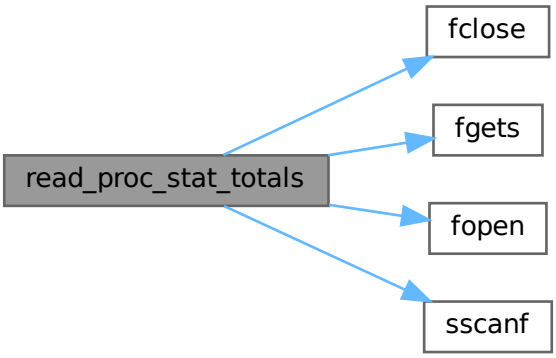
Returns

0 on success, -1 on file open or parse failure

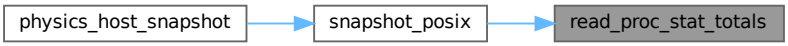
References fclose(), fgets(), fopen(), and sscanf().

Referenced by snapshot_posix().

Here is the call graph for this function:



Here is the caller graph for this function:



7.295.3.14 read_psi_file()

```
static int read_psi_file (
            const char * path,
            uint32_t * some10,
            uint32_t * some60,
            uint32_t * some300,
            uint32_t * full10,
            uint32_t * full60,
            uint32_t * full300 )  [static]
```

Read a Linux Pressure Stall Information (PSI) file.

Parses `/proc/pressure/{cpu,io,memory}` and populates the six output pointers with avg10/avg60/avg300 milli-values for both the `some` and `full` stall categories. Any output pointer may be NULL to skip that field. Returns a bitmask: bit 0 set if `some` line was found, bit 1 if `full`.

Parameters

path	Path to PSI file (e.g. `"/proc/pressure/cpu"`)
some10	avg10 for the `some` stall category (milli-units), or NULL
some60	avg60 for the `some` stall category, or NULL
some300	avg300 for the `some` stall category, or NULL
full10	avg10 for the `full` stall category, or NULL
full60	avg60 for the `full` stall category, or NULL
full300	avg300 for the `full` stall category, or NULL

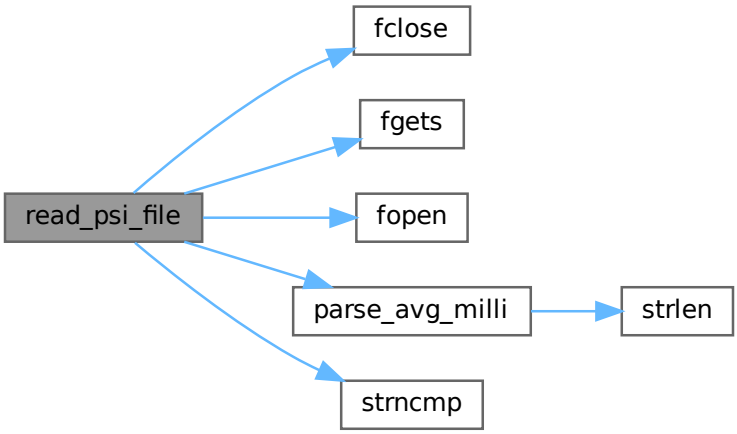
Returns

Bitmask of parsed sections (0 = none, 1 = some, 2 = full, 3 = both)

References fclose(), fgets(), fopen(), parse_avg_milli(), and strncmp().

Referenced by snapshot_posix().

Here is the call graph for this function:



Here is the caller graph for this function:

7.295.3.15 resolve_cgroup_path()

```
static int resolve_cgroup_path (
            char * out,
            size_t out_sz )  [static]
```

Resolve the cgroup v2 filesystem path for the current process.

Reads `/proc/self/cgroup`, finds the cgroup2 hierarchy entry (the line with `"::"` separator), and constructs the full sysfs path under `/sys/fs/cgroup`. The root cgroup ("/") maps to `/sys/fs/cgroup` itself.

Parameters

out	Output buffer for the resolved path
out_sz	Size of `out` in bytes

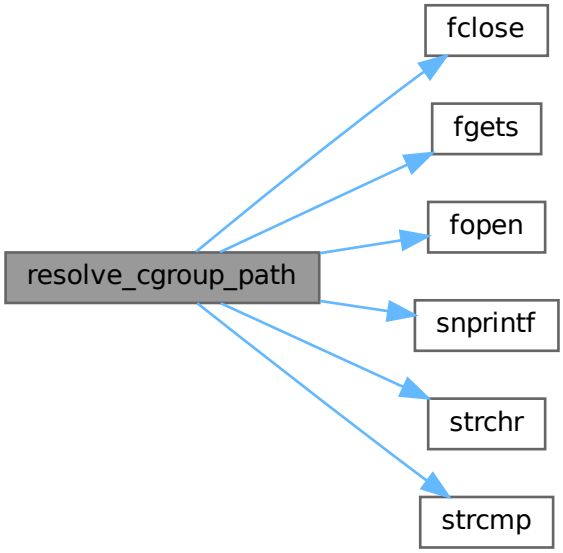
Returns

0 on success, -1 if the path could not be resolved

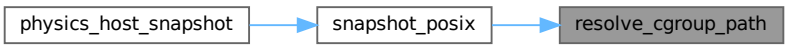
References fclose(), fgets(), fopen(), snprintf(), strchr(), and strcmp().

Referenced by snapshot_posix().

Here is the call graph for this function:



Here is the caller graph for this function:



7.295.3.16 snapshot_posix()

```
static int snapshot_posix (
            physics_host_snapshot_t * out )  [static]
```

Collect a host system snapshot using POSIX / Linux interfaces.

Populates `out` with monotonic and realtime timestamps, CPU count, scheduler policy and priority, load average (scaled × 1000), runnable thread estimate, PSI pressure readings, `/proc/stat` CPU jiffy totals, and cgroup v2 CPU usage and memory current values. Sets flags in `out->flags` to indicate which fields were successfully populated.

Parameters

out	Output snapshot struct; zeroed and populated by this function

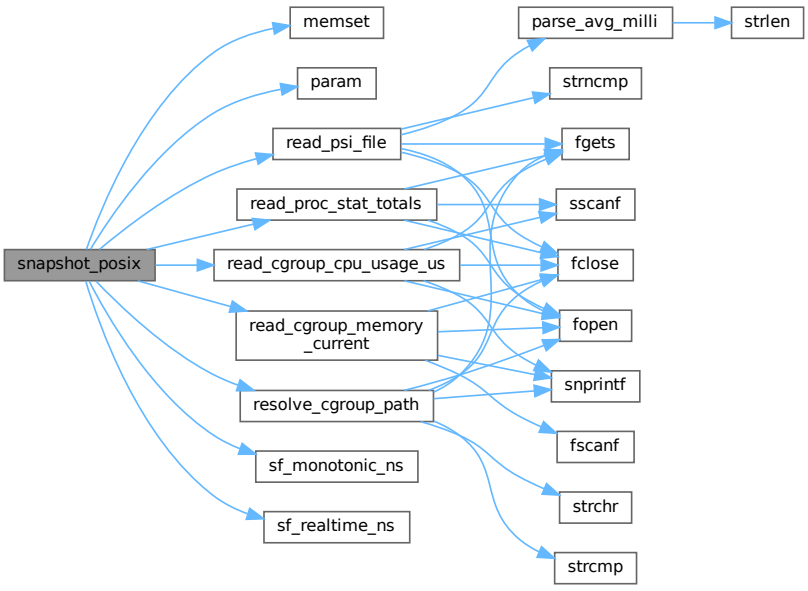
Returns

0 on success

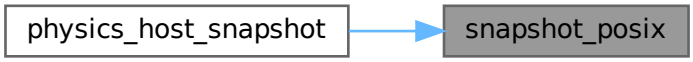
References physics_host_snapshot::backend, physics_host_snapshot::backend_seq, physics_runtime_state::backend_seq, physics_host_snapshot::cgroup_cpu_usage_us, physics_host_snapshot::cgroup_memory_current_bytes, physics_host_snapshot::cp physics_host_snapshot::cpu_idle_jiffies, physics_host_snapshot::cpu_total_jiffies, physics_host_snapshot::flags, g_runtime, physics_host_snapshot::load_avg_milli, memset(), physics_host_snapshot::monotonic_time_ns, param(), PATH_MAX, PHYSICS_HOST_BACKEND_POSIX, PHYSICS_HOST_FLAG_CGROUP, PHYSICS_HOST_FLAG_CPU_ST PHYSICS_HOST_FLAG_PSI, physics_host_snapshot::psi_cpu_full_avg10_milli, physics_host_snapshot::psi_cpu_full_avg300_milli, physics_host_snapshot::psi_cpu_full_avg60_milli, physics_host_snapshot::psi_cpu_some_avg10_milli, physics_host_snapshot::psi_ physics_host_snapshot::psi_cpu_some_avg60_milli, physics_host_snapshot::psi_io_full_avg10_milli, physics_host_snapshot::psi_io physics_host_snapshot::psi_io_full_avg60_milli, physics_host_snapshot::psi_io_some_avg10_milli, physics_host_snapshot::psi_io_s physics_host_snapshot::psi_io_some_avg60_milli, physics_host_snapshot::psi_mem_some_avg10_milli, physics_host_snapshot::ps physics_host_snapshot::psi_mem_some_avg60_milli, read_cgroup_cpu_usage_us(), read_cgroup_memory_current(), read_proc_stat_totals(), read_psi_file(), physics_host_snapshot::realtime_ns, resolve_cgroup_path(), physics_host_snapshot::runnab physics_host_snapshot::scheduler_policy, physics_host_snapshot::scheduler_priority, physics_host_snapshot::scheduler_quantum_ sf_monotonic_ns(), sf_realtime_ns(), timespec::tv_nsec, and timespec::tv_sec.

Referenced by physics_host_snapshot().

Here is the call graph for this function:



Here is the caller graph for this function:



7.295.4 Variable Documentation

7.295.4.1 g_runtime

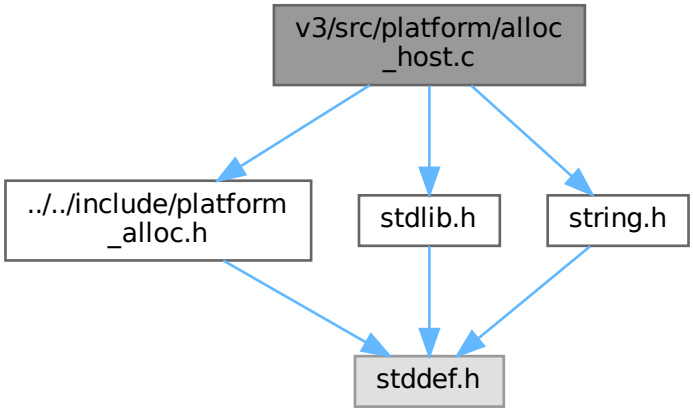
```
physics_runtime_state_t g_runtime = {0}  [static]
```

Referenced by analytics_heap_allocate(), analytics_heap_release(), physics_analytics_heap_info(), physics_analytics_publish_event(), physics_host_snapshot(), and snapshot_posix().

7.296 v3/src/platform/alloc_host.c File Reference

```
#include "../../include/platform_alloc.h"
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for alloc_host.c:



Functions

- int sf_alloc_init (void)

  Initialise the host allocator statistics.
- void * sf_malloc (size_t size)

  Allocate `size` bytes from the system heap.
- void * sf_calloc (size_t count, size_t size)

  Allocate a zero-initialised array of `count` elements of `size` bytes each.
- void * sf_realloc (void *ptr, size_t new_size)

  Resize a previously allocated block to `new_size` bytes.
- void sf_free (void *ptr)

  Release a previously allocated block back to the system heap.
- void sf_alloc_get_stats (sf_alloc_stats_t *stats)

  Copy the current allocator statistics into `stats`.

Variables

- static sf_alloc_stats_t g_stats = {0}

7.296.1 Function Documentation

7.296.1.1 sf_alloc_get_stats()

```
void sf_alloc_get_stats (
            sf_alloc_stats_t * stats )
```

Copy the current allocator statistics into `stats`.

Fills `*stats` with a snapshot of `g_stats`. Note that `used_bytes` overstates the live total when blocks have been freed or reallocated, because freed sizes are not tracked. `free_count` is accurate.

Parameters

stats	Output buffer to populate; no-op if NULL

7.296.1.2 sf_alloc_init()

```
int sf_alloc_init (
            void  )
```

Initialise the host allocator statistics.

Resets the `g_stats` struct to zero. Safe to call multiple times; subsequent calls simply reset the counters. No heap memory is allocated by this function — the underlying allocator is the system `malloc`.

Returns

0 always (cannot fail on the hosted path)

7.296.1.3 sf_calloc()

```
void * sf_calloc (
            size_t count,
            size_t size )
```

Allocate a zero-initialised array of `count` elements of `size` bytes each.

Delegates to `calloc()`. Updates statistics the same way as `sf_malloc()`: adds `count*size` to `used_bytes`, increments `alloc_count`, and updates `peak_bytes`.

Parameters

count	Number of elements; 0 returns NULL
size	Size of each element in bytes; 0 returns NULL

Returns

Pointer to zero-filled memory, or NULL on allocation failure or zero arguments

7.296.1.4 sf_free()

```
void sf_free (
            void * ptr )
```

Release a previously allocated block back to the system heap.

Delegates to `free()`. Increments `g_stats.free_count` on non-NULL `ptr`. The freed byte count is not subtracted from `used_bytes` because the original size is not tracked — see `sf_realloc()` note.

Parameters

ptr	Pointer to free; NULL is a safe no-op

7.296.1.5 sf_malloc()

```
void * sf_malloc (
            size_t size )
```

Allocate `size` bytes from the system heap.

Delegates to `malloc()`. On success, increments `g_stats.used_bytes` and `g_stats.alloc_count`, and updates `g_stats.peak_bytes` if the new used total exceeds the recorded peak.

Parameters

size	Number of bytes to allocate; 0 returns NULL

Returns

Pointer to allocated memory, or NULL on allocation failure or zero size

7.296.1.6 sf_realloc()

```
void * sf_realloc (
            void * ptr,
            size_t new_size )
```

Resize a previously allocated block to `new_size` bytes.

Delegates to `realloc()`. On success, increments `g_stats.alloc_count`. The exact byte delta is not tracked because the original allocation size is not stored alongside the pointer — `used_bytes` is therefore approximate after any `sf_realloc()` call. For accurate accounting, wrap allocations with size-tracking headers.

Parameters

ptr	Pointer to an existing allocation, or NULL (behaves like `sf_malloc`)
new_size	Desired new size in bytes; 0 may free the block (platform-defined)

Returns

Pointer to resized memory, or NULL on failure (original block unchanged)

7.296.2 Variable Documentation

7.296.2.1 g_stats

```
sf_alloc_stats_t g_stats = {0}  [static]
```

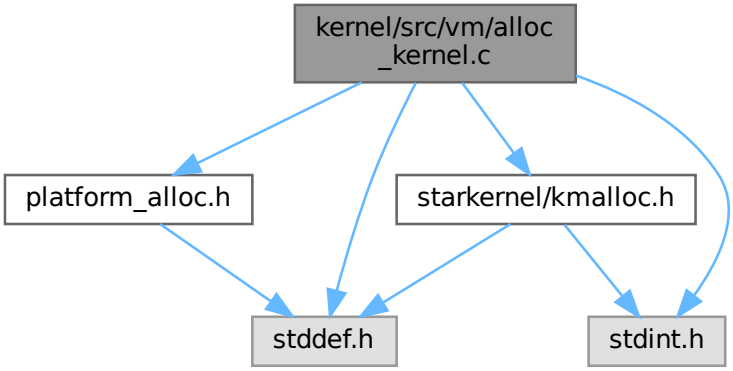
Referenced by sf_alloc_get_stats(), sf_alloc_init(), sf_calloc(), sf_free(), sf_malloc(), and sf_realloc().

7.297 kernel/src/vm/alloc_kernel.c File Reference

```
#include "platform_alloc.h"
#include "starkernel/kmalloc.h"
#include <stdint.h>
```

```
#include <stddef.h>
```

Include dependency graph for alloc_kernel.c:



Macros

- #define SF_ALIGN 8

Functions

- int sf_alloc_init (void)

  No-op: kept only for API compatibility with `platform_alloc.h`.
- void * sf_malloc (size_t size)

  Allocate `size` bytes from the system heap.
- void * sf_calloc (size_t count, size_t size)

  Allocate a zero-initialised array of `count` elements of `size` bytes each.
- void * sf_realloc (void *ptr, size_t new_size)

  Resize a previously allocated block to `new_size` bytes.
- void sf_free (void *ptr)

  Release a previously allocated block back to the system heap.
- void sf_alloc_get_stats (sf_alloc_stats_t *stats)

  Copy the current allocator statistics into `stats`.

Variables

- static size_t g_alloc_count = 0
- static size_t g_free_count = 0

7.297.1 Macro Definition Documentation

7.297.1.1 SF_ALIGN

```
#define SF_ALIGN 8
```

7.297.2 Function Documentation

7.297.2.1 sf_alloc_get_stats()

```
void sf_alloc_get_stats (
            sf_alloc_stats_t * stats )
```

Copy the current allocator statistics into `stats`.

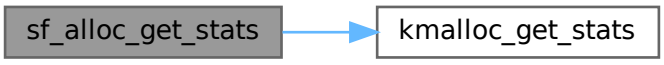
Fills `*stats` with a snapshot of `g_stats`. Note that `used_bytes` overstates the live total when blocks have been freed or reallocated, because freed sizes are not tracked. `free_count` is accurate.

Parameters

stats	Output buffer to populate; no-op if NULL

References sf_alloc_stats::alloc_count, sf_alloc_stats::free_count, g_alloc_count, g_free_count, g_stats, kmalloc_get_stats(), kmalloc_stats_t::peak_bytes, sf_alloc_stats::peak_bytes, kmalloc_stats_t::total_bytes, sf_alloc_stats::total_bytes, kmalloc_stats_t::used_bytes, and sf_alloc_stats::used_bytes.

Here is the call graph for this function:



7.297.2.2 sf_alloc_init()

```
int sf_alloc_init (
            void  )
```

No-op: kept only for API compatibility with `platform_alloc.h`.

Initialise the host allocator statistics.

The backing store is now the shared kernel heap (`kmalloc.c`), already initialised at boot (M6) well before any VM allocation can occur – there is nothing left for this function to set up, and "resetting" a heap shared by every other kernel subsystem would be actively wrong. Kept callable (matches its documented "safe to call at any point" contract) so no caller needs to change.

Returns

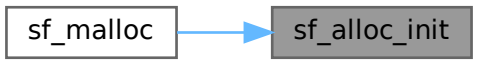
0 always (cannot fail)

References g_stats, and memset().

Referenced by sf_malloc().

Here is the call graph for this function:

Here is the caller graph for this function:



7.297.2.3 sf_calloc()

```
void * sf_calloc (
            size_t count,
            size_t size )
```

Allocate a zero-initialised array of `count` elements of `size` bytes each.

Delegates to `calloc()`. Updates statistics the same way as `sf_malloc()`: adds `count*size` to `used_bytes`, increments `alloc_count`, and updates `peak_bytes`.

Parameters

count	Number of elements; 0 returns NULL
size	Size of each element in bytes; 0 returns NULL

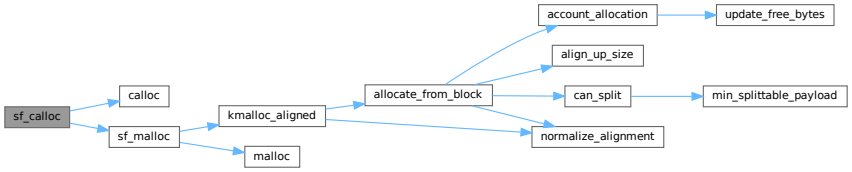
Returns

Pointer to zero-filled memory, or NULL on allocation failure or zero arguments

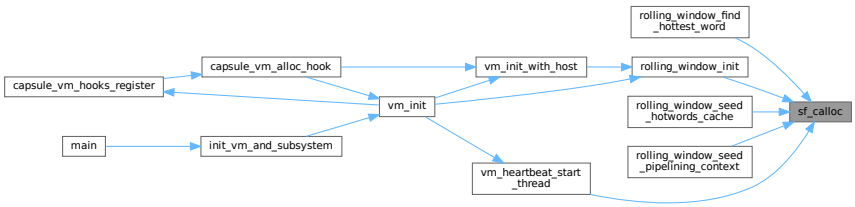
References sf_alloc_stats::alloc_count, calloc(), count, g_stats, sf_alloc_stats::peak_bytes, sf_malloc(), and sf_alloc_stats::used_bytes.

Referenced by rolling_window_find_hottest_word(), rolling_window_init(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), and vm_heartbeat_start_thread().

Here is the call graph for this function:



Here is the caller graph for this function:



7.297.2.4 sf_free()

```
void sf_free (
            void * ptr )
```

Release a previously allocated block back to the system heap.

Delegates to `free()`. Increments `g_stats.free_count` on non-NULL `ptr`. The freed byte count is not subtracted from `used_bytes` because the original size is not tracked — see `sf_realloc()` note.

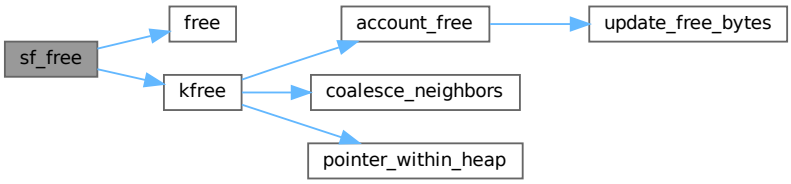
Parameters

ptr	Pointer to free; NULL is a safe no-op

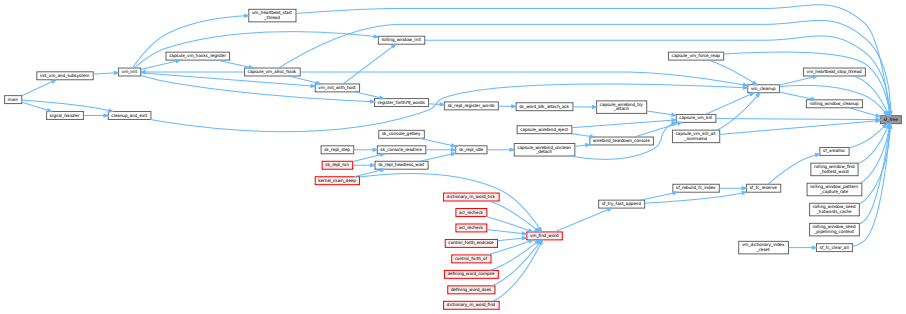
References free(), sf_alloc_stats::free_count, g_free_count, g_stats, and kfree().

Referenced by capsule_vm_alloc_hook(), capsule_vm_force_reap(), capsule_vm_kill(), capsule_vm_kill_all_nonmama(), rolling_window_cleanup(), rolling_window_find_hottest_word(), rolling_window_init(), rolling_window_pattern_capture_rate(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), sf_fc_clear_all(), sf_xrealloc(), vm_cleanup(), vm_heartbeat_start_thread(), and vm_heartbeat_stop_thread().

Here is the call graph for this function:



Here is the caller graph for this function:



7.297.2.5 sf_malloc()

```
void * sf_malloc (
            size_t size )
```

Allocate `size` bytes from the system heap.

Delegates to `malloc()`. On success, increments `g_stats.used_bytes` and `g_stats.alloc_count`, and updates `g_stats.peak_bytes` if the new used total exceeds the recorded peak.

Parameters

size	Number of bytes to allocate; 0 returns NULL

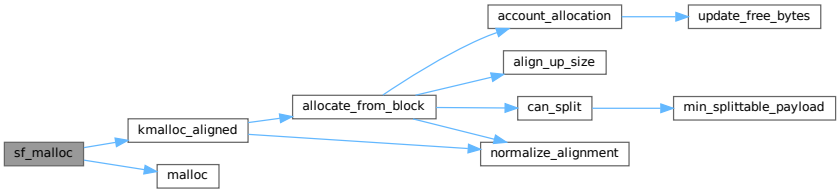
Returns

Pointer to allocated memory, or NULL on allocation failure or zero size

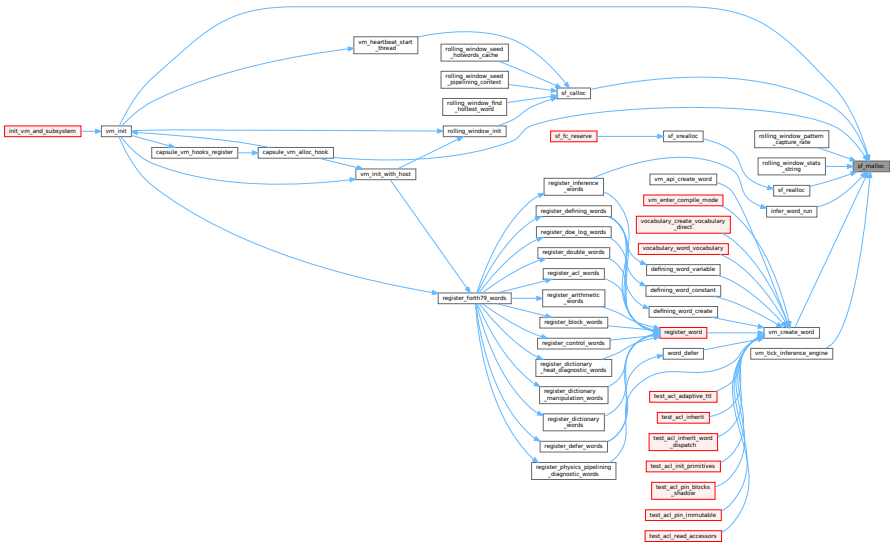
References sf_alloc_stats::alloc_count, g_alloc_count, g_stats, kmalloc_aligned(), malloc(), sf_alloc_stats::peak_bytes, SF_ALIGN, and sf_alloc_stats::used_bytes.

Referenced by capsule_vm_alloc_hook(), infer_word_run(), rolling_window_pattern_capture_rate(), rolling_window_stats_string(), sf_calloc(), sf_realloc(), vm_create_word(), vm_init(), and vm_tick_inference_engine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.297.2.6 sf_realloc()

```
void * sf_realloc (
            void * ptr,
            size_t new_size )
```

Resize a previously allocated block to `new_size` bytes.

Delegates to `realloc()`. On success, increments `g_stats.alloc_count`. The exact byte delta is not tracked because the original allocation size is not stored alongside the pointer — `used_bytes` is therefore approximate after any `sf_realloc()` call. For accurate accounting, wrap allocations with size-tracking headers.

Parameters

ptr	Pointer to an existing allocation, or NULL (behaves like `sf_malloc`)
new_size	Desired new size in bytes; 0 may free the block (platform-defined)

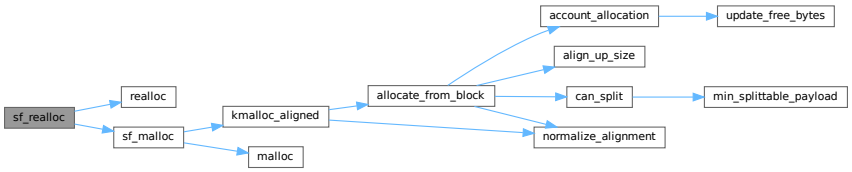
Returns

Pointer to resized memory, or NULL on failure (original block unchanged)

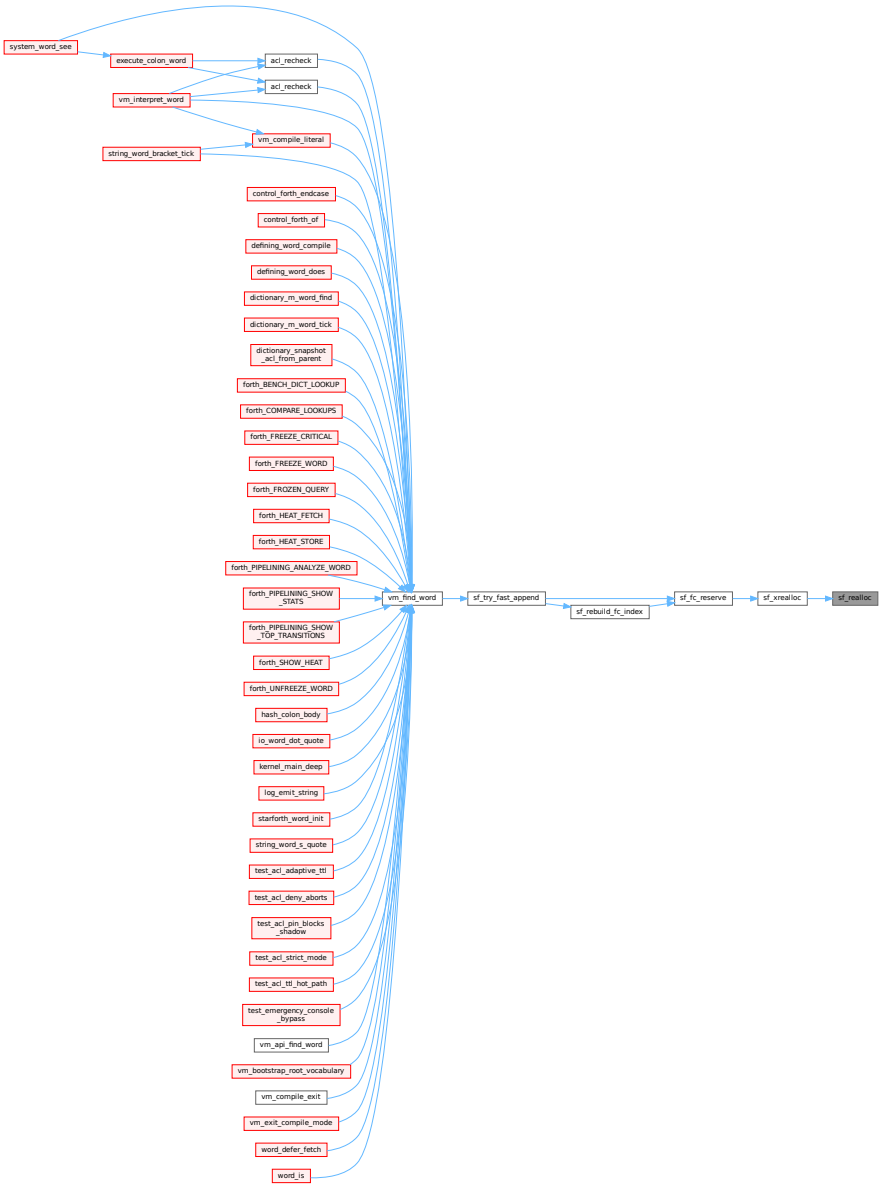
References sf_alloc_stats::alloc_count, g_stats, realloc(), and sf_malloc().

Referenced by sf_xrealloc().

Here is the call graph for this function:



Here is the caller graph for this function:



7.297.3 Variable Documentation

7.297.3.1 g_alloc_count

```
size_t g_alloc_count = 0  [static]
```

Referenced by sf_alloc_get_stats(), and sf_malloc().

7.297.3.2 g_free_count

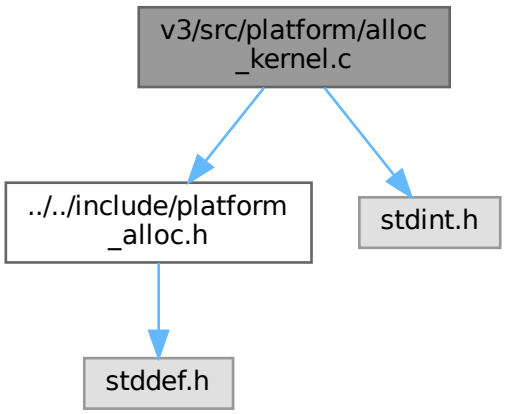
```
size_t g_free_count = 0  [static]
```

Referenced by sf_alloc_get_stats(), and sf_free().

7.298 v3/src/platform/alloc_kernel.c File Reference

```
#include "../../include/platform_alloc.h"
#include <stdint.h>
```

Include dependency graph for alloc_kernel.c:



Macros

- #define SF_ARENA_SIZE (4 ∗ 1024 ∗ 1024) /∗ 4MB default ∗/
- #define SF_ALIGN 8
- #define SF_ALIGN_UP(x) (((x) + (SF_ALIGN - 1)) & ∼(SF_ALIGN - 1))

Functions

- int sf_alloc_init (void)

  Initialise the kernel bump allocator.
- void ∗ sf_malloc (size_t size)

  Allocate `size` bytes from the system heap.
- void ∗ sf_calloc (size_t count, size_t size)

  Allocate a zero-initialised array of `count` elements of `size` bytes each.
- void ∗ sf_realloc (void ∗ptr, size_t new_size)

  Resize a previously allocated block to `new_size` bytes.
- void sf_free (void ∗ptr)

  Release a previously allocated block back to the system heap.
- void sf_alloc_get_stats (sf_alloc_stats_t ∗stats)

  Copy the current allocator statistics into `stats`.

Variables

- static uint8_t g_arena[(4 ∗1024 ∗1024) __attribute__)((aligned(8)))
- static size_t g_arena_offset = 0
- static int g_initialized = 0
- static sf_alloc_stats_t g_stats = {0}

7.298.1 Macro Definition Documentation

7.298.1.1 SF_ALIGN

```
#define SF_ALIGN 8
```

7.298.1.2 SF_ALIGN_UP

```
#define SF_ALIGN_UP(
              x ) (((x) + (SF_ALIGN - 1)) & ~(SF_ALIGN - 1))
```

7.298.1.3 SF_ARENA_SIZE

```
#define SF_ARENA_SIZE (4 * 1024 * 1024) /* 4MB default */
```

7.298.2 Function Documentation

7.298.2.1 sf_alloc_get_stats()

```
void sf_alloc_get_stats (
             sf_alloc_stats_t * stats )
```

Copy the current allocator statistics into `stats`.

Fills `*stats` with a snapshot of `g_stats`. Note that `used_bytes` overstates the live total when blocks have been freed or reallocated, because freed sizes are not tracked. `free_count` is accurate.

Parameters

stats	Output buffer to populate; no-op if NULL

References g_stats.

7.298.2.2 sf_alloc_init()

```
int sf_alloc_init (
             void  )
```

Initialise the kernel bump allocator.

Initialise the host allocator statistics.

Resets `g_arena_offset` to zero and clears all `g_stats` fields. Sets `g_initialized` so that `sf_malloc()` skips the lazy-init guard. Safe to call at any point; all prior allocations become invalid after a reset, so this must only be called before any VM allocation takes place.

Returns

0 always (cannot fail)

References sf_alloc_stats::alloc_count, sf_alloc_stats::free_count, g_arena_offset, g_initialized, g_stats, sf_alloc_stats::peak_bytes, SF_ARENA_SIZE, sf_alloc_stats::total_bytes, and sf_alloc_stats::used_bytes.

7.298.2.3 sf_calloc()

```
void * sf_calloc (
            size_t count,
            size_t size )
```

Allocate a zero-initialised array of `count` elements of `size` bytes each.

Delegates to `calloc()`. Updates statistics the same way as `sf_malloc()`: adds `count*size` to `used_bytes`, increments `alloc_count`, and updates `peak_bytes`.

Parameters

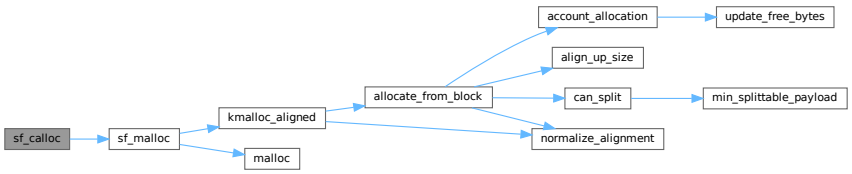
count	Number of elements; 0 returns NULL
size	Size of each element in bytes; 0 returns NULL

Returns

Pointer to zero-filled memory, or NULL on allocation failure or zero arguments

References count, and sf_malloc().

Here is the call graph for this function:



7.298.2.4 sf_free()

```
void sf_free (
            void * ptr )
```

Release a previously allocated block back to the system heap.

Delegates to `free()`. Increments `g_stats.free_count` on non-NULL `ptr`. The freed byte count is not subtracted from `used_bytes` because the original size is not tracked — see `sf_realloc()` note.

Parameters

ptr	Pointer to free; NULL is a safe no-op

References sf_alloc_stats::free_count, and g_stats.

7.298.2.5 sf_malloc()

```
void * sf_malloc (
            size_t size )
```

Allocate `size` bytes from the system heap.

Delegates to `malloc()`. On success, increments `g_stats.used_bytes` and `g_stats.alloc_count`, and updates `g_stats.peak_bytes` if the new used total exceeds the recorded peak.

Parameters

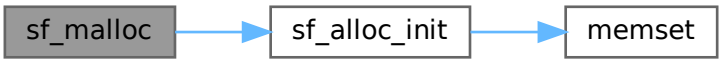
size	Number of bytes to allocate; 0 returns NULL

Returns

Pointer to allocated memory, or NULL on allocation failure or zero size

References sf_alloc_stats::alloc_count, g_arena_offset, g_initialized, g_stats, sf_alloc_stats::peak_bytes, SF_ALIGN_UP, sf_alloc_init(), SF_ARENA_SIZE, and sf_alloc_stats::used_bytes.

Here is the call graph for this function:



7.298.2.6 sf_realloc()

```
void * sf_realloc (
            void * ptr,
            size_t new_size )
```

Resize a previously allocated block to `new_size` bytes.

Delegates to `realloc()`. On success, increments `g_stats.alloc_count`. The exact byte delta is not tracked because the original allocation size is not stored alongside the pointer — `used_bytes` is therefore approximate after any `sf_realloc()` call. For accurate accounting, wrap allocations with size-tracking headers.

Parameters

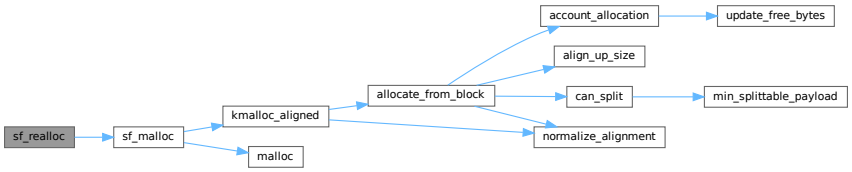
ptr	Pointer to an existing allocation, or NULL (behaves like `sf_malloc`)
new_size	Desired new size in bytes; 0 may free the block (platform-defined)

Returns

Pointer to resized memory, or NULL on failure (original block unchanged)

References sf_malloc().

Here is the call graph for this function:



7.298.3 Variable Documentation

7.298.3.1 __attribute__

```
uint8_t g_arena[(4 *1024 *1024) __attribute__) ((aligned(8))) (
            (aligned(8))  )  [static]
```

7.298.3.2 g_arena_offset

```
size_t g_arena_offset = 0  [static]
```

Referenced by sf_alloc_init(), and sf_malloc().

7.298.3.3 g_initialized

```
int g_initialized = 0  [static]
```

Referenced by sf_alloc_init(), and sf_malloc().

7.298.3.4 g_stats

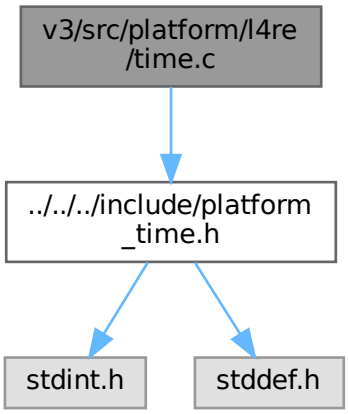
```
sf_alloc_stats_t g_stats = {0}  [static]
```

Referenced by sf_alloc_get_stats(), sf_alloc_init(), sf_free(), and sf_malloc().

7.299 v3/src/platform/l4re/time.c File Reference

```
#include "../../../include/platform_time.h"
```

Include dependency graph for time.c:



Functions

- void sf_time_init_l4re (void)

Variables

- const sf_time_backend_t sf_time_backend_l4re = {0}

7.299.1 Function Documentation

7.299.1.1 sf_time_init_l4re()

```
void sf_time_init_l4re (
            void  )
```

7.299.2 Variable Documentation

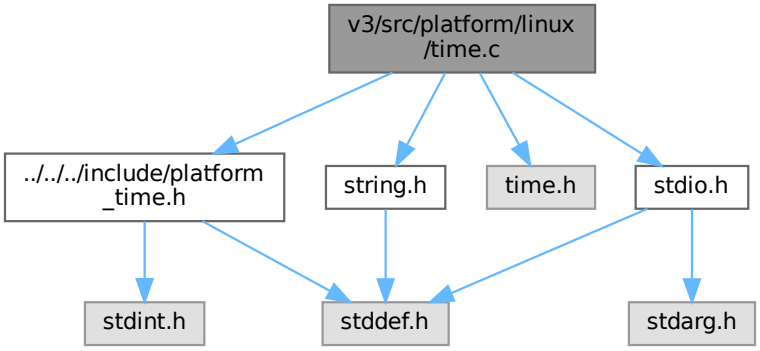
7.299.2.1 sf_time_backend_l4re

```
const sf_time_backend_t sf_time_backend_l4re = {0}
```

7.300 v3/src/platform/linux/time.c File Reference

```
#include "../../../include/platform_time.h"
#include <stdio.h>
#include <time.h>
#include <string.h>
```

Include dependency graph for time.c:



Macros

- #define _POSIX_C_SOURCE 199309L

Functions

- static sf_time_ns_t posix_get_monotonic_ns (void)

  Return the current monotonic time in nanoseconds.
- static sf_time_ns_t posix_get_realtime_ns (void)

  Return the current wall-clock time in nanoseconds since the Unix epoch.
- static int posix_set_realtime_ns (sf_time_ns_t ns_since_epoch)

  Set the system real-time clock to `ns_since_epoch`.
- static int posix_format_timestamp (sf_time_ns_t ns_since_epoch, char ∗buf, int format_24h)

  Format a nanosecond-precision wall-clock time as a human-readable string.
- static int posix_has_rtc (void)

  Probe whether a real-time clock is available.

Variables

- const sf_time_backend_t sf_time_backend_posix

7.300.1 Macro Definition Documentation

7.300.1.1 _POSIX_C_SOURCE

```
#define _POSIX_C_SOURCE 199309L
```

7.300.2 Function Documentation

7.300.2.1 posix_format_timestamp()

```
static int posix_format_timestamp (
            sf_time_ns_t ns_since_epoch,
            char * buf,
            int format_24h )  [static]
```

Format a nanosecond-precision wall-clock time as a human-readable string.

Converts `ns_since_epoch` to local time, formats hours/minutes/seconds via `strftime()`, then appends a three-digit millisecond suffix. The result is written into `buf` (must be at least `SF_TIME_STAMP_SIZE` bytes). On `localtime()` or `strftime()` failure, writes `"??:??:??.???"` into `buf` and returns -1.

Parameters

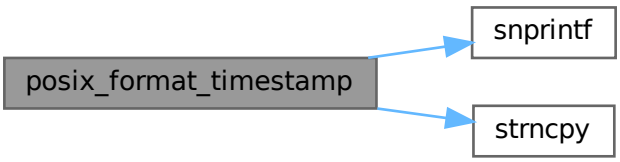
ns_since_epoch	Wall-clock time in nanoseconds since Unix epoch
buf	Output buffer (at least `SF_TIME_STAMP_SIZE` bytes); must not be NULL
format_24h	Non-zero for 24-hour format (`H`:M:S), zero for 12-hour (`I`:M:S p)

Returns

0 on success, -1 if `buf` is NULL or time formatting fails

References SF_TIME_STAMP_SIZE, snprintf(), and strncpy().

Here is the call graph for this function:



7.300.2.2 posix_get_monotonic_ns()

```
static sf_time_ns_t posix_get_monotonic_ns (
            void  )  [static]
```

Return the current monotonic time in nanoseconds.

Reads `CLOCK_MONOTONIC` via `clock_gettime()`. The clock advances continuously and is unaffected by wall-clock adjustments — suitable for measuring elapsed durations such as execution latency and decay intervals. Returns 0 on `clock_gettime()` failure (should not occur on any modern POSIX kernel).

Returns

Nanoseconds since an arbitrary but fixed epoch, or 0 on error

References timespec::tv_nsec, and timespec::tv_sec.

7.300.2.3 posix_get_realtime_ns()

```
static sf_time_ns_t posix_get_realtime_ns (
            void  )  [static]
```

Return the current wall-clock time in nanoseconds since the Unix epoch.

Reads CLOCK_REALTIME via clock_gettime(). Subject to NTP or settimeofday() adjustments — use posix_get_monotonic_ns() for interval timing. Returns 0 on clock_gettime() failure.

Returns

Nanoseconds since 1970-01-01 00:00:00 UTC, or 0 on error

References timespec::tv_nsec, and timespec::tv_sec.

7.300.2.4 posix_has_rtc()

```
static int posix_has_rtc (
            void  )  [static]
```

Probe whether a real-time clock is available.

Attempts a clock_gettime(CLOCK_REALTIME) probe. POSIX systems invariably support CLOCK_↩ REALTIME so this will return 1 in practice; the check mirrors the bare-metal backend's interface where RTC presence is genuinely uncertain.

Returns

1 if CLOCK_REALTIME is readable, 0 otherwise

7.300.2.5 posix_set_realtime_ns()

```
static int posix_set_realtime_ns (
            sf_time_ns_t ns_since_epoch )  [static]
```

Set the system real-time clock to ns_since_epoch.

Converts ns_since_epoch to a struct timespec and calls clock_settime(CLOCK_REALTIME). Requires CAP_SYS_TIME or equivalent privilege; fails with EPERM if the caller lacks permission.

Parameters

ns_since_epoch	Desired new wall-clock time in nanoseconds since Unix epoch

Returns

0 on success, non-zero errno-encoded error on failure

References timespec::tv_nsec, and timespec::tv_sec.

7.300.3 Variable Documentation

7.300.3.1 sf_time_backend_posix

```
const sf_time_backend_t sf_time_backend_posix
```

Initial value:

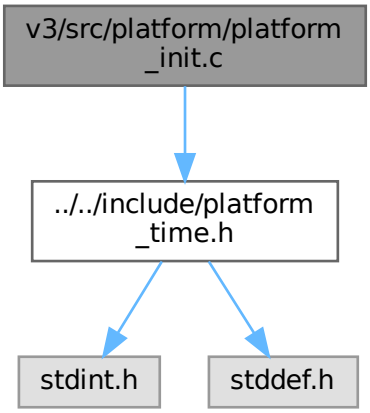
```
= {
    .get_monotonic_ns = posix_get_monotonic_ns,
    .get_realtime_ns = posix_get_realtime_ns,
    .set_realtime_ns = posix_set_realtime_ns,
    .format_timestamp = posix_format_timestamp,
    .has_rtc = posix_has_rtc,
}
```

Referenced by sf_time_init().

7.301 v3/src/platform/platform_init.c File Reference

```
#include "../../include/platform_time.h"
```

Include dependency graph for platform_init.c:



Functions

- void sf_time_init (void)

  Initialize platform time subsystem.

Variables

- const sf_time_backend_t * sf_time_backend = NULL

7.301.1 Function Documentation

7.301.1.1 sf_time_init()

```
void sf_time_init (
            void  )
```

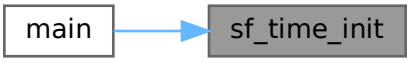
Initialize platform time subsystem.

Selects the POSIX backend (clock_gettime()). Call this once at program startup before using any time functions.

References sf_time_backend, and sf_time_backend_posix.

Referenced by main().

Here is the caller graph for this function:



7.301.2 Variable Documentation

7.301.2.1 sf_time_backend

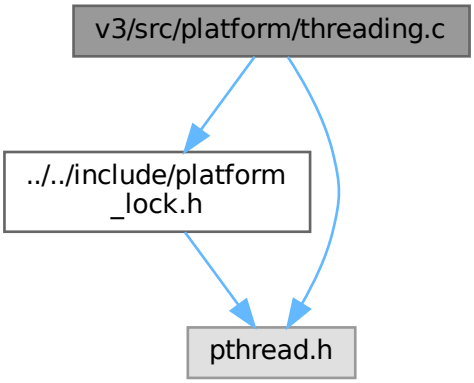
```
const sf_time_backend_t* sf_time_backend = NULL
```

Referenced by sf_format_timestamp(), sf_has_rtc(), sf_monotonic_ns(), sf_realtime_ns(), sf_set_realtime_ns(), and sf_time_init().

7.302 v3/src/platform/threading.c File Reference

```
#include "../../include/platform_lock.h"
#include <pthread.h>
```

Include dependency graph for threading.c:



Functions

- int sf_mutex_init (sf_mutex_t *mutex)

  Initialise a POSIX mutex (hosted build).
- void sf_mutex_destroy (sf_mutex_t *mutex)

  Destroy a POSIX mutex (hosted build).
- void sf_mutex_lock (sf_mutex_t *mutex)

  Acquire a POSIX mutex (hosted build).
- void sf_mutex_unlock (sf_mutex_t *mutex)

  Release a POSIX mutex (hosted build).

7.302.1 Function Documentation

7.302.1.1 sf_mutex_destroy()

```
void sf_mutex_destroy (
            sf_mutex_t * mutex )
```

Destroy a POSIX mutex (hosted build).

Delegates to `pthread_mutex_destroy()`. The mutex must not be locked when destroyed. Return value is discarded; callers that need the error code should call `pthread_mutex_destroy()` directly.

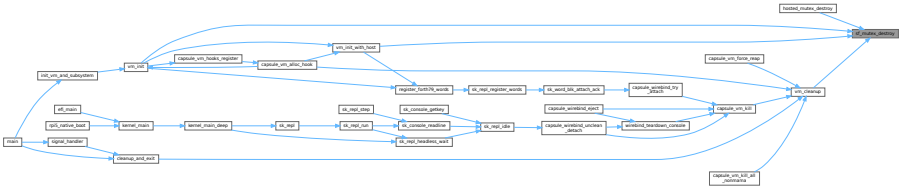
Parameters

mutex	Mutex to destroy; no-op if NULL

References sf_mutex::handle.

Referenced by hosted_mutex_destroy(), vm_cleanup(), vm_init(), and vm_init_with_host().

Here is the caller graph for this function:



7.302.1.2 sf_mutex_init()

```
int sf_mutex_init (
            sf_mutex_t * mutex )
```

Initialise a POSIX mutex (hosted build).

Delegates to `pthread_mutex_init()` with default attributes.

Parameters

mutex	Mutex to initialise; returns -1 if NULL

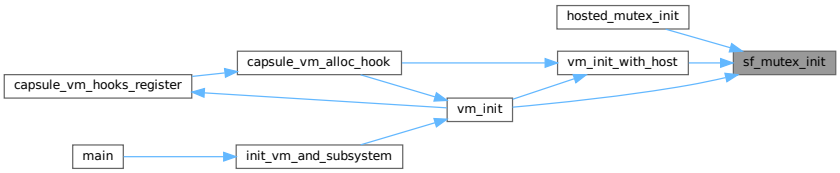
Returns

0 on success, -1 if `mutex` is NULL, or a positive `errno` value propagated from `pthread_mutex_init()`

References sf_mutex::handle.

Referenced by hosted_mutex_init(), vm_init(), and vm_init_with_host().

Here is the caller graph for this function:



7.302.1.3 sf_mutex_lock()

```
void sf_mutex_lock (
            sf_mutex_t * mutex )
```

Acquire a POSIX mutex (hosted build).

Delegates to `pthread_mutex_lock()`. Blocks until the mutex is available. Return value is discarded; a locking error will manifest as a deadlock or data race, not a silent failure.

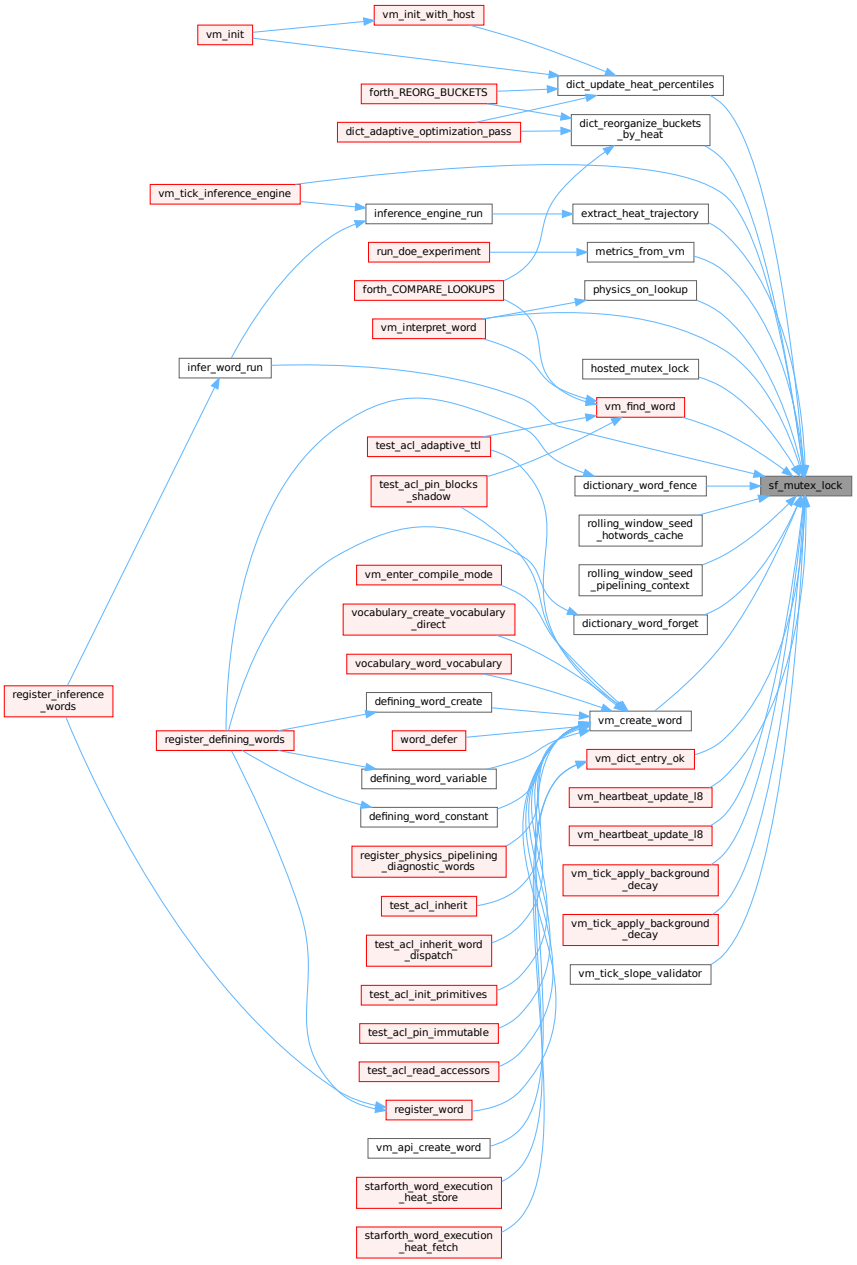
Parameters

mutex	Mutex to lock; no-op if NULL

References sf_mutex::handle.

Referenced by dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), dictionary_word_fence(), dictionary_word_forget(), extract_heat_trajectory(), hosted_mutex_lock(), infer_word_run(), metrics_from_vm(), physics_on_lookup(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), vm_create_word(), vm_dict_entry_ok(), vm_find_word(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_interpret_word(), vm_tick_apply_background_decay(), vm_tick_apply_background_decay(), vm_tick_inference_engine(), and vm_tick_slope_validator().

Here is the caller graph for this function:



7.302.1.4 sf_mutex_unlock()

```
void sf_mutex_unlock (
            sf_mutex_t * mutex )
```

Release a POSIX mutex (hosted build).

Delegates to `pthread_mutex_unlock()`. Must be called by the same thread that acquired the lock. Return value is discarded.

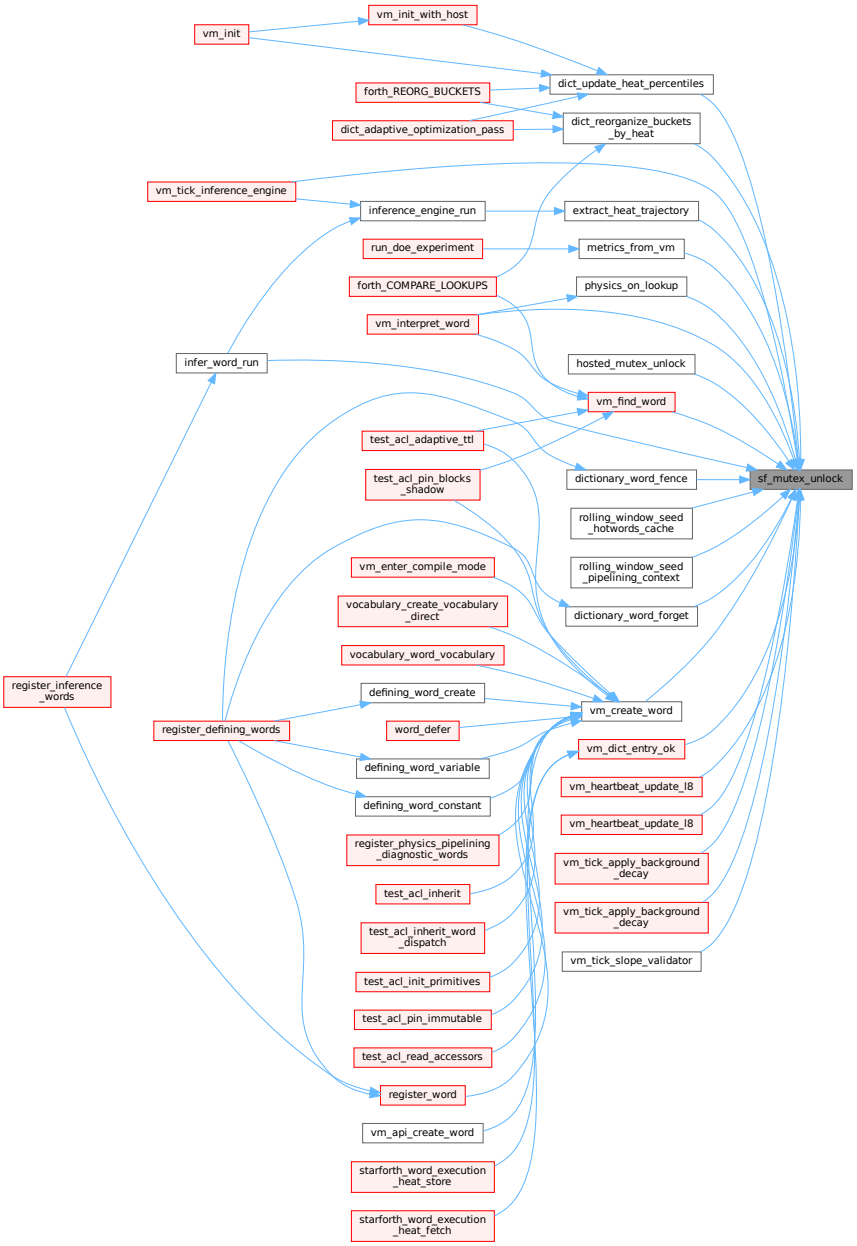
Parameters

mutex	Mutex to unlock; no-op if NULL

References sf_mutex::handle.

Referenced by dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), dictionary_word_fence(), dictionary_word_forget(), extract_heat_trajectory(), hosted_mutex_unlock(), infer_word_run(), metrics_from_vm(), physics_on_lookup(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), vm_create_word(), vm_dict_entry_ok(), vm_find_word(), vm_heartbeat_update_l8(), vm_heartbeat_update_l8(), vm_interpret_word(), vm_tick_apply_background_decay(), vm_tick_apply_background_decay(), vm_tick_inference_engine(), and vm_tick_slope_validator().

Here is the caller graph for this function:



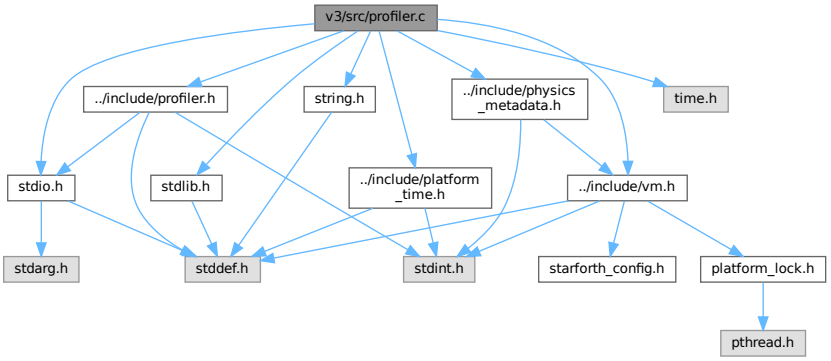
7.303 v3/src/profiler.c File Reference

Performance profiling implementation for StarForth.

```
#include "../include/profiler.h"
#include "../include/platform_time.h"
#include "../include/vm.h"
#include "../include/physics_metadata.h"
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <time.h>
```

Include dependency graph for profiler.c:



Data Structures

- struct WordStats

  Statistics for individual word execution.

Macros

- #define _POSIX_C_SOURCE 199309L

Typedefs

- typedef struct DictEntry DictEntry

Functions

- static uint64_t get_time_ns (void)

  Get current time in nanoseconds.
- static void print_word_name_safe (const DictEntry *entry)

  Safely print a word name, handling non-null-terminated names.
- int profiler_init (ProfileLevel level)

  Initialize the profiling system.
- void profiler_shutdown (void)

  Shutdown the profiler and free resources.
- static WordStats * get_word_stats (const DictEntry *entry)

  Find or create statistics entry for a word.
- void profiler_word_enter (const DictEntry *entry)

  Start timing a word execution (PROFILE_DETAILED+)
- void profiler_word_exit (const DictEntry *entry)

  End timing a word execution (PROFILE_DETAILED+)
- void profiler_word_count (const DictEntry *entry)

  Track word execution frequency without timing (PROFILE_BASIC+)

- void profiler_inc_dict_lookup (void)
- void profiler_inc_stack_op (void)
- void profiler_memory_read (size_t bytes)
- void profiler_memory_write (size_t bytes)
- static int compare_by_time (const void *a, const void *b)

  Compare two WordStats entries by total execution time.
- static int compare_by_calls (const void *a, const void *b)

  Compare two WordStats entries by call count.
- void profiler_generate_report (void)

  Generate and print detailed profiling report.
- void profiler_reset (void)

  Reset all profiling statistics.
- void profiler_set_level (ProfileLevel level)

  Change the current profiling detail level.
- uint64_t profiler_get_time_ns (void)
- void profiler_start_timer (ProfileCategory category)
- void profiler_end_timer (ProfileCategory category)
- void profiler_memory_alloc (size_t bytes)
- void profiler_inc_vm_cycles (uint64_t cycles)
- void profiler_inc_branch (void)
- void profiler_inc_loop_iter (void)
- void profiler_print_summary (void)
- void profiler_print_hotspots (void)

  Print hot word analysis with optimization recommendations.
- void profiler_print_word_analysis (void)
- void profiler_print_memory_analysis (void)
- void profiler_analyze_performance (VM *vm)
- void profiler_suggest_optimizations (void)

Variables

- struct {
    ProfileLevel level
    int enabled
    WordStats * word_stats
    size_t word_count
    size_t word_capacity
    const char * current_word
    uint64_t current_start_time
    uint64_t total_dict_lookups
    uint64_t total_stack_ops
    uint64_t total_memory_reads
    uint64_t total_memory_writes
  } profiler_state = {0}

- void * g_profiler = &profiler_state

7.303.1 Detailed Description

Performance profiling implementation for StarForth.

This module implements performance profiling functionality for the StarForth virtual machine. It tracks execution statistics including:

- Word execution counts and timing
- Dictionary lookup frequency
- Stack operations
- Memory usage patterns

The profiler supports multiple detail levels from basic to verbose profiling.

7.303.2 Macro Definition Documentation

7.303.2.1 _POSIX_C_SOURCE

```
#define _POSIX_C_SOURCE 199309L
```

7.303.3 Typedef Documentation

7.303.3.1 DictEntry

```
typedef struct DictEntry DictEntry
```

7.303.4 Function Documentation

7.303.4.1 compare_by_calls()

```
static int compare_by_calls (
            const void * a,
            const void * b )  [static]
```

Compare two WordStats entries by call count.

Parameters

a	First WordStats entry to compare
b	Second WordStats entry to compare

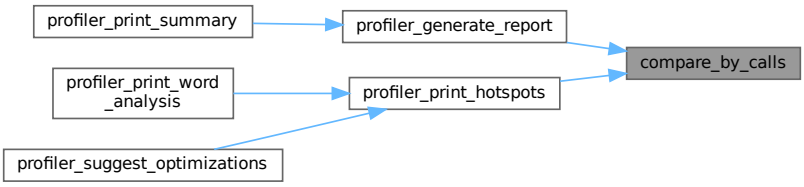
Returns

-1, 0, or 1 for qsort ordering

References WordStats::call_count.

Referenced by profiler_generate_report(), and profiler_print_hotspots().

Here is the caller graph for this function:



7.303.4.2 compare_by_time()

```
static int compare_by_time (
            const void * a,
            const void * b )  [static]
```

Compare two WordStats entries by total execution time.

Parameters

a	First WordStats entry to compare
b	Second WordStats entry to compare

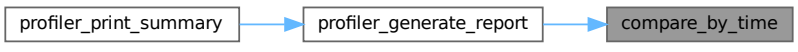
Returns

-1, 0, or 1 for qsort ordering

References WordStats::total_time_ns.

Referenced by profiler_generate_report().

Here is the caller graph for this function:



7.303.4.3 get_time_ns()

```
static uint64_t get_time_ns (
            void  )  [static]
```

Get current time in nanoseconds.

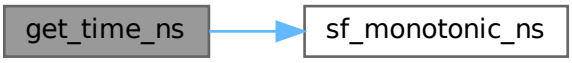
Returns

Current time in nanoseconds from monotonic clock

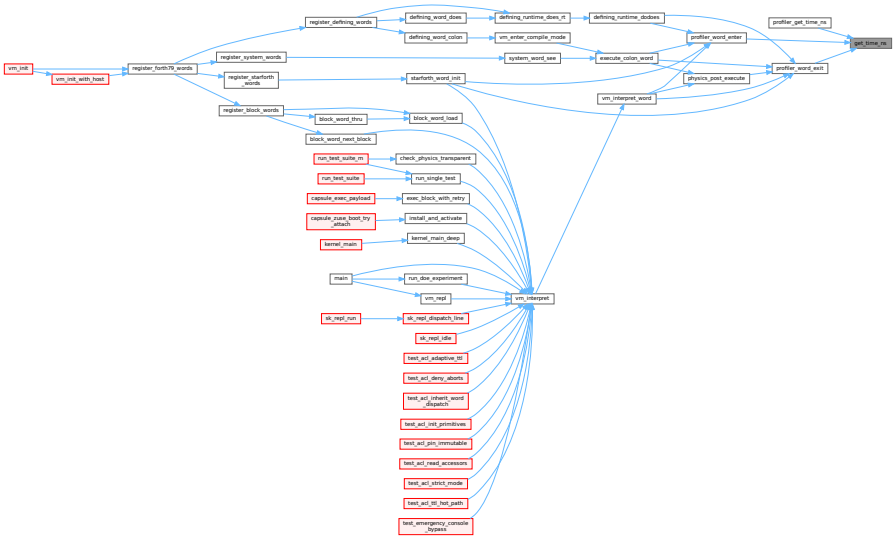
References sf_monotonic_ns().

Referenced by profiler_get_time_ns(), profiler_word_enter(), and profiler_word_exit().

Here is the call graph for this function:



Here is the caller graph for this function:



7.303.4.4 get_word_stats()

```
static WordStats * get_word_stats (
            const DictEntry * entry )  [static]
```

Find or create statistics entry for a word.

Parameters

entry	Dictionary entry to find/create stats for

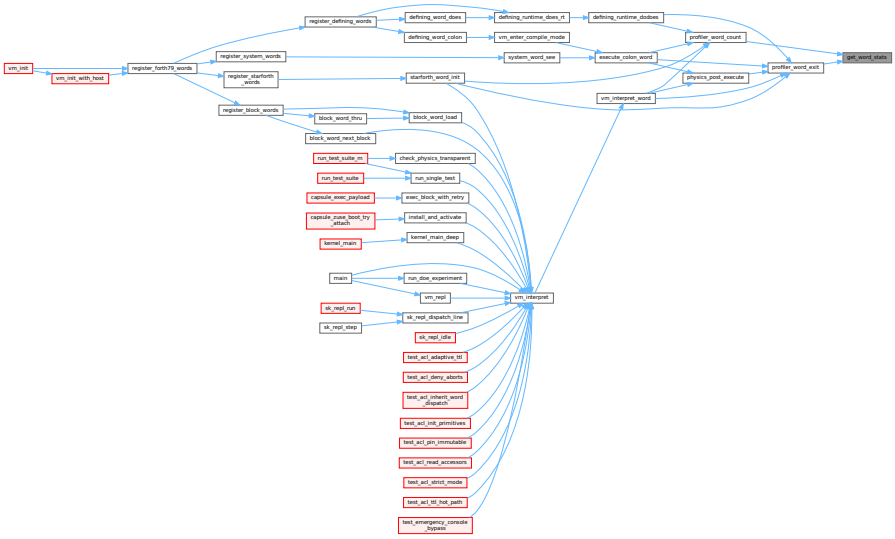
Returns

Pointer to word stats or NULL if not found/created

References WordStats::call_count, entry, WordStats::entry, WordStats::max_time_ns, WordStats::min_time_ns, profiler_state, and WordStats::total_time_ns.

Referenced by profiler_word_count(), and profiler_word_exit().

Here is the caller graph for this function:



7.303.4.5 print_word_name_safe()

```
static void print_word_name_safe (
            const DictEntry * entry )  [static]
```

Safely print a word name, handling non-null-terminated names.

Parameters

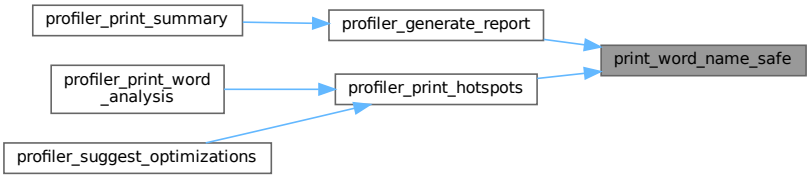
entry	Dictionary entry whose name to print

References entry, printf(), and putchar().

Referenced by profiler_generate_report(), and profiler_print_hotspots().

Here is the call graph for this function:

Here is the caller graph for this function:



7.303.4.6 profiler_analyze_performance()

```
void profiler_analyze_performance (
            VM * vm )
```

7.303.4.7 profiler_end_timer()

```
void profiler_end_timer (
            ProfileCategory category )
```

7.303.4.8 profiler_generate_report()

```
void profiler_generate_report (
            void  )
```

Generate and print detailed profiling report.

Generates a report including:

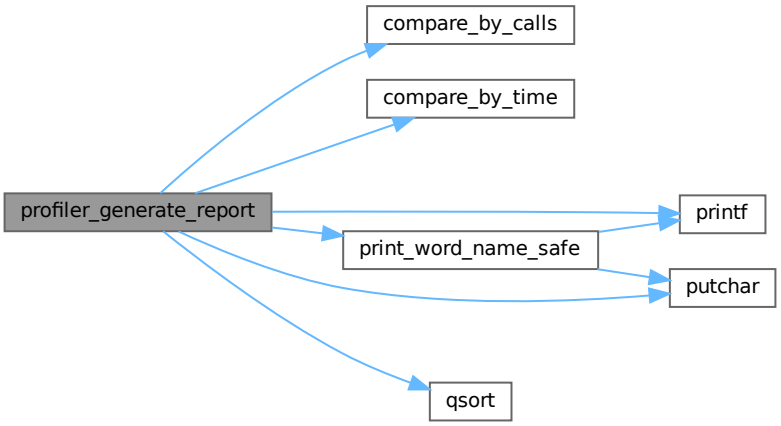
- Global statistics (dictionary lookups, stack ops, memory usage)
- Top words by execution time

- Most frequently called words

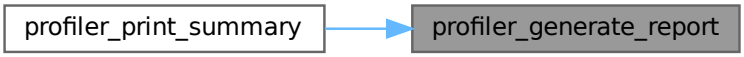
References WordStats::call_count, compare_by_calls(), compare_by_time(), WordStats::entry, limit, WordStats::max_time_ns, name_len, DictEntry::name_len, print_word_name_safe(), printf(), PROFILE_BASIC, PROFILE_DETAILED, PROFILE_VERBOSE, profiler_state, putchar(), qsort(), and WordStats::total_time_ns.

Referenced by profiler_print_summary().

Here is the call graph for this function:



Here is the caller graph for this function:

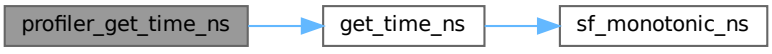


7.303.4.9 profiler_get_time_ns()

```
uint64_t profiler_get_time_ns (
            void  )
```

References get_time_ns().

Here is the call graph for this function:



7.303.4.10 profiler_inc_branch()

```
void profiler_inc_branch (
            void  )
```

7.303.4.11 profiler_inc_dict_lookup()

```
void profiler_inc_dict_lookup (
            void  )
```

References PROFILE_BASIC, and profiler_state.

7.303.4.12 profiler_inc_loop_iter()

```
void profiler_inc_loop_iter (
            void  )
```

7.303.4.13 profiler_inc_stack_op()

```
void profiler_inc_stack_op (
            void  )
```

References PROFILE_VERBOSE, and profiler_state.

7.303.4.14 profiler_inc_vm_cycles()

```
void profiler_inc_vm_cycles (
            uint64_t cycles )
```

7.303.4.15 profiler_init()

```
int profiler_init (
            ProfileLevel level )
```

Initialize the profiling system.

Parameters

level	Initial profiling detail level

Returns

0 on success, non-zero on failure

References calloc(), level, PROFILE_DISABLED, and profiler_state.

Here is the call graph for this function:

7.303.4.16 profiler_memory_alloc()

```
void profiler_memory_alloc (
            size_t bytes )
```

References bytes.

7.303.4.17 profiler_memory_read()

```
void profiler_memory_read (
            size_t bytes )
```

References bytes, PROFILE_VERBOSE, and profiler_state.

7.303.4.18 profiler_memory_write()

```
void profiler_memory_write (
            size_t bytes )
```

References bytes, PROFILE_VERBOSE, and profiler_state.

7.303.4.19 profiler_print_hotspots()

```
void profiler_print_hotspots (
            void  )
```

Print hot word analysis with optimization recommendations.

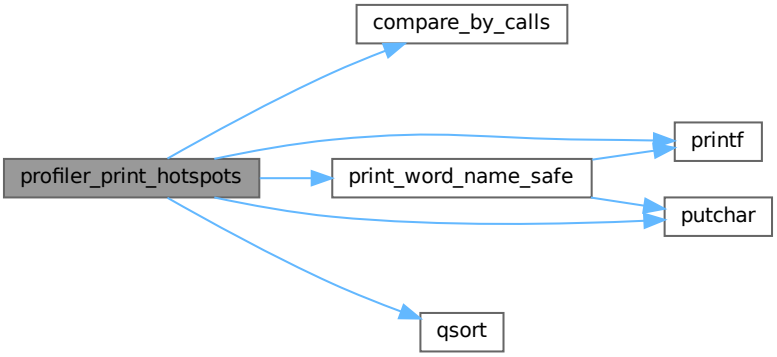
Analyzes word execution frequency and provides actionable recommendations for performance optimization including:

- Assembly optimization candidates
- Inlining opportunities
- Cache-friendly ordering

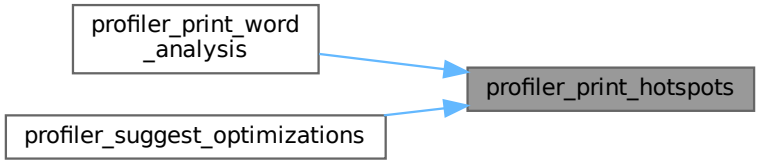
References WordStats::call_count, compare_by_calls(), WordStats::entry, limit, name_len, DictEntry::name_len, print_word_name_safe(), printf(), profiler_state, putchar(), and qsort().

Referenced by profiler_print_word_analysis(), and profiler_suggest_optimizations().

Here is the call graph for this function:



Here is the caller graph for this function:



7.303.4.20 profiler_print_memory_analysis()

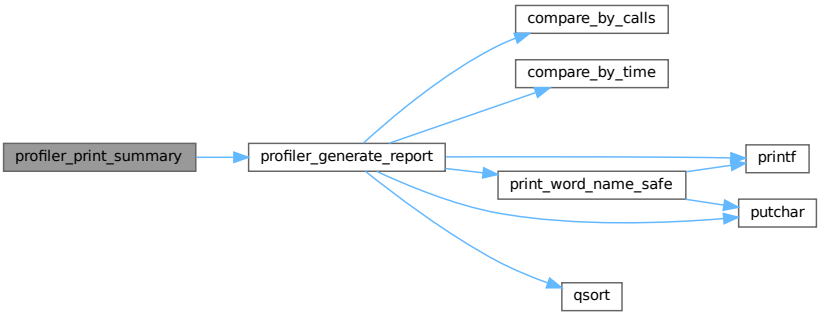
```
void profiler_print_memory_analysis (
            void  )
```

7.303.4.21 profiler_print_summary()

```
void profiler_print_summary (
            void  )
```

References profiler_generate_report().

Here is the call graph for this function:

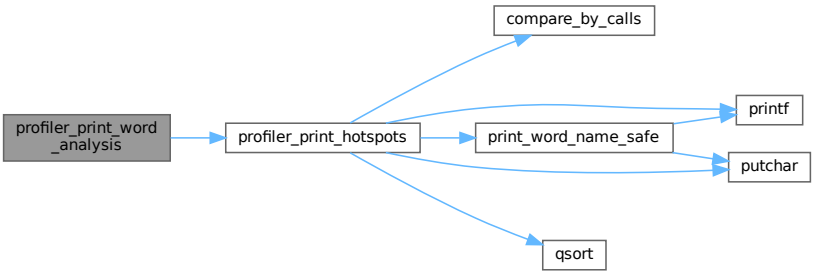


7.303.4.22 profiler_print_word_analysis()

```
void profiler_print_word_analysis (
            void  )
```

References profiler_print_hotspots().

Here is the call graph for this function:



7.303.4.23 profiler_reset()

```
void profiler_reset (
            void  )
```

Reset all profiling statistics.

References profiler_state.

7.303.4.24 profiler_set_level()

```
void profiler_set_level (
            ProfileLevel level )
```

Change the current profiling detail level.

Parameters

level	New profiling level to set

References level, and profiler_state.

7.303.4.25 profiler_shutdown()

```
void profiler_shutdown (
            void  )
```

Shutdown the profiler and free resources.

Shutdown and cleanup the profiling system.

References free(), and profiler_state.

Here is the call graph for this function:

7.303.4.26 profiler_start_timer()

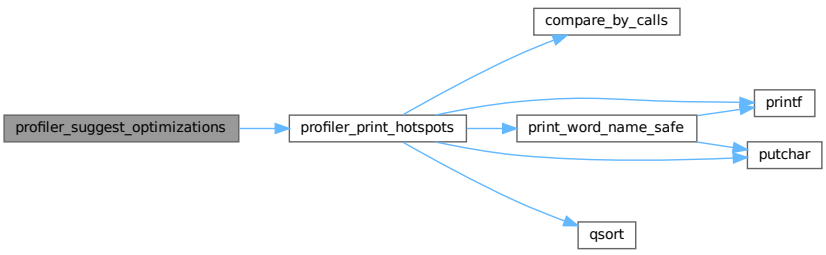
```
void profiler_start_timer (
            ProfileCategory category )
```

7.303.4.27 profiler_suggest_optimizations()

```
void profiler_suggest_optimizations (
            void  )
```

References profiler_print_hotspots().

Here is the call graph for this function:



7.303.4.28 profiler_word_count()

```
void profiler_word_count (
            const DictEntry * entry )
```

Track word execution frequency without timing (PROFILE_BASIC+)

Parameters

entry	Dictionary entry of the word being executed

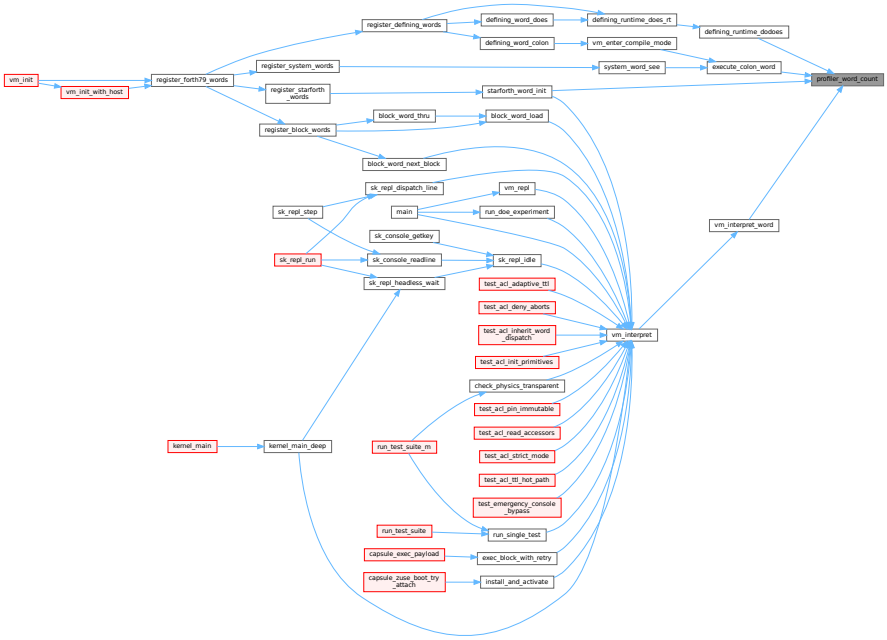
This is a lightweight alternative to profiler_word_enter/exit that only increments call counters without timing overhead. Ideal for identifying hot words (frequently executed) for optimization.

References WordStats::call_count, entry, get_word_stats(), PROFILE_DETAILED, and profiler_state.

Referenced by defining_runtime_dodoes(), execute_colon_word(), starforth_word_init(), and vm_interpret_word().

Here is the call graph for this function:

Here is the caller graph for this function:



7.303.4.29 profiler_word_enter()

```
void profiler_word_enter (
            const DictEntry * entry )
```

Start timing a word execution (PROFILE_DETAILED+)

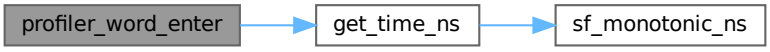
Parameters

entry	Dictionary entry of the word being executed

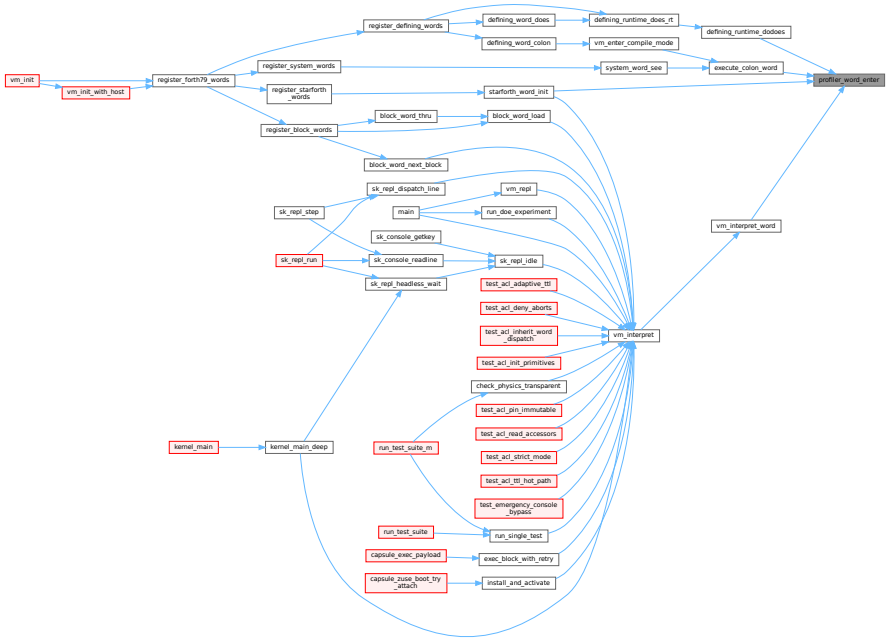
References entry, get_time_ns(), PROFILE_DETAILED, and profiler_state.

Referenced by defining_runtime_dodoes(), execute_colon_word(), starforth_word_init(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.303.4.30 profiler_word_exit()

```
void profiler_word_exit (
            const DictEntry * entry )
```

End timing a word execution (PROFILE_DETAILED+)

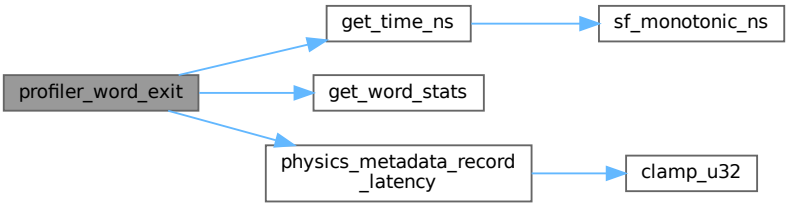
Parameters

entry	Dictionary entry of the word being executed

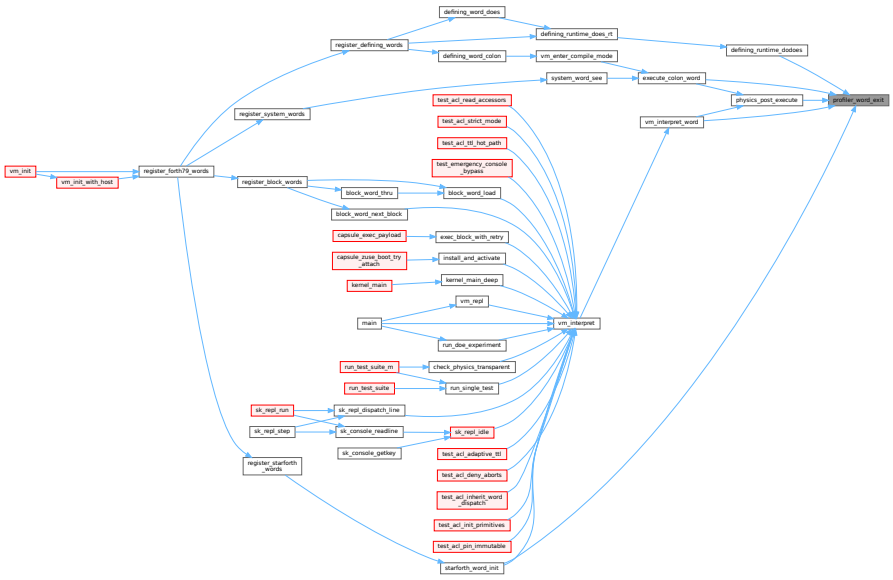
References WordStats::call_count, entry, get_time_ns(), get_word_stats(), WordStats::max_time_ns, WordStats::min_time_ns, physics_metadata_record_latency(), PROFILE_DETAILED, profiler_state, and WordStats::total_time_ns.

Referenced by defining_runtime_dodoes(), execute_colon_word(), physics_post_execute(), starforth_word_init(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.303.5 Variable Documentation

7.303.5.1 current_start_time

```
uint64_t current_start_time
```

7.303.5.2 current_word

```
const char* current_word
```

7.303.5.3 enabled

```
int enabled
```

Referenced by hotwords_cache_set_enabled(), and set_forth_79_compliance().

7.303.5.4 g_profiler

```
void* g_profiler = &profiler_state
```

7.303.5.5 level

```
ProfileLevel level
```

Referenced by log_do_emit(), log_emit_string(), log_message(), log_region_append(), log_set_level(), log_str_emit(), profiler_init(), and profiler_set_level().

7.303.5.6 [struct]

```
struct { ... } profiler_state [static]
```

Referenced by get_word_stats(), profiler_generate_report(), profiler_inc_dict_lookup(), profiler_inc_stack_op(), profiler_init(), profiler_memory_read(), profiler_memory_write(), profiler_print_hotspots(), profiler_reset(), profiler_set_level(), profiler_shutdown(), profiler_word_count(), profiler_word_enter(), and profiler_word_exit().

7.303.5.7 total_dict_lookups

```
uint64_t total_dict_lookups
```

7.303.5.8 total_memory_reads

```
uint64_t total_memory_reads
```

7.303.5.9 total_memory_writes

```
uint64_t total_memory_writes
```

7.303.5.10 total_stack_ops

```
uint64_t total_stack_ops
```

7.303.5.11 word_capacity

```
size_t word_capacity
```

7.303.5.12 word_count

```
size_t word_count
```

Referenced by forth_PIPELINING_STATS(), heartbeat_capture_tick_snapshot(), infer_word_run(), metrics_from_vm(), starforth_word_top_words(), starforth_word_word_execution_heat(), system_word_see(), vm_tick_inference_engine(), and vm_tick_slope_validator().

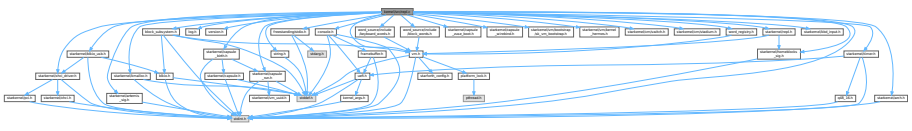
7.303.5.13 word_stats

```
WordStats* word_stats
```

7.304 kernel/src/repl.c File Reference

```
#include "starkernel/repl.h"
#include "console.h"
#include "log.h"
#include "vm.h"
#include "version.h"
#include "starkernel/timer.h"
#include "starkernel/arch.h"
#include "starkernel/xhci_driver.h"
#include "starkernel/blkio_usb.h"
#include "starkernel/kmalloc.h"
#include "starkernel/homeblocks_sig.h"
#include "starkernel/artemis_sig.h"
#include "starkernel/capsule_birth.h"
#include "starkernel/capsule_zuse_boot.h"
#include "starkernel/capsule_wirebind.h"
#include "starkernel/capsule_run.h"
#include "starkernel/vm/bootstrap/sk_vm_bootstrap.h"
#include "starkernel/vm/kernel_hermes.h"
#include "starkernel/vm/switch.h"
#include "starkernel/vm/stadium.h"
#include "block_subsystem.h"
#include "word_source/include/keyboard_words.h"
#include "starkernel/kbd_input.h"
#include "word_source/include/block_words.h"
#include "word_registry.h"
#include "freestanding/stdio.h"
#include <stdint.h>
#include <string.h>
```

Include dependency graph for repl.c:



Data Structures

- struct sk_blk_attach_pending_t
- struct sk_usb_ctrl_t

Macros

- #define SK_PROMPT_TEXT "\x1b[38;2;85;255;255mok> \x1b[39m"

  repl.c - Emergency FORTH REPL for LithosAnanke kernel
- #define SK_KH_BLK_ATTACH_BUF_SIZE 80
- #define SK_IDLE_BEAT_INTERVAL 100u /* ticks between idle service calls (1 s at 100 Hz) */
- #define SK_CURSOR_BLINK_INTERVAL 50u /* ticks between blink toggles (500 ms at 100 Hz) */

Functions

- static void sk_print_prompt (void)
- static const char * sk_console_user_prefix (void)
- void sk_repl_set_active_vm (VM *vm)

  sk_repl_set_active_vm - Redirect REPL input to a different VM (USE word).
- VM * sk_repl_get_active_vm (void)

  sk_repl_get_active_vm - Return the current USE-redirected VM, or NULL.
- static int sk_console_identity_present (void)
- blkio_dev_t * sk_repl_get_homeblocks_dev (void)

  sk_repl_get_homeblocks_dev / sk_repl_get_homeblocks_sig - The currently attached home-blocks USB drive, or NULL if none is attached / the attached drive didn't check out as HOMEBLOCKS_SIG_OK (FABRIC-2.md ?F.6/?F.9/?F.18).
- const homeblocks_sig_t * sk_repl_get_homeblocks_sig (void)
- blkio_dev_t * sk_repl_get_attached_blk_dev (void)

  sk_repl_get_attached_blk_dev - The currently attached USB block device, regardless of home-blocks recognition (FABRIC-2.md ?F.8/?F.19) – MINT's own target, since a blank/unminted drive never sets sk_repl_get_homeblocks_dev() above.
- blkio_dev_t * sk_repl_get_artemis_usb_dev (void)
- static void sk_word_blk_attach_ack (VM *vm)
- static void sk_word_kh_blk_attach_send (VM *vm)
- void sk_repl_register_words (VM *vm)

  sk_repl_register_words - Registers repl.c's own FORTH-visible words (currently just BLK-ATTACH-ACK, Artemis's storage-attach reply target – see sk_word_blk_attach_ack()'s doc comment in repl.c).
- static void sk_cursor_show (void)
- static void sk_repl_idle (VM *active_vm)
- int sk_console_getkey (VM *active_vm)

  sk_console_getkey - Real body of the standard dictionary's KEY word (called from shim.c's getchar()).
- void sk_repl_headless_wait (VM *mama)

  sk_repl_headless_wait - Idle-service loop with no interactive surface at all: no banner, no prompt, no console_getc()/readline.
- int sk_console_key_available (void)

  sk_console_key_available - Real body of the standard dictionary's ?TERMINAL word (called from sf_terminal_ready()).
- int sk_console_readline (char *buf, int size, VM *active_vm, int reanchor_prompt)

  sk_console_readline - Real body of the standard dictionary's QUERY/EXPECT words (called from shim.c's fgets()).
- static int sk_repl_line_calls_use (const char *input)
- static void sk_repl_dispatch_line (VM *vm, const char *input)
- int sk_repl_step (VM *vm)

  sk_repl_step - Execute one REPL turn on a VM and return.
- void sk_repl_run (VM *vm)

  sk_repl_run - Bare REPL loop (no banner).
- void sk_repl (VM *vm)

  sk_repl - Run the emergency FORTH REPL on the serial console.

Variables

- const char lithos_version [64] = "LithosAnanke v2.1.0 amd64"
- static VM * g_repl_active_vm = (void *)0
- static blkio_dev_t * g_homeblocks_dev = (void *)0
- static homeblocks_sig_t g_homeblocks_sig
- static int g_homeblocks_sig_valid = 0
- static blkio_dev_t * g_attached_blk_dev = (void *)0
- static blkio_dev_t * g_artemis_usb_dev = (void *)0
- static sk_usb_ctrl_t g_usb_ctrl [8u]
- static char g_kh_blk_attach_buf [80]
- static uint64_t g_last_beat_tick
- static uint64_t g_cursor_blink_tick
- static int g_cursor_visible = 1
- static int g_mama_interpreting
- static int g_idle_pump_active
- static int g_console_pending_key = -1
- static char g_kh_console_cmd_buf [SK_HERMES_CHUNK_MAX_PAYLOAD]

7.304.1 Macro Definition Documentation

7.304.1.1 SK_CURSOR_BLINK_INTERVAL

```
#define SK_CURSOR_BLINK_INTERVAL 50u /* ticks between blink toggles (500 ms at 100 Hz) */
```

7.304.1.2 SK_IDLE_BEAT_INTERVAL

```
#define SK_IDLE_BEAT_INTERVAL 100u /* ticks between idle service calls (1 s at 100 Hz) */
```

7.304.1.3 SK_KH_BLK_ATTACH_BUF_SIZE

```
#define SK_KH_BLK_ATTACH_BUF_SIZE 80
```

7.304.1.4 SK_PROMPT_TEXT

```
#define SK_PROMPT_TEXT "\x1b[38;2;85;255;255mok> \x1b[39m"
```

repl.c - Emergency FORTH REPL for LithosAnanke kernel

Direct adaptation of src/repl.c for the freestanding kernel context. Replaces libc stdio (fgets/printf/fflush) with HAL serial I/O:

- Input: console_getc() non-blocking poll with local echo and backspace
- Output: console_puts() / console_putc()

Idle spin: polls console_getc() and services the adaptive heartbeat. The timer ISR's top half (heartbeat_tick()) latches one sample per interrupt; the idle spin drains it every iteration via heartbeat_service() (item 0.8, FABRIC-0.md ?26) and calls sk_repl_idle() once per SK_IDLE_BEAT_INTERVAL ticks for coarser subsystem dispatch. On QEMU TCG the ISR must fire for ticks to advance — check "Heartbeat: N ticks" in the serial log to confirm.

Runs with interrupts enabled so the APIC heartbeat fires normally. Designed as the last thing kernel_main does before the idle loop.

7.304.2 Function Documentation

7.304.2.1 sk_console_getkey()

```
int sk_console_getkey (
            VM * active_vm )
```

sk_console_getkey - Real body of the standard dictionary's KEY word (called from shim.c's getchar()).

Blocks until a key is available from either input source (serial console or the PS2/virtio keyboard-event bridge), servicing the heartbeat/idle loop while waiting so a KEY call from inside any word never stalls the heartbeat. No echo – that's the caller's responsibility, same as any standard KEY.

Parameters

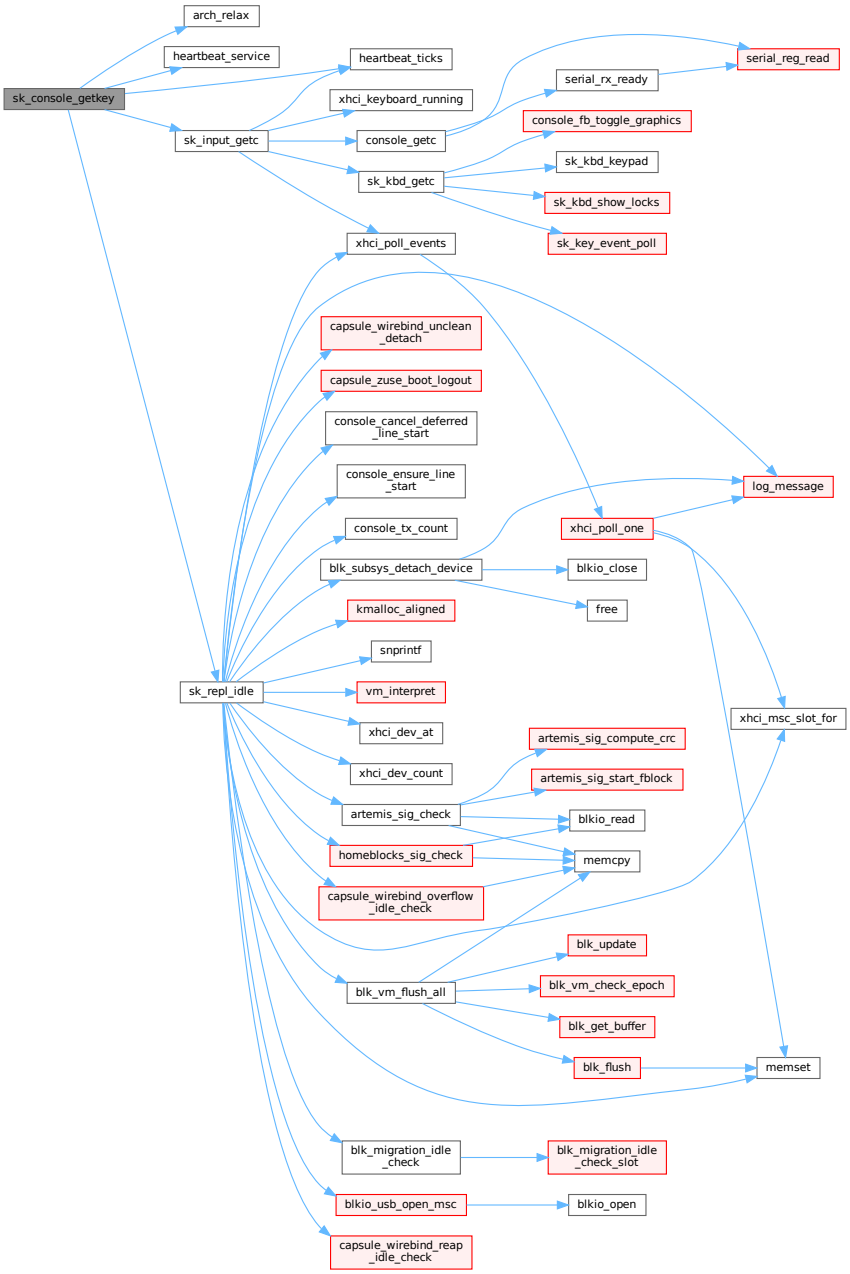
active_vm	VM whose idle dispatch runs while waiting (see sk_repl_idle()'s own doc comment on why this is a parameter rather than read via sk_repl_get_active_vm())

Returns

the key read, as an unsigned byte value

References arch_relax(), g_console_pending_key, g_last_beat_tick, g_repl_active_vm, heartbeat_service(), heartbeat_ticks(), SK_IDLE_BEAT_INTERVAL, sk_input_getc(), and sk_repl_idle().

Here is the call graph for this function:



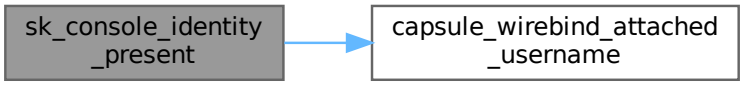
7.304.2.2 sk_console_identity_present()

```
static int sk_console_identity_present (
            void  )  [static]
```

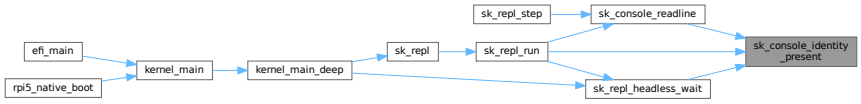
References capsule_wirebind_attached_username(), and VM::zuse_session.

Referenced by sk_console_readline(), sk_repl_headless_wait(), and sk_repl_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.304.2.3 sk_console_key_available()

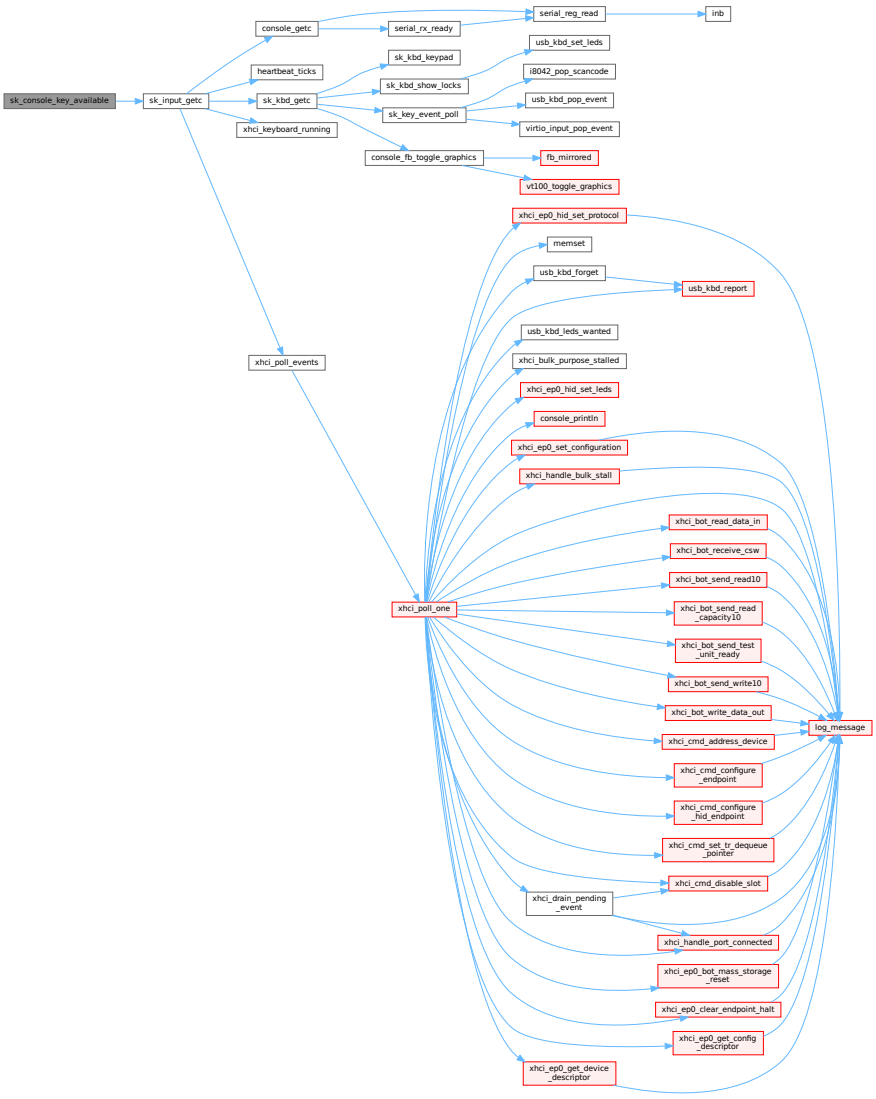
```
int sk_console_key_available (
            void  )
```

sk_console_key_available - Real body of the standard dictionary's ?TERMINAL word (called from sf_terminal_ready()).

Non-blocking peek: returns 1 if a key is ready without consuming it (a following sk_console_getkey() returns that exact key), 0 otherwise.

References g_console_pending_key, and sk_input_getc().

Here is the call graph for this function:



7.304.2.4 sk_console_readline()

```
int sk_console_readline (
            char * buf,
            int size,
            VM * active_vm,
            int reanchor_prompt )
```

sk_console_readline - Real body of the standard dictionary's QUERY/EXPECT words (called from shim.c's fgets()).

Reads one line from the console with echo and backspace support, servicing the heartbeat/ idle loop while waiting – the same line editor the REPL's own prompt uses internally, so a mid-word EXPECT behaves identically to typing at "ok>" itself.

Parameters

buf	Destination buffer
size	Buffer capacity, including the NUL terminator
active_vm	VM whose idle dispatch runs while waiting
reanchor_prompt	Nonzero to re-print the "ok> " prompt whenever an idle bottom half (heartbeat, USB attach/detach) writes to the console while this readline blocks at a bare, untyped prompt – keeps the top-level prompt as the last thing shown once the chatter dies down. Callers whose prompt line is their own (shim.c's fgets(), i.e. QUERY/EXPECT/ACCEPT) pass 0 so "ok> " never gets stamped onto their mid-word input context.

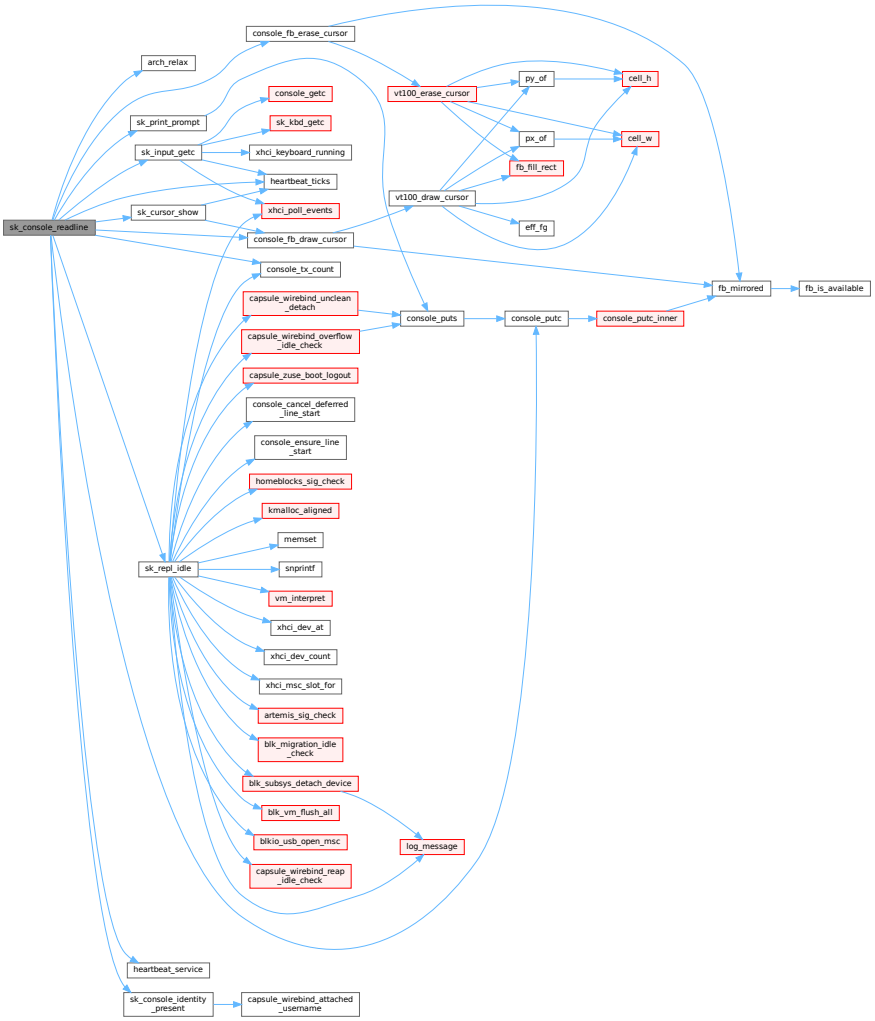
Returns

number of characters placed in buf, not counting the NUL, or -1 (2026-09-06) when reanchor_prompt is nonzero and the attached identity logged out while this call was blocked waiting for input with nothing typed yet – see repl.c's own doc comment on this function for what a caller must do with -1.

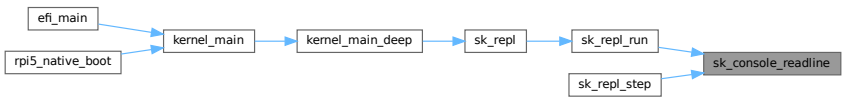
References arch_relax(), console_fb_draw_cursor(), console_fb_erase_cursor(), console_putc(), console_tx_count(), g_console_pending_key, g_cursor_blink_tick, g_cursor_visible, g_last_beat_tick, g_repl_active_vm, heartbeat_service(), heartbeat_ticks(), sk_console_identity_present(), SK_CURSOR_BLINK_INTERVAL, sk_cursor_show(), SK_IDLE_BEAT_INTERVAL, sk_input_getc(), sk_print_prompt(), and sk_repl_idle().

Referenced by sk_repl_run(), and sk_repl_step().

Here is the call graph for this function:



Here is the caller graph for this function:



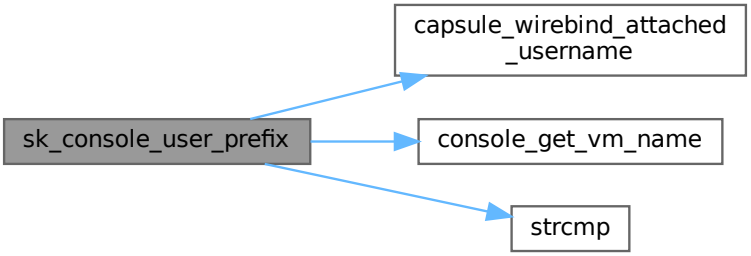
7.304.2.5 sk_console_user_prefix()

```
static const char * sk_console_user_prefix (
            void  )  [static]
```

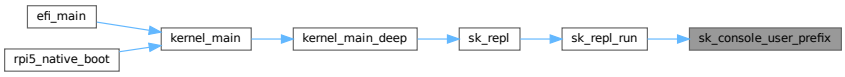
References capsule_wirebind_attached_username(), console_get_vm_name(), strcmp(), and VM::zuse_session.

Referenced by sk_repl_run().

Here is the call graph for this function:



Here is the caller graph for this function:



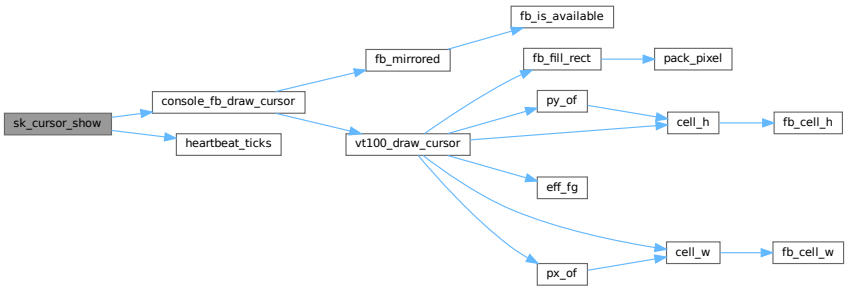
7.304.2.6 sk_cursor_show()

```
static void sk_cursor_show (
            void  )  [static]
```

References console_fb_draw_cursor(), g_cursor_blink_tick, g_cursor_visible, and heartbeat_ticks().

Referenced by sk_console_readline().

Here is the call graph for this function:



Here is the caller graph for this function:

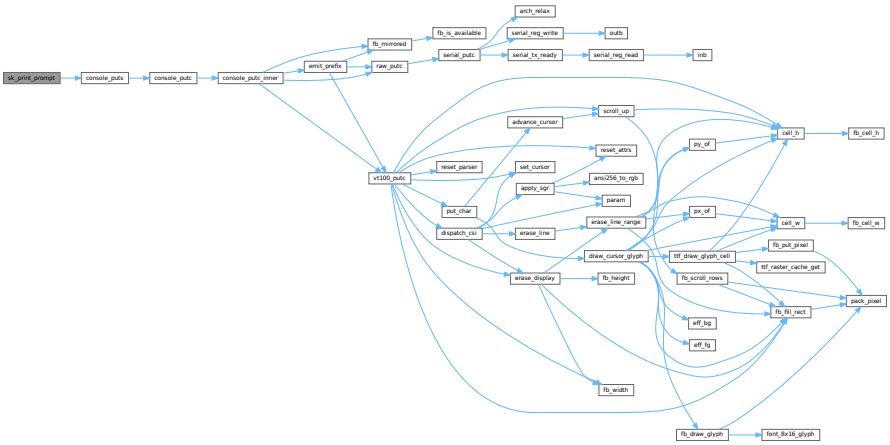
7.304.2.7 sk_print_prompt()

```
static void sk_print_prompt (
            void  )  [static]
```

References console_puts(), and SK_PROMPT_TEXT.

Referenced by sk_console_readline(), sk_repl_run(), and sk_repl_step().

Here is the call graph for this function:



Here is the caller graph for this function:

7.304.2.8 sk_repl()

```
void sk_repl (
            VM * vm )
```

sk_repl - Run the emergency FORTH REPL on the serial console.

Blocks until vm->halted is set (BYE word) or the VM encounters a halt. Runs with interrupts enabled; the APIC heartbeat continues to fire.

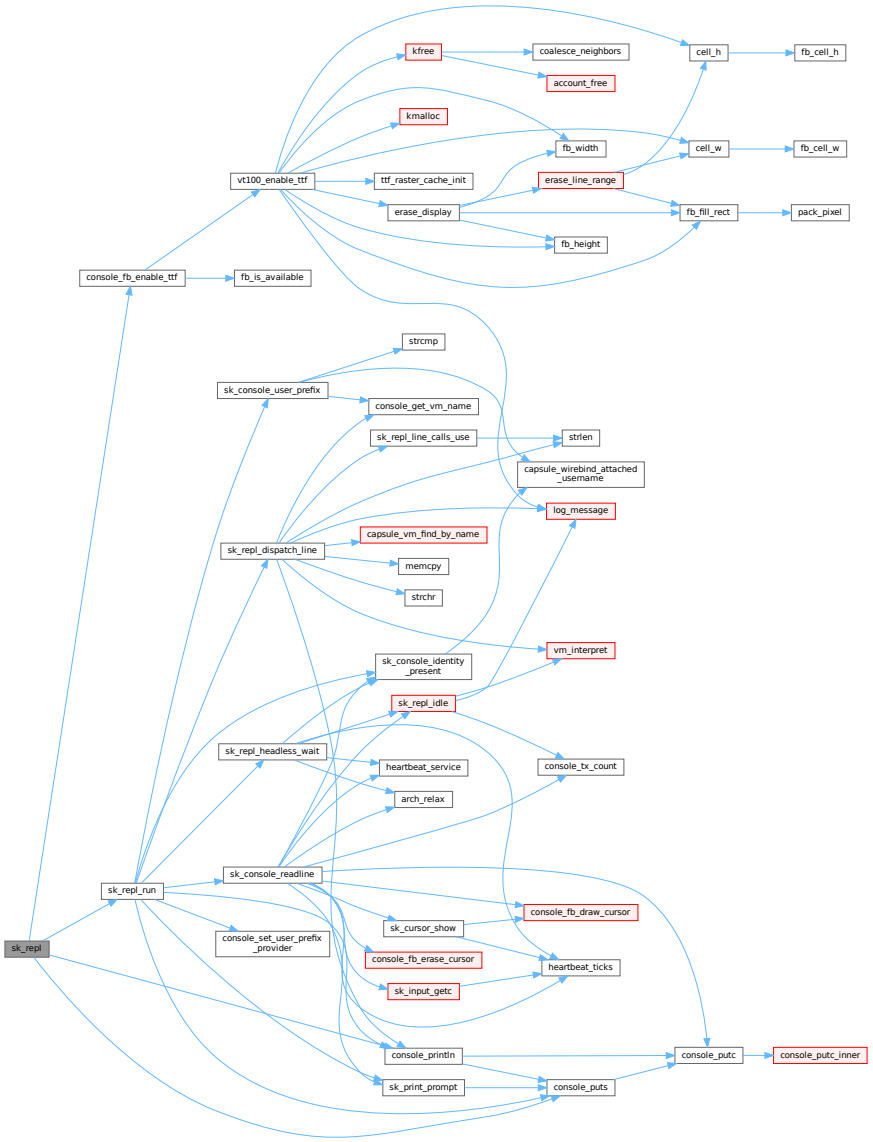
Parameters

vm	Mama VM instance (must be fully initialised)

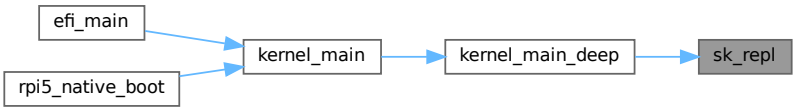
References console_fb_enable_ttf(), console_println(), console_puts(), lithos_version, sk_repl_run(), and STARFORTH_VERSION.

Referenced by kernel_main_deep().

Here is the call graph for this function:



Here is the caller graph for this function:



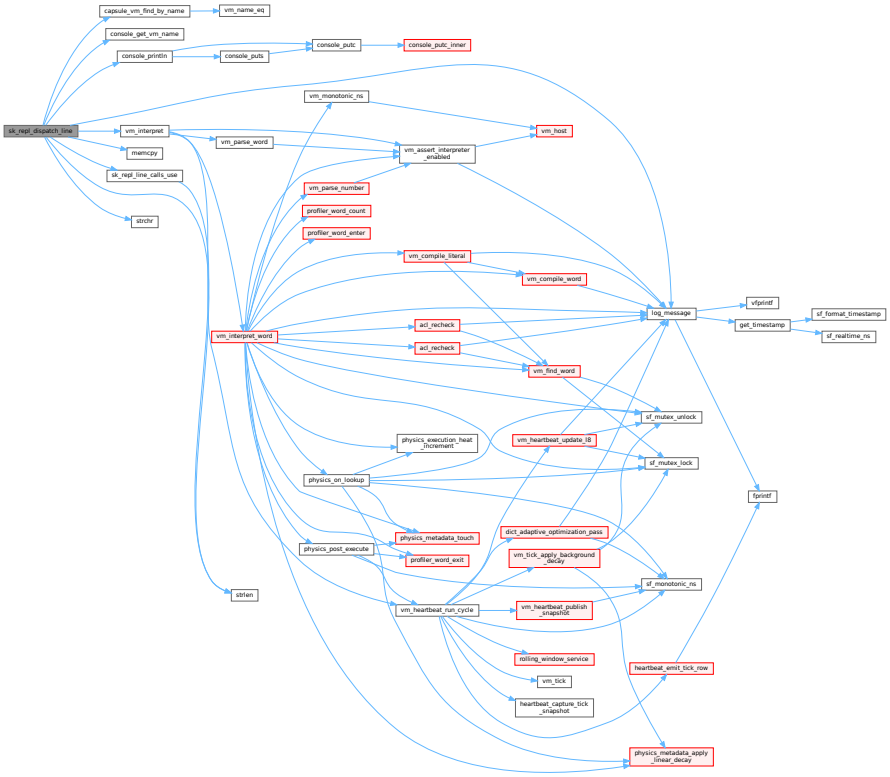
7.304.2.9 sk_repl_dispatch_line()

```
static void sk_repl_dispatch_line (
            VM * vm,
            const char * input )  [static]
```

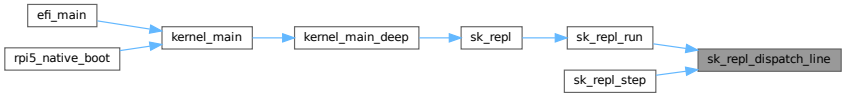
References capsule_vm_find_by_name(), console_get_vm_name(), console_println(), g_kh_console_cmd_buf, g_mama_interpreting, LOG_ERROR, log_message(), memcpy(), sk_repl_line_calls_use(), VMRegistryEntry::state, strchr(), strlen(), VMRegistryEntry::vm_id, vm_interpret(), VM_NAME_MAX, and VM_STATE_LIVE.

Referenced by sk_repl_run(), and sk_repl_step().

Here is the call graph for this function:



Here is the caller graph for this function:



7.304.2.10 sk_repl_get_active_vm()

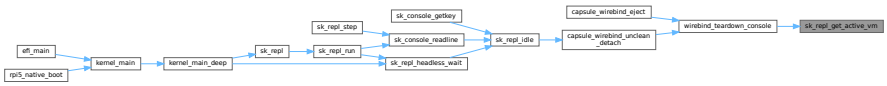
```
VM * sk_repl_get_active_vm (
            void  )
```

sk_repl_get_active_vm - Return the current USE-redirected VM, or NULL.

References g_repl_active_vm.

Referenced by wirebind_teardown_console().

Here is the caller graph for this function:



7.304.2.11 sk_repl_get_artemis_usb_dev()

```
blkio_dev_t * sk_repl_get_artemis_usb_dev (
            void  )
```

References g_artemis_usb_dev.

7.304.2.12 sk_repl_get_attached_blk_dev()

```
blkio_dev_t * sk_repl_get_attached_blk_dev (
            void  )
```

sk_repl_get_attached_blk_dev - The currently attached USB block device, regardless of home-blocks recognition (FABRIC-2.md ?F.8/?F.19) – MINT's own target, since a blank/unminted drive never sets sk_repl_get_homeblocks_dev() above.

NULL if nothing is attached.

References g_attached_blk_dev.

7.304.2.13 sk_repl_get_homeblocks_dev()

```
blkio_dev_t * sk_repl_get_homeblocks_dev (
            void  )
```

sk_repl_get_homeblocks_dev / sk_repl_get_homeblocks_sig - The currently attached home-blocks USB drive, or NULL if none is attached / the attached drive didn't check out as HOMEBLOCKS_SIG_OK (FABRIC-2.md ?F.6/?F.9/?F.18).

Both return NULL together; never one without the other.

References g_homeblocks_dev, and g_homeblocks_sig_valid.

7.304.2.14 sk_repl_get_homeblocks_sig()

```
const homeblocks_sig_t * sk_repl_get_homeblocks_sig (
            void  )
```

References g_homeblocks_sig, and g_homeblocks_sig_valid.

7.304.2.15 sk_repl_headless_wait()

```
void sk_repl_headless_wait (
            VM * mama )
```

sk_repl_headless_wait - Idle-service loop with no interactive surface at all: no banner, no prompt, no console_getc()/readline.

Runs heartbeat_service() and the same SK_IDLE_BEAT_INTERVAL-gated sk_repl_idle(mama) cadence sk_console_readline()'s own idle branch uses – so USB/WIREBIND/Zuse-attach detection, the heartbeat, and all other idle-tick subsystems keep running – until a real identity is currently attached (Zuse's own session, or a WIREBIND user), at which point it returns.

Revised 2026-09-06: originally exited on a one-way sticky "has anyone ever logged in this boot" flag (sk↩ _console_mark_login()/sk_console_ login_occurred(), both retired) – that let a real gap through, found live↩ : once the flag tripped once, it never reset, so a later full logout (nobody attached at all) fell through to a bare, unauthenticated prompt instead of going silent again. This now checks live attach state instead (repl.c's own sk_console_identity_present()), and is called from two places: once from kernel_main.c in place of an immediate sk_repl(mama) call when EMERGENCY_CONSOLE_ENABLED is off (the default, 2026-09-05) – no thumbdrive, no prompt, at boot – and again from inside sk_repl_run()'s own main loop, every time nobody is currently attached, so the same silence re-engages after any later logout mid-boot too. When EMERGENCY_CONSOLE_ENABLED is on (the debug/ recovery escape hatch), neither call site applies – the console shows immediately and stays visible regardless of attach state, exactly as before this change.

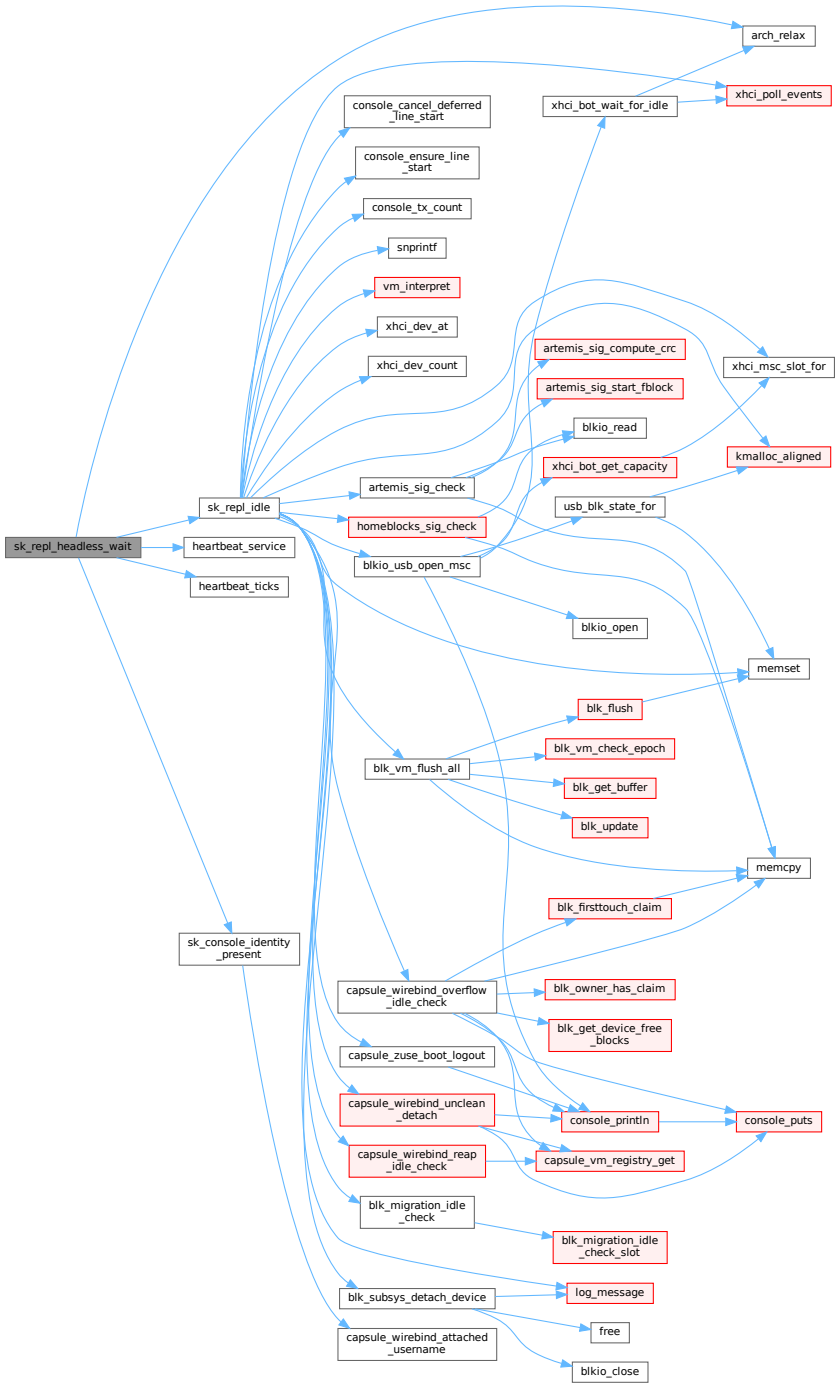
Parameters

mama	Hera's own VM instance – the idle-dispatch target, same as every other sk_repl_idle() caller uses.

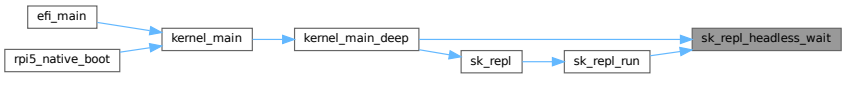
References arch_relax(), g_last_beat_tick, heartbeat_service(), heartbeat_ticks(), sk_console_identity_present(), SK_IDLE_BEAT_INTERVAL, and sk_repl_idle().

Referenced by kernel_main_deep(), and sk_repl_run().

Here is the call graph for this function:



Here is the caller graph for this function:



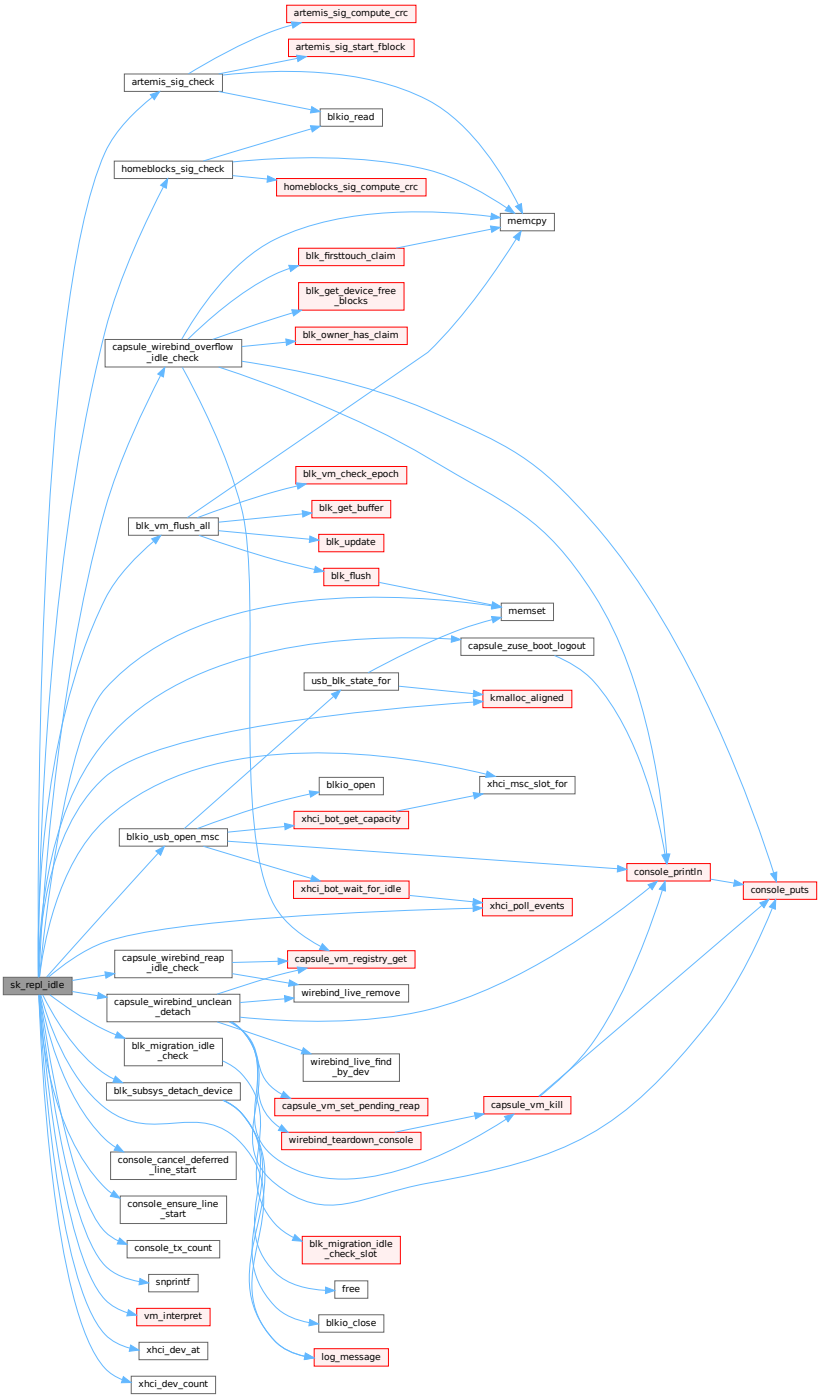
7.304.2.16 sk_repl_idle()

```
static void sk_repl_idle (
            VM * active_vm )  [static]
```

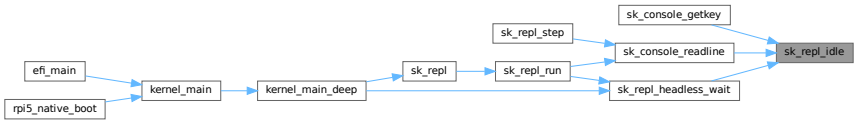
References artemis_sig_check(), ARTEMIS_SIG_OK, blk_migration_idle_check(), blk_subsys_detach_device(), blk_vm_flush_all(), blkio_usb_open_msc(), xhci_msc_slot_t::bot_msc_attach_pending, xhci_msc_slot_t::bot_msc_detach_pending, bytes, capsule_wirebind_overflow_idle_check(), capsule_wirebind_reap_idle_check(), capsule_wirebind_unclean_detach(), capsule_zuse_boot_logout(), console_cancel_deferred_line_start(), console_ensure_line_start(), console_tx_count(), sk_usb_ctrl_t::count, g_attached_blk_dev, g_homeblocks_dev, g_homeblocks_sig, g_homeblocks_sig_valid, g_idle_pump_active, g_mama_interpreting, g_usb_ctrl, HOMEBLOCKS_SIG_BAD_CRC, HOMEBLOCKS_SIG_BAD_VERSION, HOMEBLOCKS_SIG_BLANK, homeblocks_sig_check(), HOMEBLOCKS_SIG_OK, HOMEBLOCKS_SIG_READ_ERROR, HOMEBLOCKS_SIG_START_FBLOCK, sk_blk_attach_pending_t::is_artemis, kmalloc_aligned(), LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, xhci_dev_t::max_slots, memset(), sk_blk_attach_pending_t::pending, sk_usb_ctrl_t::pending, sk_blk_attach_pending_t::sig, sk_blk_attach_pending_t::sig_rc, sk_usb_ctrl_t::slots, snprintf(), vm_interpret(), xdev, xhci_dev_at(), xhci_dev_count(), XHCI_MAX_CONTROLLERS, xhci_msc_slot_for(), and xhci_poll_events().

Referenced by sk_console_getkey(), sk_console_readline(), and sk_repl_headless_wait().

Here is the call graph for this function:



Here is the caller graph for this function:



7.304.2.17 sk_repl_line_calls_use()

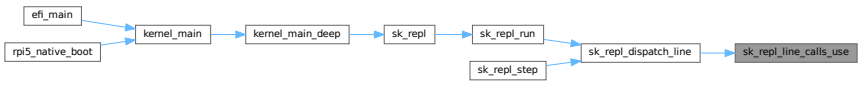
```
static int sk_repl_line_calls_use (
            const char * input )  [static]
```

References strlen().

Referenced by sk_repl_dispatch_line().

Here is the call graph for this function:

Here is the caller graph for this function:



7.304.2.18 sk_repl_register_words()

```
void sk_repl_register_words (
            VM * vm )
```

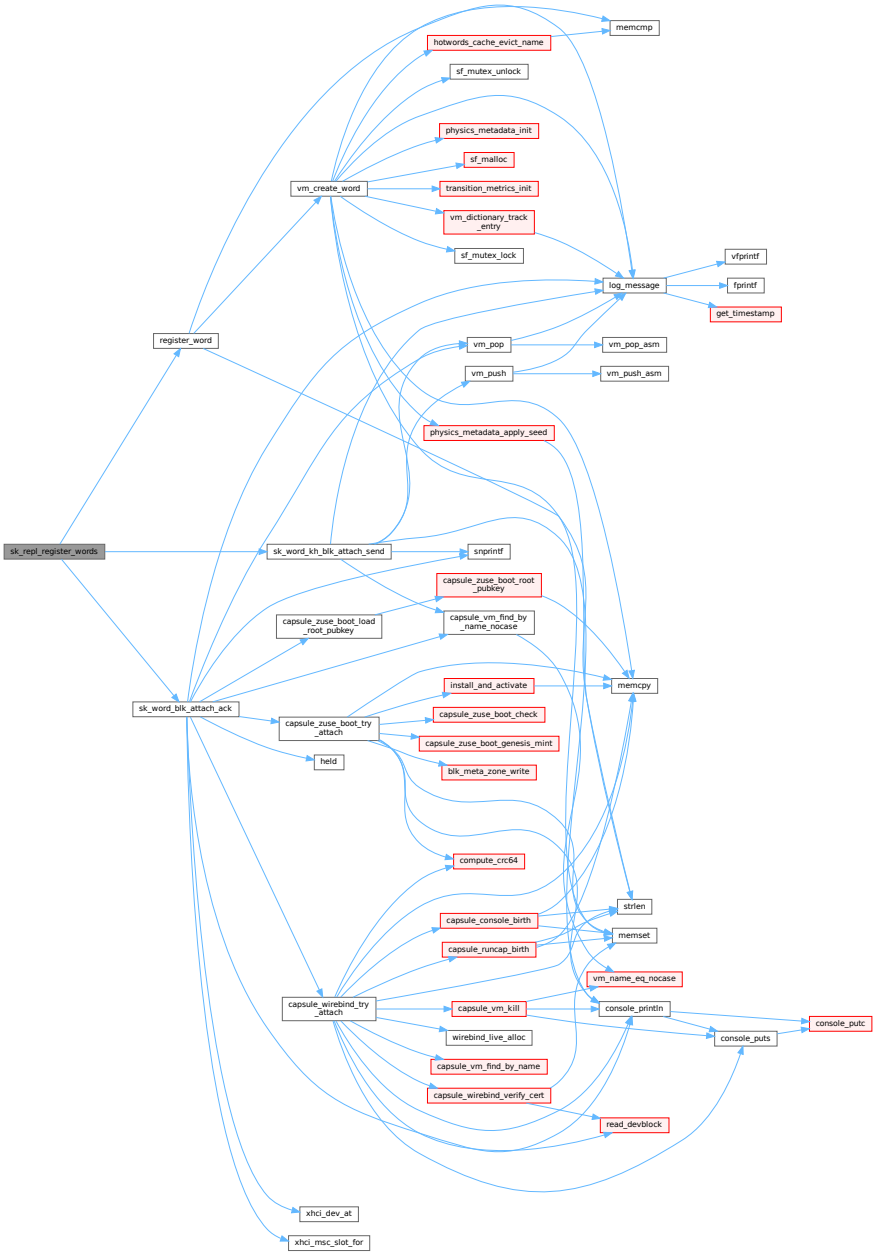
sk_repl_register_words - Registers repl.c's own FORTH-visible words (currently just BLK-ATTACH-ACK, Artemis's storage-attach reply target – see sk_word_blk_attach_ack()'s doc comment in repl.c).

Call once from register_forth79_words() alongside the other **STARKERNEL**-only registration calls.

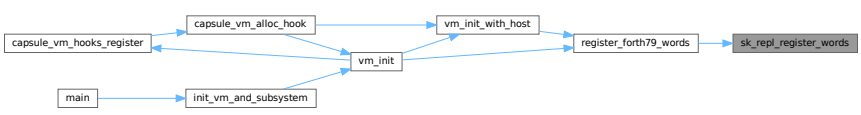
References register_word(), sk_word_blk_attach_ack(), and sk_word_kh_blk_attach_send().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.304.2.19 sk_repl_run()

```
void sk_repl_run (
            VM * vm )
```

sk_repl_run - Bare REPL loop (no banner).

Same as sk_repl but skips the version/welcome banner. Used by START to enter a child VM's interpreter loop without reprinting the header.

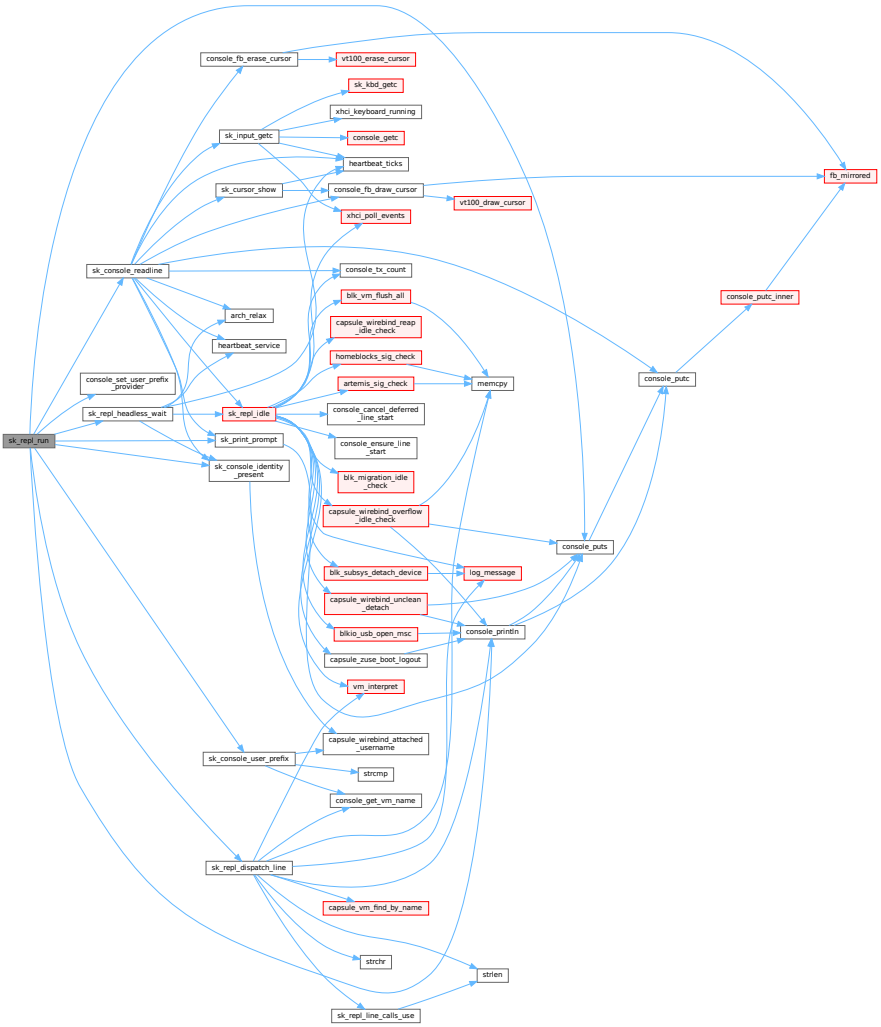
Parameters

vm	Fully initialised VM instance

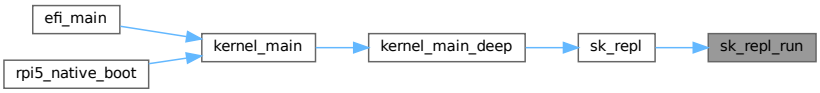
References VM::abort_requested, console_println(), console_puts(), console_set_user_prefix_provider(), VM::error, g_repl_active_vm, VM::halted, INPUT_BUFFER_SIZE, sk_console_identity_present(), sk_console_readline(), sk_console_user_prefix(), sk_print_prompt(), sk_repl_dispatch_line(), and sk_repl_headless_wait().

Referenced by sk_repl().

Here is the call graph for this function:



Here is the caller graph for this function:



7.304.2.20 sk_repl_set_active_vm()

```
void sk_repl_set_active_vm (
            VM * vm )
```

sk_repl_set_active_vm - Redirect REPL input to a different VM (USE word).

Pass NULL to restore default dispatch (Mama's VM). The change takes effect on the next REPL iteration.

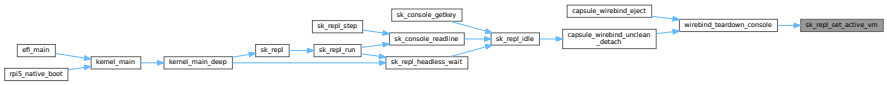
Parameters

vm	Target VM, or NULL for default

References g_repl_active_vm.

Referenced by wirebind_teardown_console().

Here is the caller graph for this function:



7.304.2.21 sk_repl_step()

```
int sk_repl_step (
            VM * vm )
```

sk_repl_step - Execute one REPL turn on a VM and return.

Prints the VM's prompt, reads one line, interprets it, prints ok/ERROR, then returns. Used by the Compudynamics VM-STEP primitive so Hera can give a single REPL quantum to a child VM without surrendering control for the full sk_repl_run() loop.

Parameters

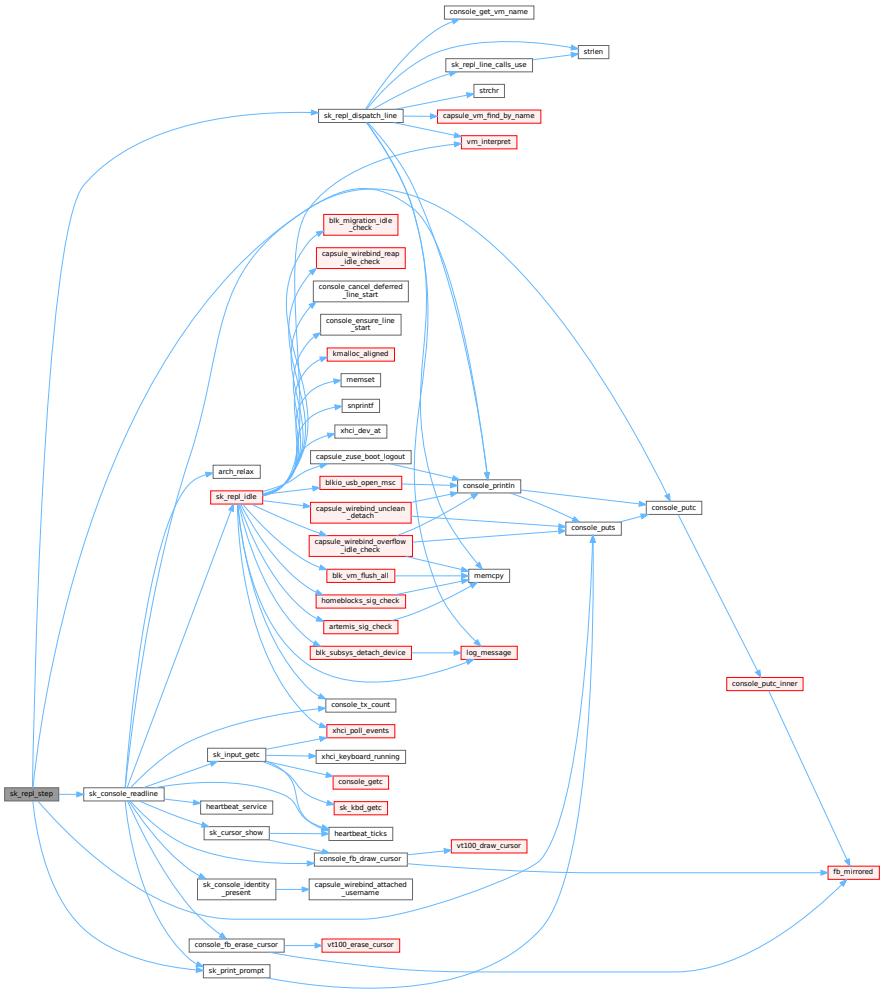
vm	Fully initialised VM instance

Returns

1 if the VM is still running, 0 if it halted during this turn

References VM::abort_requested, console_println(), console_puts(), VM::error, VM::halted, INPUT_BUFFER_SIZE, sk_console_readline(), sk_print_prompt(), and sk_repl_dispatch_line().

Here is the call graph for this function:



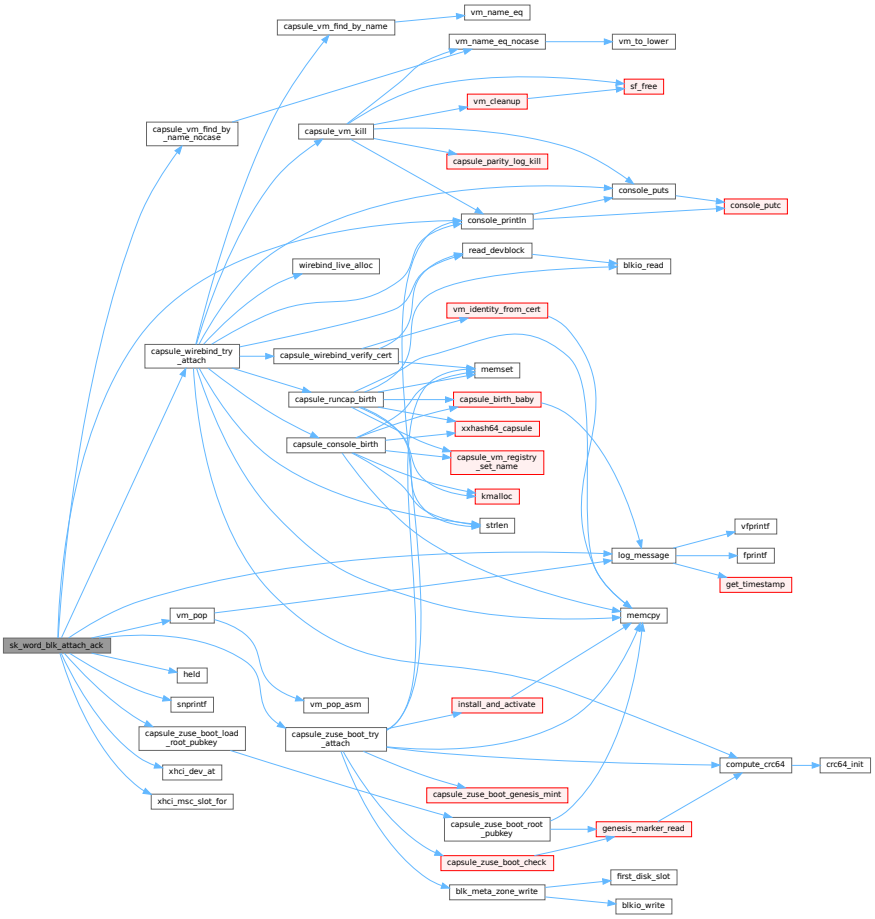
7.304.2.22 sk_word_blk_attach_ack()

```
static void sk_word_blk_attach_ack (
            VM * vm )  [static]
```

References xhci_msc_slot_t::bot_msc_attached, capsule_vm_find_by_name_nocase(), capsule_wirebind_try_attach(), capsule_zuse_boot_load_root_pubkey(), capsule_zuse_boot_try_attach(), console_println(), sk_usb_ctrl_t::count, VM::dsp, VM::error, g_artemis_usb_dev, g_attached_blk_dev, g_usb_ctrl, held(), HOMEBLOCKS_SIG_OK, sk_blk_attach_pending_t::is_artemis, LOG_ERROR, LOG_INFO, log_message(), sk_blk_attach_pending_t::pending, sk_usb_ctrl_t::pending, sk_blk_attach_pending_t::sig, sk_blk_attach_pending_t::sig_rc, slots, snprintf(), VMRegistryEntry::vm_id, vm_pop(), VMRegistryEntry::vm_ptr, xdev, xhci_dev_at(), XHCI_MAX_CONTROLLERS, and xhci_msc_slot_for().

Referenced by sk_repl_register_words().

Here is the call graph for this function:



Here is the caller graph for this function:

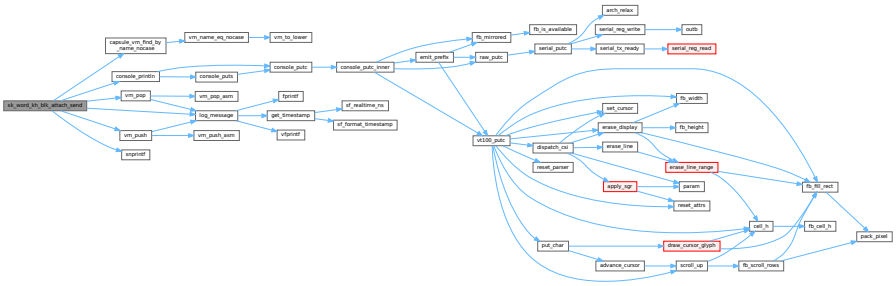
7.304.2.23 sk_word_kh_blk_attach_send()

```
static void sk_word_kh_blk_attach_send (
            VM * vm )  [static]
```

References capsule_vm_find_by_name_nocase(), console_println(), VM::dsp, VM::error, g_kh_blk_attach_buf, LOG_ERROR, log_message(), snprintf(), vm_pop(), VMRegistryEntry::vm_ptr, and vm_push().

Referenced by sk_repl_register_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.304.3 Variable Documentation

7.304.3.1 g_artemis_usb_dev

```
blkio_dev_t* g_artemis_usb_dev = (void *)0  [static]
```

Referenced by sk_repl_get_artemis_usb_dev(), and sk_word_blk_attach_ack().

7.304.3.2 g_attached_blk_dev

```
blkio_dev_t* g_attached_blk_dev = (void *)0  [static]
```

Referenced by sk_repl_get_attached_blk_dev(), sk_repl_idle(), and sk_word_blk_attach_ack().

7.304.3.3 g_console_pending_key

```
int g_console_pending_key = -1  [static]
```

Referenced by sk_console_getkey(), sk_console_key_available(), and sk_console_readline().

7.304.3.4 g_cursor_blink_tick

```
uint64_t g_cursor_blink_tick  [static]
```

Referenced by sk_console_readline(), and sk_cursor_show().

7.304.3.5 g_cursor_visible

```
int g_cursor_visible = 1  [static]
```

Referenced by sk_console_readline(), and sk_cursor_show().

7.304.3.6 g_homeblocks_dev

```
blkio_dev_t* g_homeblocks_dev = (void *)0  [static]
```

Referenced by sk_repl_get_homeblocks_dev(), and sk_repl_idle().

7.304.3.7 g_homeblocks_sig

```
homeblocks_sig_t g_homeblocks_sig  [static]
```

Referenced by sk_repl_get_homeblocks_sig(), and sk_repl_idle().

7.304.3.8 g_homeblocks_sig_valid

```
int g_homeblocks_sig_valid = 0  [static]
```

Referenced by sk_repl_get_homeblocks_dev(), sk_repl_get_homeblocks_sig(), and sk_repl_idle().

7.304.3.9 g_idle_pump_active

```
int g_idle_pump_active  [static]
```

Referenced by sk_repl_idle().

7.304.3.10 g_kh_blk_attach_buf

```
char g_kh_blk_attach_buf[80]  [static]
```

Referenced by sk_word_kh_blk_attach_send().

7.304.3.11 g_kh_console_cmd_buf

```
char g_kh_console_cmd_buf[SK_HERMES_CHUNK_MAX_PAYLOAD]  [static]
```

Referenced by sk_repl_dispatch_line().

7.304.3.12 g_last_beat_tick

```
uint64_t g_last_beat_tick  [static]
```

Referenced by sk_console_getkey(), sk_console_readline(), and sk_repl_headless_wait().

7.304.3.13 g_mama_interpreting

```
int g_mama_interpreting  [static]
```

Referenced by sk_repl_dispatch_line(), and sk_repl_idle().

7.304.3.14 g_repl_active_vm

```
VM* g_repl_active_vm = (void *)0  [static]
```

Referenced by sk_console_getkey(), sk_console_readline(), sk_repl_get_active_vm(), sk_repl_run(), and sk_repl_set_active_vm().

7.304.3.15 g_usb_ctrl

```
sk_usb_ctrl_t g_usb_ctrl[8u]  [static]
```

Referenced by sk_repl_idle(), and sk_word_blk_attach_ack().

7.304.3.16 lithos_version

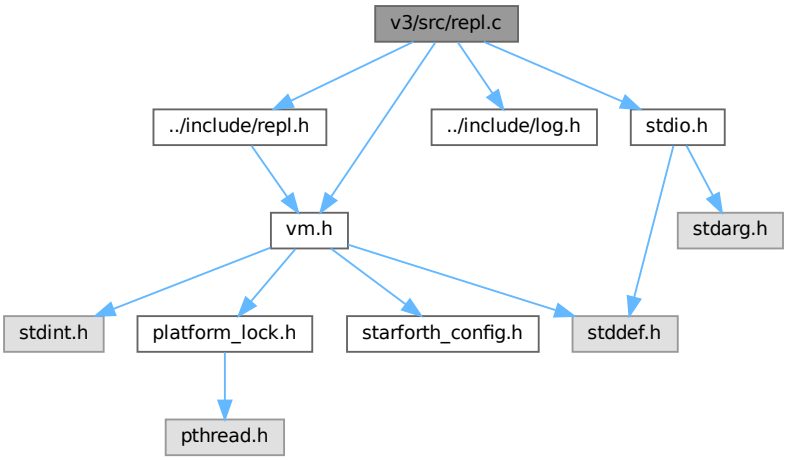
```
const char lithos_version[64] = "LithosAnanke v2.1.0 amd64"
```

Referenced by sk_repl().

7.305 v3/src/repl.c File Reference

```
#include "../include/repl.h"
#include "../include/log.h"
#include "../include/vm.h"
#include <stdio.h>
```

Include dependency graph for repl.c:



Functions

- __attribute__ ((weak))

  Extension point for non-recoverable VM errors (weak symbol).
- void vm_repl (VM *vm, int script_mode)

  Starts the Forth REPL (Read-Eval-Print Loop)

7.305.1 Function Documentation

7.305.1.1 __attribute__()

```
__attribute__ (
             (weak)  )
```

Extension point for non-recoverable VM errors (weak symbol).

Called by the REPL when `EMERGENCY_CONSOLE_ENABLED` = 0 and an error occurs. The default implementation logs the fault at `LOG_ERROR` level and sets `vm->halted` = 1, causing the REPL loop to exit cleanly.

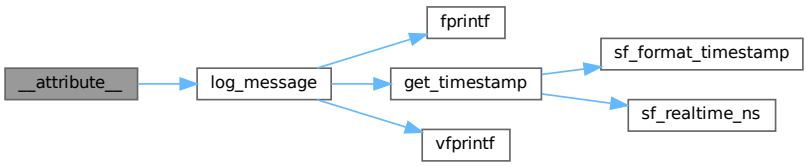
Platforms that need hardware reset, watchdog kick, or JTAG debug-probe integration should provide a strong definition in a platform-specific file (e.g., `starkernel/hal/fault.c`) to override this weak default.

Parameters

vm	Active VM whose `vm->error` triggered the fault path

References VM::halted, LOG_ERROR, and log_message().

Here is the call graph for this function:



7.305.1.2 vm_repl()

```
void vm_repl (
             VM * vm,
             int script_mode )
```

Starts the Forth REPL (Read-Eval-Print Loop)

Parameters

vm	Pointer to the VM structure
script_mode	If 1, suppress prompts and "ok" output (for piped input)

This function implements an interactive REPL for the Forth interpreter. It continuously reads input from the user, interprets it, and prints the result until the VM is halted or an error occurs.

In interactive mode (script_mode=0):

- Prints colored prompt "ok> "
- Acknowledges successful commands with " ok"
- Reports errors with " ERROR"

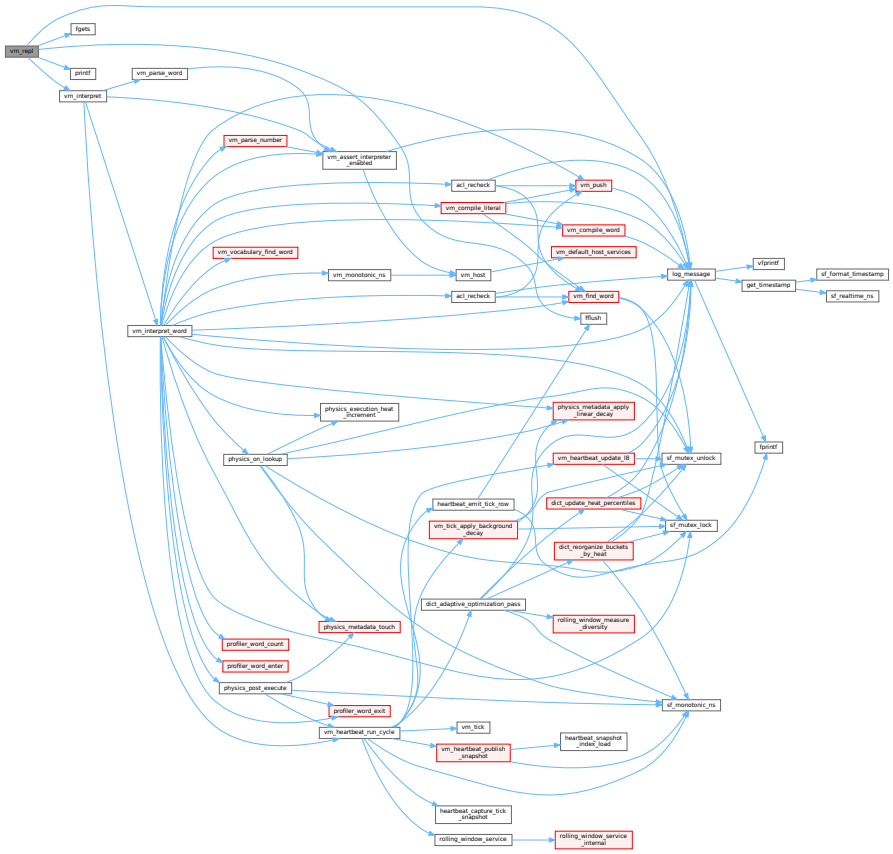
In script mode (script_mode=1):

- Suppresses all prompts and status messages
- Suitable for piped input (heredoc, pipes, redirects)

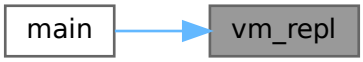
References VM::abort_requested, VM::error, fflush(), fgets(), VM::halted, LOG_INFO, log_message(), printf(), stdin, stdout, vm_interpret(), and VM::zuse_session.

Referenced by main().

Here is the call graph for this function:



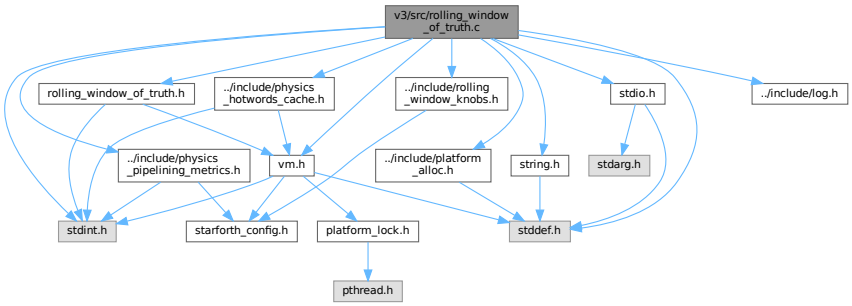
Here is the caller graph for this function:



7.306 v3/src/rolling_window_of_truth.c File Reference

```
#include "rolling_window_of_truth.h"
#include "../include/vm.h"
#include "../include/rolling_window_knobs.h"
#include "../include/physics_hotwords_cache.h"
#include "../include/physics_pipelining_metrics.h"
#include "../include/log.h"
#include "../include/platform_alloc.h"
#include <string.h>
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for rolling_window_of_truth.c:



Data Structures

- struct rolling_window_view_t

  Immutable snapshot view of a `RollingWindowOfTruth`.

Functions

- static uint64_t rolling_window_measure_diversity_view (const rolling_window_view_t *view)

  Count unique consecutive-pair transitions in a snapshot view.
- static void rolling_window_snapshot_view (const RollingWindowOfTruth *window, rolling_window_view_t *view)

Populate a `rolling_window_view_t` from the current double-buffer snapshot.

- static void rolling_window_publish_snapshot (RollingWindowOfTruth *window)

  Publish the live execution history into the inactive double-buffer slot.
- static void rolling_window_publish_snapshot_if_needed (RollingWindowOfTruth *window)

  Publish a new snapshot only if the `snapshot_pending` flag is set.
- int rolling_window_init (RollingWindowOfTruth *window)

  Initialise a `RollingWindowOfTruth`, allocating its three heap buffers.
- int rolling_window_record_execution (RollingWindowOfTruth *window, uint32_t word_id)

  Record one word execution in the circular history buffer.
- uint32_t rolling_window_get_recent_sequence (const RollingWindowOfTruth *window, uint32_t depth, uint32_t *out_sequence)

  Copy the most recent `depth` word IDs from the rolling window into `out_sequence`.
- uint32_t rolling_window_find_hottest_word (const RollingWindowOfTruth *window, uint32_t dict_size)

  Return the word ID that appears most frequently in the rolling window.
- uint64_t rolling_window_count_transition (const RollingWindowOfTruth *window, uint32_t word_a, uint32_t word_b)

  Count how many times the transition (word_a → word_b) occurs in the history.
- int rolling_window_is_warm (const RollingWindowOfTruth *window)

  Return whether the rolling window has accumulated enough history to be useful.
- char * rolling_window_stats_string (const RollingWindowOfTruth *window)

  Allocate and return a one-page status summary for a `RollingWindowOfTruth`.
- void rolling_window_reset (RollingWindowOfTruth *window)

  Reset the rolling window to an empty cold state without freeing heap memory.
- void rolling_window_cleanup (RollingWindowOfTruth *window)

  Free all heap memory held by a `RollingWindowOfTruth`.
- void rolling_window_seed_hotwords_cache (const RollingWindowOfTruth *window, struct VM *vm)

  Warm-start the hot-words cache from POST execution history.
- void rolling_window_seed_pipelining_context (const RollingWindowOfTruth *window, struct VM *vm)

  Warm-start pipelining context windows from POST execution history.
- uint64_t rolling_window_export_execution_history (const RollingWindowOfTruth *window, uint32_t *out_↩ sequence, uint64_t max_count)

  Export up to `max_count` entries from the execution history in chronological order.
- double rolling_window_pattern_capture_rate (const RollingWindowOfTruth *window, uint32_t test_window↩ _size)

  Estimate the fraction of transitions that carry unique contextual information.
- uint64_t rolling_window_measure_diversity (const RollingWindowOfTruth *window)

  Return the current pattern diversity score for the rolling window.
- static void rolling_window_run_adaptive_pass (RollingWindowOfTruth *window, const rolling_window_view_t *view)

  Adjust `effective_window_size` based on pattern diversity growth rate.
- static void rolling_window_service_internal (RollingWindowOfTruth *window)

  Drain both pending flags — snapshot and adaptive — in one call.
- void rolling_window_check_adaptive_shrink (RollingWindowOfTruth *window)

  Force an immediate adaptive window check, bypassing the accumulator gate.
- void rolling_window_service (RollingWindowOfTruth *window)

  Service the rolling window — publish pending snapshots and run adaptive sizing.

7.306.1 Function Documentation

7.306.1.1 rolling_window_check_adaptive_shrink()

```
void rolling_window_check_adaptive_shrink (
            RollingWindowOfTruth * window )
```

Force an immediate adaptive window check, bypassing the accumulator gate.

Sets `window->adaptive_pending` = 1 unconditionally and then calls `rolling_window_service_internal()` to drain it immediately. This allows external callers (e.g. the heartbeat timer or a FORTH word) to trigger a window-size evaluation outside the normal `ADAPTIVE_CHECK_FREQUENCY` cadence — for example, after a major workload transition or at benchmark checkpoints.

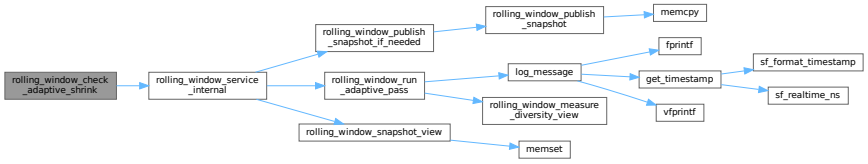
If the window is not warm the adaptive pass is a no-op internally.

Parameters

window	The `RollingWindowOfTruth` to check; no-op if NULL.

References RollingWindowOfTruth::adaptive_pending, and rolling_window_service_internal().

Here is the call graph for this function:



7.306.1.2 rolling_window_cleanup()

```
void rolling_window_cleanup (
            RollingWindowOfTruth * window )
```

Free all heap memory held by a `RollingWindowOfTruth`.

Calls `sf_free()` on `execution_history` and both `snapshot_buffers`[], setting each to NULL after freeing. Does not zero the scalar fields; the caller must not use `window` after this call without a subsequent `rolling_window_init()`.

No-op if `window` is NULL. Each buffer pointer is individually NULL-checked before freeing (defensive against partial-init failures from `rolling_window_init()`).

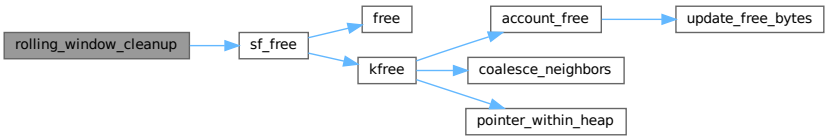
Parameters

window	The `RollingWindowOfTruth` to clean up; no-op if NULL.

References RollingWindowOfTruth::execution_history, sf_free(), and RollingWindowOfTruth::snapshot_buffers.

Referenced by vm_cleanup().

Here is the call graph for this function:



Here is the caller graph for this function:

7.306.1.3 rolling_window_count_transition()

```
uint64_t rolling_window_count_transition (
            const RollingWindowOfTruth * window,
            uint32_t word_a,
            uint32_t word_b )
```

Count how many times the transition (word_a → word_b) occurs in the history.

Scans all `ROLLING_WINDOW_SIZE` consecutive adjacent pairs in the current snapshot (wrapping at the buffer end: pair `i` is (history[i], history[(i+1) % ROLLING_WINDOW_SIZE])) and counts how many pairs match `word_a` followed by `word_b`.

Returns 0 if:

- `window` is NULL.
- `window->is_warm` is 0 (the history is too short to be meaningful).
- The snapshot's `is_warm` flag is 0.

Note: because the history is circular and the scan is linear across all slots regardless of `effective_window_size`, the count includes transitions that span the buffer wrap-around boundary. This is intentional: the full `ROLLING_WINDOW_SIZE` is used for transition counting.

Parameters

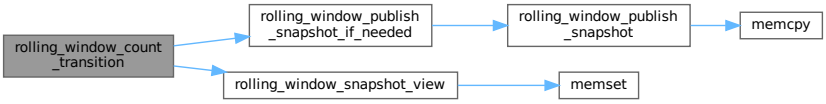
window	The `RollingWindowOfTruth` to query.
word_a	ID of the predecessor word.
word_b	ID of the successor word.

Returns

Count of (word_a → word_b) transitions; 0 if not warm.

References count, rolling_window_view_t::history, RollingWindowOfTruth::is_warm, rolling_window_view_t::is_warm, rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, and rolling_window_snapshot_view().

Here is the call graph for this function:



7.306.1.4 rolling_window_export_execution_history()

```
uint64_t rolling_window_export_execution_history (
            const RollingWindowOfTruth * window,
            uint32_t * out_sequence,
            uint64_t max_count )
```

Export up to `max_count` entries from the execution history in chronological order.

Linearises the internal circular buffer so callers get entries from oldest to newest without knowing the internal `window_pos` cursor. Two cases:

- **Pre-wrap** (`total_executions` < `ROLLING_WINDOW_SIZE`): the buffer has not yet wrapped; entries 0..`total_executions-1` are valid and are copied contiguously from `history`[0].
- **Post-wrap** (`total_executions` ≥ `ROLLING_WINDOW_SIZE`): the oldest entry is at `history`[window↩_pos]. Two `memcpy` calls stitch the two halves together:
  1. `history`[window_pos .. ROLLING_WINDOW_SIZE-1] (first part)
  2. `history`[0 .. window_pos-1] (second part, if `export_count` > `first_part`)

`export_count` is `min`(`total_executions`, `max_count`). Returns 0 if `window`, `out_sequence` are NULL, `max_count` is 0, the snapshot has no history, or `export_count` is 0.

Parameters

window	The `RollingWindowOfTruth` to export.
out_sequence	Caller-allocated output buffer of at least `max_count` `uint32_t` entries.
max_count	Maximum number of entries to export.

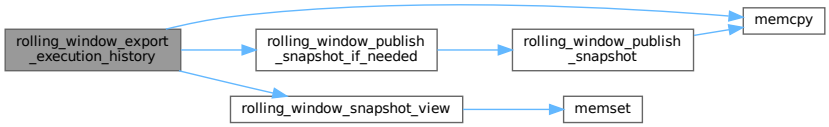
Returns

Actual number of entries written to `out_sequence`.

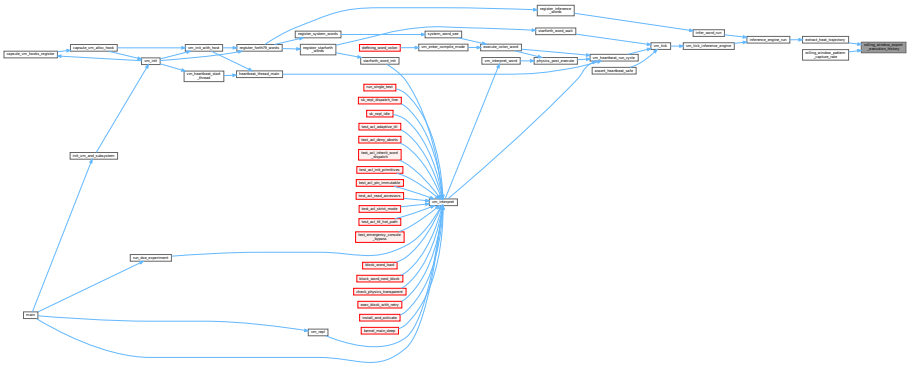
References rolling_window_view_t::history, memcpy(), rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), rolling_window_view_t::total_executions, and rolling_window_view_t::window_pos.

Referenced by extract_heat_trajectory(), and rolling_window_pattern_capture_rate().

Here is the call graph for this function:



Here is the caller graph for this function:



7.306.1.5 rolling_window_find_hottest_word()

```
uint32_t rolling_window_find_hottest_word (
            const RollingWindowOfTruth * window,
            uint32_t dict_size )
```

Return the word ID that appears most frequently in the rolling window.

Obtains the current snapshot, allocates a zero-filled frequency array of `dict_size` `uint32_t` counters via `sf_calloc`, and scans all `ROLLING_WINDOW_SIZE` history slots to tally occurrences. Then performs a linear scan for the maximum and frees the frequency array before returning.

Returns 0 immediately (without allocating) if:

- `window` is NULL.
- `window->is_warm` is 0 (not enough history to be meaningful).
- The snapshot's `is_warm` flag is 0.
- `sf_calloc` fails.

The returned word ID is the zero-based index into the dictionary's word-ID space. Word ID 0 may be returned both as a legitimate hottest word and as the not-warm sentinel; callers should check `rolling_window_is_warm()` before relying on the result.

Parameters

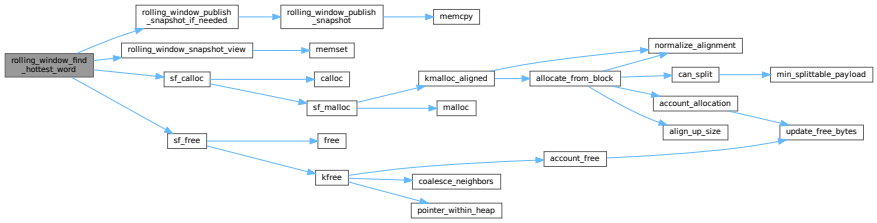
window	The `RollingWindowOfTruth` to analyse.
dict_size	Size of the dictionary's word-ID space; bounds the frequency array and the scan of the history.

Returns

Word ID of the most-frequently-executed word; 0 if the window is not warm or allocation fails.

References rolling_window_view_t::history, RollingWindowOfTruth::is_warm, rolling_window_view_t::is_warm, rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), sf_calloc(), sf_free(), and word_id.

Here is the call graph for this function:



7.306.1.6 rolling_window_get_recent_sequence()

```
uint32_t rolling_window_get_recent_sequence (
            const RollingWindowOfTruth * window,
            uint32_t depth,
            uint32_t * out_sequence )
```

Copy the most recent `depth` word IDs from the rolling window into `out_sequence`.

Calls `rolling_window_publish_snapshot_if_needed()` then reads the current double-buffer snapshot via `rolling_window_snapshot_view()`. Copies at most `min(depth`, available) entries from the most recent portion of the circular buffer into `out_sequence` in chronological order (oldest entry at index 0, most recent at index `actual_depth` - 1).

The available count is capped at `min(total_executions`, `ROLLING_WINDOW_SIZE)` so callers requesting more history than has been recorded get what is available rather than stale zeros.

Entries are addressed via:
`idx = (window_pos - actual_depth + i + ROLLING_WINDOW_SIZE) % ROLLING_WINDOW_SIZE`

Returns 0 immediately if `window`, `out_sequence` are NULL or `depth` is 0.

Parameters

window	The `RollingWindowOfTruth` to sample.
depth	Maximum number of recent word IDs to copy.
out_sequence	Caller-allocated output buffer of at least `depth` `uint32_t` entries.

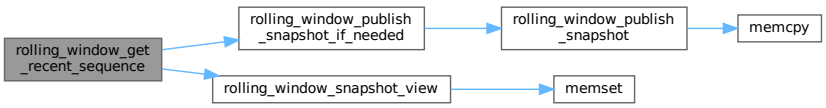
Returns

Actual number of entries written to `out_sequence`; ≤ `depth`.

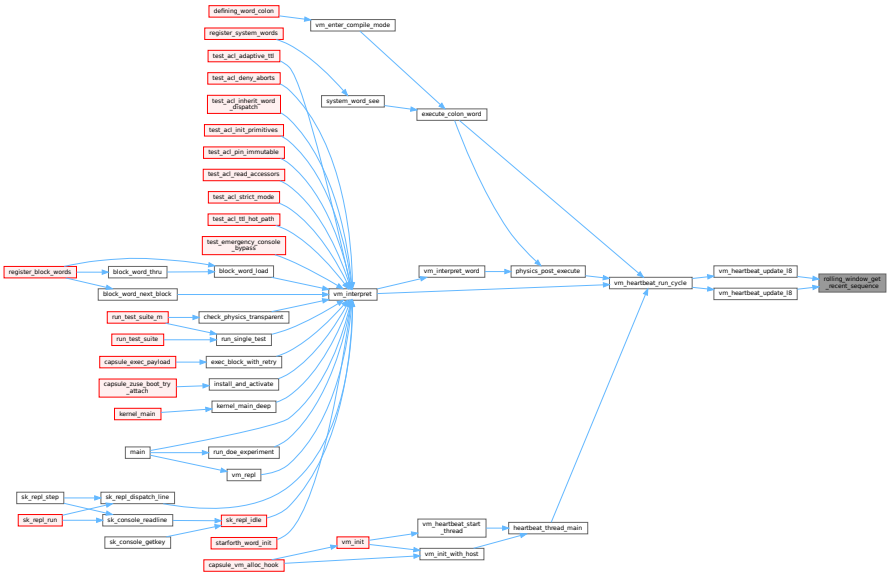
References rolling_window_view_t::history, rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), rolling_window_view_t::total_executions, and rolling_window_view_t::window_pos.

Referenced by vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the call graph for this function:



Here is the caller graph for this function:



7.306.1.7 rolling_window_init()

```
int rolling_window_init (
            RollingWindowOfTruth * window )
```

Initialise a `RollingWindowOfTruth`, allocating its three heap buffers.

Allocates and zero-initialises three `uint32_t` arrays, each of `ROLLING_WINDOW_SIZE` elements, via `sf_↩ calloc`:

- `window->execution_history` — the live circular write buffer.

- `window->snapshot_buffers[0]` — first double-buffer read slot.
- `window->snapshot_buffers[1]` — second double-buffer read slot.

If any allocation fails, all previously allocated buffers are freed and nulled before returning ?1. Initialises scalar state:

- `window_pos` = 0, `total_executions` = 0, `is_warm` = 0 (cold start).
- `effective_window_size` = `ROLLING_WINDOW_SIZE` (start at maximum).
- `snapshot_index` = 0, `snapshot_pending` = 1.
- All `snapshot_window_pos[]` / `snapshot_total_executions[]` / etc. = 0.
- `adaptive_check_accumulator` = 0, `adaptive_pending` = 0.

Immediately calls `rolling_window_publish_snapshot()` to push the zeroed state into both snapshot slots so that any early read-side call before the first `rolling_window_record_execution()` gets a coherent (empty) view.

Parameters

window	The `RollingWindowOfTruth` to initialise; must not be NULL.

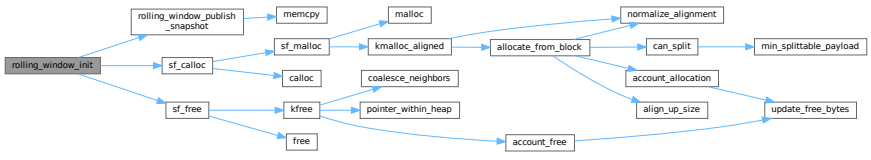
Returns

0 on success; ?1 on NULL `window` or allocation failure.

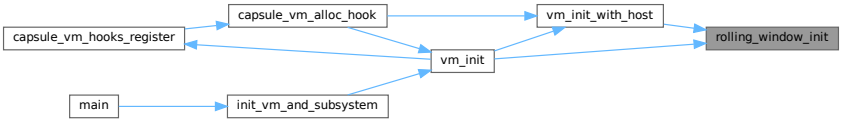
References RollingWindowOfTruth::adaptive_check_accumulator, RollingWindowOfTruth::adaptive_pending, RollingWindowOfTruth::effective_window_size, RollingWindowOfTruth::execution_history, RollingWindowOfTruth::is_warm, RollingWindowOfTruth::last_pattern_diversity, RollingWindowOfTruth::pattern_diversity_check_count, rolling_window_publish_snapsh ROLLING_WINDOW_SIZE, sf_calloc(), sf_free(), RollingWindowOfTruth::snapshot_buffers, RollingWindowOfTruth::snapshot_effectiv RollingWindowOfTruth::snapshot_index, RollingWindowOfTruth::snapshot_is_warm, RollingWindowOfTruth::snapshot_pending, RollingWindowOfTruth::snapshot_total_executions, RollingWindowOfTruth::snapshot_window_pos, RollingWindowOfTruth::total_exec and RollingWindowOfTruth::window_pos.

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.306.1.8 rolling_window_is_warm()

```
int rolling_window_is_warm (
            const RollingWindowOfTruth * window )
```

Return whether the rolling window has accumulated enough history to be useful.

Returns `window->is_warm`, which is set to 1 by `rolling_window_record_execution()` once `total_executions` $\geq$ 1024. Before that threshold the window is considered "cold" — its contents do not yet represent a stable distribution of the program's word usage, and analysis functions (`rolling_window_find_hottest_word()`, `rolling_window_count_transition()`, adaptive sizing) return early with conservative defaults when `is_warm` is 0.

Parameters

window	The `RollingWindowOfTruth` to query; NULL → returns 0.

Returns

1 if the window is warm ($\geq$ 1024 executions); 0 otherwise.

References RollingWindowOfTruth::is_warm.

7.306.1.9 rolling_window_measure_diversity()

```
uint64_t rolling_window_measure_diversity (
            const RollingWindowOfTruth * window )
```

Return the current pattern diversity score for the rolling window.

Triggers a lazy snapshot publish via `rolling_window_publish_snapshot_if_needed()`, acquires a read-side view via `rolling_window_snapshot_view()`, and delegates to the internal `rolling_window_measure_diversity_view()` to count the number of unique adjacent-pair transitions in the active window.

Higher diversity means the window is observing a wide variety of word sequences, indicating that the execution pattern is volatile and heat-stratified lookup is more beneficial than stable-pattern nearest-first lookup. Lower diversity means the program is executing a narrow, repetitive sequence — pattern knowledge is concentrated and reliable.

Used by `dict_adaptive_optimization_pass()` to select lookup strategy (threshold = 70 unique transitions) and by `rolling_window_run_adaptive_pass()` to calibrate `effective_window_size`.

Parameters

window	The `RollingWindowOfTruth` to measure.

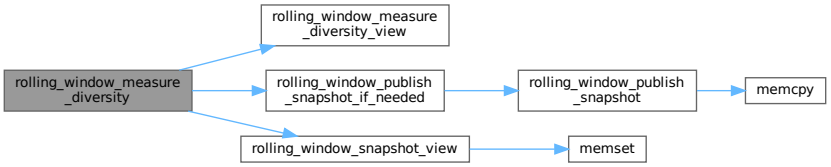
Returns

Count of unique adjacent-pair transitions in the active window; 0 if the window has fewer than 2 executions or `window` is NULL.

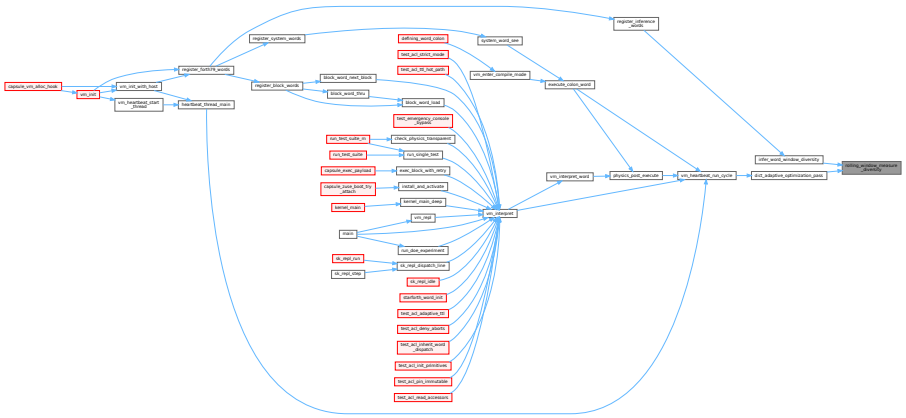
References rolling_window_measure_diversity_view(), rolling_window_publish_snapshot_if_needed(), and rolling_window_snapshot_view().

Referenced by dict_adaptive_optimization_pass(), and infer_word_window_diversity().

Here is the call graph for this function:



Here is the caller graph for this function:



7.306.1.10 rolling_window_measure_diversity_view()

```
static uint64_t rolling_window_measure_diversity_view (
            const rolling_window_view_t * view )  [static]
```

Count unique consecutive-pair transitions in a snapshot view.

Scans the `view`'s history buffer over the most recent `effective_window_size` entries (when warm) or all `ROLLING_WINDOW_SIZE` entries (when cold) and counts the number of adjacent pairs (history[i], history[i+1]) where `history`[i+1] $\neq$ `history`[i].

This is the primary pattern-diversity metric used by:

- `dict_adaptive_optimization_pass()` — to choose between heat-aware and naive lookup strategies (diversity $>$ 70 $\rightarrow$ heat-aware).
- `rolling_window_run_adaptive_pass()` — to decide whether to shrink or grow `effective_window_size`.

Returns 0 if `view` is NULL, has no history, or has fewer than 2 total executions.

Parameters

view	Immutable snapshot view; must have `history` != NULL.

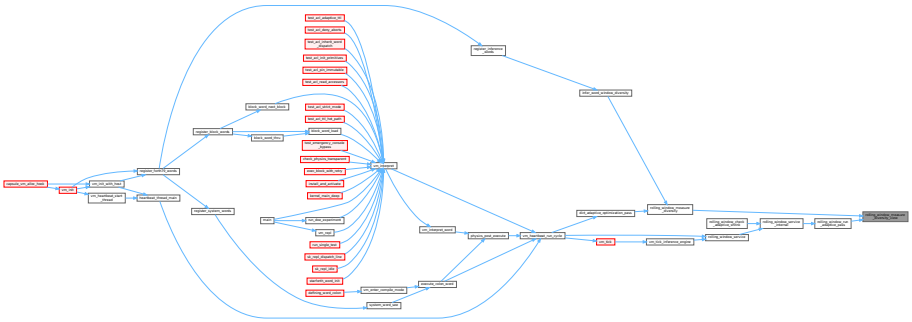
Returns

Count of unique adjacent-pair transitions in the active window.

References rolling_window_view_t::effective_window_size, rolling_window_view_t::history, rolling_window_view_t::is_warm, ROLLING_WINDOW_SIZE, rolling_window_view_t::total_executions, and rolling_window_view_t::window_pos.

Referenced by rolling_window_measure_diversity(), and rolling_window_run_adaptive_pass().

Here is the caller graph for this function:



7.306.1.11 rolling_window_pattern_capture_rate()

```
double rolling_window_pattern_capture_rate (
            const RollingWindowOfTruth * window,
            uint32_t test_window_size )
```

Estimate the fraction of transitions that carry unique contextual information.

Measures how many of the recorded adjacent-pair transitions produce a non-zero hash when the preceding `test_window_size` word IDs are incorporated:

```
pattern_hash = (... ((word[i-ws] × 31 + word[i-ws+1]) × 31 + ...) × 31 + word[i+1])
            mod 0x7FFFFFFF
```

A non-zero hash is counted as a "captured" pattern. The capture rate is:

```
rate = patterns_seen / total_transitions × 100  (clamped to [0, 100])
```

In practice, hash collisions with the zero sentinel are extraordinarily rare (probability ~1/2^31), so nearly all transitions contribute to `patterns_seen`. The function thus measures whether `test_window_size` is wide enough to produce a non-trivial hash — a proxy for whether the window introduces meaningful context beyond a single predecessor.

Linearises the circular buffer via `rolling_window_export_execution_history()` before scanning. Returns 0.0 if `window` or `test_window_size` < 1, if the linearised history has fewer than 2 entries, or if `sf_malloc` fails.

Parameters

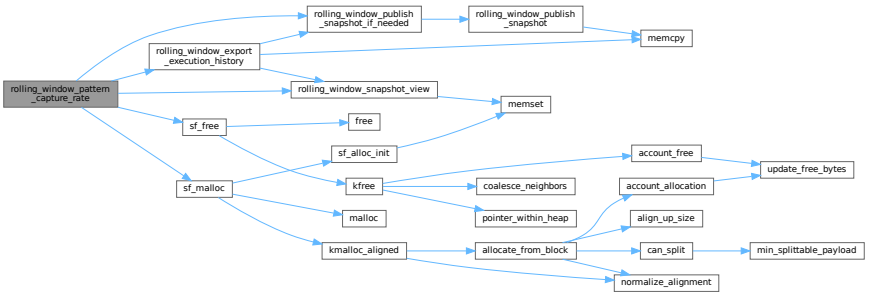
window	The RollingWindowOfTruth to analyse.
test_window_size	Number of preceding word IDs to fold into each pattern hash.

Returns

Capture rate as a percentage in [0.0, 100.0].

References rolling_window_view_t::history, rolling_window_export_execution_history(), rolling_window_publish_snapshot_if_needed, ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), sf_free(), sf_malloc(), and rolling_window_view_t::total_executions.

Here is the call graph for this function:



7.306.1.12 rolling_window_publish_snapshot()

```
static void rolling_window_publish_snapshot (
            RollingWindowOfTruth * window )  [static]
```

Publish the live execution history into the inactive double-buffer slot.

Implements the write side of the lock-free double buffer:

1. Identifies the inactive slot as `write_idx` = `snapshot_index` XOR 1.
2. Copies `window->execution_history` (the live circular buffer) into `snapshot_buffers`[write_idx] via `memcpy`.
3. Copies the current `window_pos`, `total_executions`, `effective_window_size`, and `is_warm` into the corresponding `snapshot_*` parallel arrays.
4. Flips `snapshot_index` to `write_idx` with `__ATOMIC_RELEASE`, making the new snapshot visible to readers.
5. Clears `snapshot_pending` with `__ATOMIC_RELEASE`.

Must only be called from the writer (main VM thread). No-op if `window`, `window->execution_history`, or `snapshot_buffers`[0] are NULL.

Parameters

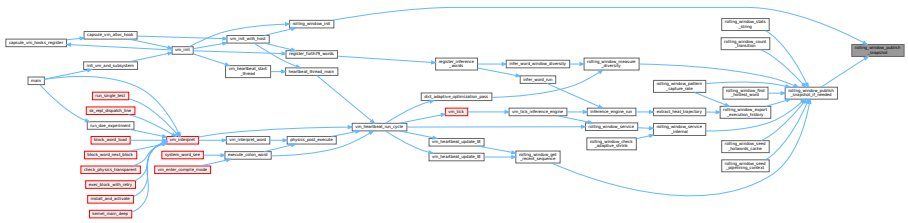
window	The `RollingWindowOfTruth` to snapshot.

References RollingWindowOfTruth::effective_window_size, RollingWindowOfTruth::execution_history, RollingWindowOfTruth::is_warm, memcpy(), ROLLING_WINDOW_SIZE, RollingWindowOfTruth::snapshot_buffers, RollingWindowOfTruth::snapshot_effective_window, RollingWindowOfTruth::snapshot_index, RollingWindowOfTruth::snapshot_is_warm, RollingWindowOfTruth::snapshot_pending, RollingWindowOfTruth::snapshot_total_executions, RollingWindowOfTruth::snapshot_window_pos, RollingWindowOfTruth::total_exec, and RollingWindowOfTruth::window_pos.

Referenced by rolling_window_init(), and rolling_window_publish_snapshot_if_needed().

Here is the call graph for this function:

Here is the caller graph for this function:



7.306.1.13 rolling_window_publish_snapshot_if_needed()

```
static void rolling_window_publish_snapshot_if_needed (
            RollingWindowOfTruth * window )  [static]
```

Publish a new snapshot only if the `snapshot_pending` flag is set.

Atomically exchanges `window->snapshot_pending` with 0 using `__ATOMIC_ACQ_REL`. If the exchanged value was non-zero (meaning `rolling_window_record_execution()` has written new data since the last publish), calls `rolling_window_publish_snapshot()` to make the current state visible to readers. Otherwise returns without copying.

This is the lazy-publish counterpart to `rolling_window_record_execution()`'s eager flag set. By batching publish work to read-side call sites (`rolling_window_get_recent_sequence()`, `rolling_window_find_hottest_word()`, etc.), unnecessary `memcpy` of the full `ROLLING_WINDOW_SIZE` buffer is avoided when no new data has arrived.

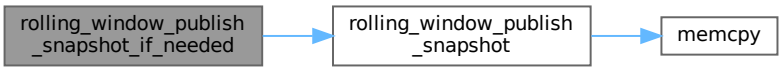
Parameters

window	The `RollingWindowOfTruth` to conditionally snapshot. No-op if NULL.

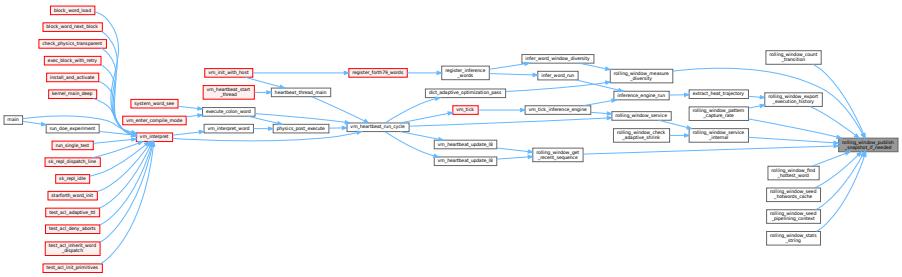
References rolling_window_publish_snapshot(), and RollingWindowOfTruth::snapshot_pending.

Referenced by rolling_window_count_transition(), rolling_window_export_execution_history(), rolling_window_find_hottest_word(), rolling_window_get_recent_sequence(), rolling_window_measure_diversity(), rolling_window_pattern_capture_rate(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), rolling_window_service_internal(), and rolling_window_stats_string().

Here is the call graph for this function:



Here is the caller graph for this function:



7.306.1.14 rolling_window_record_execution()

```
int rolling_window_record_execution (
            RollingWindowOfTruth * window,
            uint32_t word_id )
```

Record one word execution in the circular history buffer.

Writes `word_id` into `window->execution_history`[window_pos], advances `window_pos` modulo `ROLLING_WINDOW_SIZE` (circular wrap), increments `total_executions`, and checks the warm threshold:

- Once `total_executions` $\geq$ 1024 and `is_warm` is still 0, sets `is_warm` = 1. The 1024-execution threshold is calibrated to the POST harness which generates approximately 1450 executions; reaching 1024 guarantees a statistically meaningful execution distribution before diversity or percentile analysis begins.

Sets `snapshot_pending` = 1 to signal that a new snapshot should be published on the next read-side access. When the window is warm and `adaptive_check_accumulator` reaches `ADAPTIVE_CHECK_↩ FREQUENCY`, also sets `adaptive_pending` = 1 to request an `effective_window_size` update.

Called from the interpreter hot path on every word dispatch. All operations are O(1) with no locks and no allocations.

Parameters

window	The `RollingWindowOfTruth` to record into.
word_id	Stable numeric ID of the word just executed.

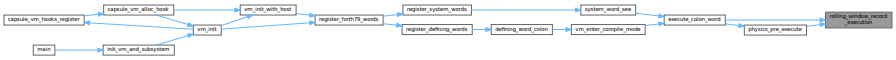
Returns

0 on success; ?1 if `window` or `execution_history` are NULL.

References RollingWindowOfTruth::adaptive_check_accumulator, ADAPTIVE_CHECK_FREQUENCY, RollingWindowOfTruth::adapti RollingWindowOfTruth::execution_history, RollingWindowOfTruth::is_warm, ROLLING_WINDOW_SIZE, RollingWindowOfTruth::snap RollingWindowOfTruth::total_executions, RollingWindowOfTruth::window_pos, and word_id.

Referenced by execute_colon_word(), and physics_pre_execute().

Here is the caller graph for this function:



7.306.1.15 rolling_window_reset()

```
void rolling_window_reset (
            RollingWindowOfTruth * window )
```

Reset the rolling window to an empty cold state without freeing heap memory.

Zeros `execution_history` via `memset` and resets the scalar fields: `window_pos` = 0, `total_executions` = 0, `is_warm` = 0. Sets `snapshot_pending` = 1 so that the next read will publish an empty snapshot.

The three heap buffers (`execution_history`, `snapshot_buffers`[0/1]) are preserved; only their contents are zeroed. Intended for benchmark harnesses that need a clean baseline between test runs without the overhead of `rolling_window_cleanup()` + `rolling_window_init()`.

Parameters

window	The `RollingWindowOfTruth` to reset; no-op if NULL or `execution_history` is NULL.

References RollingWindowOfTruth::execution_history, RollingWindowOfTruth::is_warm, memset(), ROLLING_WINDOW_SIZE, RollingWindowOfTruth::snapshot_pending, RollingWindowOfTruth::total_executions, and RollingWindowOfTruth::window_pos.

Here is the call graph for this function:

7.306.1.16 rolling_window_run_adaptive_pass()

```
static void rolling_window_run_adaptive_pass (
            RollingWindowOfTruth * window,
            const rolling_window_view_t * view )  [static]
```

Adjust `effective_window_size` based on pattern diversity growth rate.

Runs the core adaptive window logic using the snapshot `view`. Measures the current diversity via `rolling_window_measure_diversity_view()`, computes the fractional growth rate relative to `window->last_pattern_diversity` in Q16.16 fixed-point:

```
growth_rate_q48 = (diversity_delta « 16) / last_pattern_diversity
```

and compares it against a threshold:

```
threshold_q48 = (ADAPTIVE_GROWTH_THRESHOLD « 16) / 100
```

Shrink path (`growth_rate_q48 < threshold_q48`): diversity is plateauing; reducing the window saves scan work. The new size is:

```
new_size = max(effective_window_size × ADAPTIVE_SHRINK_RATE / 100,
               ADAPTIVE_MIN_WINDOW_SIZE)
```

Already at `ADAPTIVE_MIN_WINDOW_SIZE` → no change (logs "Floor Reached").

Grow path (`growth_rate_q48 ≥ threshold_q48`): diversity is still increasing; the larger window is capturing new patterns. The new size is:

```
new_size = min(effective_window_size × 100 / ADAPTIVE_SHRINK_RATE,
               ROLLING_WINDOW_SIZE)
```

Already at `ROLLING_WINDOW_SIZE` → no change (logs "Ceiling Reached").

On the very first call after init, `last_pattern_diversity` == 0; only the baseline is recorded and the function returns without resizing. All decisions are logged at `LOG_DEBUG` for offline analysis.

No-op if `window` or `view` are NULL or the view is not warm.

Parameters

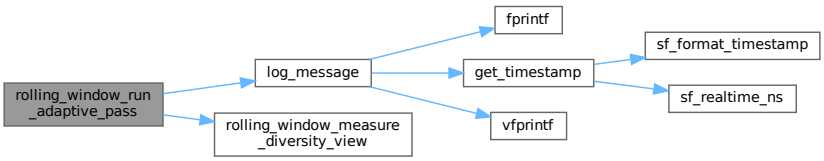
window	The `RollingWindowOfTruth` whose `effective_window_size` and `last_pattern_diversity` are updated.
view	Current immutable snapshot; must be warm.

References ADAPTIVE_GROWTH_THRESHOLD, ADAPTIVE_MIN_WINDOW_SIZE, ADAPTIVE_SHRINK_RATE, RollingWindowOfTruth::effective_window_size, rolling_window_view_t::history, rolling_window_view_t::is_warm,

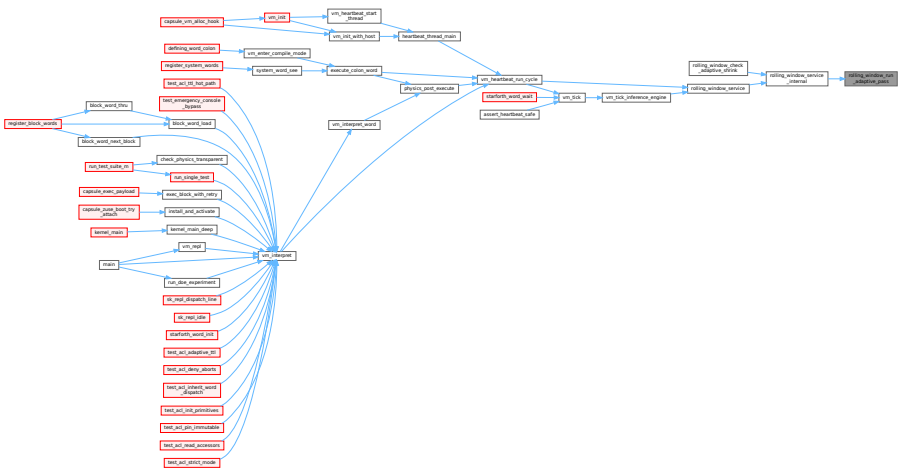
RollingWindowOfTruth::last_pattern_diversity, LOG_DEBUG, log_message(), RollingWindowOfTruth::pattern_diversity_check_count, rolling_window_measure_diversity_view(), ROLLING_WINDOW_SIZE, and rolling_window_view_t::total_executions.

Referenced by rolling_window_service_internal().

Here is the call graph for this function:



Here is the caller graph for this function:



7.306.1.17 rolling_window_seed_hotwords_cache()

```
void rolling_window_seed_hotwords_cache (
            const RollingWindowOfTruth * window,
            struct VM * vm )
```

Warm-start the hot-words cache from POST execution history.

Called exactly once after POST completes and before the REPL begins, so that the first interactive command benefits from cache entries derived from real POST execution data rather than starting cold. The algorithm:

1. Allocates a zero-filled frequency array of `DICTIONARY_SIZE` `uint32_t` counters via `sf_calloc`.
2. Obtains the current snapshot and tallies occurrences of each word ID over all `ROLLING_WINDOW_SIZE` history slots.
3. Computes a promotion threshold: words appearing more than the average frequency across all non-zero-frequency words are considered "hot".

4. Iterates over the frequency array; for each word above the threshold, looks up the live `DictEntry` via `vm_dictionary_lookup_by_word_id()` (under `vm->dict_lock`), updates `entry->execution_heat` to the observed frequency count, and calls `hotwords_cache_promote()` to insert it into the cache.

5. Logs the count of promoted words at `LOG_INFO`.

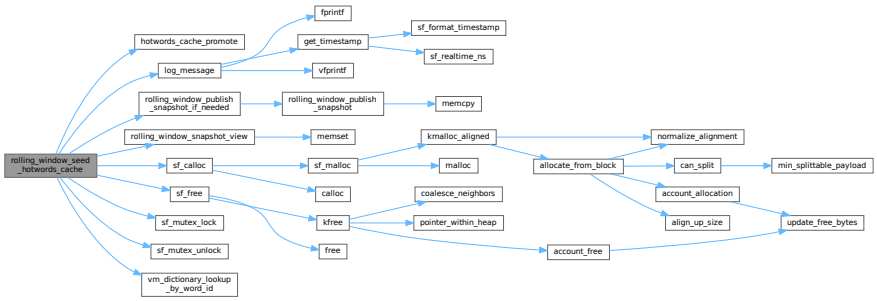
No-op if `window` is NULL, not warm, `vm` is NULL, or `vm->hotwords_cache` is NULL. Also no-op if the snapshot's `is_warm` flag is 0.

Parameters

window	Warm `RollingWindowOfTruth` containing POST execution data.
vm	The VM whose `hotwords_cache` is seeded and whose `dict_lock` is held during `DictEntry` updates.

References VM::dict_lock, DICTIONARY_SIZE, entry, rolling_window_view_t::history, VM::hotwords_cache, hotwords_cache_promote(), RollingWindowOfTruth::is_warm, rolling_window_view_t::is_warm, LOG_INFO, log_message(), rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), sf_calloc(), sf_free(), sf_mutex_lock(), sf_mutex_unlock(), vm_dictionary_lookup_by_word_id(), and word_id.

Here is the call graph for this function:



7.306.1.18 rolling_window_seed_pipelining_context()

```
void rolling_window_seed_pipelining_context (
            const RollingWindowOfTruth * window,
            struct VM * vm )
```

Warm-start pipelining context windows from POST execution history.

Called exactly once after POST completes and before the REPL begins, to populate each word's `transition_metrics` context window with real predecessor data so that speculative prefetch can begin making informed decisions on the first interactive command. The algorithm:

1. Allocates a sliding `context[]` buffer of `context_window_size` `uint32_t` entries (max(`TRANSITION_WINDOW_SIZE`, 2)) and initialises it with the first `context_window_size` entries from the history.

2. Scans all `ROLLING_WINDOW_SIZE` consecutive adjacent pairs in the snapshot:

- Looks up `current_entry = vm_dictionary_lookup_by_word_id(current_id)` (under `vm->dict_lock`, released immediately).
- If `current_entry->transition_metrics` is set, calls `transition_metrics_record_context()` with the current `context[]` and `next_id` to populate the Phase 1 context-transition counter.
- Shifts the `context[]` buffer left by one, appending `next_id` at the right end (sliding window advance).

3. Logs seeded-words and seeded-transitions counts at `LOG_INFO`.

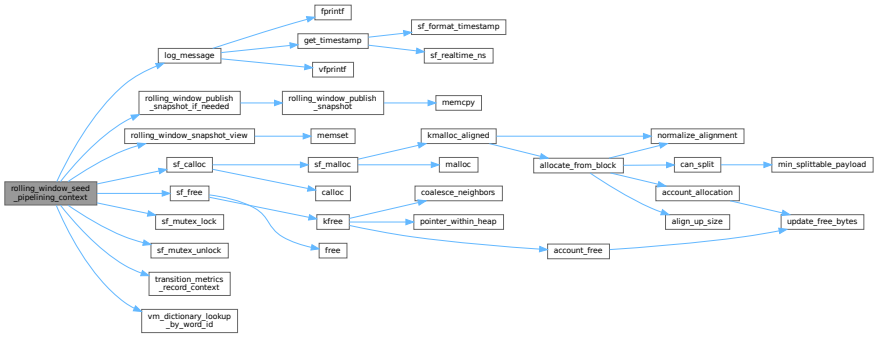
No-op if `window` is NULL, not warm, `vm` is NULL, `ENABLE_PIPELINING` is 0, or the context buffer allocation fails.

Parameters

window	Warm `RollingWindowOfTruth` containing POST execution data.
vm	The VM whose per-word `transition_metrics` structures are seeded; `dict_lock` is briefly held per lookup.

References VM::dict_lock, DICTIONARY_SIZE, ENABLE_PIPELINING, rolling_window_view_t::history, RollingWindowOfTruth::is_wa
rolling_window_view_t::is_warm, LOG_INFO, log_message(), rolling_window_publish_snapshot_if_needed(),
ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), sf_calloc(), sf_free(), sf_mutex_lock(), sf_mutex_unlock(),
DictEntry::transition_metrics, transition_metrics_record_context(), TRANSITION_WINDOW_SIZE, and vm_dictionary_lookup_by_wo

Here is the call graph for this function:



7.306.1.19 rolling_window_service()

```
void rolling_window_service (
            RollingWindowOfTruth * window )
```

Service the rolling window — publish pending snapshots and run adaptive sizing.

The public entry point for the rolling window's periodic maintenance. Intended to be called from the heartbeat tick handler or the REPL idle loop at a rate determined by the caller (typically 100 Hz from the APIC timer ISR via `heartbeat_tick()`). Delegates entirely to `rolling_window_service_internal()`:

- Flushes the pending snapshot if `snapshot_pending` is set.
- Runs `rolling_window_run_adaptive_pass()` if `adaptive_pending` is set.

Unlike `rolling_window_check_adaptive_shrink()`, this function does NOT force `adaptive_pending = 1`; it only services flags that the record path has already set. The normal adaptive-pass cadence is therefore governed entirely by `ADAPTIVE_CHECK_FREQUENCY` executions between checks.

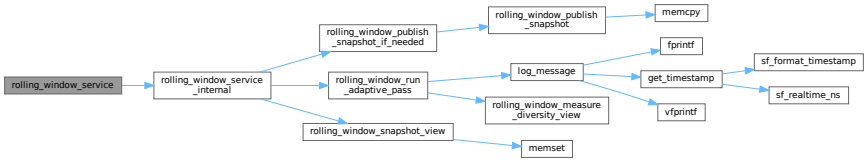
Parameters

window	The `RollingWindowOfTruth` to service; no-op if NULL.

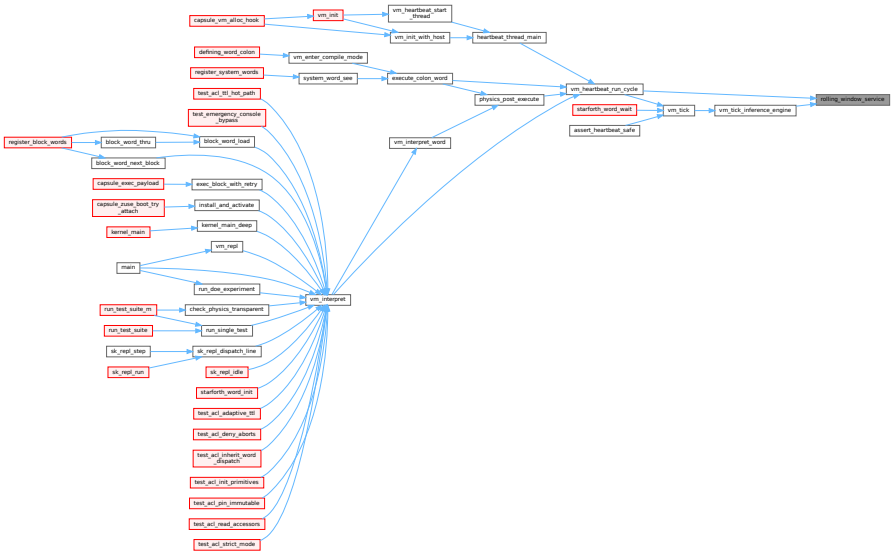
References rolling_window_service_internal().

Referenced by vm_heartbeat_run_cycle(), and vm_tick_inference_engine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.306.1.20 rolling_window_service_internal()

```
static void rolling_window_service_internal (
            RollingWindowOfTruth * window )  [static]
```

Drain both pending flags — snapshot and adaptive — in one call.

Called from both `rolling_window_check_adaptive_shrink()` and `rolling_window_service()` as the single internal dispatch point:

1. Calls `rolling_window_publish_snapshot_if_needed()` to flush any pending execution history into the double-buffer snapshot.
2. Checks `window->adaptive_pending`; if set, acquires a view, runs `rolling_window_run_adaptive_pass()`, then clears `adaptive_pending`.

No-op if `window` is NULL.

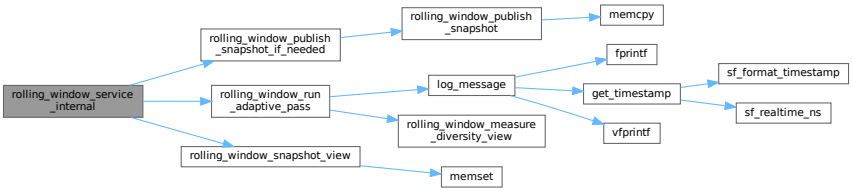
Parameters

window	The `RollingWindowOfTruth` to service.

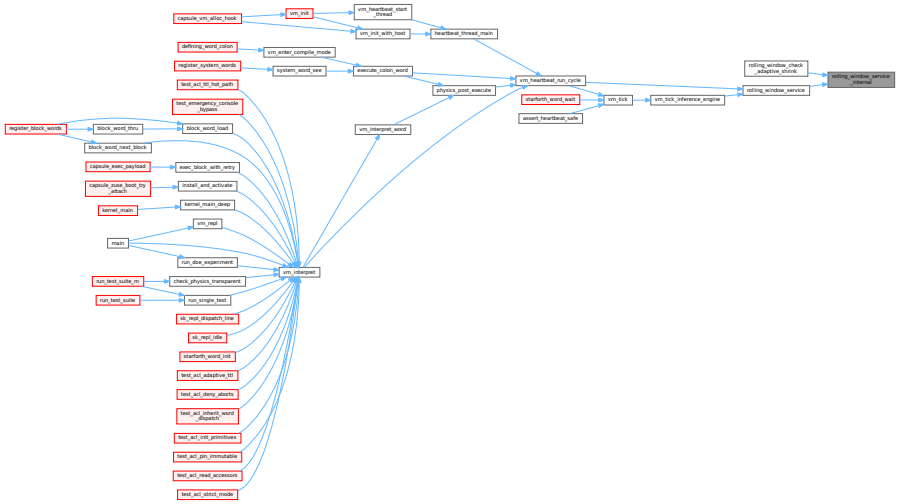
References RollingWindowOfTruth::adaptive_pending, rolling_window_publish_snapshot_if_needed(), rolling_window_run_adaptive_ and rolling_window_snapshot_view().

Referenced by rolling_window_check_adaptive_shrink(), and rolling_window_service().

Here is the call graph for this function:



Here is the caller graph for this function:



7.306.1.21 rolling_window_snapshot_view()

```
static void rolling_window_snapshot_view (
            const RollingWindowOfTruth * window,
            rolling_window_view_t * view )  [static]
```

Populate a `rolling_window_view_t` from the current double-buffer snapshot.

Reads `window->snapshot_index` with `__ATOMIC_ACQUIRE` to determine which of the two snapshot buffers (`snapshot_buffers`[0] or [1]) is the most recently published. Fills `view` with the corresponding history pointer and metadata fields (`window_pos`, `total_executions`, `effective_window_size`, `is_warm`). If `window` or `view` is NULL, `view` is zeroed and the function returns early.

Callers hold no lock — the acquire barrier on `snapshot_index` ensures they see a coherent snapshot that was fully written before `rolling_window_publish_snapshot()` released it via `__ATOMIC_RELEASE`.

Parameters

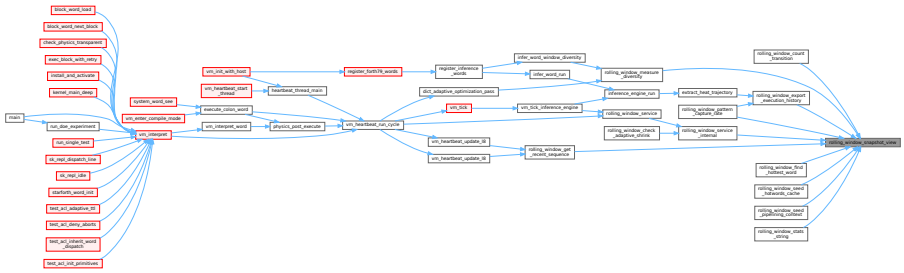
window	The `RollingWindowOfTruth` to sample.
view	Output `rolling_window_view_t`; zeroed on NULL `window`.

References rolling_window_view_t::effective_window_size, rolling_window_view_t::history, rolling_window_view_t::is_warm, memset(), RollingWindowOfTruth::snapshot_buffers, RollingWindowOfTruth::snapshot_effective_window_size, RollingWindowOfTruth::snapshot_index, RollingWindowOfTruth::snapshot_is_warm, RollingWindowOfTruth::snapshot_total_executio RollingWindowOfTruth::snapshot_window_pos, rolling_window_view_t::total_executions, and rolling_window_view_t::window_pos.

Referenced by rolling_window_count_transition(), rolling_window_export_execution_history(), rolling_window_find_hottest_word(), rolling_window_get_recent_sequence(), rolling_window_measure_diversity(), rolling_window_pattern_capture_rate(), rolling_window_seed_hotwords_cache(), rolling_window_seed_pipelining_context(), rolling_window_service_internal(), and rolling_window_stats_string().

Here is the call graph for this function:

Here is the caller graph for this function:



7.306.1.22 rolling_window_stats_string()

```
char * rolling_window_stats_string (
            const RollingWindowOfTruth * window )
```

Allocate and return a one-page status summary for a `RollingWindowOfTruth`.

Allocates a 1024-byte buffer via `sf_malloc` and fills it with a multi-line report covering:

- `total_executions` — lifetime word-execution count.
- `ROLLING_WINDOW_SIZE` — compile-time capacity constant.

- Warm/cold status: "WARM (representative)" or "COLD (warming up)".
- `window_pos` / `ROLLING_WINDOW_SIZE` — current write-cursor position.

Values are read from the current double-buffer snapshot obtained via `rolling_window_snapshot_view()`; a lazy publish is triggered first by `rolling_window_publish_snapshot_if_needed()`.

The caller is responsible for calling `sf_free()` on the returned pointer. Returns NULL if `window` is NULL or `sf_malloc` fails.

Parameters

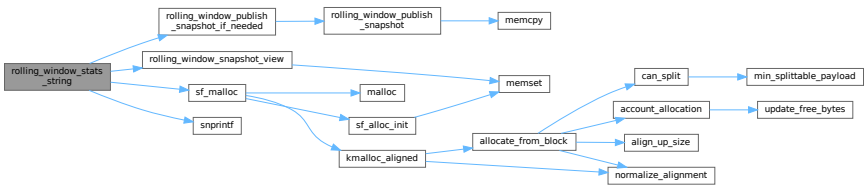
window	The `RollingWindowOfTruth` to describe.

Returns

Heap-allocated NUL-terminated summary string; caller must `sf_free()` it. NULL on failure.

References rolling_window_view_t::is_warm, rolling_window_publish_snapshot_if_needed(), ROLLING_WINDOW_SIZE, rolling_window_snapshot_view(), sf_malloc(), snprintf(), rolling_window_view_t::total_executions, and rolling_window_view_t::window

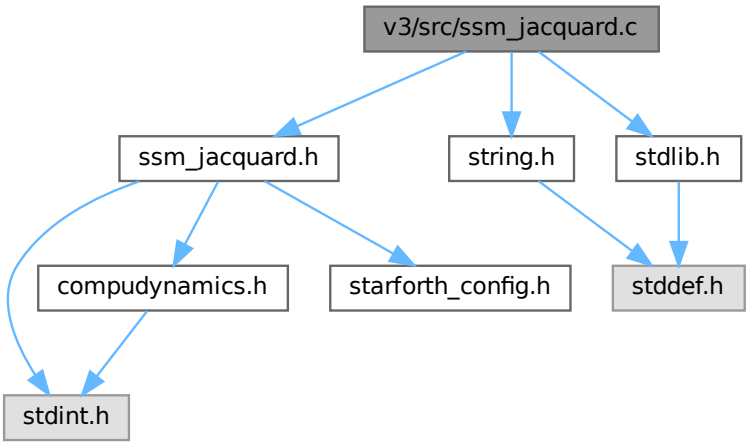
Here is the call graph for this function:



7.307 v3/src/ssm_jacquard.c File Reference

```
#include "ssm_jacquard.h"
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for ssm_jacquard.c:



Functions

- void ssm_l8_init (ssm_l8_state_t *state, ssm_l8_mode_t initial_mode)

  Initialise the L8 Jacquard steady-state machine.
- void ssm_l8_update (const ssm_l8_metrics_t *metrics, ssm_l8_state_t *state)

  Update the L8 mode selector using current runtime metrics.
- void ssm_apply_mode (const ssm_l8_state_t *state, ssm_config_t *config)

  Apply the current L8 mode to a runtime configuration struct.
- const char * ssm_l8_mode_name (ssm_l8_mode_t mode)

  Return a human-readable name string for the given L8 mode.
- static void ssm_l8_seed_from_doe (CDConfigTable *table)

  Seed the adaptive UCB config table with DoE prior scores.
- static void ssm_l8_tick_score_and_select (CDConfigTable *table, ssm_config_t *config, const ssm_l8_metrics_t *metrics, uint32_t current_window, uint32_t locality_q16)

  Score the current config from this tick's outcome and UCB-select this tick's config.
- void ssm_l8_init_table (ssm_l8_state_t *state)

  Allocate and initialise the adaptive UCB config table.
- void ssm_l8_free_table (ssm_l8_state_t *state)

  Free the adaptive UCB config table owned by `state`.
- void ssm_l8_update_table (ssm_l8_state_t *state, const ssm_l8_metrics_t *metrics, ssm_config_t *config, uint32_t current_window, const uint32_t *recent_word_ids, uint32_t recent_word_ids_count)

  Tick the adaptive UCB table: classify, score, and reselect.
- void ssm_apply_mode_from_table (const ssm_l8_state_t *state, ssm_config_t *config)

  Apply the UCB-selected config bits to a runtime configuration struct.
- void ssm_l8_force_config (ssm_l8_state_t *state, ssm_config_t *config, uint8_t config_idx)

  Force the adaptive table onto an externally-chosen config.

7.307.1 Function Documentation

7.307.1.1 ssm_apply_mode()

```
void ssm_apply_mode (
            const ssm_l8_state_t * state,
            ssm_config_t * config )
```

Apply the current L8 mode to a runtime configuration struct.

Apply L8 mode to SSM configuration (legacy threshold path)

Extracts the four loop-enable bits (L2/L3/L5/L6) from `state->current_mode` and writes them into `config`. Used by the legacy (non-table) L8 path when the adaptive UCB table is not present.

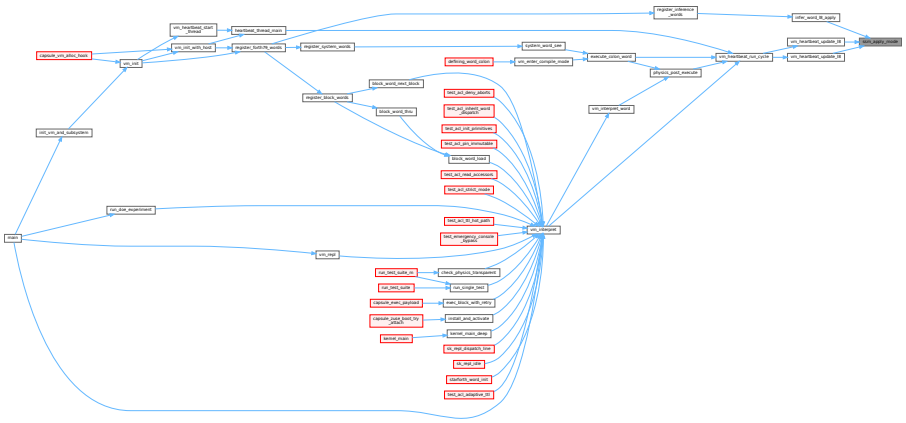
Parameters

state	L8 state containing the committed current mode
config	Runtime configuration to update

References ssm_l8_state_t::current_mode, ssm_config_t::L2_rolling_window, ssm_config_t::L3_linear_decay, ssm_config_t::L5_window_inference, and ssm_config_t::L6_decay_inference.

Referenced by infer_word_l8_apply(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the caller graph for this function:



7.307.1.2 ssm_apply_mode_from_table()

```
void ssm_apply_mode_from_table (
            const ssm_l8_state_t * state,
            ssm_config_t * config )
```

Apply the UCB-selected config bits to a runtime configuration struct.

Apply the current table config to ssm_config_t Used on first tick to set the initial DoE-seeded config immediately.

Reads the current winning config entry from the adaptive table and writes all 7 of its bits into `config`. Use this instead of `ssm_apply_mode()` when the adaptive table is present (`state->table` != NULL).

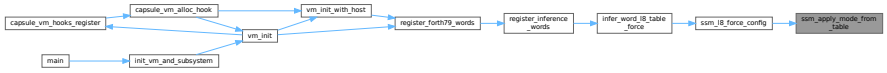
Parameters

state	L8 state with an attached, initialised adaptive table
config	Runtime configuration to update

References CDConfigEntry::config_bits, CDConfigTable::current_config, CDConfigTable::entries, ssm_config_t::L1_heat_tracking, ssm_config_t::L2_rolling_window, ssm_config_t::L3_linear_decay, ssm_config_t::L4_pipelining, ssm_config_t::L5_window_inference, ssm_config_t::L6_decay_inference, ssm_config_t::L7_adaptive_heartrate, SSM_CFG_L1, SSM_CFG_L2, SSM_CFG_L3, SSM_CFG_L4, SSM_CFG_L5, SSM_CFG_L6, SSM_CFG_L7, and ssm_l8_state_t::table.

Referenced by ssm_l8_force_config().

Here is the caller graph for this function:



7.307.1.3 ssm_l8_force_config()

```
void ssm_l8_force_config (
            ssm_l8_state_t * state,
            ssm_config_t * config,
            uint8_t config_idx )
```

Force the adaptive table onto an externally-chosen config.

Force the adaptive table onto an externally-chosen config (e.g.

Sets `current_config` directly (as if the bandit's own UCB selection had picked it) and applies its bits immediately. Since scoring and UCB reselection now run on every heartbeat tick (rev r: one clock, no batching of any kind), this is a one-tick nudge, not a hold: the forced config earns exactly one tick's worth of reward signal into its own regime_score, then the very next tick's ssm_l8_update_table() call scores it and reselects on its own, same as any other tick. Calling this repeatedly (e.g. once per DoE draw, faster than the VM's own tick cadence) keeps a config in effect in practice, not because it's pinned – because it's what's currently forced, most recently.

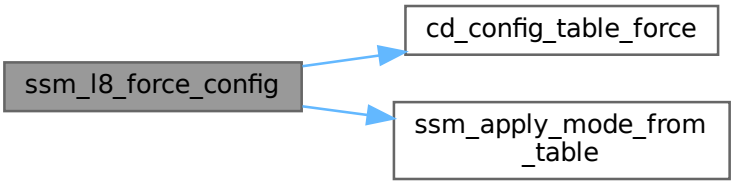
Parameters

state	L8 state with an attached, initialised adaptive table
config	Runtime configuration to update immediately
config_idx	Desired config, masked to the 128-entry (7-bit) space

References cd_config_table_force(), ssm_apply_mode_from_table(), and ssm_l8_state_t::table.

Referenced by infer_word_l8_table_force().

Here is the call graph for this function:



Here is the caller graph for this function:

7.307.1.4 ssm_l8_free_table()

```
void ssm_l8_free_table (
            ssm_l8_state_t * state )
```

Free the adaptive UCB config table owned by `state`.

Free the adaptive config table.

Safe to call with `state->table` == NULL. Sets `state->table` to NULL after freeing so the state falls back to legacy mode.

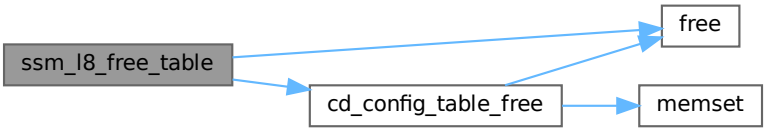
Parameters

state	L8 state whose table is freed

References cd_config_table_free(), free(), and ssm_l8_state_t::table.

Referenced by vm_cleanup().

Here is the call graph for this function:



Here is the caller graph for this function:

7.307.1.5 ssm_l8_init()

```
void ssm_l8_init (
            ssm_l8_state_t * state,
            ssm_l8_mode_t initial_mode )
```

Initialise the L8 Jacquard steady-state machine.

Initialize L8 state (legacy fields only; call ssm_l8_init_table separately)

Sets `current_mode` and `pending_mode` to `initial_mode`, zeroes `hysteresis_counter`, and sets `table` to NULL. Callers must invoke `ssm_l8_init_table()` separately if the adaptive UCB table is required.

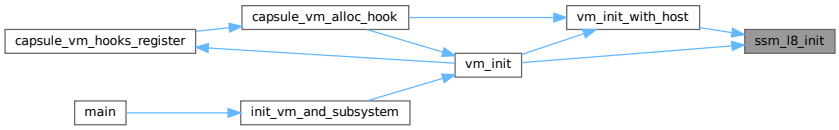
Parameters

state	Pointer to the L8 state to initialise
initial_mode	Starting mode (typically `SSM_MODE_C0`)

References ssm_l8_state_t::current_mode, ssm_l8_state_t::hysteresis_counter, ssm_l8_state_t::pending_mode, and ssm_l8_state_t::table.

Referenced by vm_init(), and vm_init_with_host().

Here is the caller graph for this function:



7.307.1.6 ssm_l8_init_table()

```
void ssm_l8_init_table (
            ssm_l8_state_t * state )
```

Allocate and initialise the adaptive UCB config table.

Allocate and seed the adaptive config table Sets state->table; must be called after ssm_l8_init().

Mallocs and initialises a `CDConfigTable` sized by `cd_tuning_word()`, seeds it with `ssm_l8_seed_from_doe()`, and attaches it to `state->table`. If either allocation fails, `state->table` is set to NULL and the system falls back to legacy mode (`ssm_apply_mode()`).

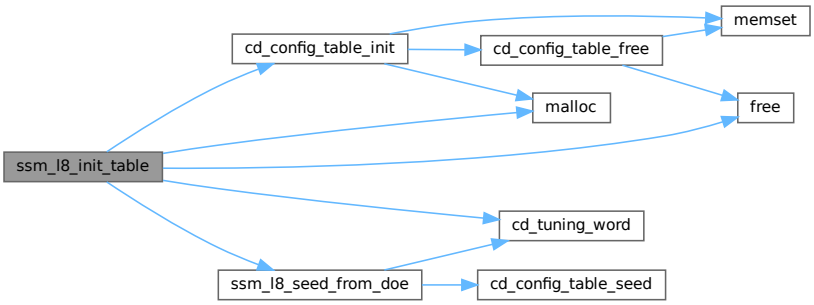
Parameters

state	L8 state that will own the table

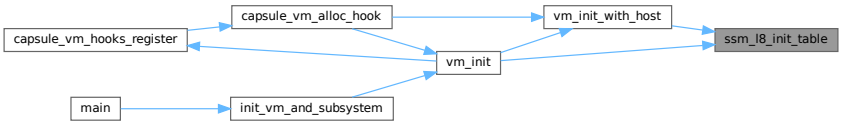
References cd_config_table_init(), cd_tuning_word(), free(), malloc(), ssm_l8_seed_from_doe(), and ssm_l8_state_t::table.

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.307.1.7 ssm_l8_mode_name()

```
const char * ssm_l8_mode_name (
            ssm_l8_mode_t mode )
```

Return a human-readable name string for the given L8 mode.

Get human-readable mode name.

Returns a static string such as "C4_TEMPORAL" or "C15_FULL_ADAPTIVE". Modes marked ? TOP 5% are the configurations favoured by the DoE study. Returns "UNKNOWN" for out-of-range values.

Parameters

mode	L8 mode enumeration value

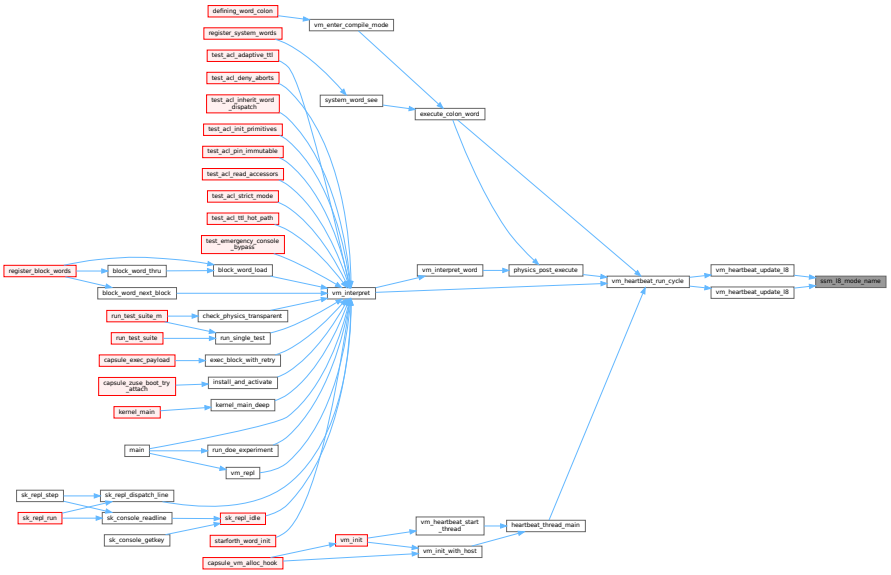
Returns

Pointer to a static null-terminated string (never NULL)

References SSM_MODE_C0, SSM_MODE_C1, SSM_MODE_C10, SSM_MODE_C11, SSM_MODE_C12, SSM_MODE_C13, SSM_MODE_C14, SSM_MODE_C15, SSM_MODE_C2, SSM_MODE_C3, SSM_MODE_C4, SSM_MODE_C5, SSM_MODE_C6, SSM_MODE_C7, SSM_MODE_C8, and SSM_MODE_C9.

Referenced by vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the caller graph for this function:



7.307.1.8 ssm_l8_seed_from_doe()

```
static void ssm_l8_seed_from_doe (
            CDConfigTable * table )  [static]
```

Seed the adaptive UCB config table with DoE prior scores.

Computes benefit scores derived from the DoE top-5% analysis for all `cd_tuning_word()`.num_configs (128) configs and hands them to `cd_config_table_seed()`. The five top-5% configs (C4/C7/C9/C11/C12 in 4-bit space → indices 17/23/35/39/49 in 7-bit space) receive 80% of `cd_tuning_word()`.score_max; configs involving L1 or L4 (known expensive loops) receive 5–10%; all others receive 40%. Sets the starting config to index 17 (C4-equivalent: L3+L7 only).

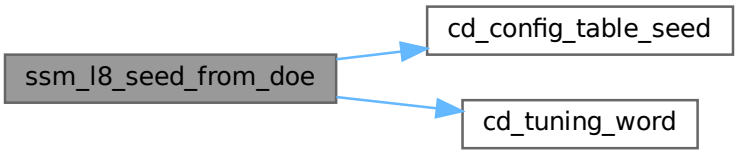
Parameters

table	Pointer to the CDConfigTable to seed (already initialised via cd_config_table_init())

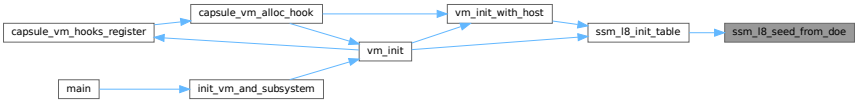
References cd_config_table_seed(), cd_tuning_word(), CDTuning::num_configs, CDTuning::score_max, SSM_CFG_L1, and SSM_CFG_L4.

Referenced by ssm_l8_init_table().

Here is the call graph for this function:



Here is the caller graph for this function:



7.307.1.9 ssm_l8_tick_score_and_select()

```
static void ssm_l8_tick_score_and_select (
            CDConfigTable * table,
            ssm_config_t * config,
            const ssm_l8_metrics_t * metrics,
            uint32_t current_window,
            uint32_t locality_q16 )  [static]
```

Score the current config from this tick's outcome and UCB-select this tick's config.

Computes a joint convergence signal from this tick's ANOVA outcome and the relative delta between this tick's and the previous tick's (window, locality) pair. Updates the regime-specific benefit score for the current config using exponential decay plus a signed delta – the decay is what provides smoothing over time; there's no separate multi-tick batching on top of it (VM-FLEET-ATTRACTOR-DESIGN-20260705.md rev r: one clock, the heartbeat tick – a tick is a tick). Then runs UCB1 over all 128 configs for the current regime, selecting the one with the highest upper confidence bound as this tick's config.

Parameters

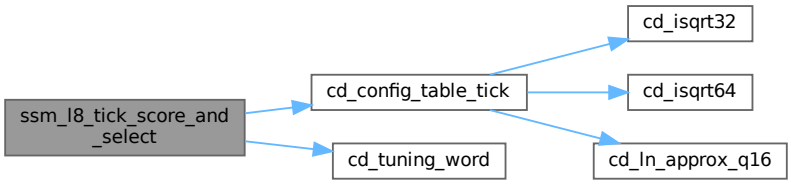
table	Adaptive config table
config	Runtime config to update with the newly selected bits
metrics	This tick's metrics snapshot (for the ANOVA signal)
current_window	Current rolling window width, this tick
locality_q16	Current locality signal from cd_classify_ids(), this tick

References cd_config_table_tick(), cd_tuning_word(), CDConfigEntry::config_bits, CDConfigTable::current_config, CDConfigTable::current_regime, CDConfigTable::entries, ssm_l8_metrics_t::inference_early_exited, ssm_l8_metrics_t::inference_ran

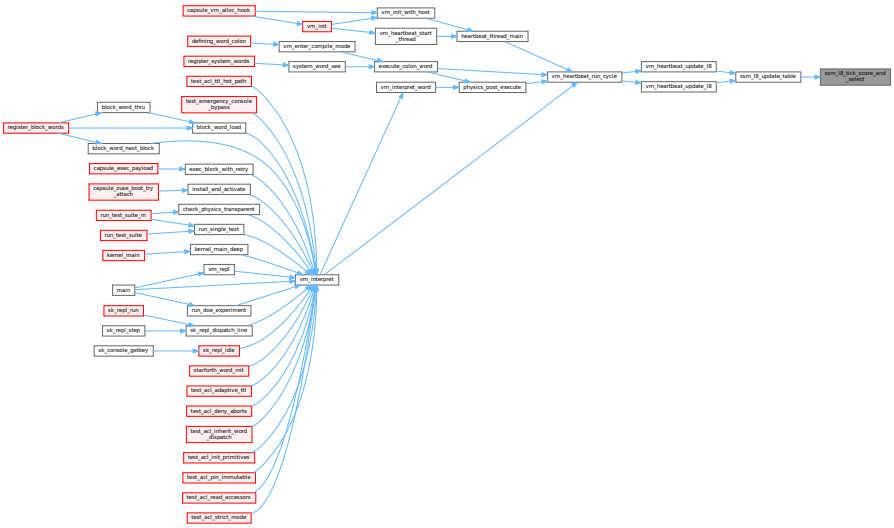
ssm_config_t::L1_heat_tracking, ssm_config_t::L2_rolling_window, ssm_config_t::L3_linear_decay, ssm_config_t::L4_pipelining, ssm_config_t::L5_window_inference, ssm_config_t::L6_decay_inference, ssm_config_t::L7_adaptive_heartrate, SSM_CFG_L1, SSM_CFG_L2, SSM_CFG_L3, SSM_CFG_L4, SSM_CFG_L5, SSM_CFG_L6, and SSM_CFG_L7.

Referenced by ssm_l8_update_table().

Here is the call graph for this function:



Here is the caller graph for this function:



7.307.1.10 ssm_l8_update()

```
void ssm_l8_update (
            const ssm_l8_metrics_t * metrics,
            ssm_l8_state_t * state )
```

Update the L8 mode selector using current runtime metrics.

Update L8 state based on current metrics (legacy threshold path)

Classifies `metrics` into four binary feature bits (L2/L3/L5/L6) derived from DoE top-5% analysis and computes a target mode. Applies hysteresis: the mode only commits after `SSM_HYSTERESIS_TICKS` consecutive ticks with the same target. Called by the heartbeat thread on every tick.

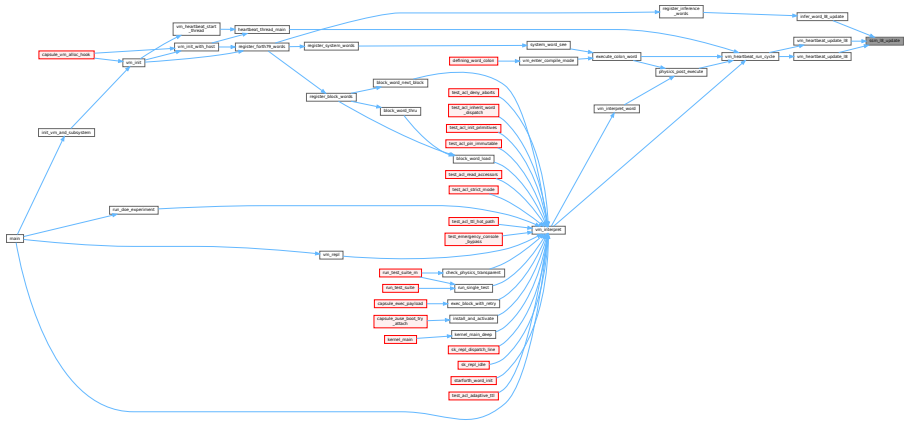
Parameters

metrics	Runtime metrics snapshot (entropy, CV, temporal decay)
state	L8 state to update; `current_mode` is written on commit

References ssm_l8_state_t::current_mode, ssm_l8_metrics_t::cv, ssm_l8_metrics_t::entropy, ssm_l8_state_t::hysteresis_counter, ssm_l8_state_t::pending_mode, SSM_CV_HIGH_THRESHOLD, SSM_ENTROPY_HIGH_THRESHOLD, SSM_HYSTERESIS_TICK SSM_MODE_C0, SSM_TEMPORAL_DECAY_LOW_THRESHOLD, SSM_TEMPORAL_DECAY_THRESHOLD, and ssm_l8_metrics_t::temporal_decay.

Referenced by infer_word_l8_update(), vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the caller graph for this function:



7.307.1.11 ssm_l8_update_table()

```
void ssm_l8_update_table (
            ssm_l8_state_t * state,
            const ssm_l8_metrics_t * metrics,
            ssm_config_t * config,
            uint32_t current_window,
            const uint32_t * recent_word_ids,
            uint32_t recent_word_ids_count )
```

Tick the adaptive UCB table: classify, score, and reselect.

Classifies the current tick into one of 8 regimes (entropy x CV x locality) and stores it unconditionally, then scores the current config from this tick's outcome and UCB-reselects, every single call – no batching, no derived counter of any kind (VM-FLEET-ATTRACTOR-DESIGN- 20260705.md rev r: one clock, the heartbeat tick, applied uniformly). The locality component of both the regime and the joint-convergence reward comes from cd_classify_ids() over recent_word_ids – purely execution-count-derived, no wall-clock input (rev s; see header comment on this function's declaration for why that matters). Also mirrors the winning L2/L3/L5/L6 bits back into `state->current_mode` for diagnostic display. No-op if `state->table` is NULL.

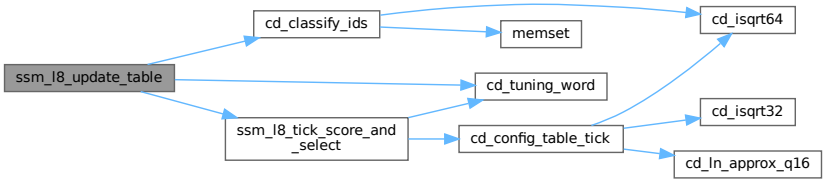
Parameters

state	L8 state with attached adaptive table
metrics	Current runtime metrics snapshot
config	Runtime config updated this tick
current_window	Current rolling window width
recent_word_ids	Recent word-execution-ID history, oldest first
recent_word_ids_count	Number of valid entries in recent_word_ids

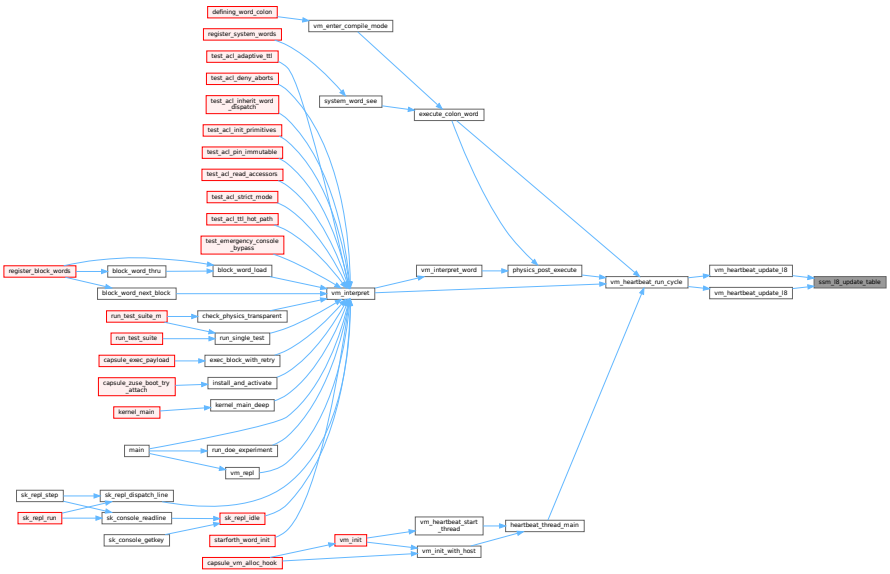
References cd_classify_ids(), cd_tuning_word(), CDConfigEntry::config_bits, CDConfigTable::current_config, ssm_l8_state_t::current_mode, CDConfigTable::current_regime, ssm_l8_metrics_t::cv, CDConfigTable::entries, ssm_l8_metrics_t::entropy, CDTuning::locality_high_threshold_q16, CDTuning::locality_lookback, CDRegimeMetrics::locality_q16, SSM_CFG_L2, SSM_CFG_L3, SSM_CFG_L5, SSM_CFG_L6, SSM_CV_HIGH_THRESHOLD, SSM_ENTROPY_HIGH_THRESHO ssm_l8_tick_score_and_select(), and ssm_l8_state_t::table.

Referenced by vm_heartbeat_update_l8(), and vm_heartbeat_update_l8().

Here is the call graph for this function:



Here is the caller graph for this function:



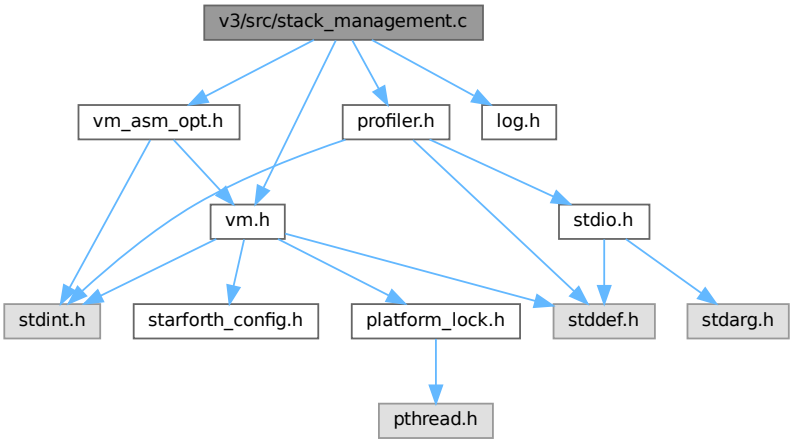
7.308 v3/src/stack_management.c File Reference

Implementation of stack operations for the Forth virtual machine.

```
#include "vm.h"
#include "log.h"
#include "profiler.h"
```

```
#include "vm_asm_opt.h"
```

Include dependency graph for stack_management.c:



Functions

- void vm_push (VM *vm, cell_t value)

  Push a value onto the data stack.
- cell_t vm_pop (VM *vm)

  Pop a value from the data stack.
- void vm_rpush (VM *vm, cell_t value)

  Push a value onto the return stack.
- cell_t vm_rpop (VM *vm)

  Pop a value from the return stack.

7.308.1 Detailed Description

Implementation of stack operations for the Forth virtual machine.

This module provides the core stack manipulation functions for both the data stack and return stack of the Forth virtual machine. It includes optimized assembly implementations for x86_64 and ARM64 architectures when available.

7.308.2 Function Documentation

7.308.2.1 vm_pop()

```
cell_t vm_pop (
            VM * vm )
```

Pop a value from the data stack.

Parameters

vm	Pointer to the VM structure

Returns

The popped value

Note

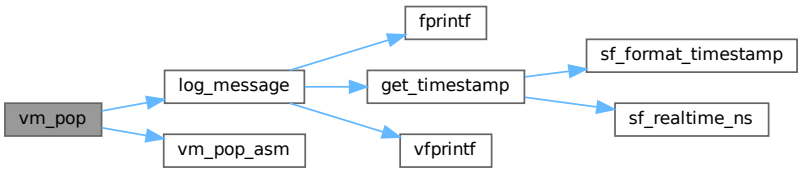
Returns 0 and sets error flag if stack underflow occurs

Uses assembly optimization if available for x86_64 or ARM64

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), PROFILE_INC_STACK_OP, and vm_pop_asm().

Referenced by arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), block_word_acl_allow_fetch(), block_word_acl_allow_store(), block_word_acl_ttl_fetch(), block_word_acl_ttl_store(), block_word_blk_attach(), block_word_block(), block_word_buffer(), block_word_confirm_format(), block_word_list(), block_word_load(), block_word_owner_fetch(), block_word_relocate(), block_word_thru(), control_forth_0branch(), control_forth_runtime_do(), control_forth_runtime_plus_loop(), control_forth_runtime_qdo(), defining_word_constant(), defining_word_literal(), dictionary_m_word_cfa(), dictionary_m_word_link_from(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), dictionary_m_word_traverse(), dictionary_word_2comma(), dictionary_word_allot(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_sp_store(), double_word_2drop(), double_word_2rot(), double_word_2swap(), double_word_2to_r(), double_word_d_equals(), double_word_d_less(), double_word_d_minus(), double_word_d_plus(), double_word_d_two_slash(), double_word_d_two_star(), double_word_d_zero_equals(), double_word_d_zero_less(), double_word_dmax(), double_word_dmin(), double_word_dnegate(), double_word_s_to_d(), editor_word_edit(), editor_word_l(), editor_word_s(), fbw_plot(), format_word_d_dot(), format_word_d_dot_r(), format_word_dot(), format_word_dot_r(), format_word_dump(), format_word_end_conversion(), format_word_hash(), format_word_hash_s(), format_word_hold(), format_word_question(), format_word_sign(), format_word_u_dot(), format_word_u_dot_r(), forth_acl_allow_store(), forth_acl_mode_store(), forth_acl_ttl_store(), forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), hosted_lifecycle_birth(), hosted_lifecycle_kill(), hosted_lifecycle_pause(), hosted_lifecycle_resume(), hosted_lifecycle_use(), io_word_type(), log_str_emit(), log_word_set_level(), logical_word_and(), logical_word_equals(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), memory_word_2fetch(), memory_word_2store(), memory_word_cells(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), mixed_math_word_m_minus(), mixed_math_word_m_plus(), mixed_math_word_m_slash_mod(), mixed_math_word_m_star(), mixed_math_word_mod(), mixed_math_word_mod(), mixed_math_word_slash_mod(), mixed_math_word_star_slash(), mixed_math_word_star_slash_mod(), pop_xt(), return_stack_word_to_r(), runtime_abortq(), sk_word_blk_attach_ack(), sk_word_kh_blk_attach_send(), stack_word_pick(), starforth_word_execution_heat_fetc starforth_word_execution_heat_store(), starforth_word_random(), starforth_word_seed(), starforth_word_top_words(), starforth_word_wait(), starforth_word_zuse_pubkey_fetch(), string_word_blank(), string_word_cmove(), string_word_cmove_greater() string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_minus_trailing(), string_word_number(), string_word_scan(), string_word_search(), string_word_skip(), string_word_word(), system_word_abort_quote(), system_word_execute(), system_word_reboot(), test_acl_adaptive_ttl(), test_acl_read_accessors(), vm_api_pop(), word_divide(), word_divmod(), and word_is().

Here is the call graph for this function:



7.308.2.2 vm_push()

```
void vm_push (
            VM * vm,
            cell_t value )
```

Push a value onto the data stack.

Parameters

vm	Pointer to the VM structure
value	Value to push onto the stack

Note

Performs bounds checking to prevent stack overflow

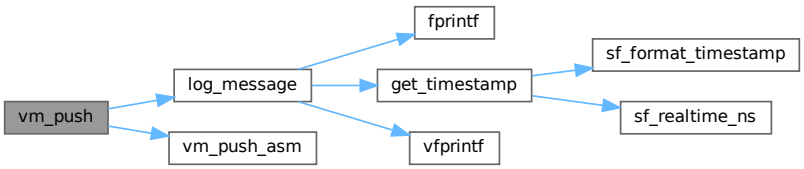
Uses assembly optimization if available for x86_64 or ARM64

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), PROFILE_INC_STACK_OP, STACK_SIZE, and vm_push_asm().

Referenced by acl_recheck(), acl_recheck(), arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), block_word_acl_allow_fetch(), block_word_acl_ttl_fetch(), block_word_blk_attach(), block_word_block(), block_word_buffer(), block_word_owner_fetch(), block_word_scr(), block_word_thru(), control_forth_I(), control_forth_J(), defining_runtime_constant(), defining_runtime_create(), defining_runtime_dodoes(), defining_runtime_lit(), defining_runtime_variable(), defining_word_state(), dictionary_m_word_cfa(), dictionary_m_word_find(), dictionary_m_word_link_from(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_tick(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), dictionary_m_word_traverse(), dictionary_word_here(), dictionary_word_latest(), dictionary_word_pad(), dictionary_word_sp_fetch(), double_word_2dup(), double_word_2over(), double_word_2r_fetch(), double_word_2r_from(), double_word_2rot(), double_word_2swap(), double_word_d_equals(), double_word_d_less(), double_word_d_minus(), double_word_d_plus(), double_word_d_two_slash(), double_word_d_two_star(), double_word_d_zero_equals(), double_word_d_zero_less(), double_word_dmax(), double_word_dmin(), double_word_dnegate(), double_word_s_to_d(), editor_word_edit(), emit_cell(), fbw_height(), fbw_width(), format_word_base(), format_word_end_conversion(), format_word_hash(), format_word_hash_s(), format_word_sign(), forth_acl_allow_fetch(), forth_acl_heat_fetch(), forth_acl_mode_fetch(), forth_acl_pinned_query(), forth_acl_ttl_fetch(), forth_acl_word_id(), kbw_debug(), kbw_key_event(), kbw_scan(), kbw_vdebug(), kbw_vevent(), log_word_debug(), log_word_error(), log_word_get_level(), log_word_info(), log_word_test(), log_word_warn(),

logical_word_and(), logical_word_equals(), logical_word_false(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_true(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), memory_word_2fetch(), memory_word_cells(), memory_word_cfetch(), memory_word_fetch(), mixed_math_word_m_minus(), mixed_math_word_m_plus(), mixed_math_word_m_slash_mod(), mixed_math_word_m_star(), mixed_math_word_mod(), mixed_math_word_mod(), mixed_math_word_slash_mod(), mixed_math_word_star_slash(), mixed_math_word_star_slash_mod(), return_stack_word_r_fetch(), return_stack_word_r_from(), sk_word_kh_blk_attach_send(), stack_word_pick(), starforth_word_execution_heat_fetch(), starforth_word_heartbeat_ticks(), starforth_word_init(), starforth_word_random(), starforth_word_zuse_cert_installed_query(), starforth_word_zuse_pubkey_fetch(), starforth_word_zuse_session(), string_runtime_s_quote(), string_word_bl(), string_word_bracket_tick(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_minus_trailing(), string_word_number(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), string_word_source(), string_word_span(), string_word_tib(), string_word_to_in(), string_word_word(), system_word_79_standard(), test_acl_adaptive_ttl(), test_acl_inherit_word_dispatch(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), vm_api_push(), vm_compile_literal(), vm_interpret_word(), vocabulary_word_context(), vocabulary_word_current(), vocabulary_word_paren_find(), word_defer_fetch(), word_divide(), and word_divmod().

Here is the call graph for this function:



7.308.2.3 vm_rpop()

```
cell_t vm_rpop (
            VM * vm )
```

Pop a value from the return stack.

Parameters

vm	Pointer to the VM structure

Returns

The popped value

Note

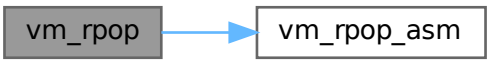
Returns 0 and sets error flag if stack underflow occurs

Uses assembly optimization if available for x86_64 or ARM64

References VM::error, VM::return_stack, VM::rsp, and vm_rpop_asm().

Referenced by execute_colon_word(), return_stack_word_r_from(), and vm_api_rpop().

Here is the call graph for this function:



Here is the caller graph for this function:

7.308.2.4 vm_rpush()

```
void vm_rpush (
            VM * vm,
            cell_t value )
```

Push a value onto the return stack.

Parameters

vm	Pointer to the VM structure
value	Value to push onto the return stack

Note

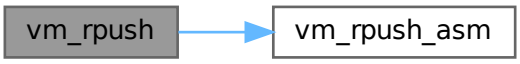
Sets error flag if stack overflow occurs

Uses assembly optimization if available for x86_64 or ARM64

References VM::error, VM::return_stack, VM::rsp, STACK_SIZE, and vm_rpush_asm().

Referenced by execute_colon_word(), return_stack_word_to_r(), and vm_api_rpush().

Here is the call graph for this function:



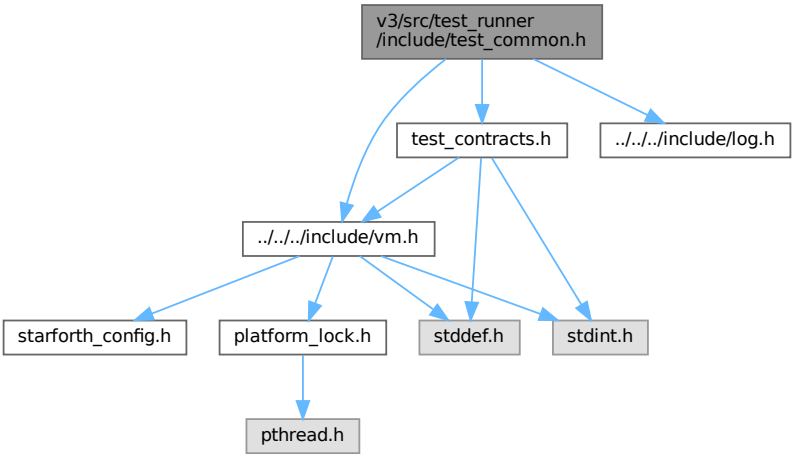
Here is the caller graph for this function:

7.309 v3/src/test_runner/include/test_common.h File Reference

Common test utilities and structures for StarForth test runner.

```
#include "../../../include/vm.h"
#include "../../../include/log.h"
#include "test_contracts.h"
```

Include dependency graph for test_common.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct TestCase

  Structure representing a single test case.
- struct WordTestSuite

  Collection of test cases for a single Forth word.

Macros

Test Configuration

- #define MAX_TEST_OUTPUT 1024

  Maximum length of test output buffer.
- #define MAX_TEST_INPUT 256

  Maximum length of test input buffer.
- #define MAX_TESTS_PER_WORD 20

  Maximum number of test cases per word.

Enumerations

- enum TestType { TEST_NORMAL = 0 , TEST_EDGE_CASE = 1 , TEST_ERROR_CASE = 2 }

  Types of test cases.

Functions

Core Test Execution Functions

- TestResult run_single_test (VM *vm, const char *word_name, const TestCase *test)

  Executes a single test case.
- void run_test_suite (VM *vm, const WordTestSuite *suite)

  Executes a complete test suite for a Forth word.
- void run_test_suite_m (VM *vm, const WordTestSuite *suite, WordContract module_contract)

  Runs all tests in a suite with an inherited module-level contract.
- void print_module_summary (const char *module_name, int pass, int fail, int skip, int error)

  Prints test module execution summary.

Test Assertion Functions

- int assert_stack_depth (VM *vm, int expected_depth)

  Verifies VM stack depth matches expected value.
- int assert_stack_top (VM *vm, int expected_value)

  Verifies top value on VM stack matches expected value.
- int assert_vm_error (VM *vm, int should_have_error)

  Verifies VM error state matches expectation.

VM State Management Functions

- void save_vm_state (VM *vm, int *dsp, int *rsp, int *error, vm_mode_t *mode)

  Saves current VM state.
- void restore_vm_state (VM *vm, int dsp, int rsp, int error, vm_mode_t mode)

  Restores previously saved VM state.
- void save_dict_state (VM *vm, DictEntry **latest, size_t *here)

  Saves current dictionary state before test execution.
- void restore_dict_state (VM *vm, DictEntry *latest, size_t here)

  Restores dictionary state after test execution.

Variables

- int fail_fast

  Global flag to control test execution behavior.

7.309.1 Detailed Description

Common test utilities and structures for StarForth test runner.

This file contains test configuration, structures, and utility functions used throughout the StarForth test framework.

7.309.2 Macro Definition Documentation

7.309.2.1 MAX_TEST_INPUT

```
#define MAX_TEST_INPUT 256
```

Maximum length of test input buffer.

7.309.2.2 MAX_TEST_OUTPUT

```
#define MAX_TEST_OUTPUT 1024
```

Maximum length of test output buffer.

7.309.2.3 MAX_TESTS_PER_WORD

```
#define MAX_TESTS_PER_WORD 20
```

Maximum number of test cases per word.

7.309.3 Enumeration Type Documentation

7.309.3.1 TestType

```
enum TestType
```

Types of test cases.

Enumerator

TEST_NORMAL	Standard test case.
TEST_EDGE_CASE	Edge case test.
TEST_ERROR_CASE	Error condition test.

7.309.4 Function Documentation

7.309.4.1 assert_stack_depth()

```
int assert_stack_depth (
            VM * vm,
            int expected_depth )
```

Verifies VM stack depth matches expected value.

Parameters

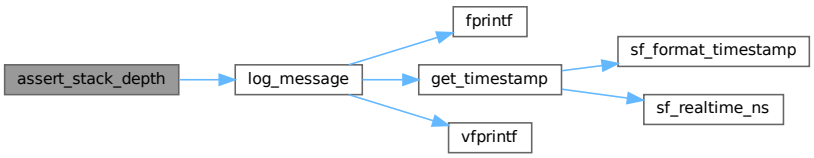
vm	Pointer to VM instance
expected_depth	Expected stack depth

Returns

1 if assertion passes, 0 if it fails

References VM::dsp, LOG_ERROR, and log_message().

Here is the call graph for this function:



7.309.4.2 assert_stack_top()

```
int assert_stack_top (
            VM * vm,
            int expected_value )
```

Verifies top value on VM stack matches expected value.

Parameters

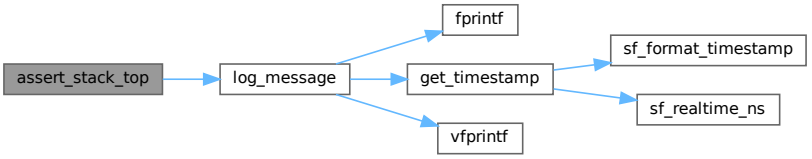
vm	Pointer to VM instance
expected_value	Expected value on top of stack

Returns

1 if assertion passes, 0 if it fails

References VM::data_stack, VM::dsp, LOG_ERROR, and log_message().

Here is the call graph for this function:



7.309.4.3 assert_vm_error()

```
int assert_vm_error (
            VM * vm,
            int should_have_error )
```

Verifies VM error state matches expectation.

Parameters

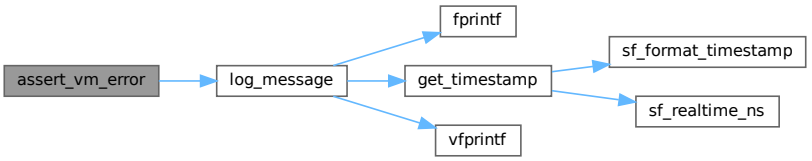
vm	Pointer to VM instance
should_have_error	Expected error state (1 for error, 0 for no error)

Returns

1 if assertion passes, 0 if it fails

References VM::error, LOG_ERROR, and log_message().

Here is the call graph for this function:



7.309.4.4 print_module_summary()

```
void print_module_summary (
            const char * module_name,
```

```
int pass,
int fail,
int skip,
int error )
```

Prints test module execution summary.

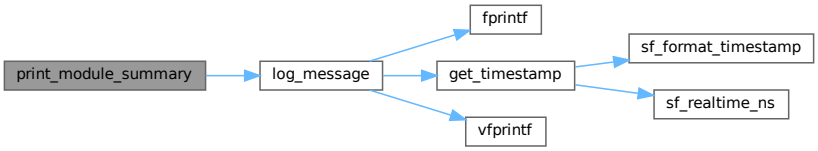
Parameters

module_name	Name of the test module
pass	Number of passed tests
fail	Number of failed tests
skip	Number of skipped tests
error	Number of tests with errors

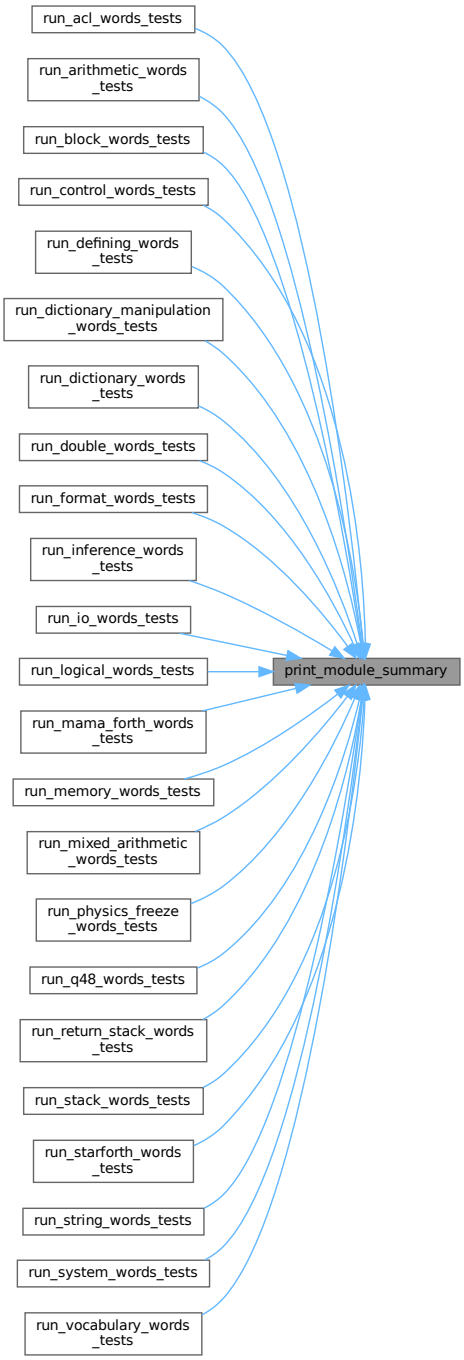
References log_message(), and LOG_TEST.

Referenced by run_acl_words_tests(), run_arithmetic_words_tests(), run_block_words_tests(), run_control_words_tests(), run_defining_words_tests(), run_dictionary_manipulation_words_tests(), run_dictionary_words_tests(), run_double_words_tests(), run_format_words_tests(), run_inference_words_tests(), run_io_words_tests(), run_logical_words_tests(), run_mama_forth_words_tests(), run_memory_words_tests(), run_mixed_arithmetic_words_tests(), run_physics_freeze_words_tests(run_q48_words_tests(), run_return_stack_words_tests(), run_stack_words_tests(), run_starforth_words_tests(), run_string_words_tests(), run_system_words_tests(), and run_vocabulary_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.309.4.5 restore_dict_state()

```
void restore_dict_state (
            VM * vm,
            DictEntry * latest,
            size_t here )
```

Restores dictionary state after test execution.

Parameters

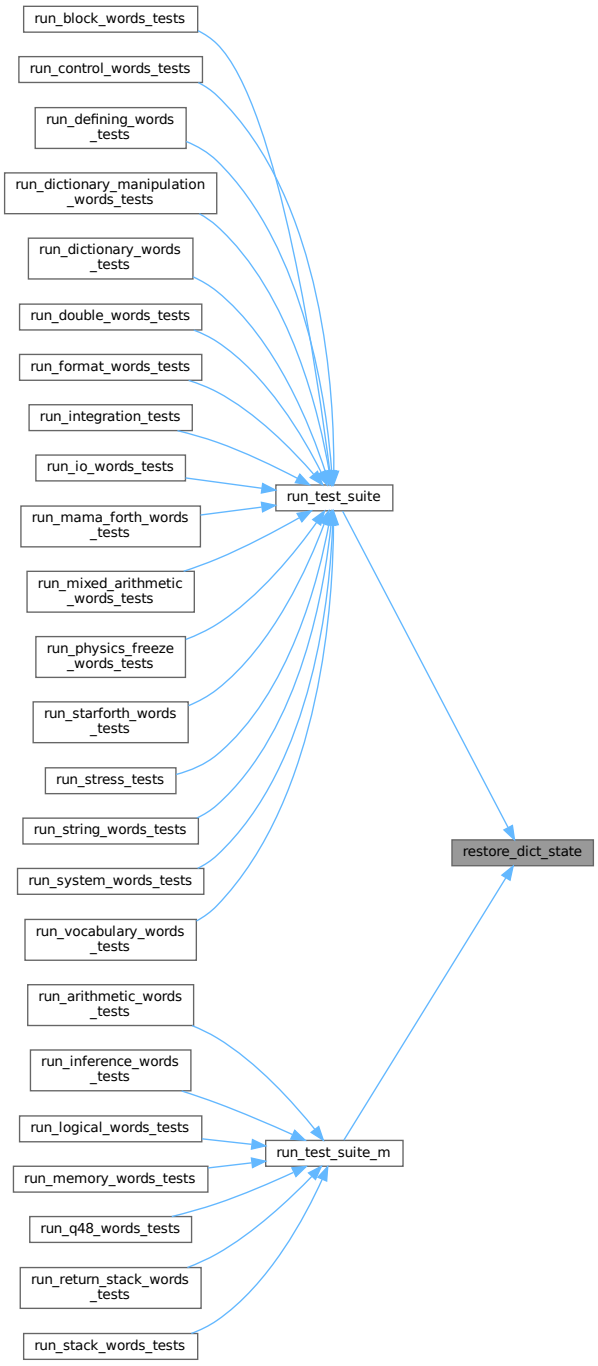
vm	Pointer to VM instance
latest	Latest dictionary entry to restore
here	HERE pointer to restore

Removes all words added during test by restoring dictionary pointers. This prevents test pollution of the dictionary and ensures each test starts with a clean slate.

References VM::here, and VM::latest.

Referenced by run_test_suite(), and run_test_suite_m().

Here is the caller graph for this function:



7.309.4.6 restore_vm_state()

```
void restore_vm_state (
            VM * vm,
            int dsp,
            int rsp,
```

```
    int error,
    vm_mode_t mode )
```

Restores previously saved VM state.

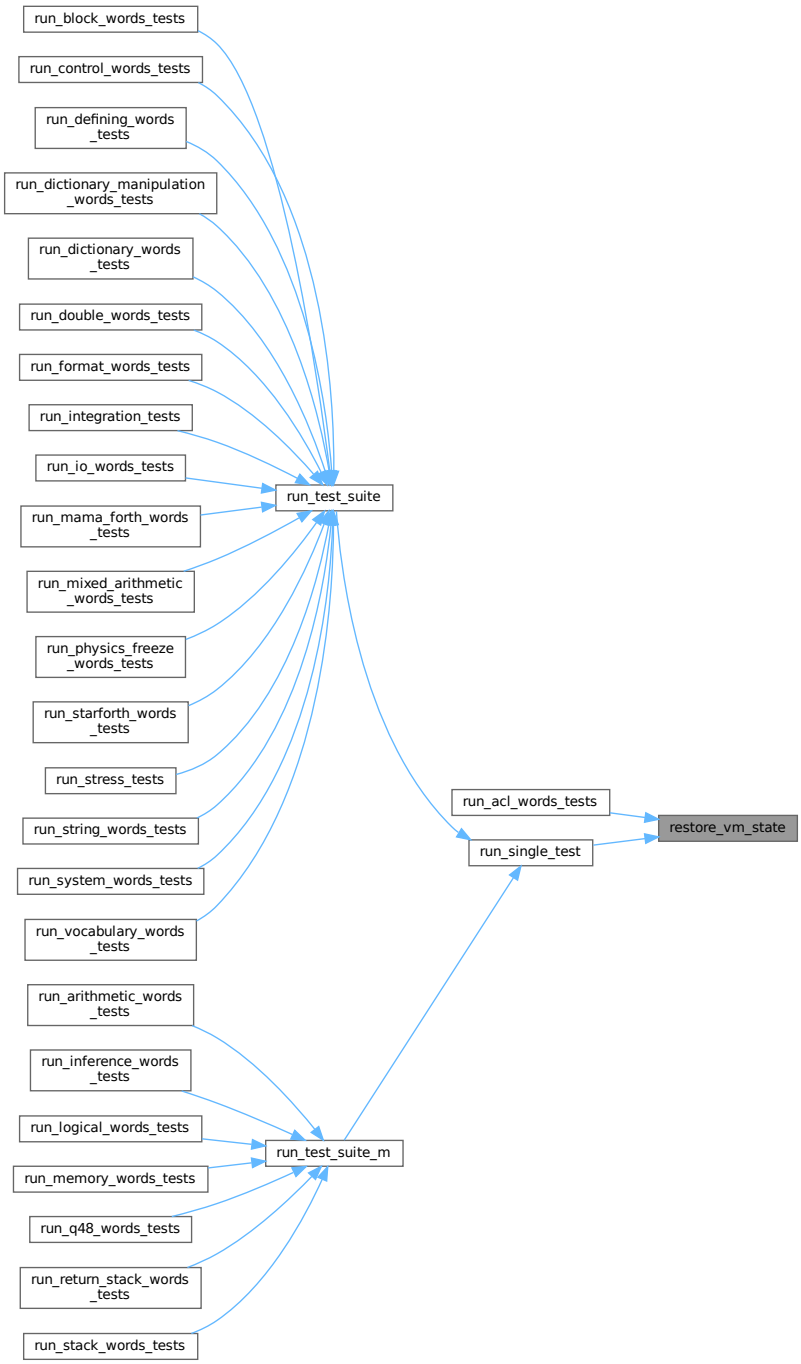
Parameters

vm	Pointer to VM instance
dsp	Data stack pointer to restore
rsp	Return stack pointer to restore
error	Error state to restore
mode	VM mode to restore

References VM::abort_requested, VM::compiling_word, VM::current_executing_entry, VM::dsp, VM::error, VM::exit_colon, VM::mode, and VM::rsp.

Referenced by run_acl_words_tests(), and run_single_test().

Here is the caller graph for this function:



7.309.4.7 run_single_test()

```
TestResult run_single_test (
            VM * vm,
            const char * word_name,
            const TestCase * test )
```

Executes a single test case.

Parameters

vm	Pointer to VM instance
word_name	Name of the word being tested
test	Pointer to test case definition

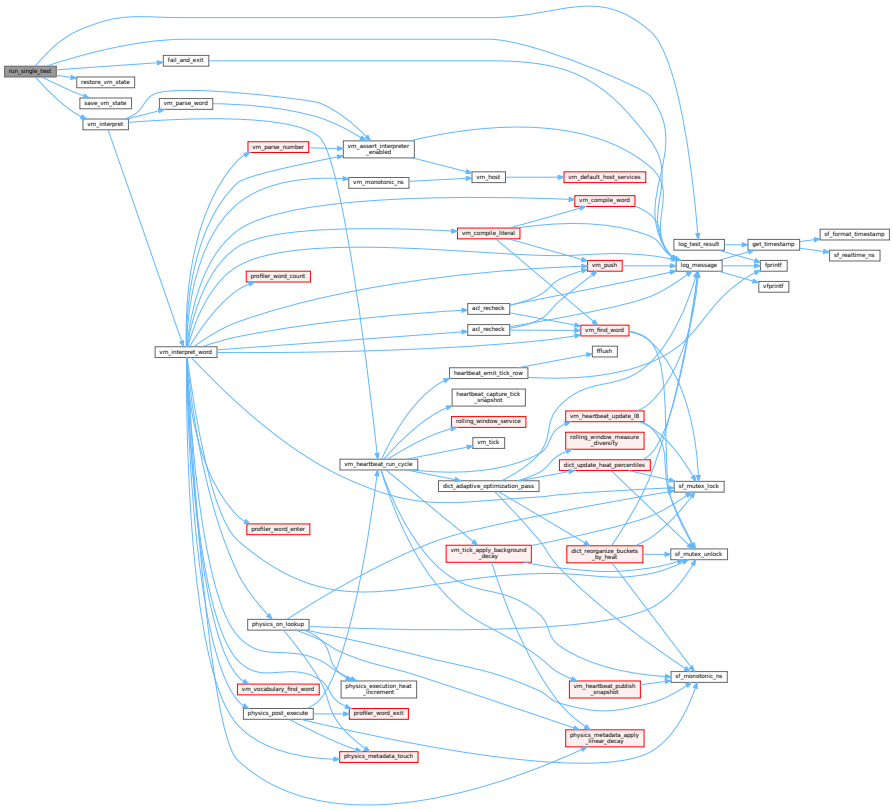
Returns

Test result (PASS, FAIL, SKIP, or ERROR)

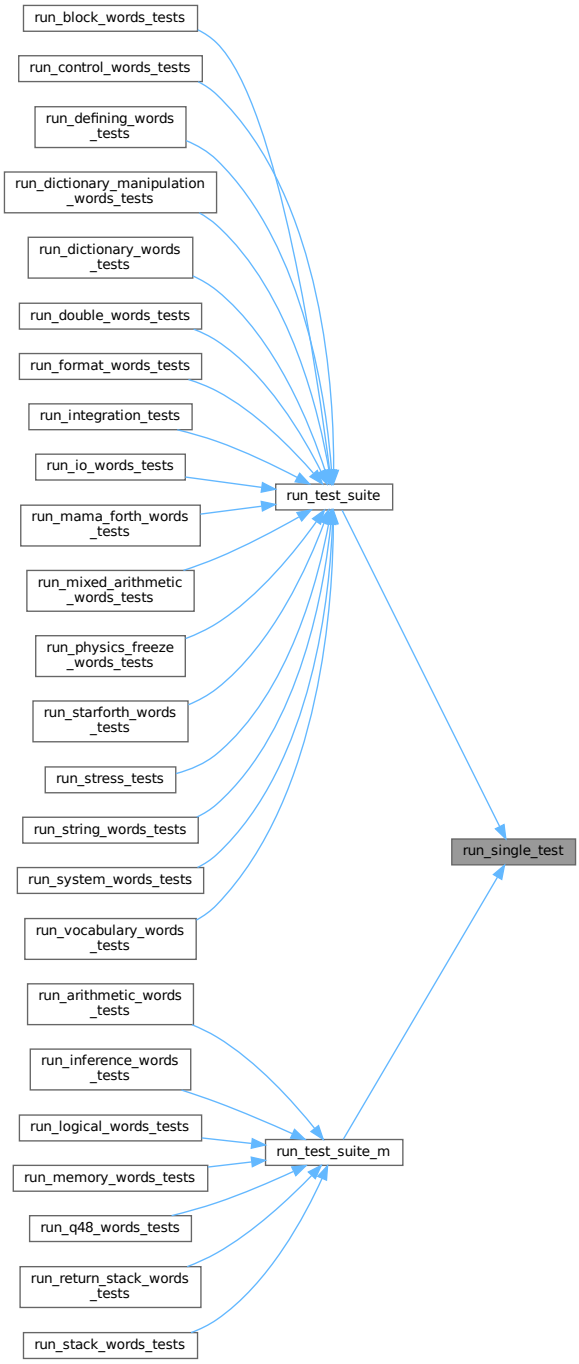
References VM::error, TestCase::expected, fail_and_exit(), fail_fast, TestCase::implemented, TestCase::input, LOG_ERROR, log_message(), LOG_TEST, log_test_result(), TestCase::name, restore_vm_state(), save_vm_state(), TestCase::should_error, TEST_FAIL, TEST_PASS, TEST_SKIP, and vm_interpret().

Referenced by run_test_suite(), and run_test_suite_m().

Here is the call graph for this function:



Here is the caller graph for this function:



7.309.4.8 run_test_suite()

```
void run_test_suite (
            VM * vm,
            const WordTestSuite * suite )
```

Executes a complete test suite for a Forth word.

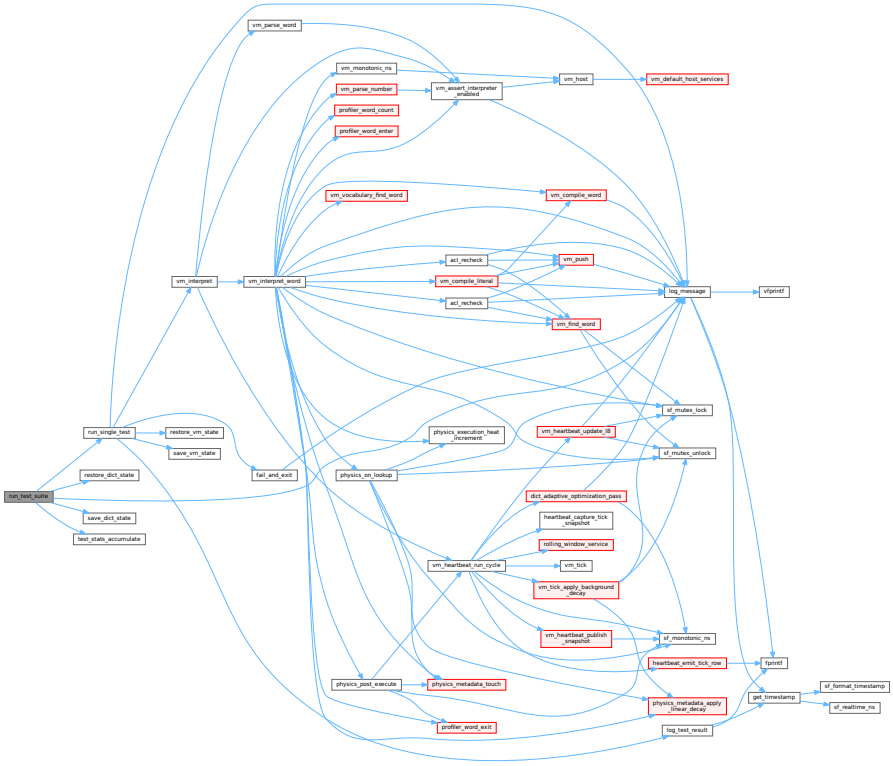
Parameters

vm	Pointer to VM instance
suite	Pointer to test suite definition

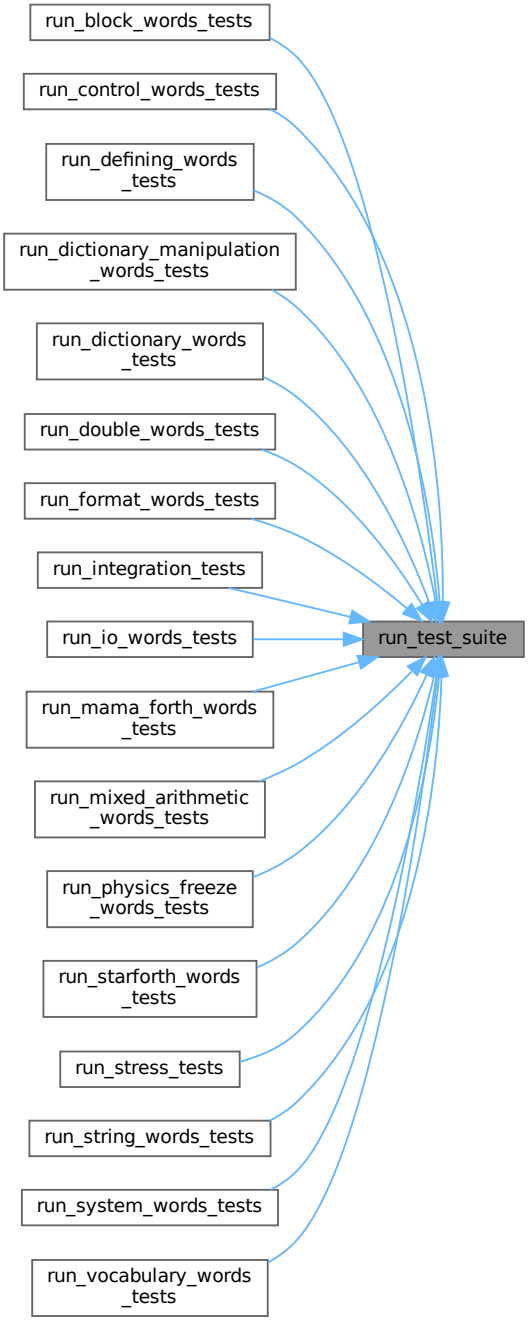
References TestCase::implemented, log_message(), LOG_TEST, TestCase::name, restore_dict_state(), run_single_test(), save_dict_state(), WordTestSuite::test_count, TEST_ERROR, TEST_FAIL, TEST_PASS, TEST_SKIP, test_stats_accumulate(), WordTestSuite::tests, and WordTestSuite::word_name.

Referenced by run_block_words_tests(), run_control_words_tests(), run_defining_words_tests(), run_dictionary_manipulation_words_ run_dictionary_words_tests(), run_double_words_tests(), run_format_words_tests(), run_integration_tests(), run_io_words_tests(), run_mama_forth_words_tests(), run_mixed_arithmetic_words_tests(), run_physics_freeze_words_tests(), run_starforth_words_tests(), run_stress_tests(), run_string_words_tests(), run_system_words_tests(), and run_vocabulary_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.309.4.9 run_test_suite_m()

```
void run_test_suite_m (
            VM * vm,
            const WordTestSuite * suite,
            WordContract module_contract )
```

Runs all tests in a suite with an inherited module-level contract.

Contract resolution order (first non-NONE wins): test->contract > suite->suite_contract > module_contract

When CONTRACTS_ENABLED=1, each passing TEST_NORMAL test is re-run under physics perturbation to witness A4' (word_physics_transparent).

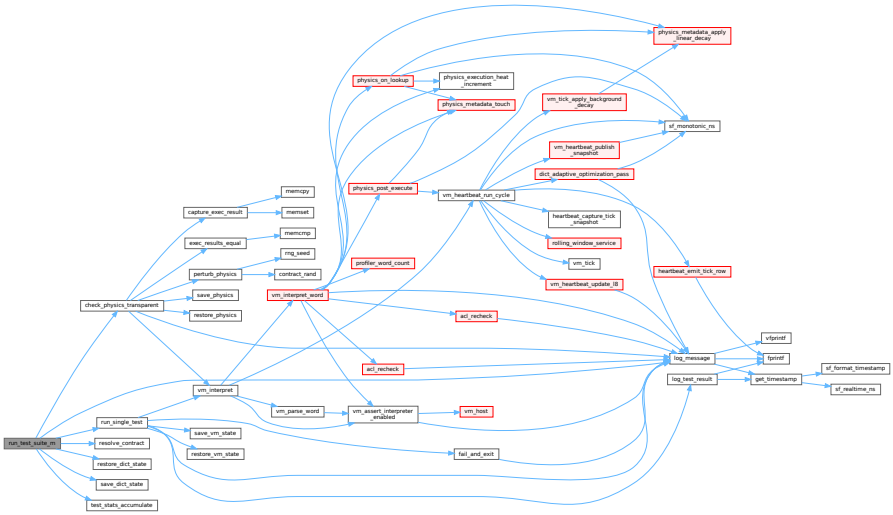
Parameters

vm	Pointer to the VM instance
suite	Test suite to execute
module_contract	Default contract for tests that don't specify one

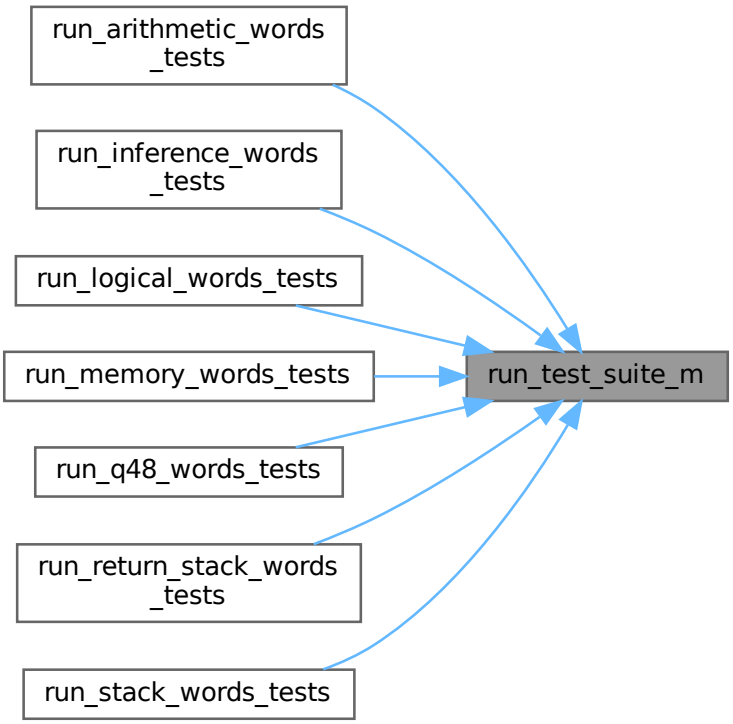
References check_physics_transparent(), TestCase::contract, CONTRACT_PHYSICS_TRANSPARENT, WordContract::flags, WordContract::fuzz_rounds, TestCase::implemented, TestCase::input, log_message(), LOG_TEST, TestCase::name, resolve_contract(), restore_dict_state(), run_single_test(), save_dict_state(), TestCase::should_error, WordTestSuite::suite_contract, WordTestSuite::test_count, TEST_ERROR, TEST_ERROR_CASE, TEST_FAIL, TEST_PASS, TEST_SKIP, test_stats_accumulate(), WordTestSuite::tests, TestCase::type, and WordTestSuite::word_name.

Referenced by run_arithmetic_words_tests(), run_inference_words_tests(), run_logical_words_tests(), run_memory_words_tests(), run_q48_words_tests(), run_return_stack_words_tests(), and run_stack_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.309.4.10 save_dict_state()

```
void save_dict_state (
            VM * vm,
            DictEntry ** latest,
            size_t * here )
```

Saves current dictionary state before test execution.

Parameters

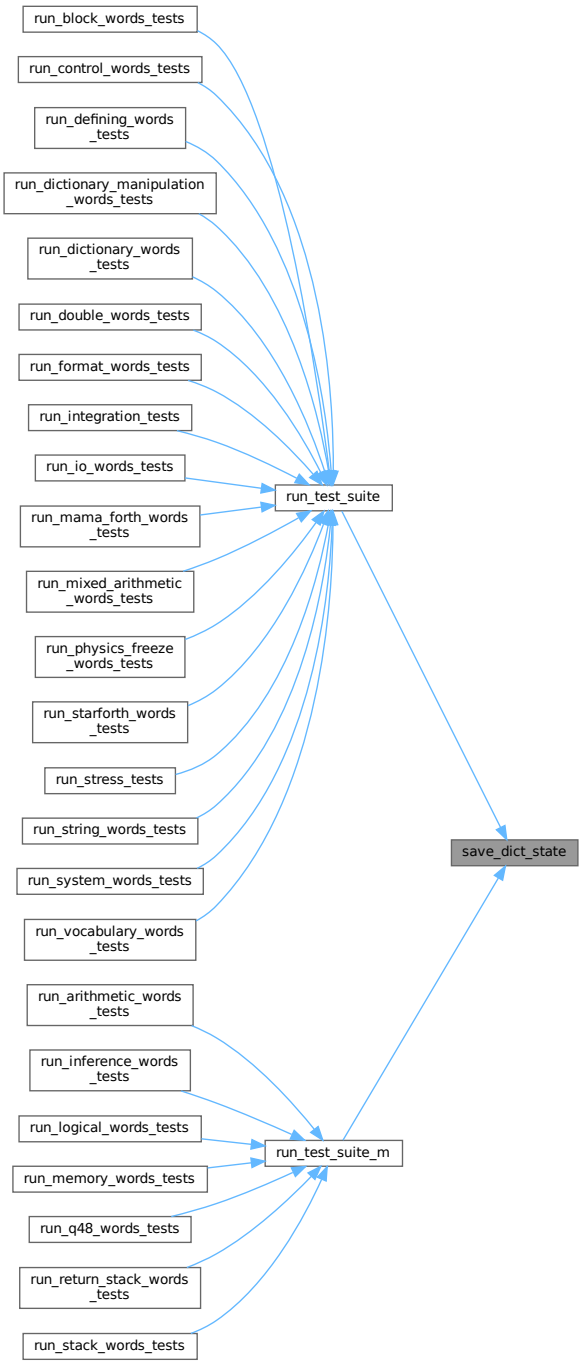
vm	Pointer to VM instance
latest	Pointer to store latest dictionary entry
here	Pointer to store HERE pointer

Captures dictionary state so tests can be cleaned up after execution

References VM::here, and VM::latest.

Referenced by run_test_suite(), and run_test_suite_m().

Here is the caller graph for this function:



7.309.4.11 save_vm_state()

```
void save_vm_state (
            VM * vm,
            int * dsp,
            int * rsp,
```

```
    int * error,
    vm_mode_t * mode )
```

Saves current VM state.

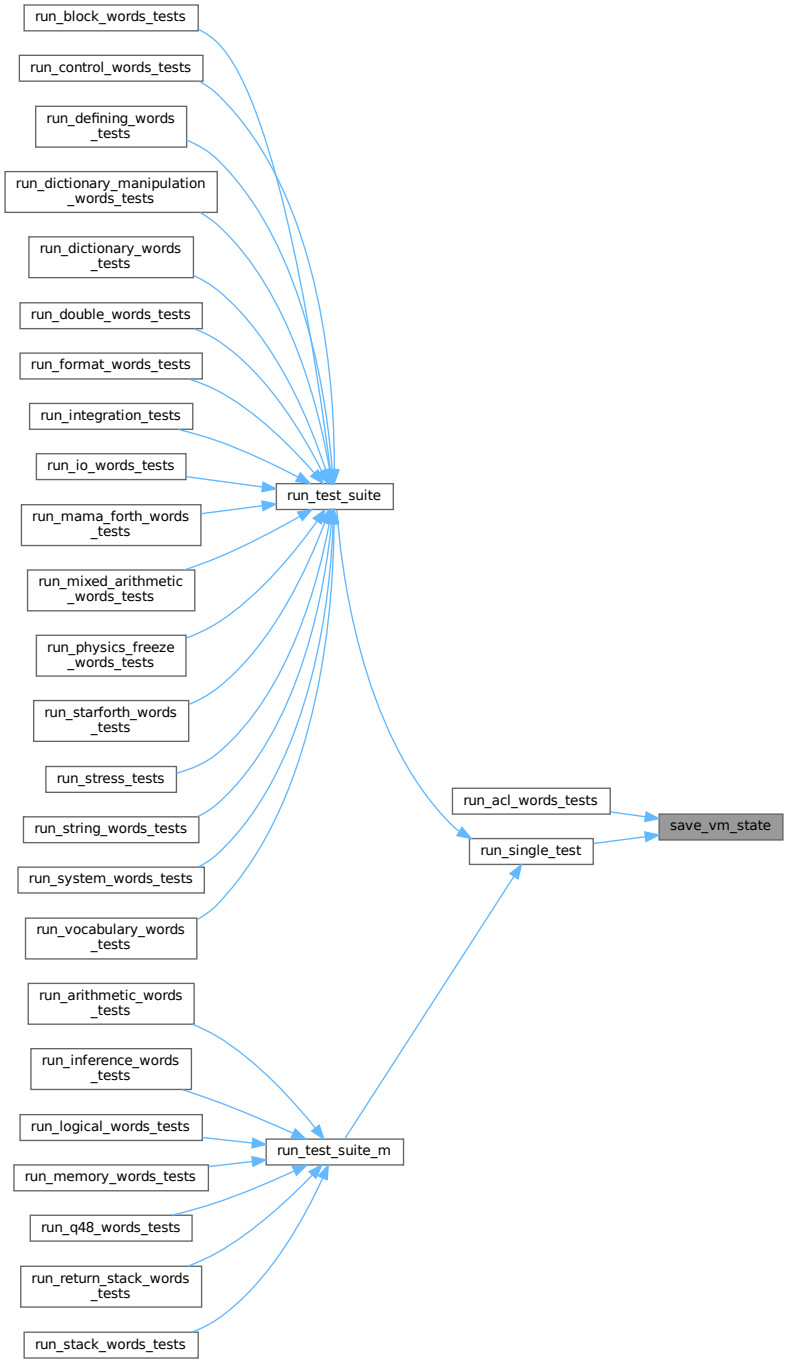
Parameters

vm	Pointer to VM instance
dsp	Pointer to store data stack pointer
rsp	Pointer to store return stack pointer
error	Pointer to store error state
mode	Pointer to store VM mode

References VM::dsp, VM::error, VM::mode, and VM::rsp.

Referenced by run_acl_words_tests(), and run_single_test().

Here is the caller graph for this function:



7.309.5 Variable Documentation

7.309.5.1 fail_fast

```
int fail_fast  [extern]
```

Global flag to control test execution behavior.

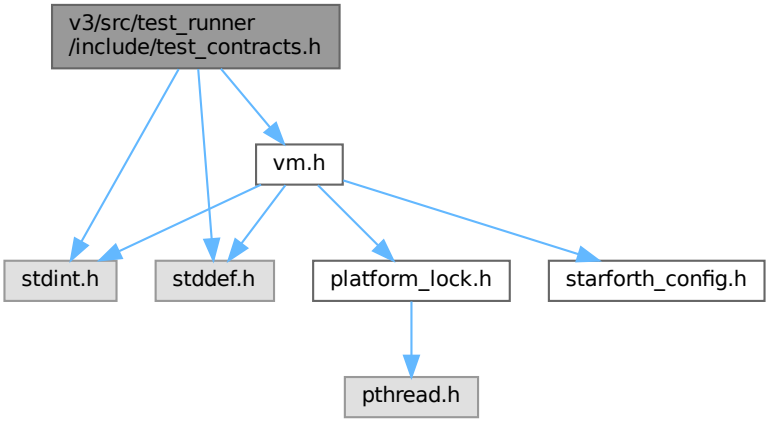
When set to non-zero, causes test runner to exit immediately on first failure

Referenced by main(), and run_single_test().

7.310 v3/src/test_runner/include/test_contracts.h File Reference

```
#include "vm.h"
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for test_contracts.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct WordContract
- struct ExecResult
- struct PhysicsSnapshot

Macros

- #define CONTRACT_NONE 0u
- #define CONTRACT_PHYSICS_TRANSPARENT (1u << 0)
- #define CONTRACT_HEARTBEAT_SAFE (1u << 1)
- #define CONTRACT_DEFAULT_FUZZ_ROUNDS 4

Typedefs

- typedef uint32_t contract_flags_t

Functions

- int contract_get_violation_count (void)

  Return the current count of accumulated contract violations.
- void contract_reset_violation_count (void)

  Reset the contract violation counter to zero.
- void capture_exec_result (VM *vm, ExecResult *r)

  Snapshot the execution-domain fields of a VM into an `ExecResult`.
- int exec_results_equal (const ExecResult *a, const ExecResult *b)

  Compare two `ExecResult` snapshots for execution-domain equality.
- void save_physics (VM *vm, PhysicsSnapshot *s)

  Save the key physics scalars touched by the seven feedback loops.
- void restore_physics (VM *vm, const PhysicsSnapshot *s)

  Restore the physics scalars from a previously saved snapshot.
- void perturb_physics (VM *vm, int round)

  Randomly perturb key physics scalars to expose A4' violations.
- int check_physics_transparent (VM *vm, const char *word_name, const char *input, int should_error, int implemented, int fuzz_rounds)

  A4' contract check — verify a word is transparent to physics state.
- int assert_heartbeat_safe (VM *vm)

  A1 contract check — verify `vm_tick()` does not modify the execution domain.

Variables

- int global_contract_violations

7.310.1 Macro Definition Documentation

7.310.1.1 CONTRACT_DEFAULT_FUZZ_ROUNDS

```
#define CONTRACT_DEFAULT_FUZZ_ROUNDS 4
```

7.310.1.2 CONTRACT_HEARTBEAT_SAFE

```
#define CONTRACT_HEARTBEAT_SAFE (1u << 1)
```

7.310.1.3 CONTRACT_NONE

```
#define CONTRACT_NONE 0u
```

7.310.1.4 CONTRACT_PHYSICS_TRANSPARENT

```
#define CONTRACT_PHYSICS_TRANSPARENT (1u << 0)
```

7.310.2 Typedef Documentation

7.310.2.1 contract_flags_t

```
typedef uint32_t contract_flags_t
```

7.310.3 Function Documentation

7.310.3.1 assert_heartbeat_safe()

```
int assert_heartbeat_safe (
            VM * vm )
```

A1 contract check — verify `vm_tick()` does not modify the execution domain.

Axiom A1 (heartbeat isolation): the background heartbeat tick must never alter stacks, error state, or any other execution-domain field observable by FORTH words. This function captures an `ExecResult` before and after one `vm_tick()` call and compares them with `exec_results_equal()`.

On violation, logs [A1 CONTRACT] VIOLATION with before/after dsp and rsp values, and increments `global_contract_violations`.

Parameters

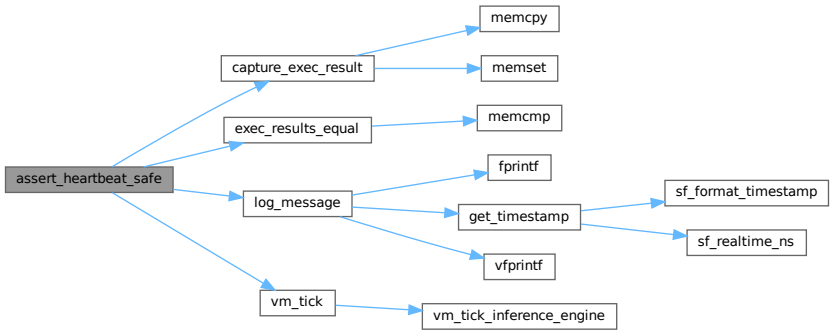
vm	VM instance; stacks and error state must already be in a known reference state before calling.

Returns

1 if the heartbeat left the execution domain unchanged, 0 on violation.

References capture_exec_result(), ExecResult::dsp, exec_results_equal(), global_contract_violations, LOG_ERROR, log_message(), ExecResult::rsp, and vm_tick().

Here is the call graph for this function:



7.310.3.2 capture_exec_result()

```
void capture_exec_result (
            VM * vm,
            ExecResult * r )
```

Snapshot the execution-domain fields of a VM into an `ExecResult`.

Captures `dsp`, `rsp`, `error`, and the live portion of the data stack (indices 0 ... dsp?1). The remainder of `r->stack` is zeroed for deterministic `memcmp()` comparisons in `exec_results_equal()`.

Only execution-domain fields are captured — physics scalars, heartbeat state, and rolling-window counters are deliberately excluded.

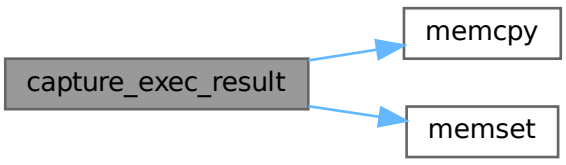
Parameters

vm	VM instance to snapshot.
r	Output `ExecResult` struct to populate.

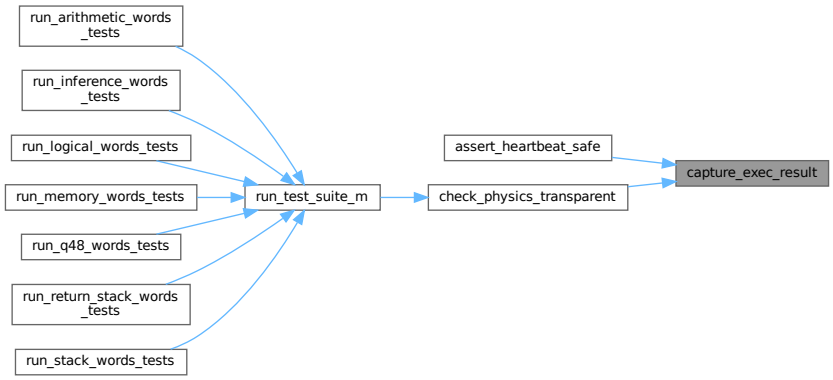
References VM::data_stack, VM::dsp, ExecResult::dsp, VM::error, ExecResult::error, memcpy(), memset(), VM::rsp, ExecResult::rsp, ExecResult::stack, and STACK_SIZE.

Referenced by assert_heartbeat_safe(), and check_physics_transparent().

Here is the call graph for this function:



Here is the caller graph for this function:



7.310.3.3 check_physics_transparent()

```
int check_physics_transparent (
            VM * vm,
            const char * word_name,
            const char * input,
            int should_error,
            int implemented,
            int fuzz_rounds )
```

A4' contract check — verify a word is transparent to physics state.

Axiom A4' (physics transparency): a FORTH word's execution-domain result (`ExecResult`) must be identical regardless of the values of the seven physics feedback-loop scalars.

Protocol:

1. Run `input` with current (unperturbed) physics; capture baseline.
2. For each of `fuzz_rounds` rounds: seed RNG, perturb physics, re-run `input`, capture result, compare to baseline.
3. On divergence, log the violation and increment `global_contract_violations`.
4. Always restore original physics state before returning.

Error-expected tests (`should_error`) and unimplemented words (`implemented` == 0) are skipped: exec-equivalent divergence is legitimate for error paths.

Parameters

vm	VM instance.
word_name	Human-readable word name for diagnostic messages.
input	FORTH source string to execute (same each round).
should_error	Non-zero if the test is expected to raise an error.
implemented	Non-zero if the word is implemented and should be checked.
fuzz_rounds	Number of perturbation rounds; 0 → CONTRACT_DEFAULT_FUZZ_ROUNDS.

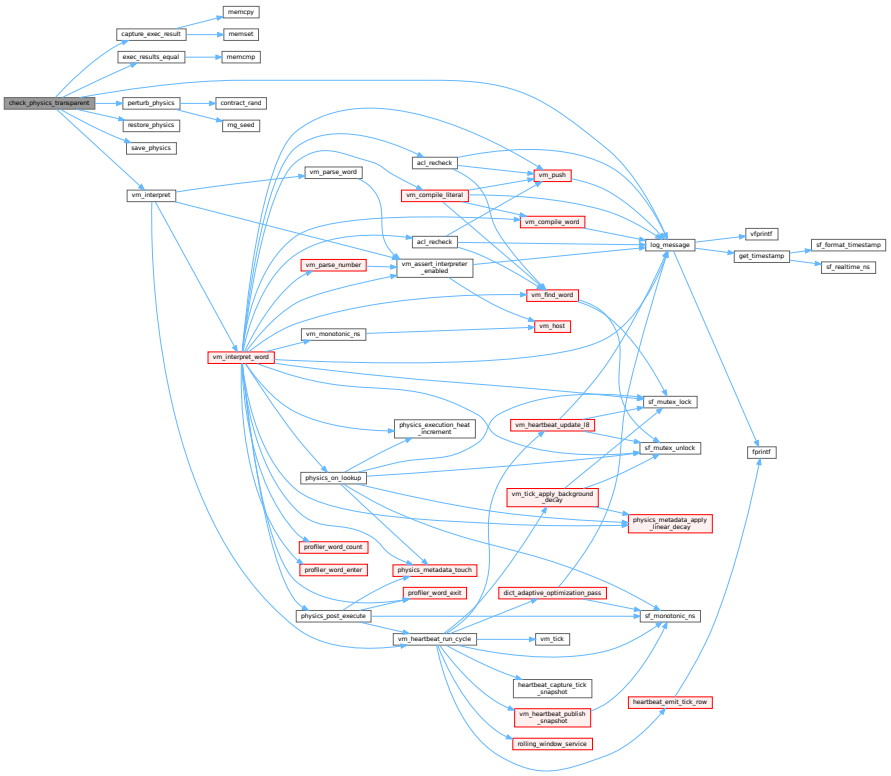
Returns

1 if all rounds passed (or test skipped), 0 on any violation.

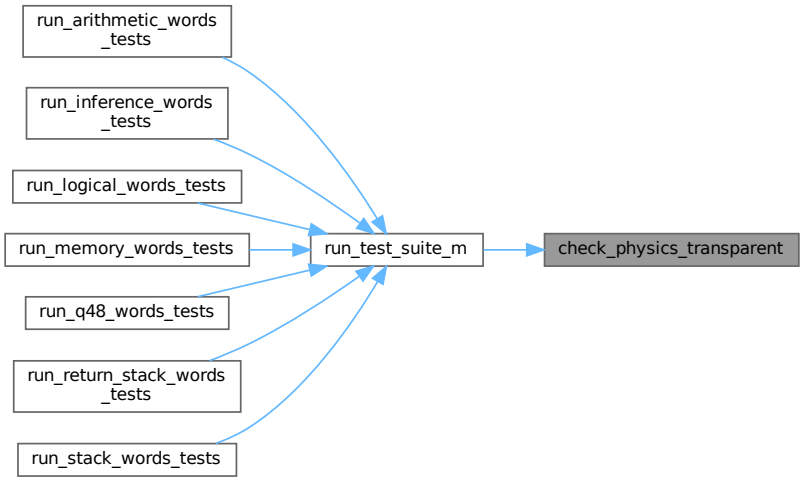
References VM::abort_requested, capture_exec_result(), CONTRACT_DEFAULT_FUZZ_ROUNDS, VM::dsp, ExecResult::dsp, VM::error, exec_results_equal(), VM::exit_colon, global_contract_violations, LOG_ERROR, log_message(), perturb_physics(), restore_physics(), VM::rsp, save_physics(), and vm_interpret().

Referenced by run_test_suite_m().

Here is the call graph for this function:



Here is the caller graph for this function:



7.310.3.4 contract_get_violation_count()

```
int contract_get_violation_count (
            void  )
```

Return the current count of accumulated contract violations.

Reads `global_contract_violations` directly; same-TU access avoids GOT double-dereference in `-fPIC` PE32+ builds where `R_X86_64_REX_GOTPCRELX` is not relaxed to a LEA instruction.

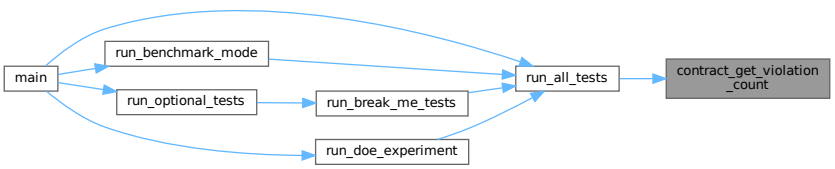
Returns

Number of contract violations recorded since last reset.

References global_contract_violations.

Referenced by run_all_tests().

Here is the caller graph for this function:



7.310.3.5 contract_reset_violation_count()

```
void contract_reset_violation_count (
            void  )
```

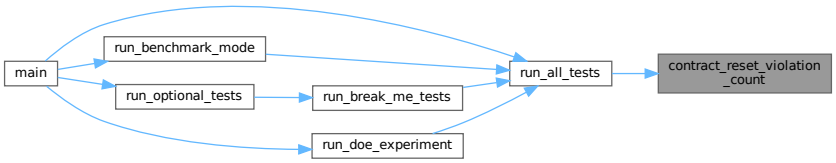
Reset the contract violation counter to zero.

Call before a fresh round of contract checks so that counts from earlier test phases do not accumulate across module boundaries.

References global_contract_violations.

Referenced by run_all_tests().

Here is the caller graph for this function:



7.310.3.6 exec_results_equal()

```
int exec_results_equal (
            const ExecResult * a,
            const ExecResult * b )
```

Compare two `ExecResult` snapshots for execution-domain equality.

Returns 1 only if `dsp`, `rsp`, `error`, and the live stack contents are all identical. Stack comparison uses `memcmp()` over the live portion (0 ... dsp?1 cells); the zeroed tail guarantees no false positives from uninitialised padding.

Parameters

a	First snapshot.
b	Second snapshot.

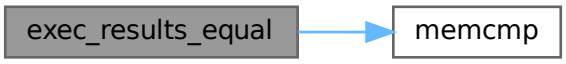
Returns

1 if execution-domain is identical, 0 if any field differs.

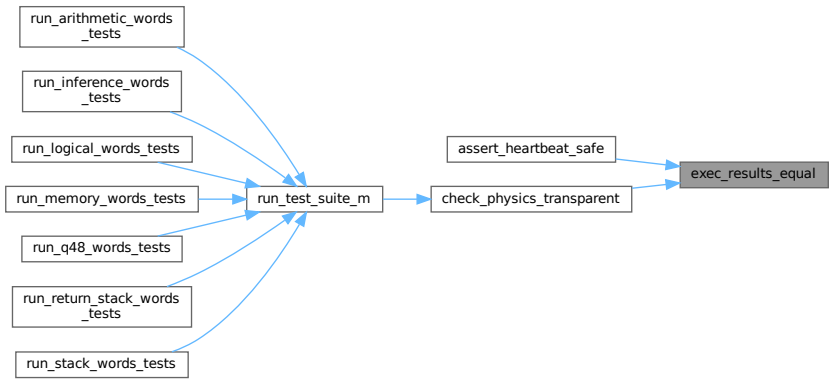
References ExecResult::dsp, ExecResult::error, memcmp(), ExecResult::rsp, and ExecResult::stack.

Referenced by assert_heartbeat_safe(), and check_physics_transparent().

Here is the call graph for this function:



Here is the caller graph for this function:



7.310.3.7 perturb_physics()

```
void perturb_physics (
            VM * vm,
            int round )
```

Randomly perturb key physics scalars to expose A4' violations.

Seeds the RNG via `rng_seed(round)` for deterministic but distinct perturbations per round, then overwrites the eight tracked physics fields with random values that still satisfy their well-formedness invariants:

- `decay_slope_q48`: LSB forced to 1 (must be $>$ 0 per `slope_wf`)
- `effective_window_size`: [1, ROLLING_WINDOW_SIZE] (never 0)
- `tick_target_ns`: LSB forced to 1 (must be $>$ 0 per `hb_state_wf`)
- Heat thresholds: arbitrary positive values (no strict invariant)
- `prefetch_attempts` / `total_executions`: arbitrary counts

After perturbation, running the same FORTH input must produce an identical `ExecResult`; divergence is an A4' violation.

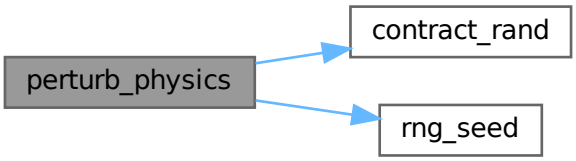
Parameters

vm	VM instance to perturb.
round	Fuzz round index (used to seed the RNG).

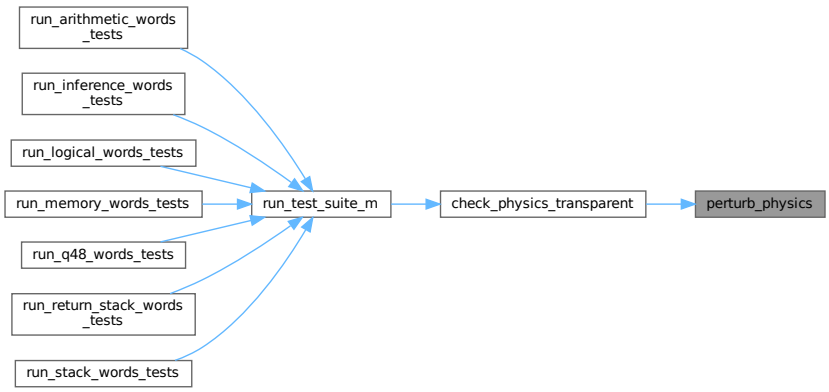
References contract_rand(), VM::decay_slope_q48, RollingWindowOfTruth::effective_window_size, VM::heartbeat, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch rng_seed(), VM::rolling_window, ROLLING_WINDOW_SIZE, HeartbeatState::tick_target_ns, and RollingWindowOfTruth::total_execu

Referenced by check_physics_transparent().

Here is the call graph for this function:



Here is the caller graph for this function:



7.310.3.8 restore_physics()

```
void restore_physics (
            VM * vm,
            const PhysicsSnapshot * s )
```

Restore the physics scalars from a previously saved snapshot.

Writes back the eight fields saved by save_physics(), undoing any perturbations made by perturb_physics(). Called at the end of each fuzz round and after the full check_physics_transparent() loop so the VM state is clean for the next contract check.

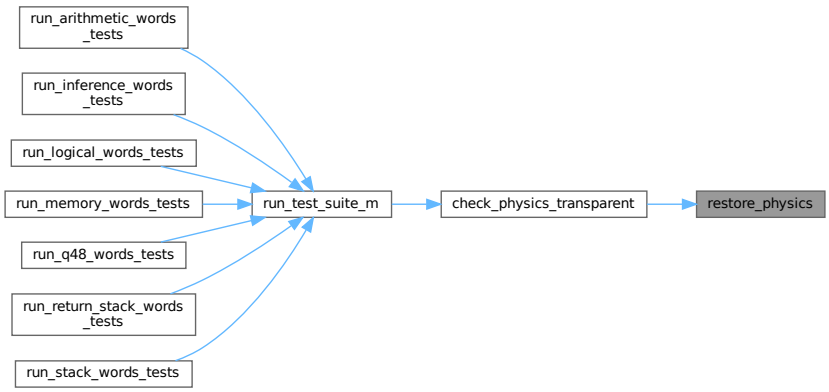
Parameters

vm	VM instance to restore.
s	Source `PhysicsSnapshot` to restore from.

References VM::decay_slope_q48, PhysicsSnapshot::decay_slope_q48, PhysicsSnapshot::eff_window_size, RollingWindowOfTruth::effective_window_size, VM::heartbeat, PhysicsSnapshot::heat_25th, PhysicsSnapshot::heat_50th, PhysicsSnapshot::heat_75th, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PhysicsSnapshot::prefetch_attempts, VM::rolling_window, HeartbeatState::tick_target_ns, PhysicsSnapshot::tick_target_ns, RollingWindowOfTruth::total_executions, and PhysicsSnapshot::total_executions.

Referenced by check_physics_transparent().

Here is the caller graph for this function:



7.310.3.9 save_physics()

```
void save_physics (
            VM * vm,
            PhysicsSnapshot * s )
```

Save the key physics scalars touched by the seven feedback loops.

Captures the eight fields that `perturb_physics()` will randomise: `decay_slope_q48`, `effective_window_size`, `tick_target_ns`, the three heat percentile thresholds, `prefetch_attempts`, and `total_executions`. Used as a bracket pair with `restore_physics()` around each fuzz round so the VM returns to a known state.

Parameters

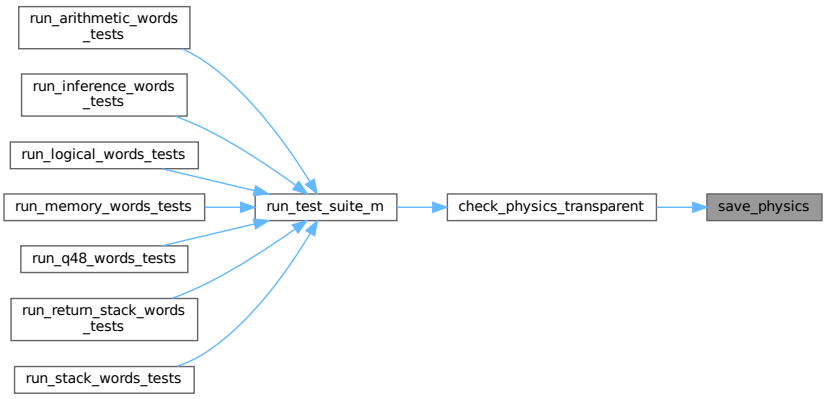
vm	VM instance to snapshot.
s	Output `PhysicsSnapshot` to populate.

References VM::decay_slope_q48, PhysicsSnapshot::decay_slope_q48, PhysicsSnapshot::eff_window_size,

RollingWindowOfTruth::effective_window_size, VM::heartbeat, PhysicsSnapshot::heat_25th, PhysicsSnapshot::heat_50th, PhysicsSnapshot::heat_75th, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PhysicsSnapshot::prefetch_attempts, VM::rolling_window, HeartbeatState::tick_target_ns, PhysicsSnapshot::tick_target_ns, RollingWindowOfTruth::total_executions, and PhysicsSnapshot::total_executions.

Referenced by check_physics_transparent().

Here is the caller graph for this function:



7.310.4 Variable Documentation

7.310.4.1 global_contract_violations

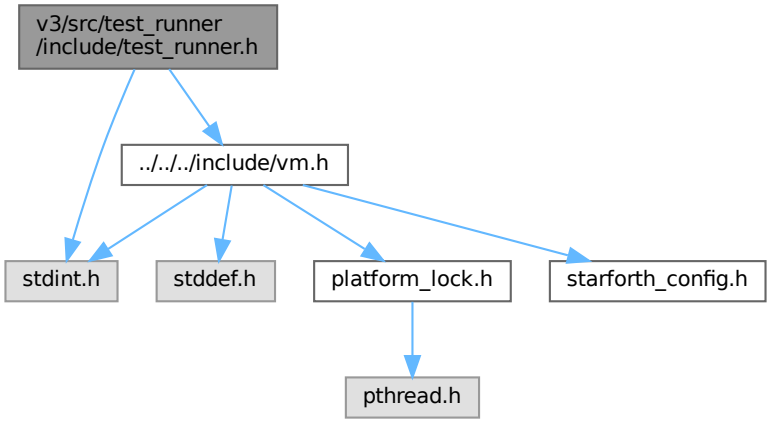
```
int global_contract_violations  [extern]
```

Referenced by assert_heartbeat_safe(), check_physics_transparent(), contract_get_violation_count(), and contract_reset_violation_count().

7.311 v3/src/test_runner/include/test_runner.h File Reference

```
#include "../../../include/vm.h"
#include <stdint.h>
```

Include dependency graph for test_runner.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct TestStats

  Structure holding test execution statistics.
- struct TestModule

  Structure representing a test module.

Functions

- void test_stats_accumulate (int tests, int pass, int fail, int skip, int error)
- void run_stack_words_tests (VM *vm)

  Executes all test suites for stack manipulation words.
- void run_return_stack_words_tests (VM *vm)

  Executes all test suites for return stack manipulation words.
- void run_memory_words_tests (VM *vm)

  Executes the test suite for memory manipulation words.
- void run_arithmetic_words_tests (VM *vm)

  Executes all arithmetic word test suites.
- void run_logical_words_tests (VM *vm)

  Executes all logical words test suites.
- void run_mixed_arithmetic_words_tests (VM *vm)

  Module 7 entry point — run all mixed-arithmetic word POST tests.
- void run_double_words_tests (VM *vm)

  Executes all test suites for double number words.
- void run_dictionary_words_tests (VM *vm)

Executes all dictionary word test suites.

- void run_io_words_tests (VM *vm)

  Executes all I/O words test suites.
- void run_format_words_tests (VM *vm)

  Executes all format words test suites.
- void run_string_words_tests (VM *vm)

  Executes all string manipulation word tests.
- void run_dictionary_manipulation_words_tests (VM *vm)

  Executes all dictionary manipulation words test suites.
- void run_vocabulary_words_tests (VM *vm)

  Executes all vocabulary word test suites.
- void run_block_words_tests (VM *vm)

  Executes all block words test suites.
- void run_system_words_tests (VM *vm)

  Executes all system word tests.
- void run_defining_words_tests (VM *vm)

  Executes all test suites for Forth defining words.
- void run_control_words_tests (VM *vm)

  Executes all control words test suites.
- void run_starforth_words_tests (VM *vm)

  Executes all StarForth words test suites.
- void run_acl_words_tests (VM *vm)

  Module 24 entry point — run all word-level ACL POST tests.
- void run_q48_words_tests (VM *vm)

  Executes all Q48.16 fixed-point word test suites.
- void run_inference_words_tests (VM *vm)

  Executes the proof-covered subset of inference-engine word test suites.
- void run_physics_freeze_words_tests (VM *vm)

  Executes the proof-covered subset of physics-freeze word test suites.
- void run_mama_forth_words_tests (VM *vm)

  Executes all Mama FORTH vocabulary test suites.
- void enable_benchmark_mode (int iterations)

  Enable benchmark mode for test execution.
- void run_stress_tests (VM *vm)

  Execute comprehensive stress test suites.
- void run_integration_tests (VM *vm)

  Executes all integration test suites for StarForth.
- void run_all_tests (VM *vm)

  Run all test modules with optional benchmarking.
- void run_module_tests (VM *vm, const char *module_name)

  Run tests for a specific module.
- void run_word_tests (VM *vm, const char *word_name)

  Run tests for a specific Forth word.
- void run_break_me_tests (VM *vm)

  Run ultra-comprehensive break-me diagnostic suite.

Variables

- TestStats global_test_stats

  Global statistics for test execution.

7.311.1 Function Documentation

7.311.1.1 enable_benchmark_mode()

```
void enable_benchmark_mode (
            int iterations )
```

Enable benchmark mode for test execution.

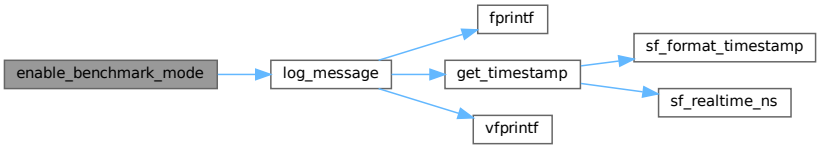
Parameters

iterations	Number of times each test should be repeated

References benchmark_iterations, benchmark_mode, LOG_INFO, and log_message().

Referenced by run_benchmark_mode().

Here is the call graph for this function:



Here is the caller graph for this function:

7.311.1.2 run_acl_words_tests()

```
void run_acl_words_tests (
            VM * vm )
```

Module 24 entry point — run all word-level ACL POST tests.

Saves the current VM state (dsp, rsp, error, mode, emergency_console) before each sub-test and restores it between tests so failures do not cascade. Runs eleven tests covering:

1. ACL-PIN one-way ratchet (4 assertions)

2. ACL-INHERIT semantics (4 assertions)
3. emergency_console bypass (1 assertion)
4. ACL denial aborts word execution (2 assertions)
5. TTL hot-path decrement (1 assertion)
6. STRICT mode — TTL stays 0 (1 assertion or conditional skip)
7. Adaptive TTL via ACL-TTL-COMPUTE (1 assertion or conditional skip)
8. Pin blocks shadowing (3 assertions)
9. Read accessors — ACL-MODE@/PINNED?/TTL@/ALLOW@/HEAT@/WORD-ID (6 assertions)
10. ACL-INHERIT as an interpreted word, not just its C twin (2 assertions)
11. ACL-INIT-PRIMITIVES resets unpinned, skips pinned (4 assertions)

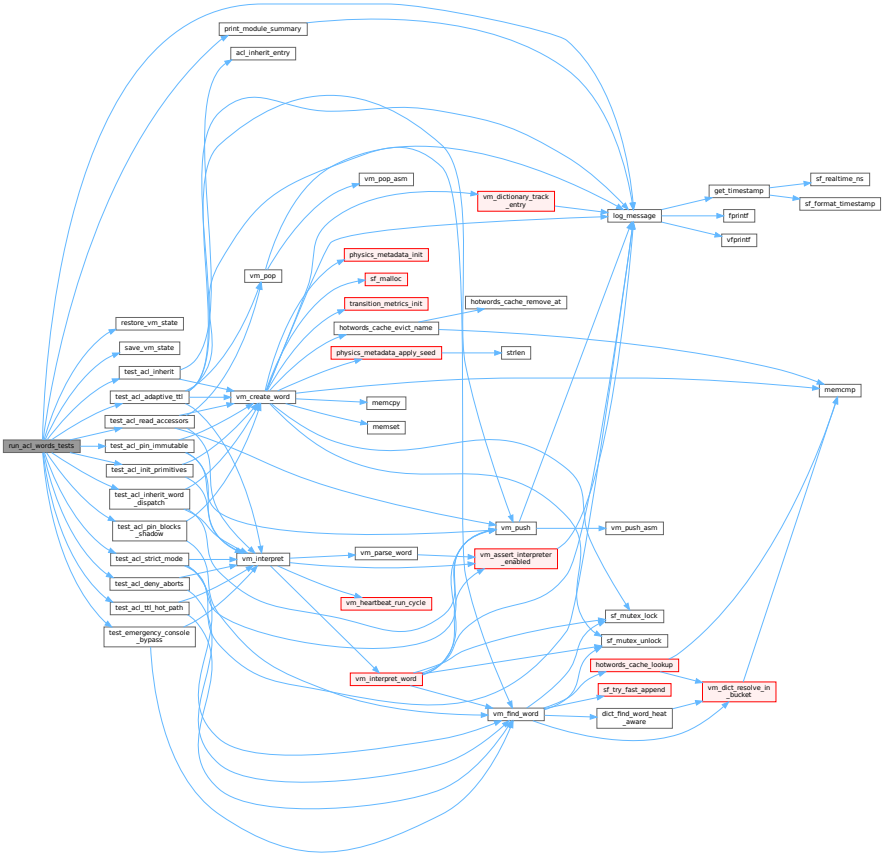
Tests that depend on `ACL.4th` being loaded are conditionally skipped during bare POST (before `ACL.4th` runs) and counted as passes so the POST score is not penalised by load-order.

Parameters

vm	VM instance; must be initialised and have the standard word vocabulary available.

References VM::emergency_console, LOG_INFO, log_message(), print_module_summary(), restore_vm_state(), save_vm_state(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_inherit(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_blocks_shadow(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_bypass(), tests_failed, and tests_passed.

Here is the call graph for this function:



7.311.1.3 run_all_tests()

```
void run_all_tests (
            VM * vm )
```

Run all test modules with optional benchmarking.

Executes all registered test modules sequentially, collecting statistics and/or benchmark data depending on the current mode.

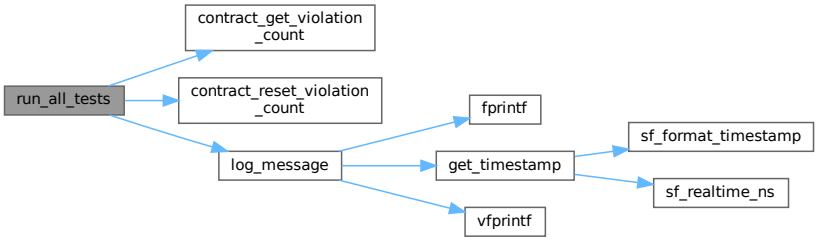
Parameters

vm	Pointer to the Forth virtual machine instance

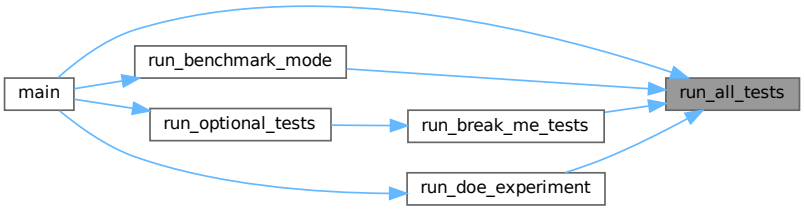
References contract_get_violation_count(), contract_reset_violation_count(), global_test_stats, LOG_ERROR, LOG_INFO, log_message(), LOG_WARN, TestModule::module_name, test_modules, TestStats::total_error, TestStats::total_fail, TestStats::total_pass, TestStats::total_skip, and TestStats::total_tests.

Referenced by main(), run_benchmark_mode(), run_break_me_tests(), and run_doe_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:



7.311.1.4 run_arithmetic_words_tests()

```
void run_arithmetic_words_tests (
            VM * vm )
```

Executes all arithmetic word test suites.

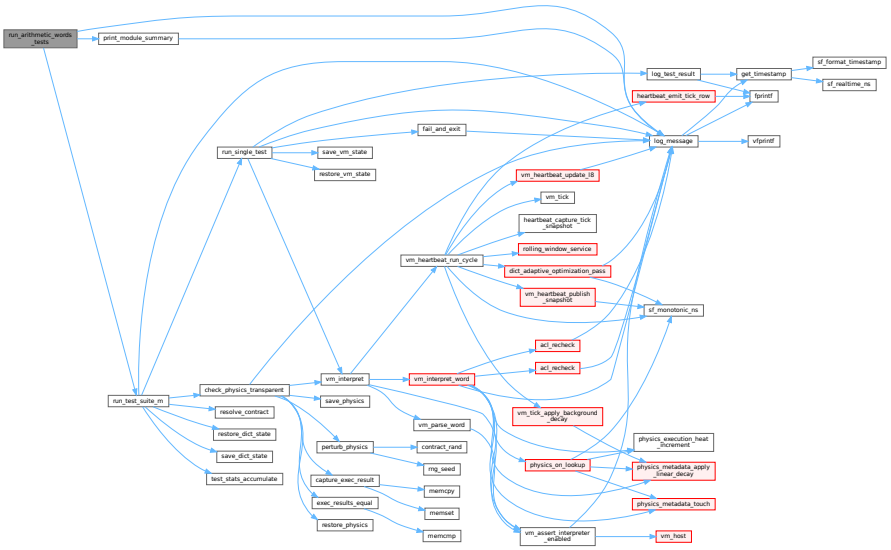
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all test cases for arithmetic operations including basic operations (+, -, ∗, /), modulo operations (MOD, /MOD), and numeric functions (ABS, NEGATE, MIN, MAX)

References arithmetic_word_suites, CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.5 run_block_words_tests()

```
void run_block_words_tests (
            VM * vm )
```

Executes all block words test suites.

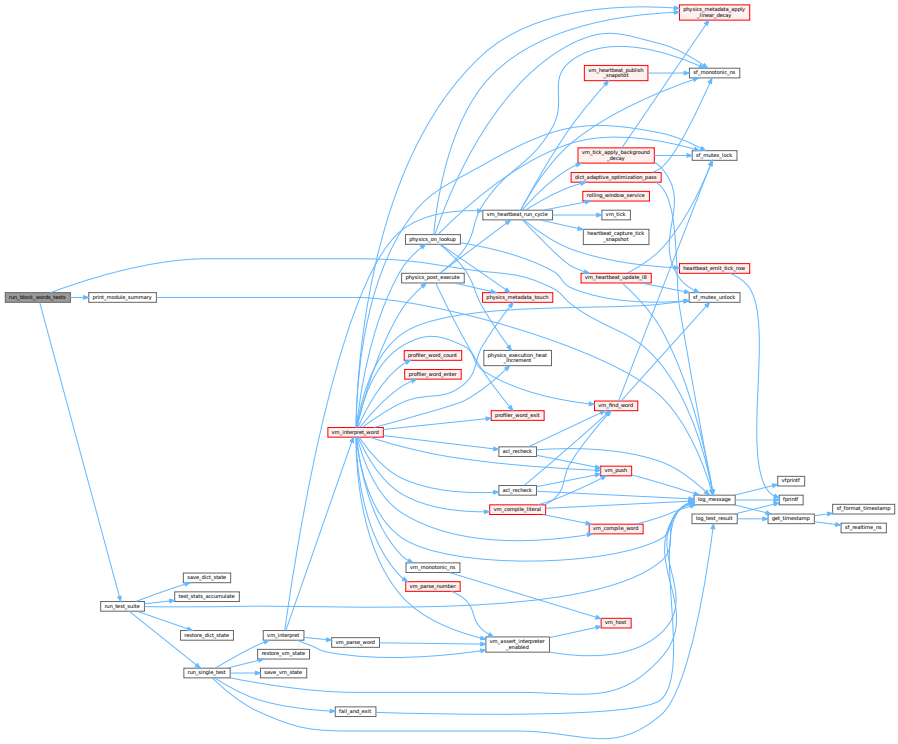
Parameters

vm	Pointer to the FORTH virtual machine instance

Runs through all test cases for block manipulation words, including block access, buffer management, and block I/O operations. Results are logged and summarized after completion.

References block_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.6 run_break_me_tests()

```
void run_break_me_tests (
            VM * vm )
```

Run ultra-comprehensive break-me diagnostic suite.

Parameters

vm	Pointer to the VM instance

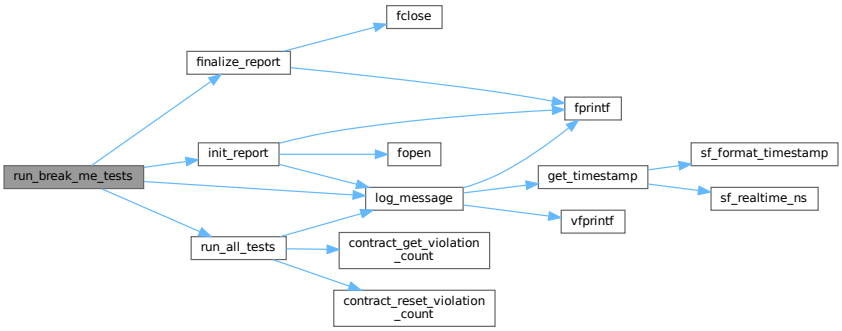
Executes the most exhaustive test battery ever designed for StarForth, generates a detailed markdown report in docs/BREAK_ME_REPORT.md, and includes a surprise easter egg at the end.

Run ultra-comprehensive break-me diagnostic suite.

References finalize_report(), global_test_stats, init_report(), LOG_ERROR, LOG_INFO, log_message(), run_all_tests(), TestStats::total_tests, and total_tests_run.

Referenced by run_optional_tests().

Here is the call graph for this function:



Here is the caller graph for this function:

7.311.1.7 run_control_words_tests()

```
void run_control_words_tests (
            VM * vm )
```

Executes all control words test suites.

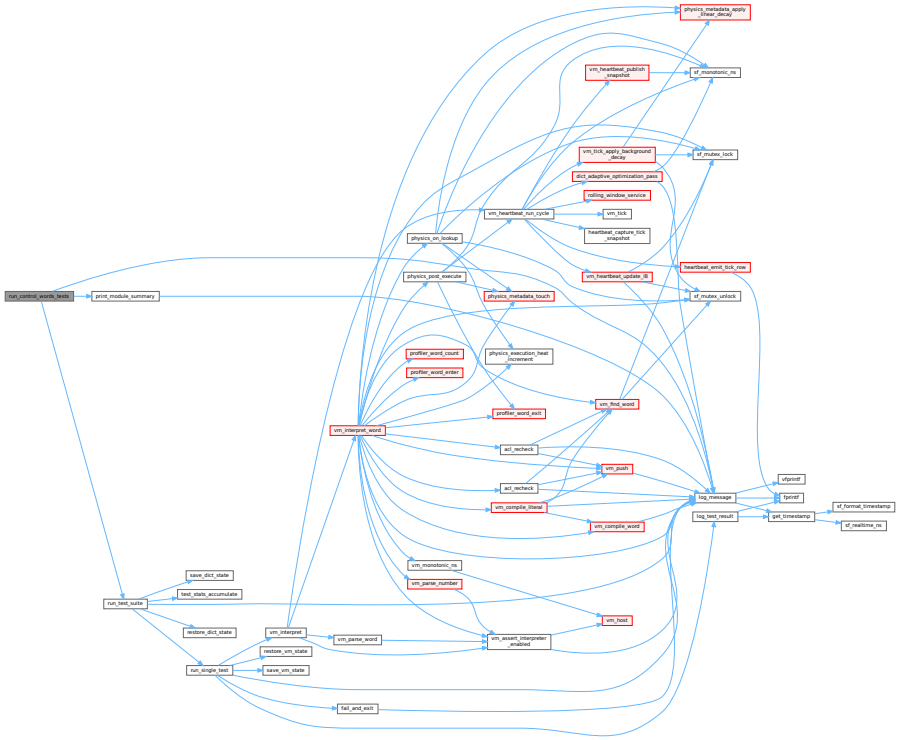
Parameters

vm	Pointer to the FORTH virtual machine instance

Runs through all defined test cases for control words, testing conditional branches and loop constructs

References control_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.8 run_defining_words_tests()

```
void run_defining_words_tests (
            VM * vm )
```

Executes all test suites for Forth defining words.

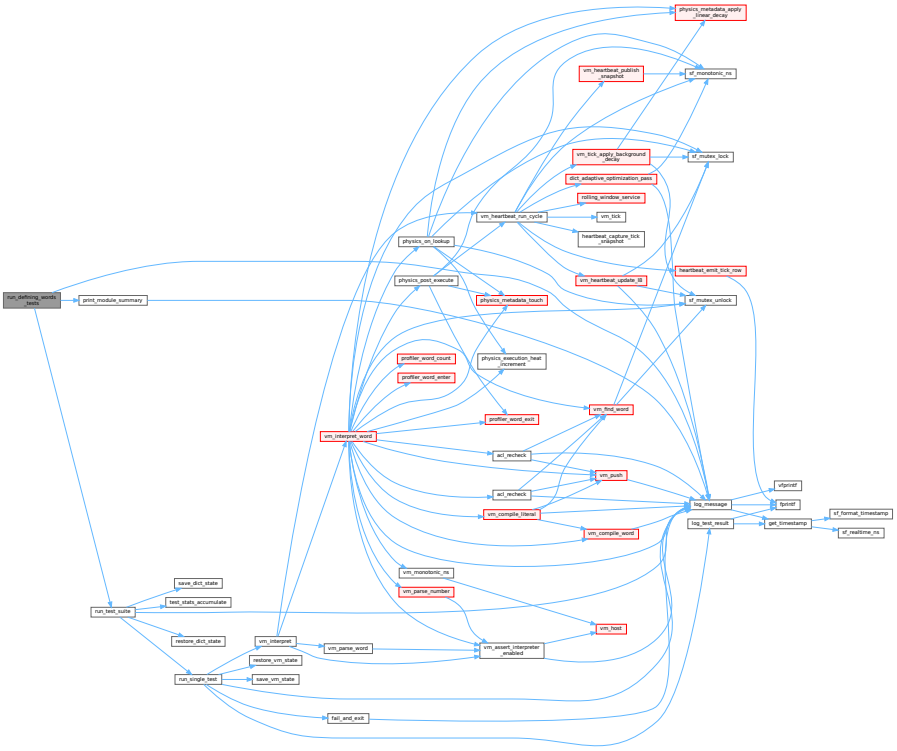
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through each test suite in defining_word_suites array, executing individual test cases and logging results_↩ run_01_2025_12_08

References defining_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.9 run_dictionary_manipulation_words_tests()

```
void run_dictionary_manipulation_words_tests (
            VM * vm )
```

Executes all dictionary manipulation words test suites.

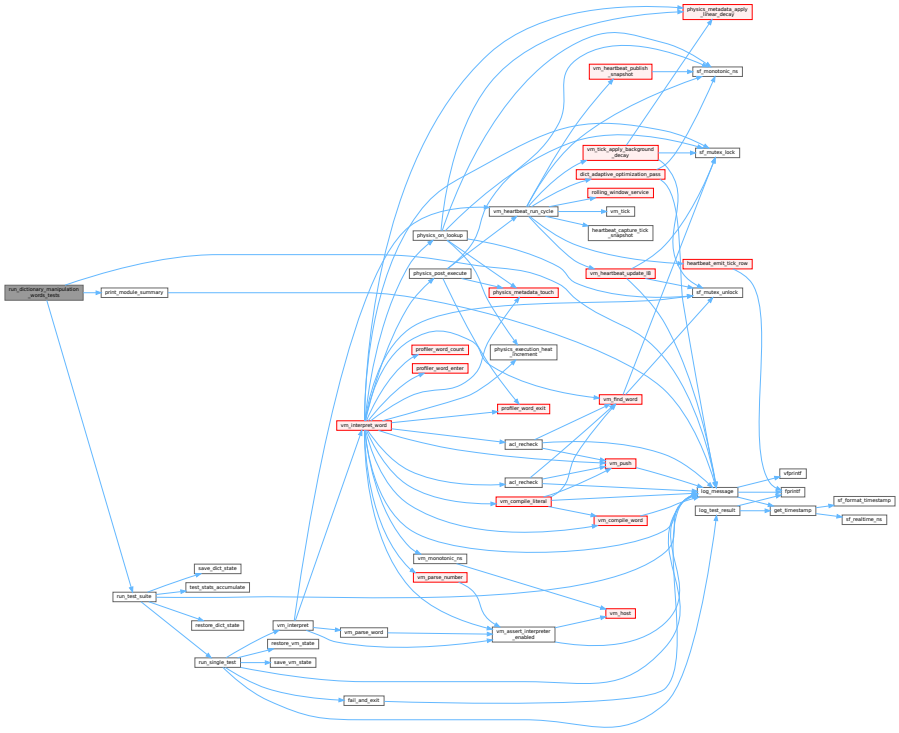
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all test cases for CREATE, FORGET, IMMEDIATE, FIND, DEFINITIONS, SMUDGE, LATEST, >BODY, and HIDDEN words

References dict_manip_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.10 run_dictionary_words_tests()

```
void run_dictionary_words_tests (
            VM * vm )
```

Executes all dictionary word test suites.

Parameters

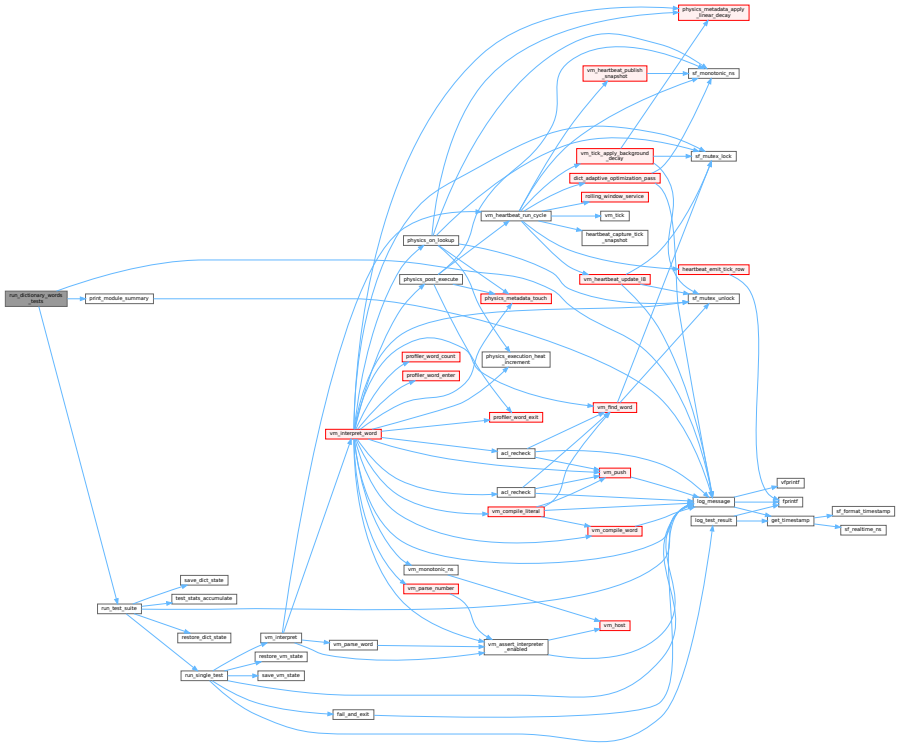
vm	Pointer to the FORTH virtual machine instance

This function runs through all dictionary word test suites, including tests for:

- HERE (dictionary pointer)
- ALLOT (memory allocation)
- Comma operations (, C, 2,)
- PAD (scratch pad area)
- Stack pointer operations (SP@ SP!)
- LATEST (most recent dictionary entry)
- Compilation sequences
- Stack management operations

References dictionary_word_suites, LOG_INFO, log_message(), print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.11 run_double_words_tests()

```
void run_double_words_tests (
            VM * vm )
```

Executes all test suites for double number words.

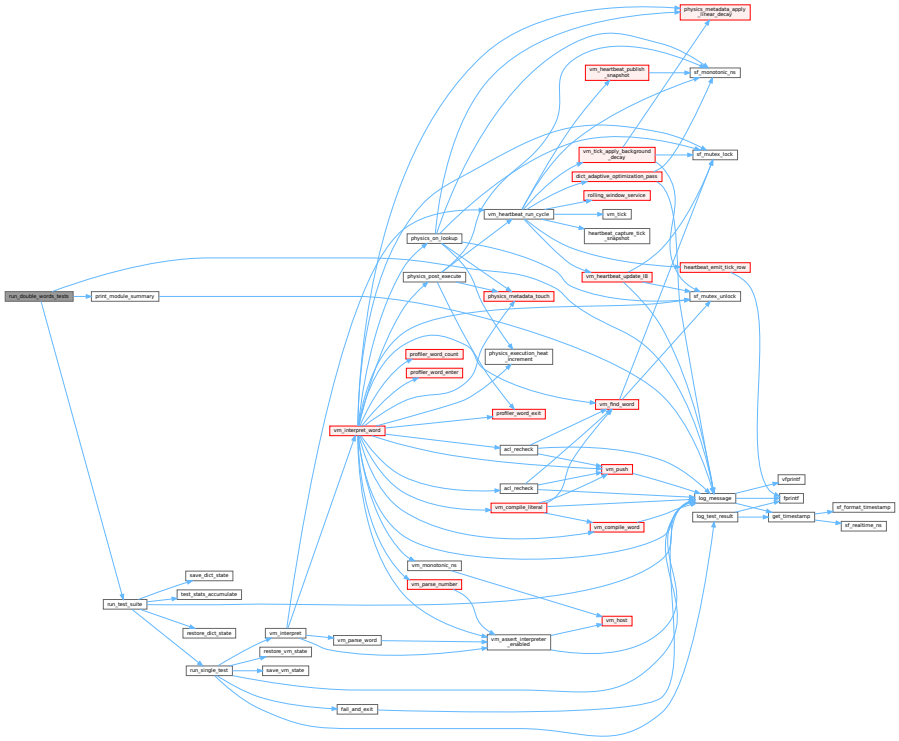
This function runs comprehensive tests for all double number word operations, including stack manipulation (2DROP, 2DUP, etc.), arithmetic operations (D+, D-, etc.), and comparison operations (D=, D<, etc.).

Parameters

vm	Pointer to the virtual machine instance where tests will be executed

References double_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.12 run_format_words_tests()

```
void run_format_words_tests (
            VM * vm )
```

Executes all format words test suites.

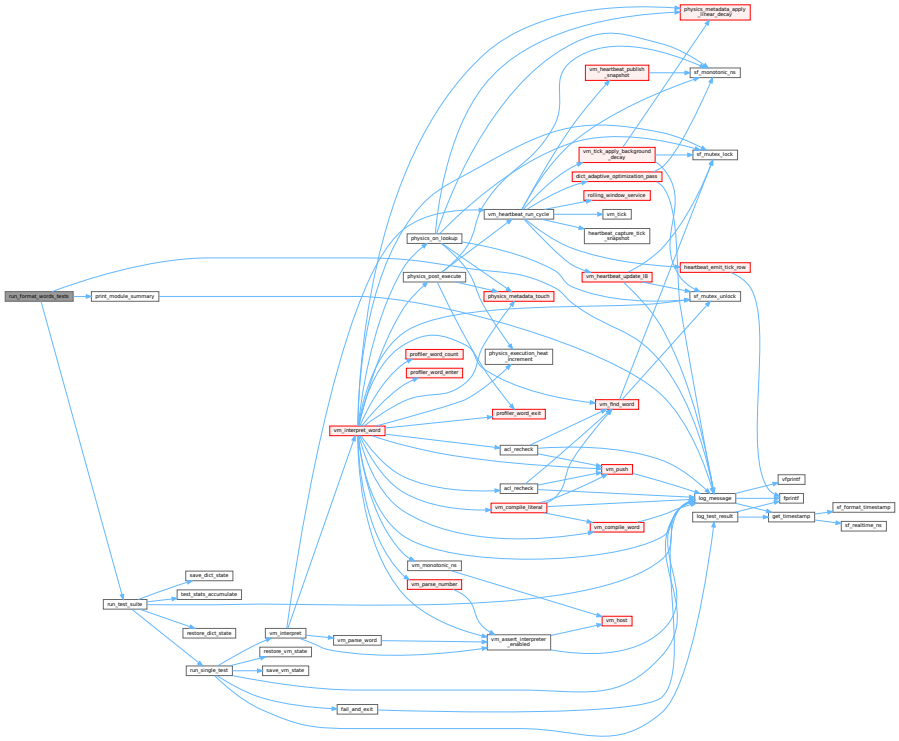
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all test cases for number base manipulation and formatting words

References format_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.13 run_inference_words_tests()

```
void run_inference_words_tests (
            VM * vm )
```

Executes the proof-covered subset of inference-engine word test suites.

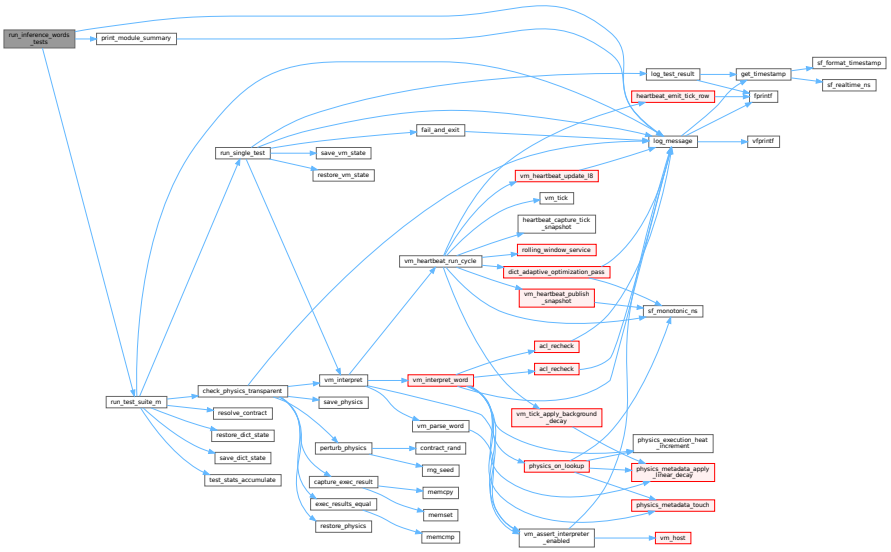
Parameters

vm	Pointer to the Forth virtual machine instance

Covers the 8 words proof/COVERAGE.md marks as proof-covered in inference_words.c: the three array-based primitives (Q.VARIANCE, INFER-DECAY-SLOPE, INFER-WINDOW-WIDTH), INFER-RUN, and the four ∗-@ output accessors (INFER-WINDOW@/DECAY@/VARIANCE@/FIT@) plus INFER-EARLY-EXIT. Does not cover the L8 Jacquard or Bayesian posterior words in the same file — not claimed as proof-covered.

References CONTRACT_NONE, inference_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.14 run_integration_tests()

```
void run_integration_tests (
            VM * vm )
```

Executes all integration test suites for StarForth.

Parameters

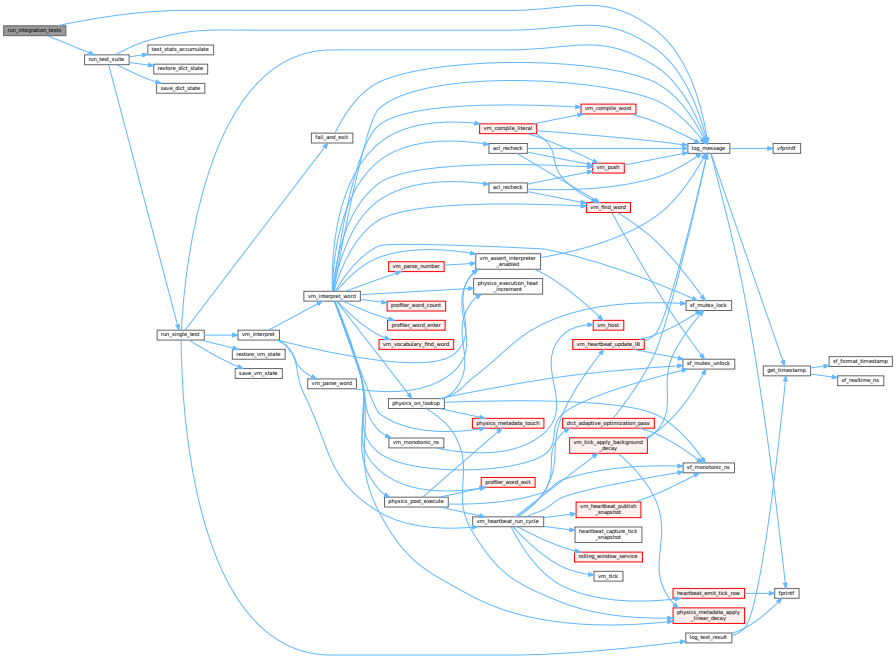
vm	Pointer to the Forth virtual machine instance

This function runs through all defined integration test suites, including:

- Complete Programs suite (prime checker, array operations, etc)
- Data Structures suite (linked list, min/max tracker)
- Algorithms suite (sorting, GCD)

References integration_suites, LOG_INFO, log_message(), and run_test_suite().

Here is the call graph for this function:



7.311.1.15 run_io_words_tests()

```
void run_io_words_tests (
            VM * vm )
```

Executes all I/O words test suites.

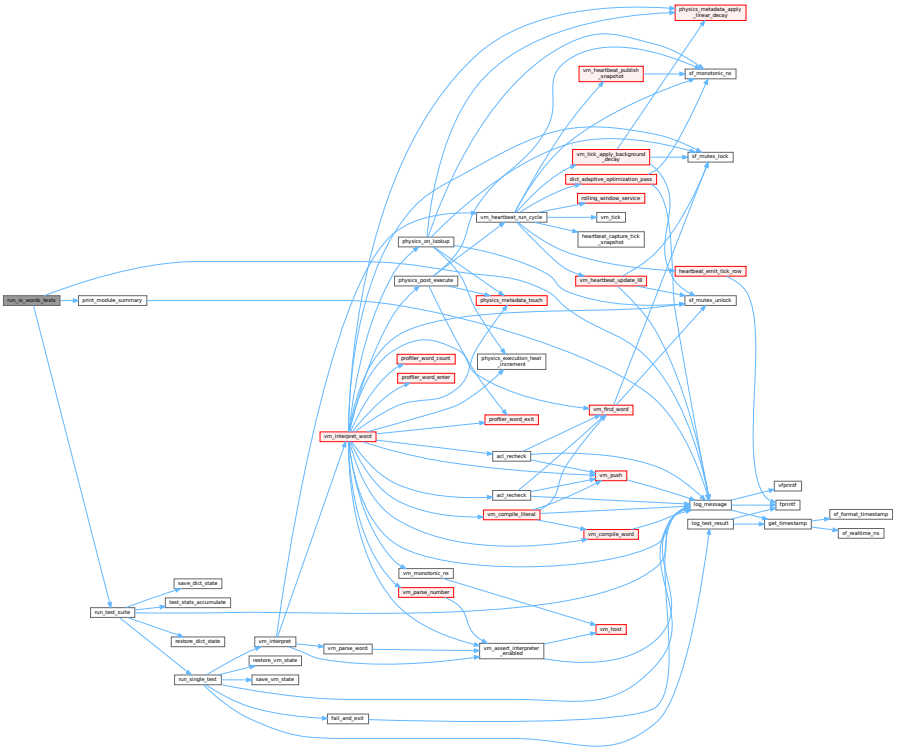
Parameters

vm	Pointer to the FORTH virtual machine instance

Runs through all defined test suites for I/O words, logging results_run_01_2025_12_08 and providing a summary of the test execution

References io_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.16 run_logical_words_tests()

```
void run_logical_words_tests (
            VM * vm )
```

Executes all logical words test suites.

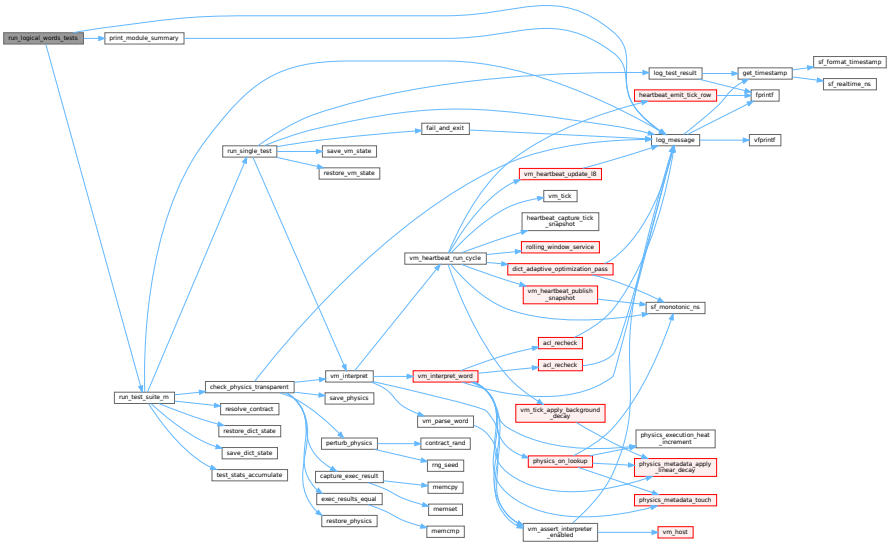
This function runs comprehensive tests for all FORTH logical and comparison operators, including stack underflow checks and edge cases.

Parameters

vm	Pointer to the FORTH virtual machine instance

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, logical_word_suites, print_module_summary(), run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.17 run_mama_forth_words_tests()

```
void run_mama_forth_words_tests (
            VM * vm )
```

Executes all Mama FORTH vocabulary test suites.

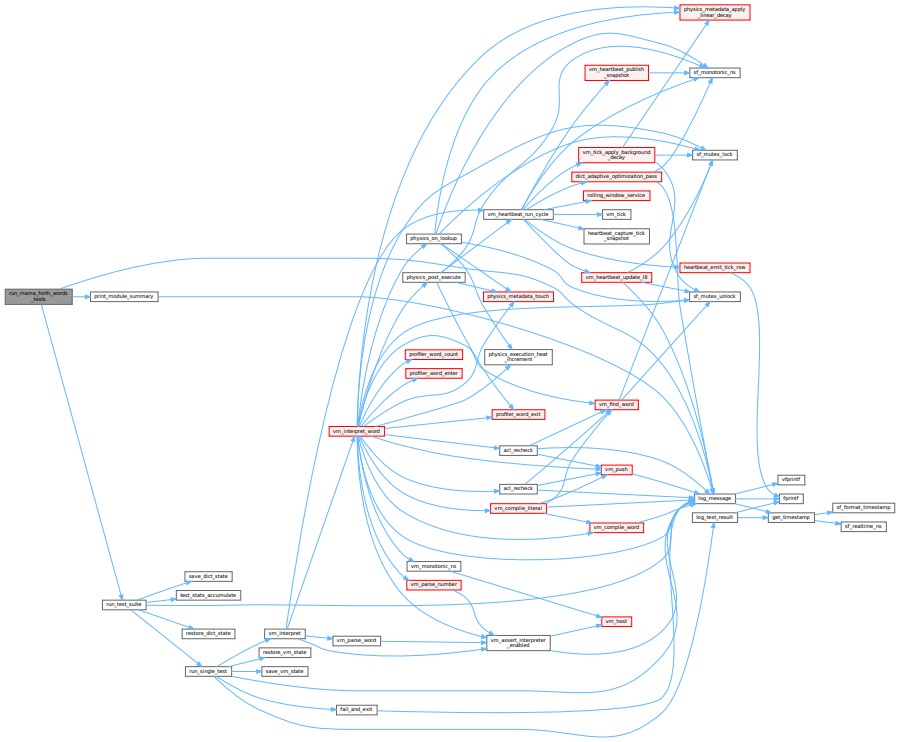
In hosted builds, all tests will be skipped since MAMA_IMPLEMENTED=0. In kernel builds, tests will execute against the capsule system.

Parameters

vm	Pointer to the Forth virtual machine instance

References LOG_INFO, log_message(), mama_word_suites, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.18 run_memory_words_tests()

```
void run_memory_words_tests (
            VM * vm )
```

Executes the test suite for memory manipulation words.

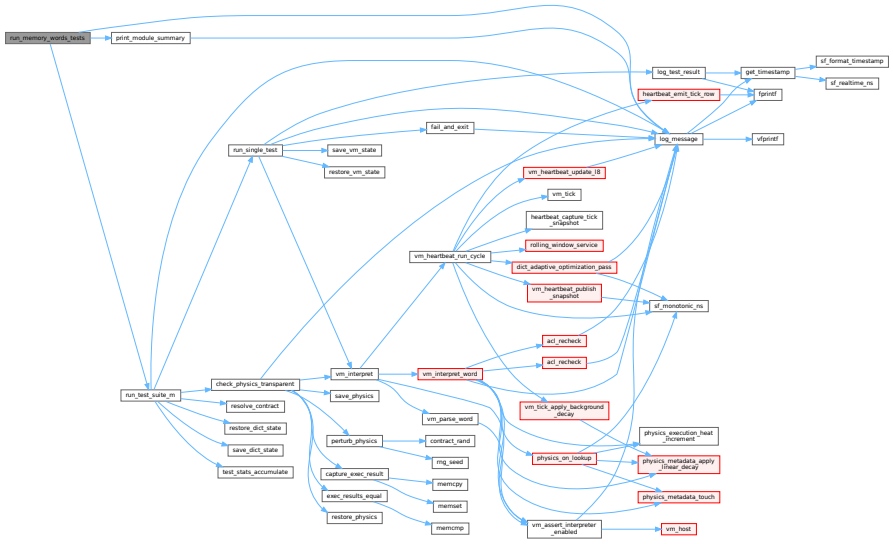
Runs through a series of test cases for Forth memory operations including store, fetch, allot, and other memory-related words

Parameters

vm	Pointer to the Forth virtual machine instance

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, memory_word_suites, print_module_summary(), run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.19 run_mixed_arithmetic_words_tests()

```
void run_mixed_arithmetic_words_tests (
            VM * vm )
```

Module 7 entry point — run all mixed-arithmetic word POST tests.

Iterates `mixed_arithmetic_word_suites` and delegates each to `run_test_suite()`. Covers five word families:

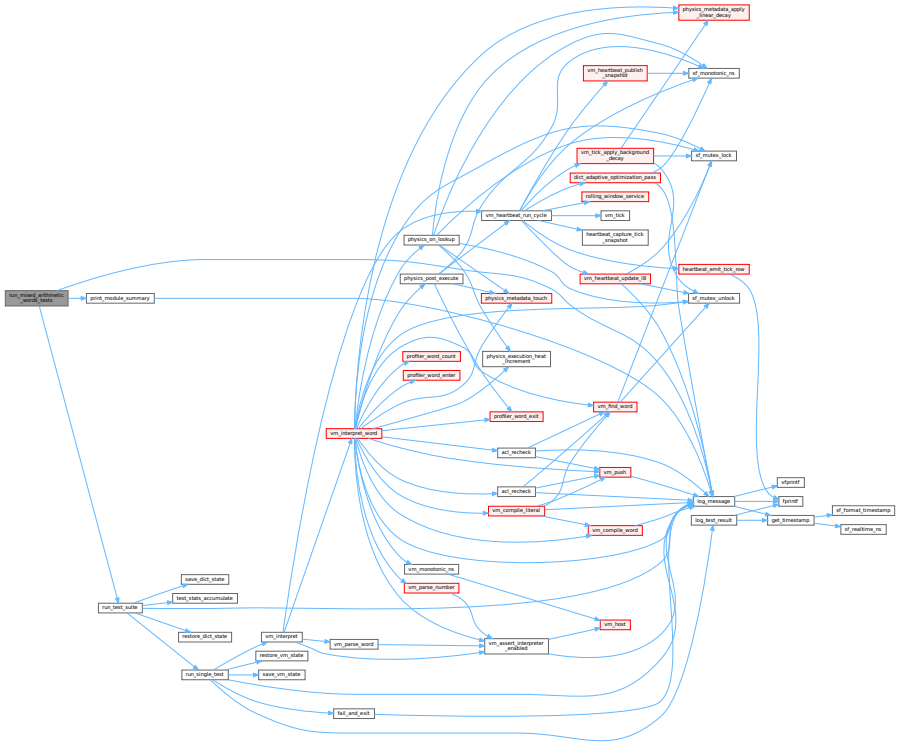
- `*-/` (n1 n2 n3 – n4) — scale: n1*n2/n3 in 64-bit intermediate
- `*-/MOD` (n1 n2 n3 – rem quot) — scale with remainder
- `M+` (d n – d') — add single to double
- `M*` (n1 n2 – d) — signed multiply to double
- `M/MOD` (d n – rem quot) — double divided by single

Parameters

vm	VM instance; must be initialised with the standard vocabulary.

References LOG_INFO, log_message(), LOG_TEST, mixed_arithmetic_word_suites, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.20 run_module_tests()

```
void run_module_tests (
            VM * vm,
            const char * module_name )
```

Run tests for a specific module.

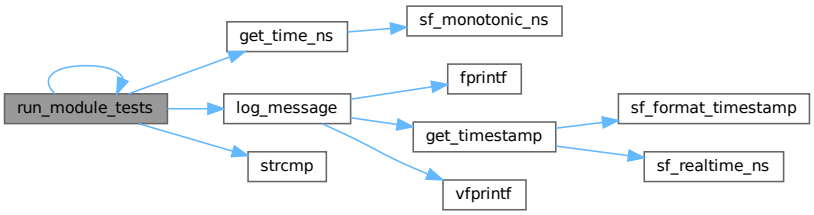
Parameters

vm	Pointer to the Forth virtual machine instance
module_name	Name of the module to test

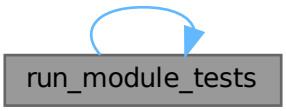
References benchmark_iterations, benchmark_mode, get_time_ns(), LOG_ERROR, LOG_INFO, log_message(), LOG_WARN, TestModule::module_name, run_module_tests(), TestModule::run_module_tests, strcmp(), and test_modules.

Referenced by run_module_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.311.1.21 run_physics_freeze_words_tests()

```
void run_physics_freeze_words_tests (
            VM * vm )
```

Executes the proof-covered subset of physics-freeze word test suites.

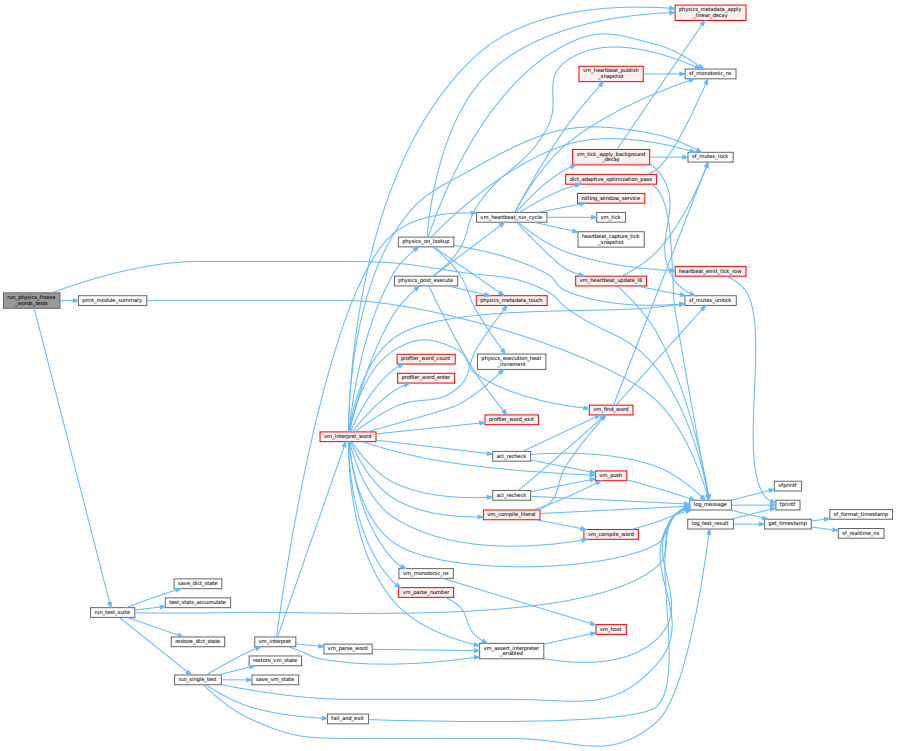
Parameters

vm	Pointer to the Forth virtual machine instance

Covers the 6 words proof/StarForth_Physics_Freeze_Words.thy gives real lemmas for: FREEZE-WORD, UNFREEZE-WORD, FROZEN?, HEAT!, HEAT@, DECAY-RATE. Does not cover FREEZE-CRITICAL, SHOW-HEAT, or ALL-HEATS — those carry only trivial/not-modelled lemmas in the same theory file.

References LOG_INFO, log_message(), LOG_TEST, physics_freeze_word_suites, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.22 run_q48_words_tests()

```
void run_q48_words_tests (
            VM * vm )
```

Executes all Q48.16 fixed-point word test suites.

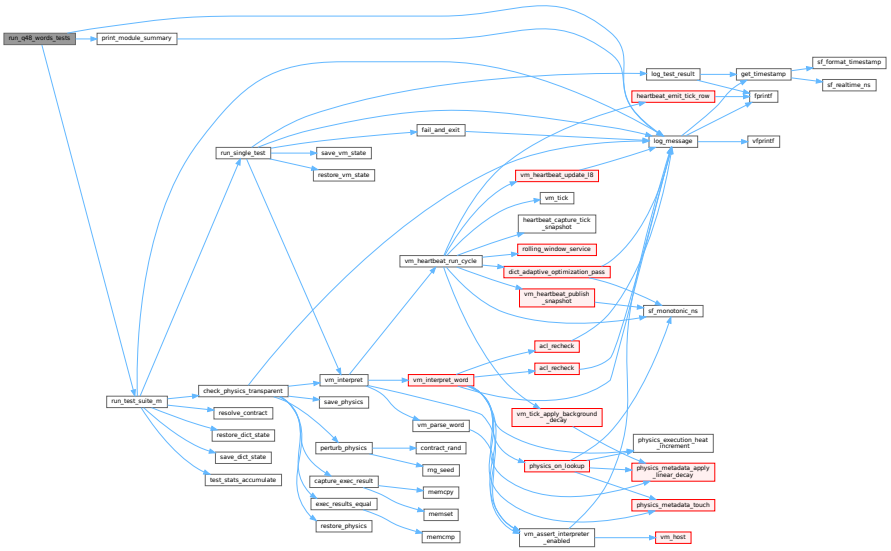
Parameters

vm	Pointer to the Forth virtual machine instance

Covers all 23 words registered by register_q48_words() (q48_words.c): arithmetic (+, -, ∗, /), sign (ABS, NEG), transcendental approximations (LOG, EXP, SQRT, SIN, COS), conversions (FROM-INT, TO-INT), constants (1, 0, SCALE), comparisons (=, <, >, 0=, MAX, MIN), and PRINT.

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), q48_word_suites, run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.23 run_return_stack_words_tests()

```
void run_return_stack_words_tests (
            VM * vm )
```

Executes all test suites for return stack manipulation words.

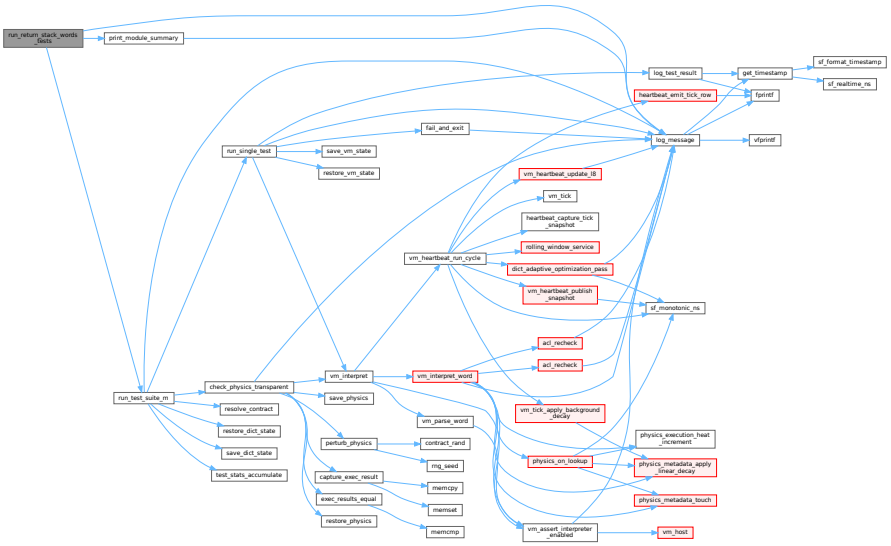
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all defined test cases for >R, R>, and R@ words and reports results_run_01_2025_12_08 through the logging system

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), return_stack_word_suites, run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.24 run_stack_words_tests()

```
void run_stack_words_tests (
            VM * vm )
```

Executes all test suites for stack manipulation words.

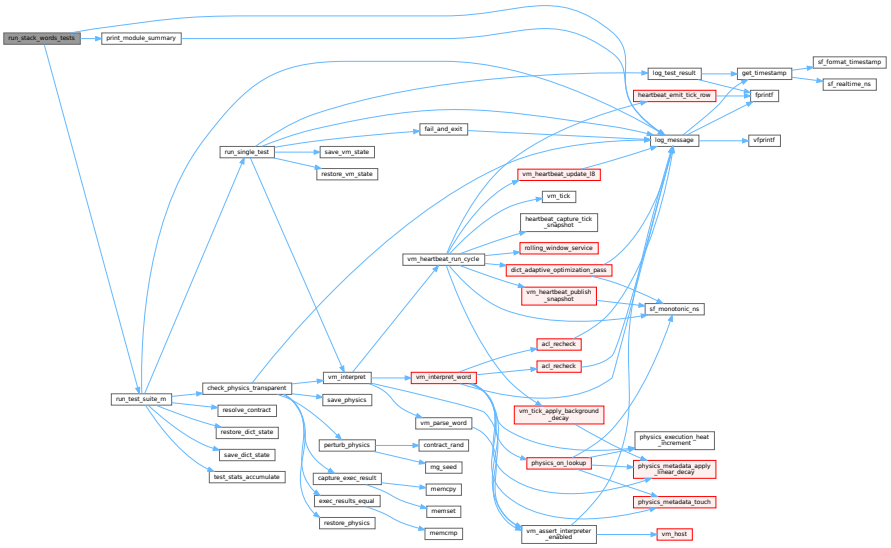
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all defined test cases for stack manipulation words and reports results_run_01_2025_12_08 through the logging system

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite_m(), stack_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.25 run_starforth_words_tests()

```
void run_starforth_words_tests (
            VM * vm )
```

Executes all StarForth words test suites.

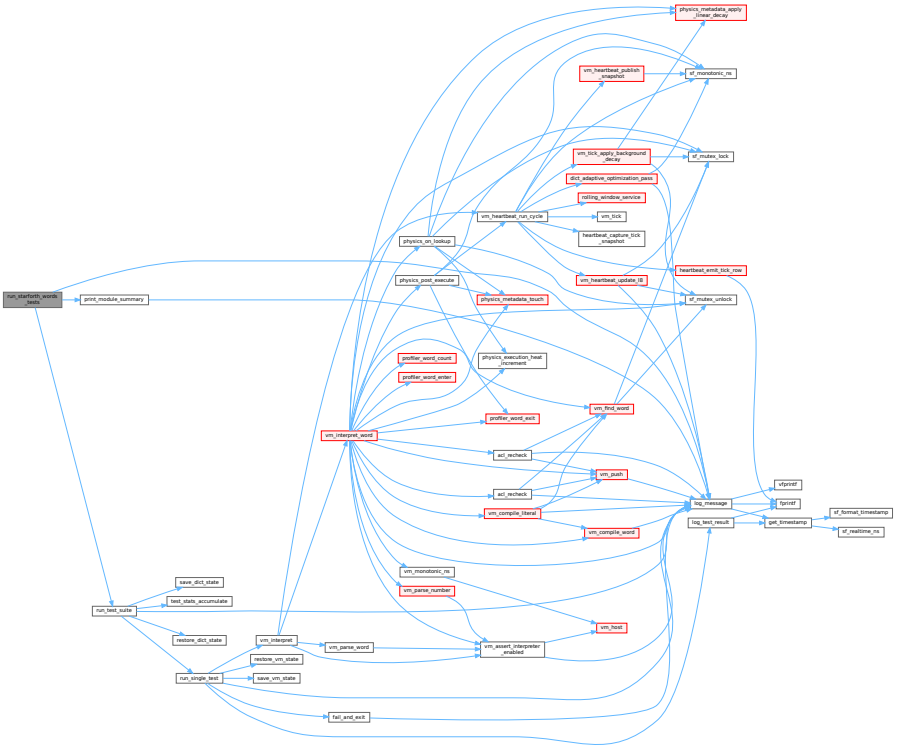
Parameters

vm	Pointer to the FORTH virtual machine instance

Runs through all test cases for StarForth-specific implementation words, including entropy tracking, comments, and initialization.

References LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), starforth_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.26 run_stress_tests()

```
void run_stress_tests (
            VM * vm )
```

Execute comprehensive stress test suites.

Runs an extensive set of stress tests to verify the VM's behavior under extreme conditions. Tests include:

- Stack operations and boundaries
- Deep nesting and control flow
- Numeric boundaries and overflow
- Dictionary pressure
- Memory allocation patterns
- Error recovery
- Mixed complexity scenarios

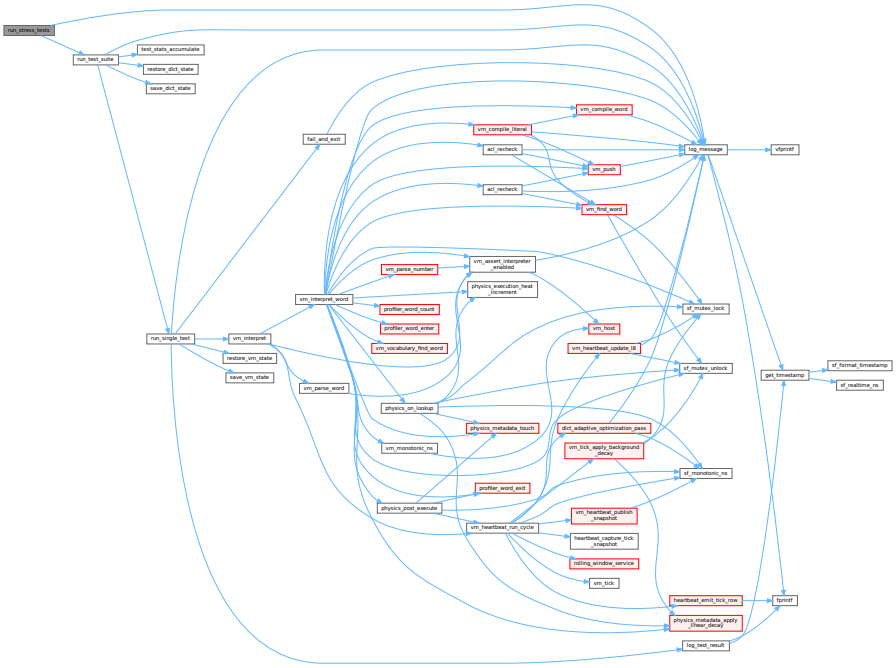
All tests are FORTH-79 compliant.

Parameters

vm	Pointer to the Forth virtual machine instance

References global_test_stats, LOG_INFO, log_message(), LOG_TEST, run_test_suite(), stress_suites, TestStats::total_error, TestStats::total_fail, TestStats::total_pass, TestStats::total_skip, and TestStats::total_tests.

Here is the call graph for this function:



7.311.1.27 run_string_words_tests()

```
void run_string_words_tests (
            VM * vm )
```

Executes all string manipulation word tests.

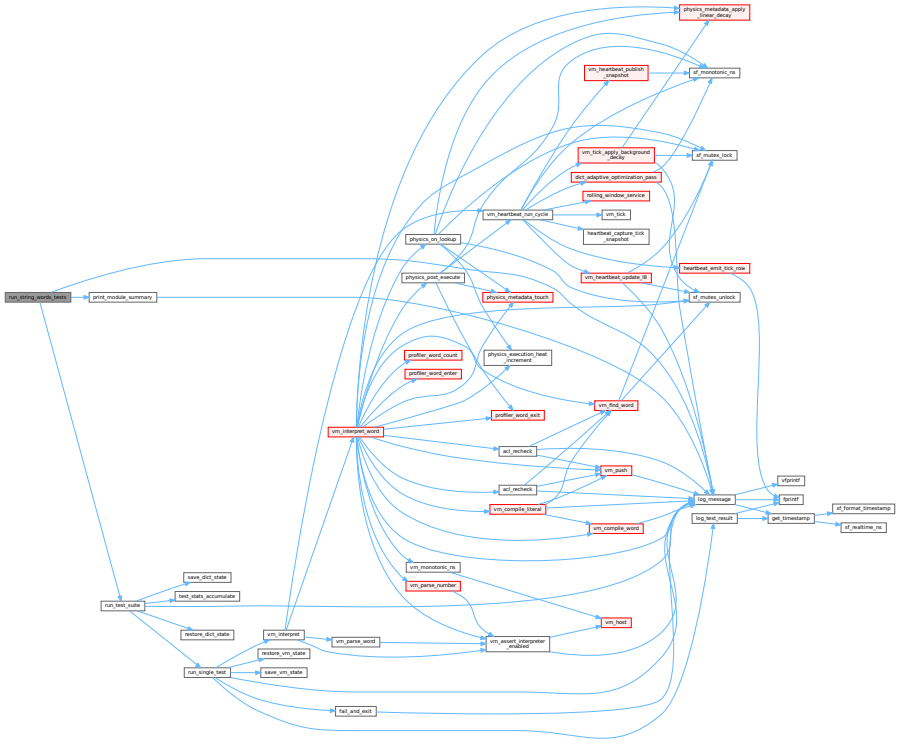
Runs test suites for FORTH string operations including COUNT, -TRAILING, CMOVE, CMOVE>, COMPARE, SEARCH, SCAN, SKIP, and BLANK

Parameters

vm	Pointer to the Forth virtual machine instance

References LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), string_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.28 run_system_words_tests()

```
void run_system_words_tests (
            VM * vm )
```

Executes all system word tests.

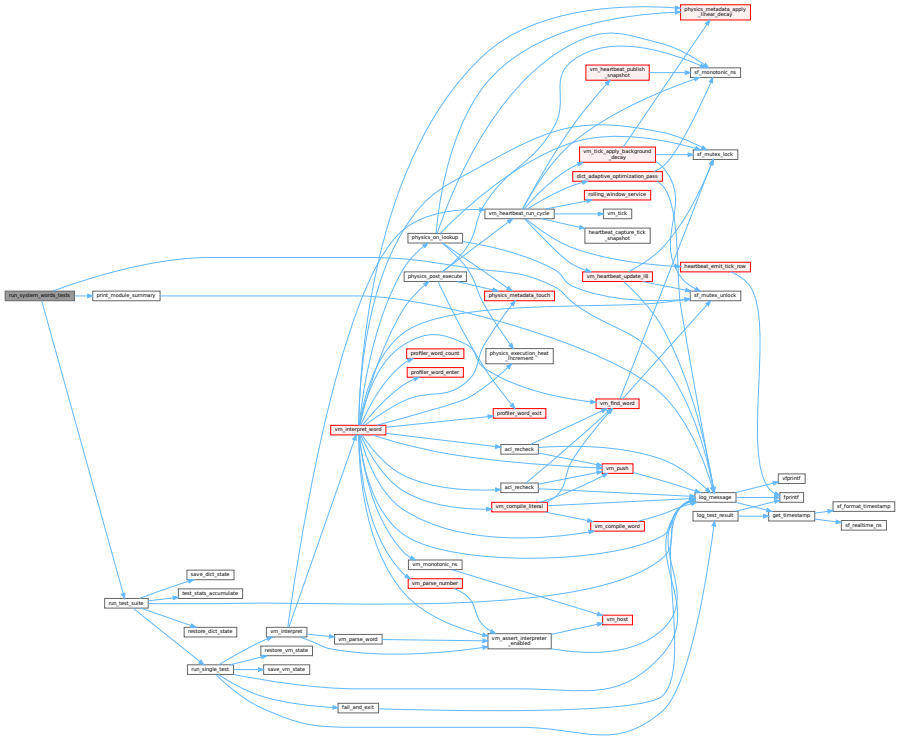
Runs test suites for FORTH system control operations including QUIT, ABORT, ABORT", BYE, COLD, and WARM. Some tests are marked as unimplemented due to their system-level effects

Parameters

vm	Pointer to the Forth virtual machine instance

References LOG_INFO, log_message(), LOG_TEST, LOG_WARN, print_module_summary(), run_test_suite(), system_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.29 run_vocabulary_words_tests()

```
void run_vocabulary_words_tests (
            VM * vm )
```

Executes all vocabulary word test suites.

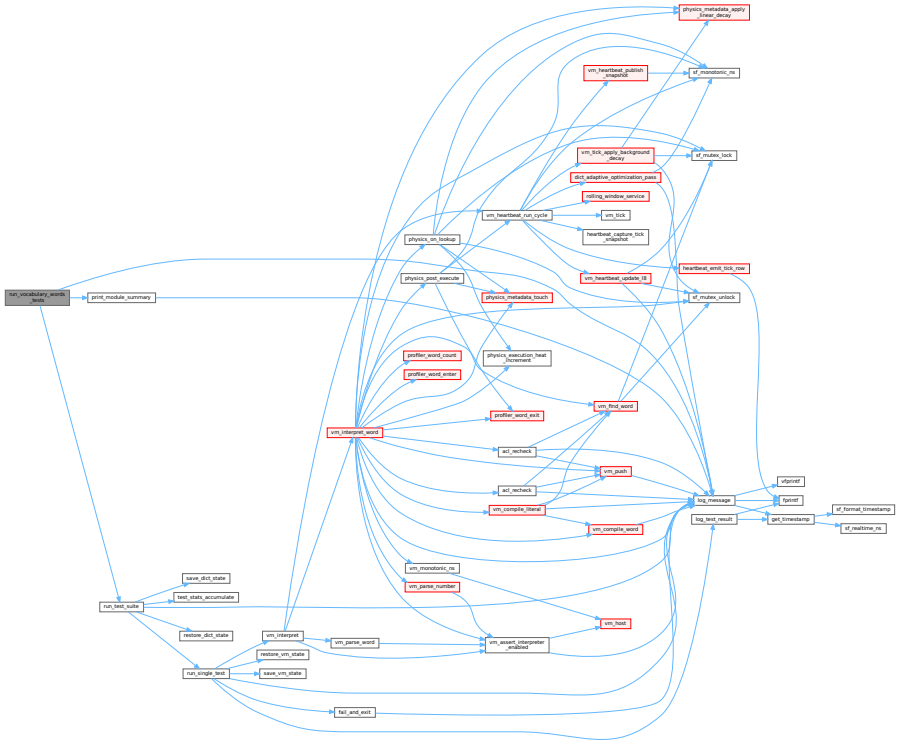
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all vocabulary-related test suites, executing each test case and logging the results_run_01_2025_↩ 12_08. Tests vocabulary creation, selection, and manipulation.

References LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), vocabulary_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.311.1.30 run_word_tests()

```
void run_word_tests (
            VM * vm,
            const char * word_name )
```

Run tests for a specific Forth word.

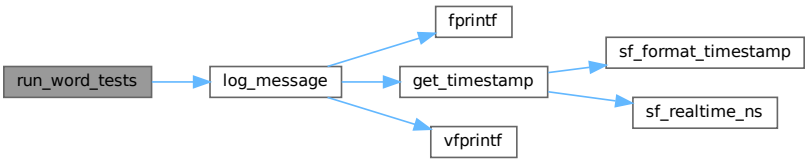
Searches all modules for tests related to the specified word and executes them.

Parameters

vm	Pointer to the Forth virtual machine instance
word_name	Name of the Forth word to test

References LOG_INFO, log_message(), LOG_WARN, TestModule::module_name, and test_modules.

Here is the call graph for this function:



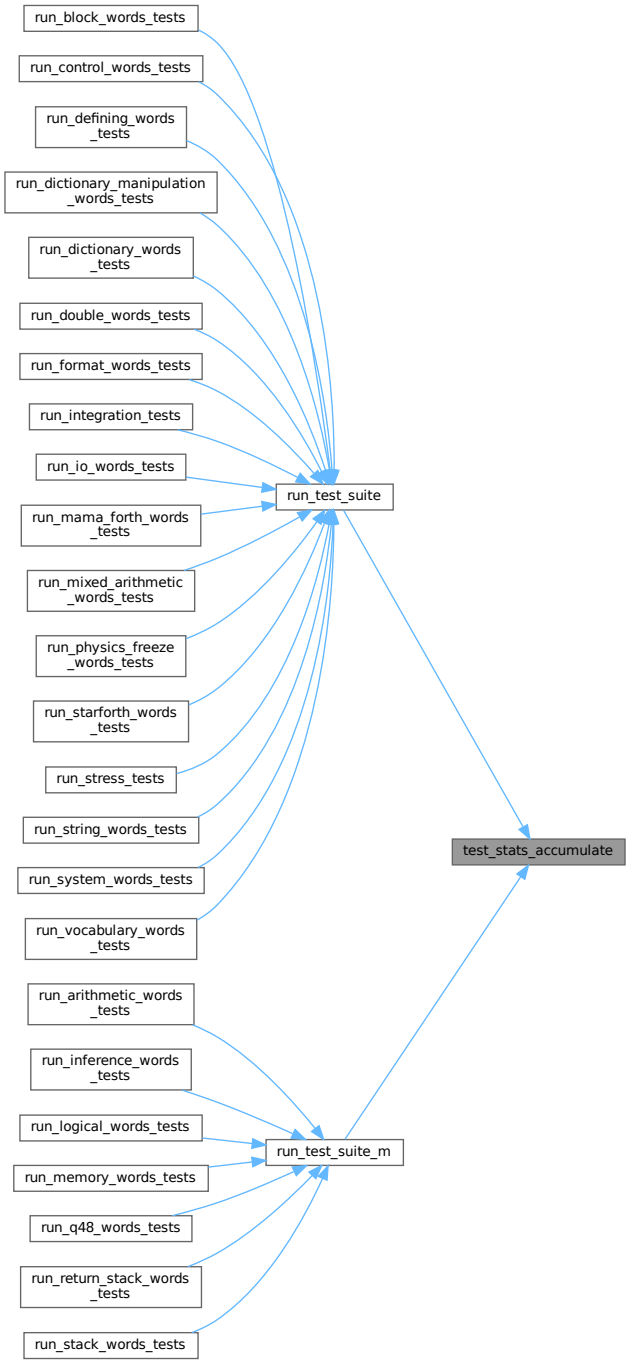
7.311.1.31 test_stats_accumulate()

```
void test_stats_accumulate (
            int tests,
            int pass,
            int fail,
            int skip,
            int error )
```

References global_test_stats, TestStats::total_error, TestStats::total_fail, TestStats::total_pass, TestStats::total_skip, and TestStats::total_tests.

Referenced by run_test_suite(), and run_test_suite_m().

Here is the caller graph for this function:



7.311.2 Variable Documentation

7.311.2.1 global_test_stats

```
TestStats global_test_stats  [extern]
```

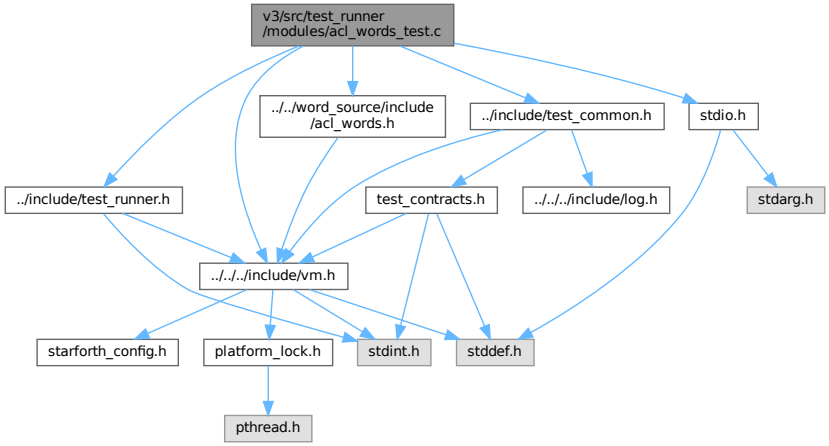
Global statistics for test execution.

Referenced by run_all_tests(), run_break_me_tests(), run_stress_tests(), sk_parity_collect(), and test_stats_accumulate().

7.312 v3/src/test_runner/modules/acl_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
#include "../../../include/vm.h"
#include "../../word_source/include/acl_words.h"
#include <stdio.h>
```

Include dependency graph for acl_words_test.c:



Macros

- #define ACL_ASSERT(cond, msg)

Functions

- static void test_acl_pin_immutable (VM *vm)

  Test 1 — ACL-PIN is a one-way ratchet.
- static void test_acl_inherit (VM *vm)

  Test 2 — ACL-INHERIT copies mode, clears pin, and resets TTL/allow.
- static void test_emergency_console_bypass (VM *vm)

  Test 3 — emergency_console flag bypasses a denied word.
- static void test_acl_deny_aborts (VM *vm)

  Test 4 — ACL denial aborts execution without running the word body.
- static void test_acl_ttl_hot_path (VM *vm)

  Test 5 — ACL TTL hot-path decrements without invoking recheck.
- static void test_acl_strict_mode (VM *vm)

  Test 6 — STRICT mode keeps TTL at 0 after every recheck.
- static void test_acl_adaptive_ttl (VM *vm)

  Test 7 — Adaptive TTL scales with execution heat via ACL-TTL-COMPUTE.
- static void test_acl_pin_blocks_shadow (VM *vm)

  Test 8 — A pinned word cannot be shadowed by redefinition.
- static void test_acl_read_accessors (VM *vm)

Test 9 — ACL-MODE@/ACL-PINNED?/ACL-TTL@/ACL-ALLOW@/ACL-HEAT@/ ACL-WORD-ID read a word's ACL fields via the interpreter.

- static void test_acl_inherit_word_dispatch (VM *vm)

  Test 10 — ACL-INHERIT, as an interpreted FORTH word (src dst –).

- static void test_acl_init_primitives (VM *vm)

  Test 11 — ACL-INIT-PRIMITIVES resets unpinned words, skips pinned ones.

- void run_acl_words_tests (VM *vm)

  Module 24 entry point — run all word-level ACL POST tests.

Variables

- static int tests_passed = 0
- static int tests_failed = 0

7.312.1 Macro Definition Documentation

7.312.1.1 ACL_ASSERT

```
#define ACL_ASSERT(
                cond,
                msg )
```

Value:

```
    do { \
        if (cond) { \
            tests_passed++; \
            log_message(LOG_DEBUG, "  PASS: %s", msg); \
        } else { \
            tests_failed++; \
            log_message(LOG_ERROR, "  FAIL: %s", msg); \
        } \
    } while (0)
```

7.312.2 Function Documentation

7.312.2.1 run_acl_words_tests()

```
void run_acl_words_tests (
              VM * vm )
```

Module 24 entry point — run all word-level ACL POST tests.

Saves the current VM state (dsp, rsp, error, mode, emergency_console) before each sub-test and restores it between tests so failures do not cascade. Runs eleven tests covering:

1. ACL-PIN one-way ratchet (4 assertions)
2. ACL-INHERIT semantics (4 assertions)
3. emergency_console bypass (1 assertion)
4. ACL denial aborts word execution (2 assertions)
5. TTL hot-path decrement (1 assertion)

6. STRICT mode — TTL stays 0 (1 assertion or conditional skip)
7. Adaptive TTL via ACL-TTL-COMPUTE (1 assertion or conditional skip)
8. Pin blocks shadowing (3 assertions)
9. Read accessors — ACL-MODE@/PINNED?/TTL@/ALLOW@/HEAT@/WORD-ID (6 assertions)
10. ACL-INHERIT as an interpreted word, not just its C twin (2 assertions)
11. ACL-INIT-PRIMITIVES resets unpinned, skips pinned (4 assertions)

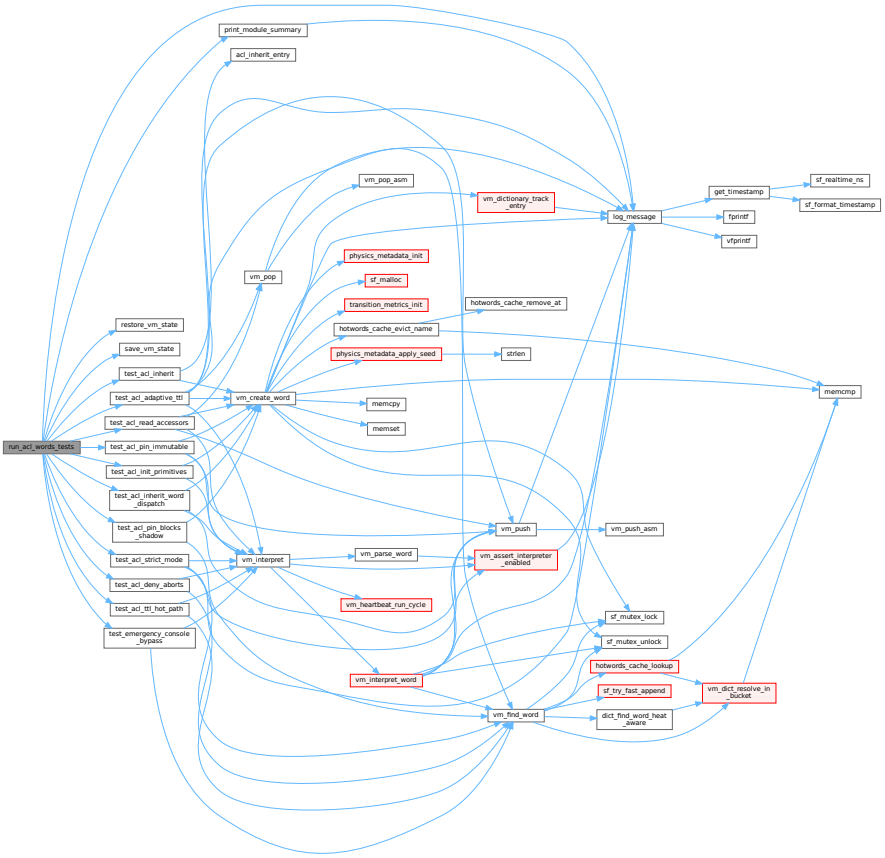
Tests that depend on `ACL.4th` being loaded are conditionally skipped during bare POST (before `ACL.4th` runs) and counted as passes so the POST score is not penalised by load-order.

Parameters

vm	VM instance; must be initialised and have the standard word vocabulary available.

References VM::emergency_console, LOG_INFO, log_message(), print_module_summary(), restore_vm_state(), save_vm_state(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_inherit(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_blocks_shadow(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), test_emergency_console_bypass(), tests_failed, and tests_passed.

Here is the call graph for this function:



7.312.2.2 test_acl_adaptive_ttl()

```
static void test_acl_adaptive_ttl (
            VM * vm )  [static]
```

Test 7 — Adaptive TTL scales with execution heat via ACL-TTL-COMPUTE.

Sets `execution_heat=2560` on a temporary entry and invokes `ACL-TTL-COMPUTE` if the word is present (i.e., `ACL.4th` is loaded). The expected formula is `ttl` = heat/4 + 256 = 640 + 256 = 896.

If `ACL-TTL-COMPUTE` is not present (ACL.4th not yet loaded), the test is conditionally skipped and counted as a pass.

Expected assertions (1, or skip):

- `ACL-TTL-COMPUTE` returns 896 for heat=2560.

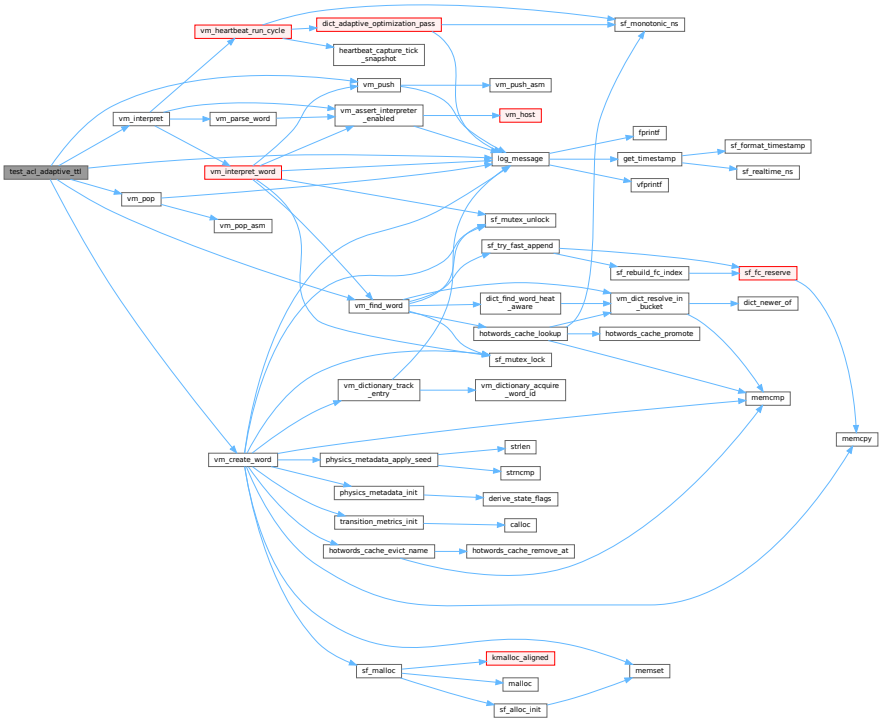
Parameters

vm	VM instance.

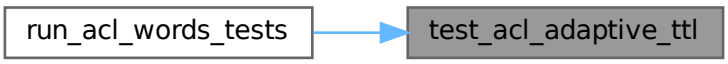
References DictEntry::acl_allow, ACL_ASSERT, DictEntry::acl_pinned, VM::dsp, VM::error, DictEntry::execution_heat, LOG_ERROR, LOG_INFO, log_message(), tests_failed, tests_passed, vm_create_word(), vm_find_word(), vm_interpret(), vm_pop(), and vm_push().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.3 test_acl_deny_aborts()

```
static void test_acl_deny_aborts (
            VM * vm )  [static]
```

Test 4 — ACL denial aborts execution without running the word body.

Defines a word that pushes a sentinel (99), then denies it with `acl_allow=0` and `acl_ttl=1` (hot-path decrement triggers the deny check without invoking `acl_recheck()`). Verifies that executing the word sets either `vm->abort_requested`, `vm->error`, or leaves the stack depth unchanged — and that the sentinel was never pushed.

`acl_ttl=1` is used rather than 0 to stay on the hot path (TTL decrements to 0 → check; avoids the fail-open cold-path recheck).

Expected assertions (2):

- Execution was aborted (abort_requested || error || stack unchanged)
- Stack depth is unchanged (sentinel was not pushed)

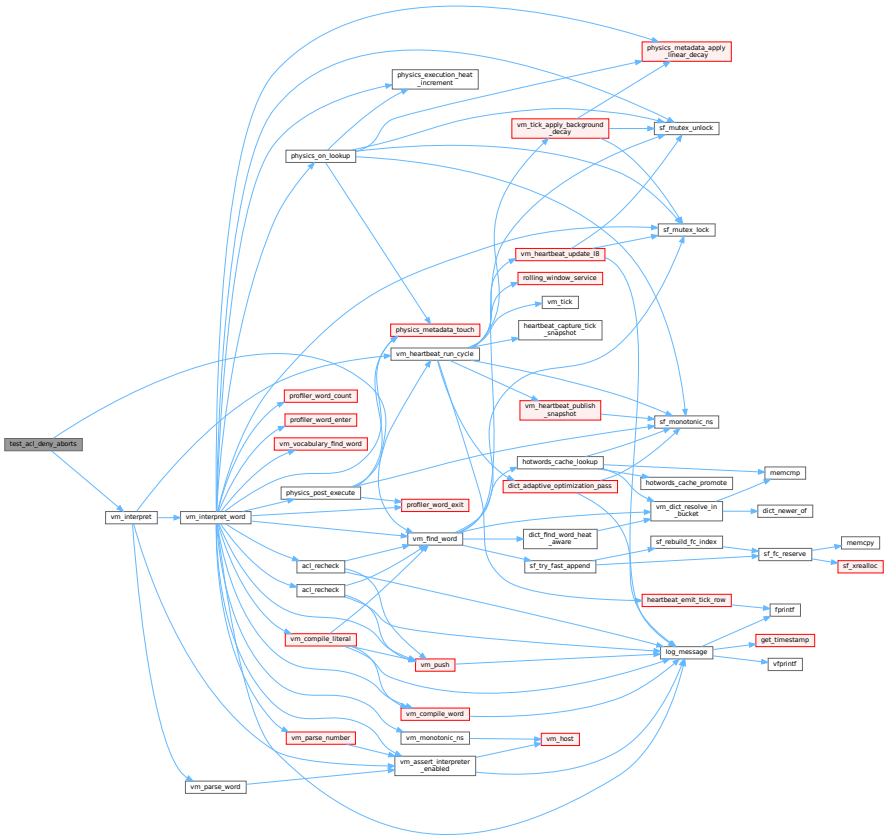
Parameters

vm	VM instance.

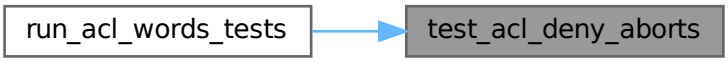
References VM::abort_requested, DictEntry::acl_allow, ACL_ASSERT, DictEntry::acl_pinned, DictEntry::acl_ttl, VM::dsp, VM::emergency_console, VM::error, tests_failed, vm_find_word(), and vm_interpret().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.4 test_acl_inherit()

```
static void test_acl_inherit (
            VM * vm )  [static]
```

Test 2 — ACL-INHERIT copies mode, clears pin, and resets TTL/allow.

Constructs a source entry with STRICT mode, pinned, TTL=5000, allow=0, and a destination entry with a different mode and also pinned. Calls `acl_inherit_entry()` directly and checks the resulting destination state. Verifies the Isabelle/HOL lemma `ACL_Inherit_Clears_Pin`.

Expected assertions (4):

- `dst->acl_mode` == ACL_MODE_STRICT (mode copied from src)
- `dst->acl_pinned` == 0 (pin cleared regardless of dst's original pin)
- `dst->acl_ttl` == 0 (TTL reset to 0)
- `dst->acl_allow` == 1 (allow reset to 1)

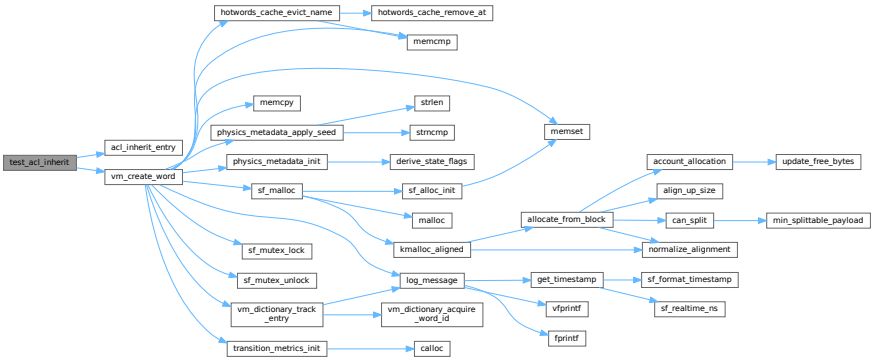
Parameters

vm	VM instance (used for word creation; not modified beyond that).

References DictEntry::acl_allow, ACL_ASSERT, acl_inherit_entry(), DictEntry::acl_mode, ACL_MODE_STRICT, ACL_MODE_TTL, DictEntry::acl_pinned, DictEntry::acl_ttl, tests_failed, and vm_create_word().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:

7.312.2.5 test_acl_inherit_word_dispatch()

```
static void test_acl_inherit_word_dispatch (
            VM * vm )  [static]
```

Test 10 — ACL-INHERIT, as an interpreted FORTH word (src dst –).

`test_acl_inherit` above already proves the underlying `acl_inherit_entry()` C function is correct (Isabelle lemma `ACL_Inherit_Clears_Pin`) by calling it directly. This test instead exercises the FORTH-level dispatch path itself — pushing two xts and interpreting "ACL-INHERIT" — which nothing in this file previously did, so the word's own stack-popping/argument-order code was untested.

Expected assertions (2):

- `dst->acl_mode` == ACL_MODE_STRICT after interpreting ACL-INHERIT
- `dst->acl_pinned` == 0 after interpreting ACL-INHERIT

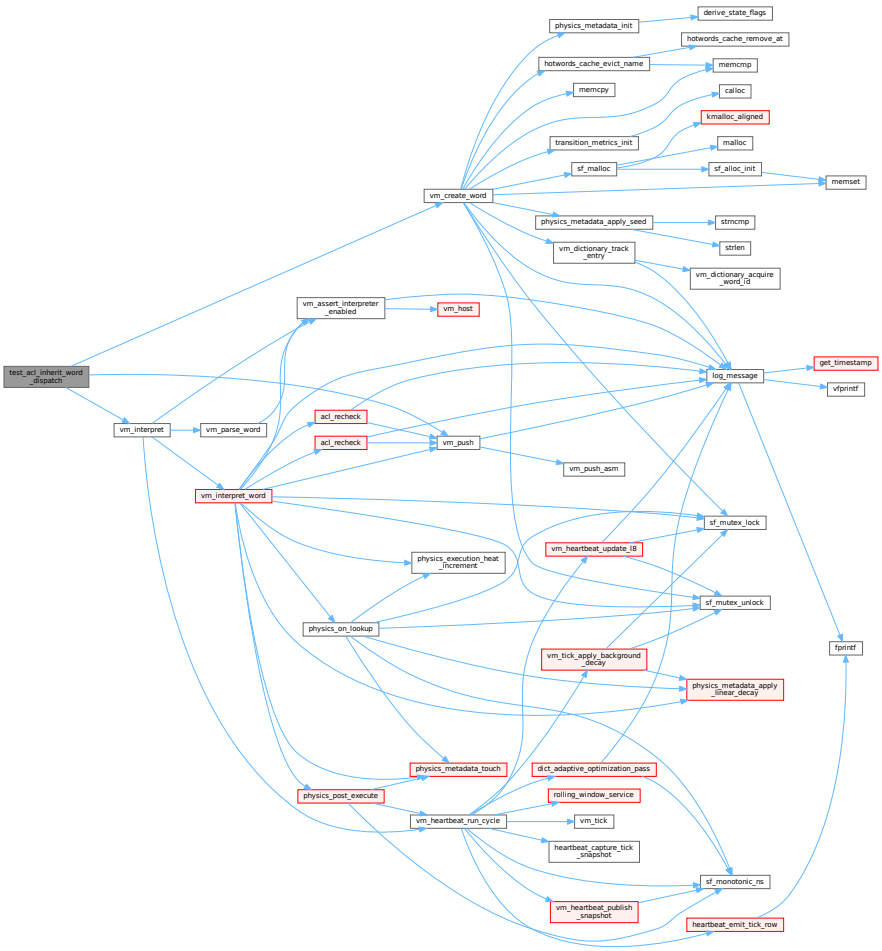
Parameters

vm	VM instance.

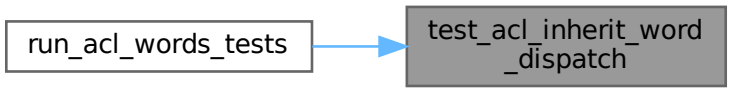
References ACL_ASSERT, DictEntry::acl_mode, ACL_MODE_STRICT, ACL_MODE_TTL, DictEntry::acl_pinned, VM::error, tests_failed, vm_create_word(), vm_interpret(), and vm_push().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.6 test_acl_init_primitives()

```
static void test_acl_init_primitives (
            VM * vm )  [static]
```

Test 11 — ACL-INIT-PRIMITIVES resets unpinned words, skips pinned ones.

Creates two fresh words with non-default ACL state: one left unpinned, one pinned. Interprets "ACL-INIT-↩ PRIMITIVES" (which walks the entire dictionary) and verifies the unpinned word was reset to the permissive defaults (`acl_ttl=0`, `acl_allow=1`, `acl_mode=ACL_MODE_TTL`) while the pinned word's deliberately-different state was left untouched.

Expected assertions (4):

- unpinned word's `acl_ttl` reset to 0
- unpinned word's `acl_allow` reset to 1
- unpinned word's `acl_mode` reset to ACL_MODE_TTL
- pinned word's `acl_ttl` is NOT reset (still its original value)

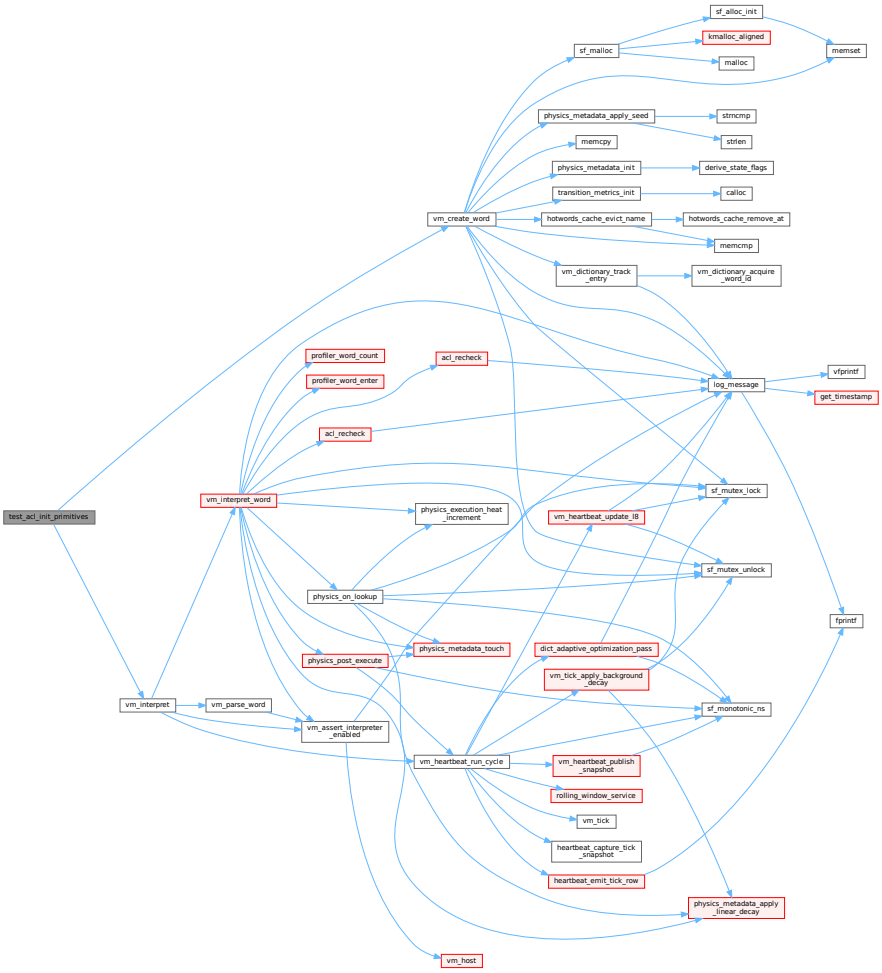
Parameters

vm	VM instance.

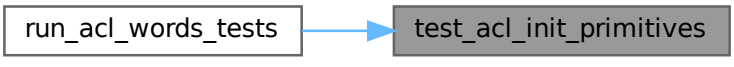
References DictEntry::acl_allow, ACL_ASSERT, DictEntry::acl_mode, ACL_MODE_STRICT, ACL_MODE_TTL, DictEntry::acl_pinned, DictEntry::acl_ttl, VM::error, tests_failed, vm_create_word(), and vm_interpret().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.7 test_acl_pin_blocks_shadow()

```
static void test_acl_pin_blocks_shadow (
            VM * vm )  [static]
```

Test 8 — A pinned word cannot be shadowed by redefinition.

Creates a word, pins it, then attempts to create another word with the same name via `vm_create_word()`. Verifies that the second create returns NULL and sets `vm->error`, and that the original pinned entry is still the one returned by `vm_find_word()`.

This enforces the Isabelle/HOL lemma `ACL_Pin_Monotone:` pin is permanent and cannot be overridden even by a new definition.

Expected assertions (3):

- `vm_create_word()` returns NULL for the shadowed name
- `vm->error` is non-zero after the blocked create
- `vm_find_word()` still returns the original pinned entry

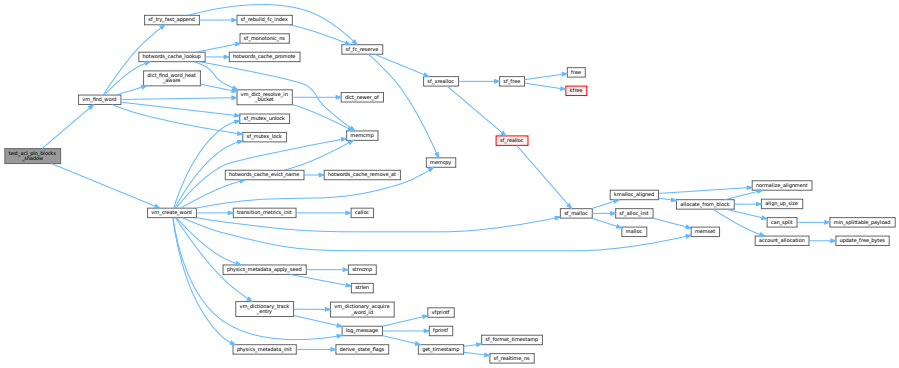
Parameters

vm	VM instance.

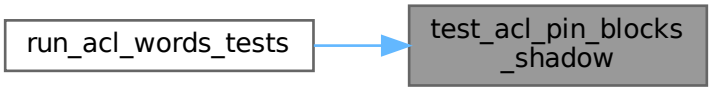
References ACL_ASSERT, DictEntry::acl_pinned, VM::error, tests_failed, vm_create_word(), and vm_find_word().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.8 test_acl_pin_immutable()

```
static void test_acl_pin_immutable (
            VM * vm )  [static]
```

Test 1 — ACL-PIN is a one-way ratchet.

Creates a temporary word and pins it via `ACL-PIN`. Verifies that subsequent calls to `ACL-MODE!`, `ACL-TTL!`, and `ACL-ALLOW!` are all no-ops while the word is pinned. The pin flag itself must remain 1.

Expected assertions (4):

- `acl_pinned` == 1 after ACL-PIN
- `acl_mode` unchanged after ACL-MODE! on pinned entry
- `acl_ttl` unchanged after ACL-TTL! on pinned entry
- `acl_allow` unchanged after ACL-ALLOW! on pinned entry

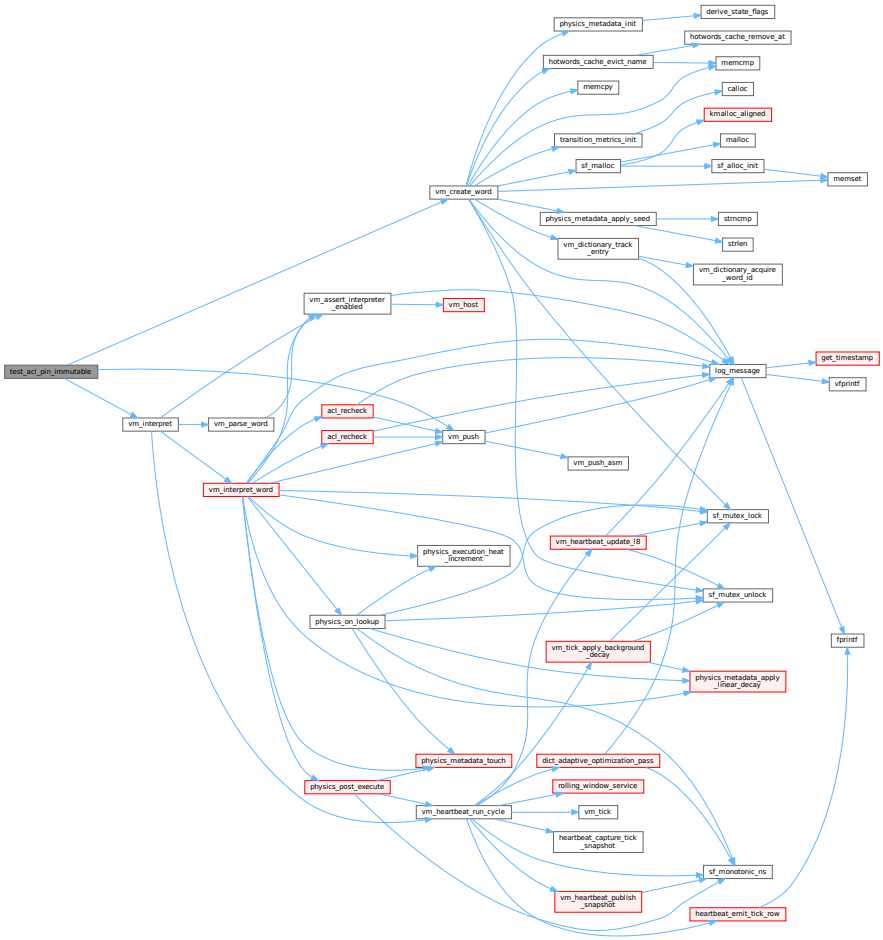
Parameters

vm	VM instance (shared state; caller restores after this function).

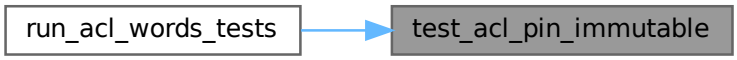
References DictEntry::acl_allow, ACL_ASSERT, DictEntry::acl_mode, ACL_MODE_STRICT, ACL_MODE_TTL, DictEntry::acl_pinned, DictEntry::acl_ttl, VM::error, tests_failed, vm_create_word(), vm_interpret(), and vm_push().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.9 test_acl_read_accessors()

```
static void test_acl_read_accessors (
            VM * vm )  [static]
```

Test 9 — ACL-MODE@/ACL-PINNED?/ACL-TTL@/ACL-ALLOW@/ACL-HEAT@/ ACL-WORD-ID read a word's ACL fields via the interpreter.

Proof-covered field accessors (`proof/StarForth_ACL_*.thy`) that had no interpreter-level POST coverage before this test — every other ACL test in this file exercises the write accessors (`ACL-MODE!`/`ACL-TTL!`/`ACL-ALLOW!`) or calls the underlying C functions directly, but never pushed an xt and interpreted these six read words.

Sets each field to a distinctive non-default value directly on a fresh test word, then pushes its xt and interprets each accessor in turn, checking the value returned on the stack.

Expected assertions (6):

- ACL-MODE@ returns `ACL_MODE_STRICT`
- ACL-PINNED? returns -1 (true) for a pinned entry
- ACL-TTL@ returns 4242
- ACL-ALLOW@ returns 0 (false)
- ACL-HEAT@ returns the entry's `execution_heat`
- ACL-WORD-ID returns the entry's `word_id`

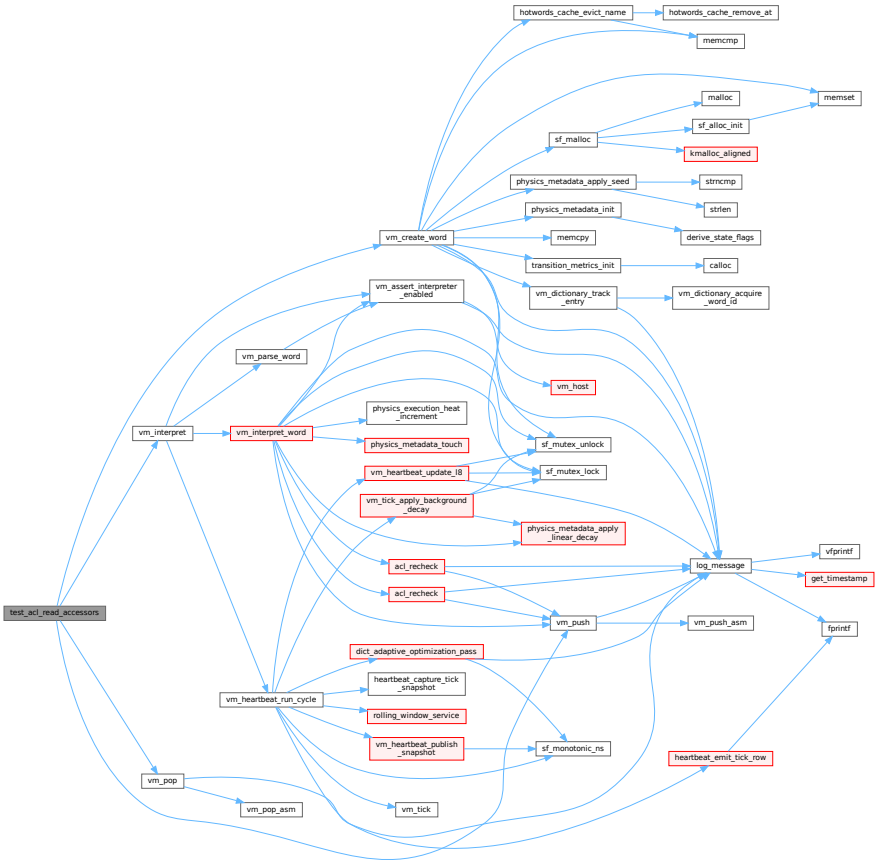
Parameters

vm	VM instance.

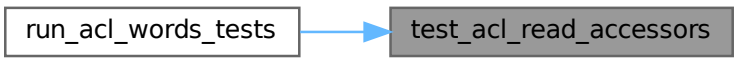
References DictEntry::acl_allow, ACL_ASSERT, DictEntry::acl_mode, ACL_MODE_STRICT, DictEntry::acl_pinned, DictEntry::acl_ttl, VM::error, DictEntry::execution_heat, tests_failed, vm_create_word(), vm_interpret(), vm_pop(), vm_push(), and DictEntry::word_id.

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.10 test_acl_strict_mode()

```
static void test_acl_strict_mode (
            VM * vm )  [static]
```

Test 6 — STRICT mode keeps TTL at 0 after every recheck.

In `ACL_MODE_STRICT`, `ACL-RECHECK` must reset `acl_ttl` to 0 after each allow decision so the recheck fires on every subsequent execution.

If `ACL.4th` is not loaded, `ACL-RECHECK` is absent and `acl_recheck()` fails open (`acl_ttl` = `ACL_TTL_OPEN`). In that case the test is conditionally skipped (counted as a pass) since the ACL system is not yet active.

Expected assertions (1, or skip counted as 1 pass):

- `e->acl_ttl` == 0 after one execution in STRICT mode with ACL loaded.

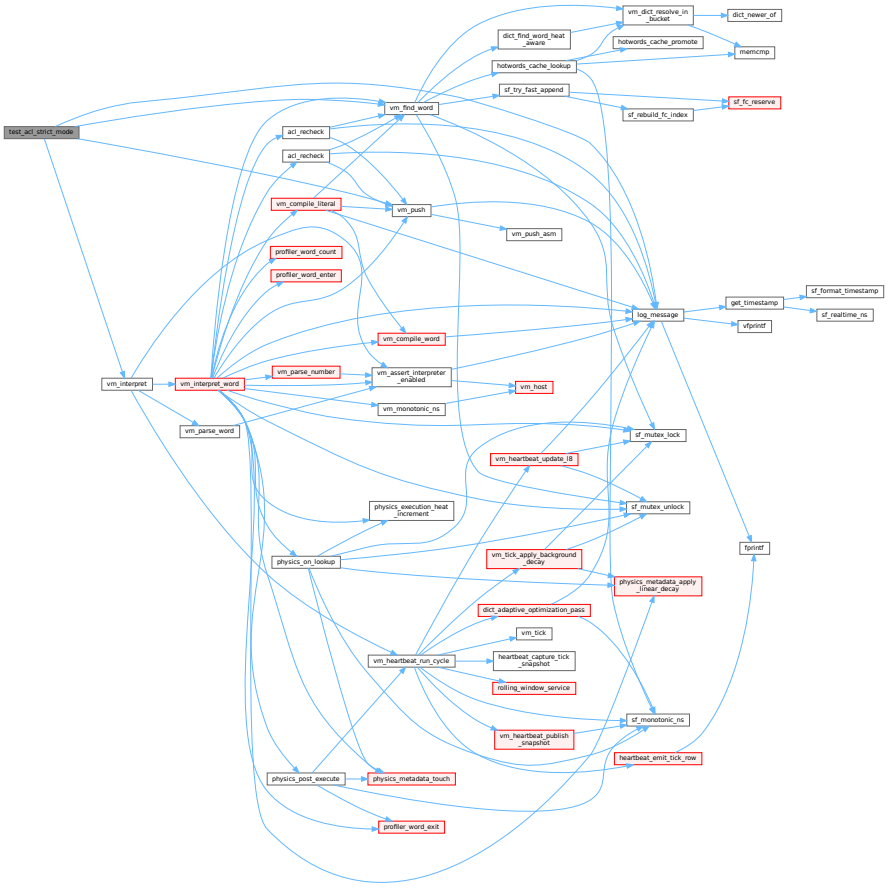
Parameters

vm	VM instance.

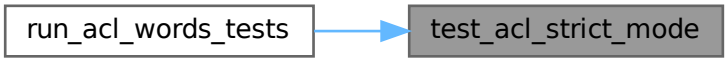
References DictEntry::acl_allow, ACL_ASSERT, DictEntry::acl_mode, ACL_MODE_STRICT, DictEntry::acl_pinned, DictEntry::acl_ttl, VM::dsp, VM::error, LOG_INFO, log_message(), tests_failed, tests_passed, vm_find_word(), vm_interpret(), and vm_push().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.11 test_acl_ttl_hot_path()

```
static void test_acl_ttl_hot_path (
            VM * vm )  [static]
```

Test 5 — ACL TTL hot-path decrements without invoking recheck.

Defines a word with `acl_ttl=10` and `acl_allow=1`, then executes it three times via the outer interpreter. Each outer-interpreter execution decrements `acl_ttl` by 1, so the expected final value is 7.

This confirms the hot-path optimisation: the interpreter only calls `acl_recheck()` when `acl_ttl` reaches 0, not on every invocation.

Expected assertions (1):

- `e->acl_ttl` == 7 after three executions (10 → 9 → 8 → 7)

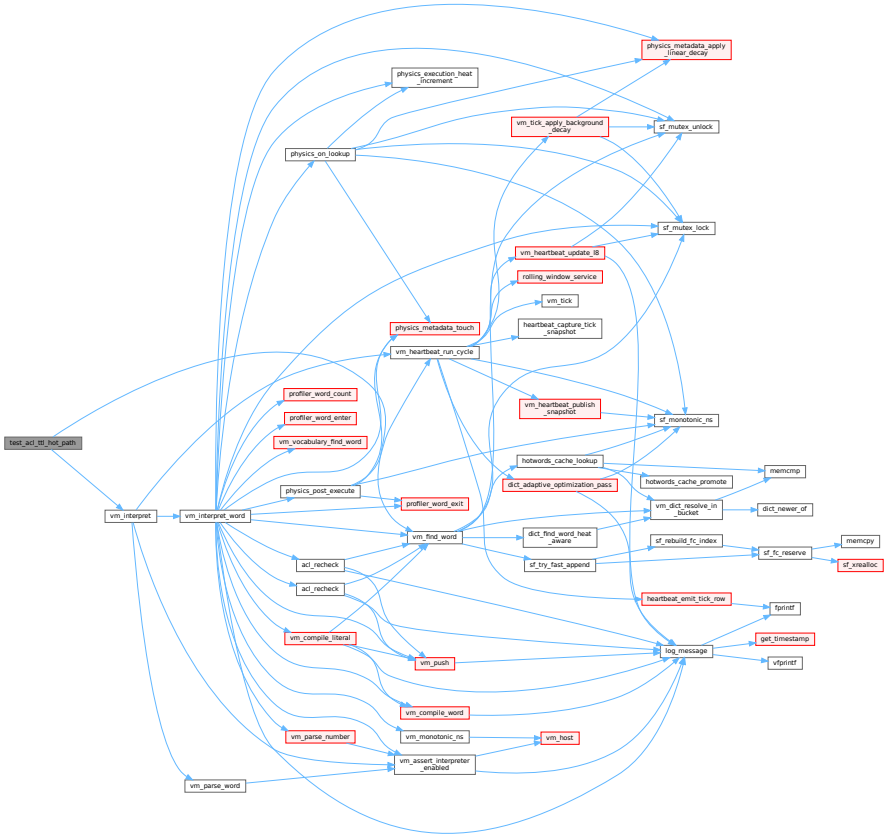
Parameters

vm	VM instance.

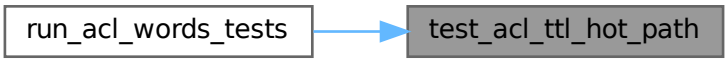
References DictEntry::acl_allow, ACL_ASSERT, DictEntry::acl_mode, ACL_MODE_TTL, DictEntry::acl_pinned, DictEntry::acl_ttl, VM::emergency_console, VM::error, tests_failed, vm_find_word(), and vm_interpret().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.2.12 test_emergency_console_bypass()

```
static void test_emergency_console_bypass (
            VM * vm )  [static]
```

Test 3 — emergency_console flag bypasses a denied word.

Defines a FORTH word that pushes 42, then sets its `acl_allow` to 0 to simulate a policy denial. Executes the word with `emergency_console=1` and verifies that the word body ran (stack depth increased by 1).

The `emergency_console` bypass is required by the bare-metal fault handler so that critical recovery words can always execute.

Expected assertions (1):

- `vm->dsp` == saved_dsp + 1 (word ran despite deny)

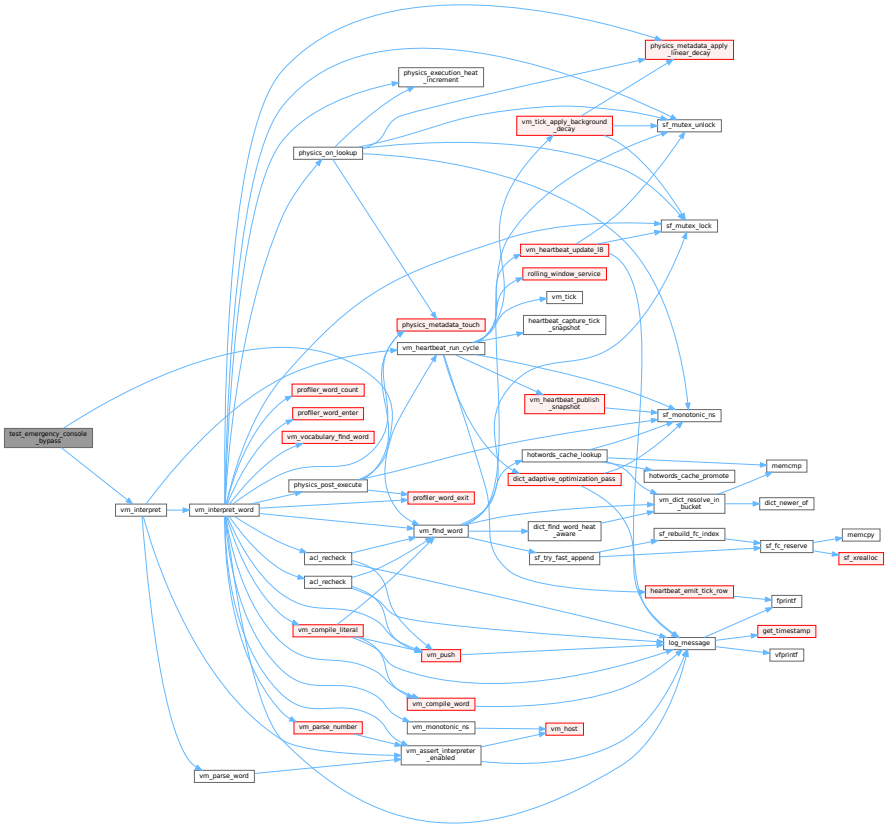
Parameters

vm	VM instance.

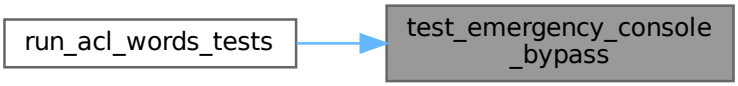
References DictEntry::acl_allow, ACL_ASSERT, DictEntry::acl_pinned, DictEntry::acl_ttl, VM::dsp, VM::emergency_console, VM::error, tests_failed, vm_find_word(), and vm_interpret().

Referenced by run_acl_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.312.3 Variable Documentation

7.312.3.1 tests_failed

```
int tests_failed = 0  [static]
```

Referenced by run_acl_words_tests(), test_acl_adaptive_ttl(), test_acl_deny_aborts(), test_acl_inherit(), test_acl_inherit_word_dispatch(), test_acl_init_primitives(), test_acl_pin_blocks_shadow(), test_acl_pin_immutable(), test_acl_read_accessors(), test_acl_strict_mode(), test_acl_ttl_hot_path(), and test_emergency_console_bypass().

7.312.3.2 tests_passed

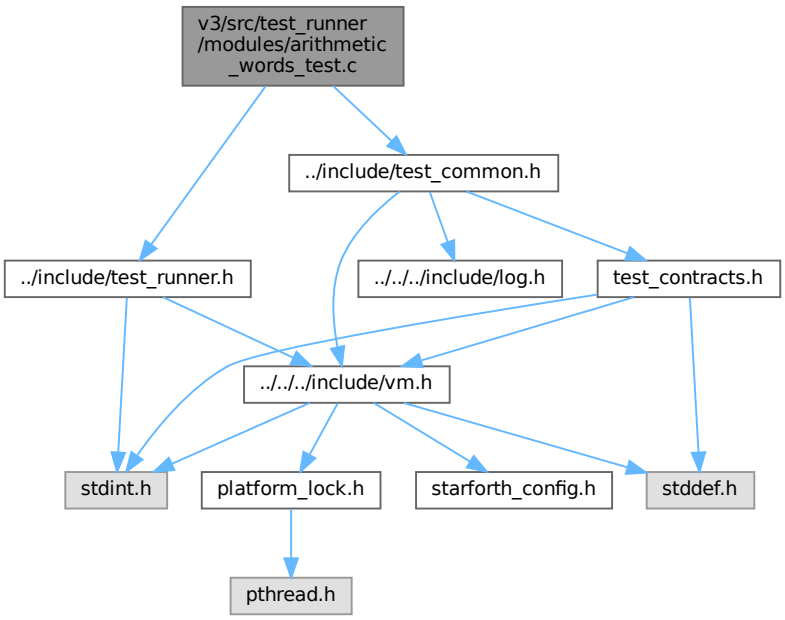
```
int tests_passed = 0  [static]
```

Referenced by run_acl_words_tests(), test_acl_adaptive_ttl(), and test_acl_strict_mode().

7.313 v3/src/test_runner/modules/arithmetic_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for arithmetic_words_test.c:



Functions

- void run_arithmetic_words_tests (VM *vm)

  Executes all arithmetic word test suites.

Variables

- static WordTestSuite arithmetic_word_suites [ ]

7.313.1 Function Documentation

7.313.1.1 run_arithmetic_words_tests()

```
void run_arithmetic_words_tests (
            VM * vm )
```

Executes all arithmetic word test suites.

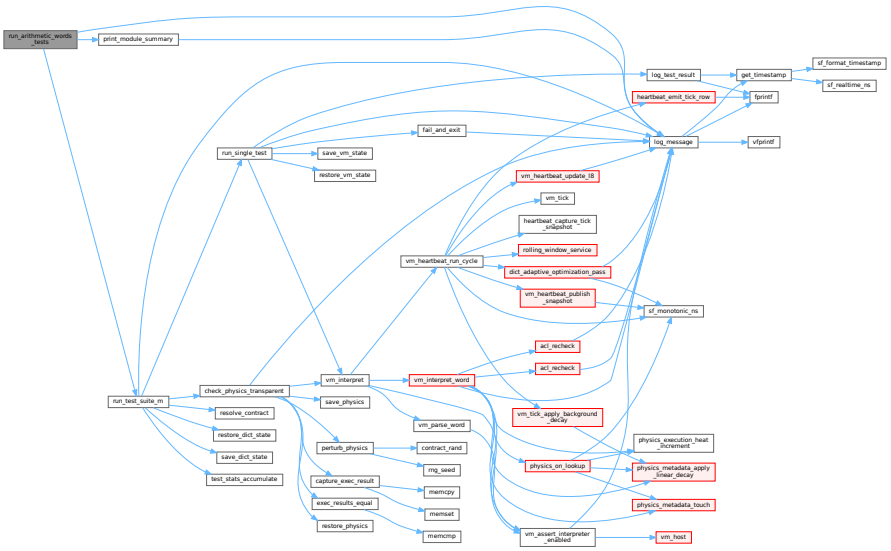
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all test cases for arithmetic operations including basic operations (+, -, ∗, /), modulo operations (MOD, /MOD), and numeric functions (ABS, NEGATE, MIN, MAX)

References arithmetic_word_suites, CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.313.2 Variable Documentation

7.313.2.1 arithmetic_word_suites

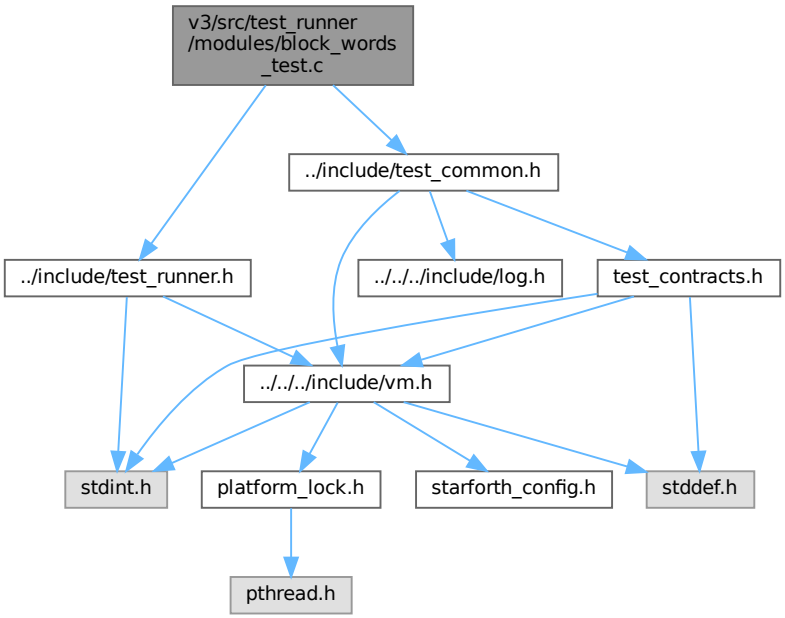
```
WordTestSuite arithmetic_word_suites[ ]  [static]
```

Referenced by run_arithmetic_words_tests().

7.314 v3/src/test_runner/modules/block_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for block_words_test.c:



Functions

- void run_block_words_tests (VM *vm)

  Executes all block words test suites.

Variables

- static WordTestSuite block_word_suites [ ]

  Test suites for FORTH block words (Module 16)

7.314.1 Function Documentation

7.314.1.1 run_block_words_tests()

```
void run_block_words_tests (
            VM * vm )
```

Executes all block words test suites.

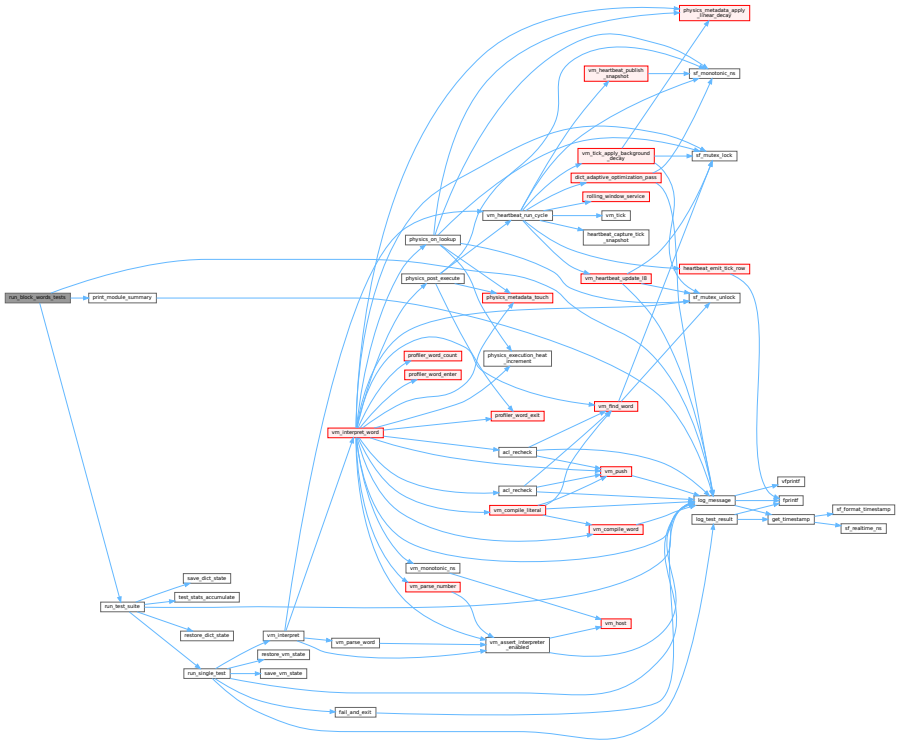
Parameters

vm	Pointer to the FORTH virtual machine instance

Runs through all test cases for block manipulation words, including block access, buffer management, and block I/O operations. Results are logged and summarized after completion.

References block_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.314.2 Variable Documentation

7.314.2.1 block_word_suites

```
WordTestSuite block_word_suites[]  [static]
```

Test suites for FORTH block words (Module 16)

Contains test cases for the following FORTH words:

- BLOCK: Block access
- BUFFER: Buffer management
- UPDATE: Block modification marking
- SAVE-BUFFERS: Buffer flushing

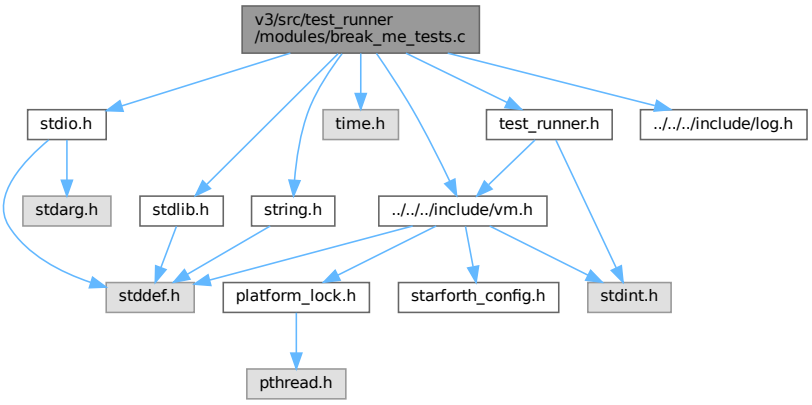
- EMPTY-BUFFERS: Buffer invalidation
- FLUSH: Buffer saving and invalidation
- LOAD: Block interpretation
- LIST: Block display
- SCR: Current block number
- THRU: Load block range
- -->: Continue to next block

Referenced by run_block_words_tests().

7.315 v3/src/test_runner/modules/break_me_tests.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "test_runner.h"
#include "../../../include/log.h"
#include "../../../include/vm.h"
```

Include dependency graph for break_me_tests.c:



Functions

- static int init_report (void)

  Initialize markdown report file.
- static void finalize_report (VM *vm)

  Finalize report with system info and easter egg.
- void run_break_me_tests (VM *vm)

  Main entry point for –break-me mode.

Variables

- static FILE * report_file = NULL
- static time_t test_start_time
- static time_t test_end_time
- static int total_tests_run = 0

7.315.1 Function Documentation

7.315.1.1 finalize_report()

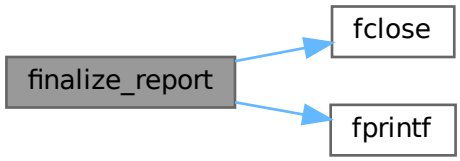
```
static void finalize_report (
            VM * vm )  [static]
```

Finalize report with system info and easter egg.

References fclose(), fprintf(), report_file, STACK_SIZE, test_end_time, test_start_time, total_tests_run, and VM_MEMORY_SIZE.

Referenced by run_break_me_tests().

Here is the call graph for this function:



Here is the caller graph for this function:

7.315.1.2 init_report()

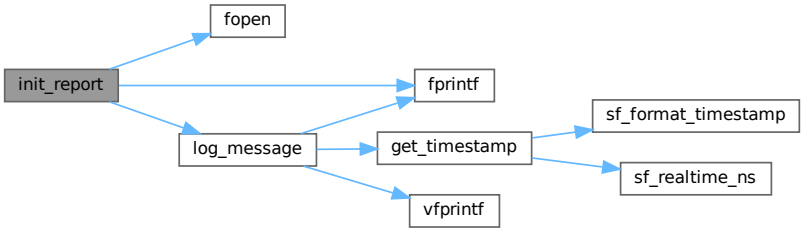
```
static int init_report (
            void  )  [static]
```

Initialize markdown report file.

References fopen(), fprintf(), LOG_ERROR, log_message(), report_file, and test_start_time.

Referenced by run_break_me_tests().

Here is the call graph for this function:



Here is the caller graph for this function:

7.315.1.3 run_break_me_tests()

```
void run_break_me_tests (
            VM * vm )
```

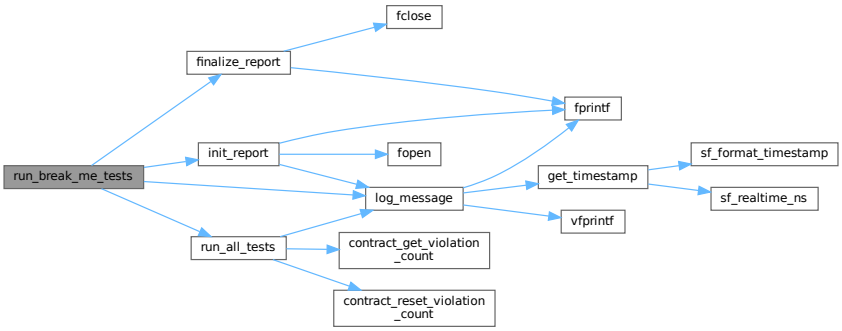
Main entry point for –break-me mode.

Run ultra-comprehensive break-me diagnostic suite.

References finalize_report(), global_test_stats, init_report(), LOG_ERROR, LOG_INFO, log_message(), run_all_tests(), TestStats::total_tests, and total_tests_run.

Referenced by run_optional_tests().

Here is the call graph for this function:



Here is the caller graph for this function:

7.315.2 Variable Documentation

7.315.2.1 report_file

```
FILE* report_file = NULL  [static]
```

Referenced by finalize_report(), and init_report().

7.315.2.2 test_end_time

```
time_t test_end_time  [static]
```

Referenced by finalize_report().

7.315.2.3 test_start_time

```
time_t test_start_time  [static]
```

Referenced by finalize_report(), and init_report().

7.315.2.4 total_tests_run

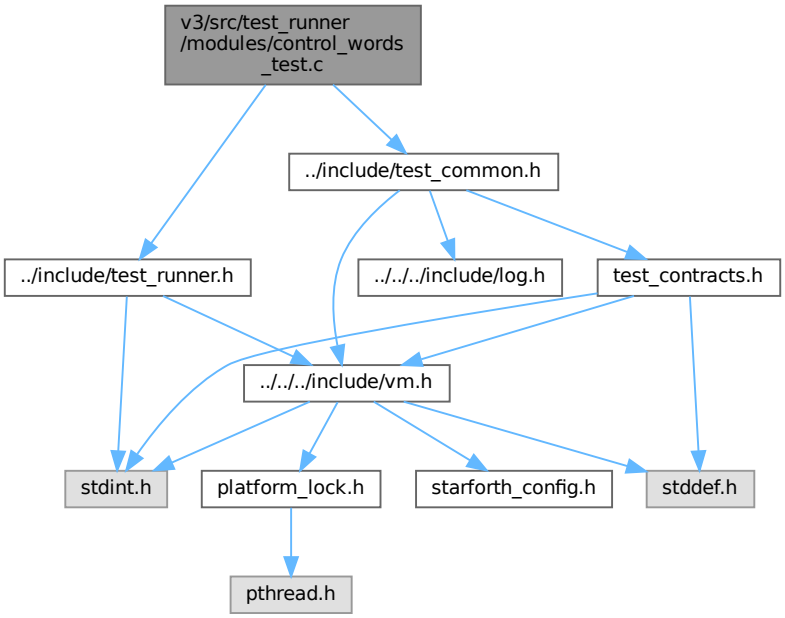
```
int total_tests_run = 0  [static]
```

Referenced by finalize_report(), and run_break_me_tests().

7.316 v3/src/test_runner/modules/control_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for control_words_test.c:



Functions

- void run_control_words_tests (VM *vm)

  Executes all control words test suites.

Variables

- static WordTestSuite control_word_suites [ ]

  Test suites for FORTH control words.

7.316.1 Function Documentation

7.316.1.1 run_control_words_tests()

```
void run_control_words_tests (
            VM * vm )
```

Executes all control words test suites.

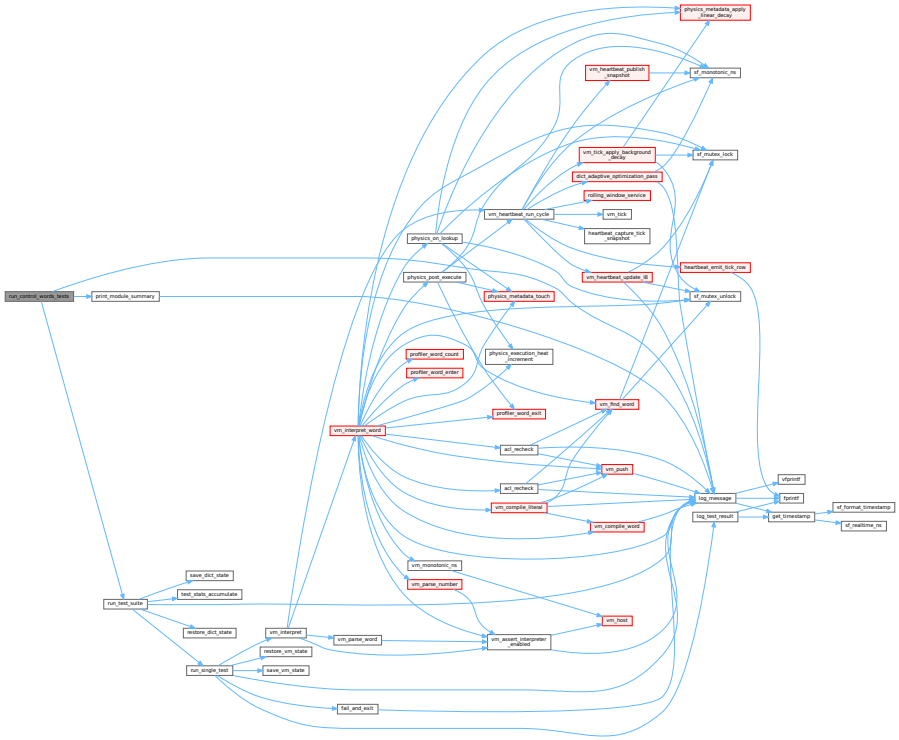
Parameters

vm	Pointer to the FORTH virtual machine instance

Runs through all defined test cases for control words, testing conditional branches and loop constructs

References control_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.316.2 Variable Documentation

7.316.2.1 control_word_suites

```
WordTestSuite control_word_suites[]  [static]
```

Test suites for FORTH control words.

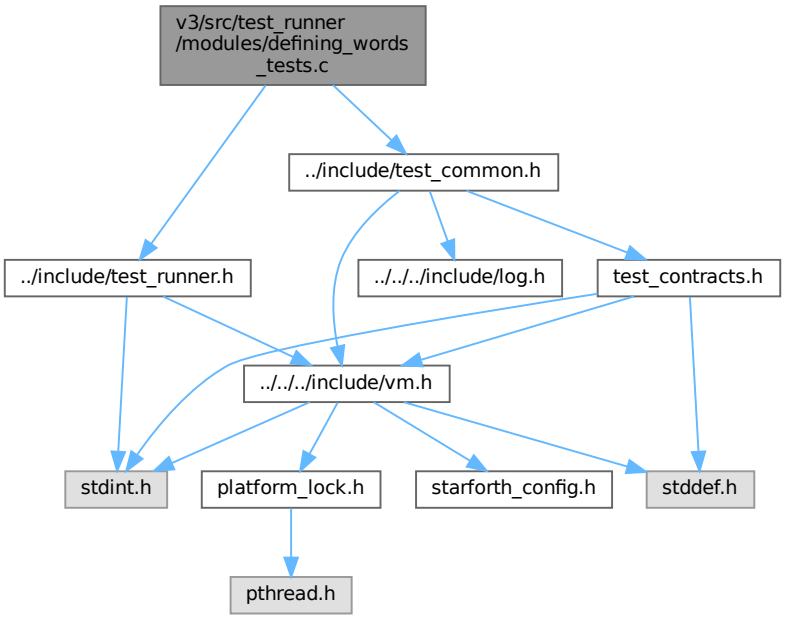
Contains test cases for all control flow words in Module 12, including conditional branches and loop structures

Referenced by run_control_words_tests().

7.317 v3/src/test_runner/modules/defining_words_tests.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for defining_words_tests.c:



Functions

- void run_defining_words_tests (VM *vm)

  Executes all test suites for Forth defining words.

Variables

- static WordTestSuite defining_word_suites [ ]

  Test suites for Forth defining words (Module 13)

7.317.1 Function Documentation

7.317.1.1 run_defining_words_tests()

```
void run_defining_words_tests (
            VM * vm )
```

Executes all test suites for Forth defining words.

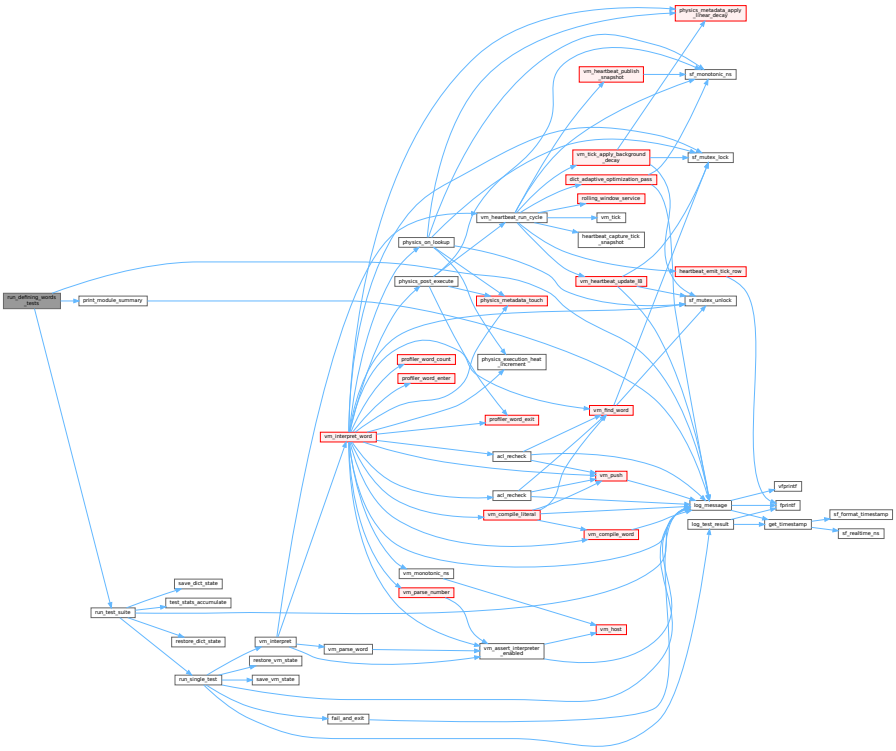
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through each test suite in defining_word_suites array, executing individual test cases and logging results_↩ run_01_2025_12_08

References defining_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.317.2 Variable Documentation

7.317.2.1 defining_word_suites

```
WordTestSuite defining_word_suites[]  [static]
```

Test suites for Forth defining words (Module 13)

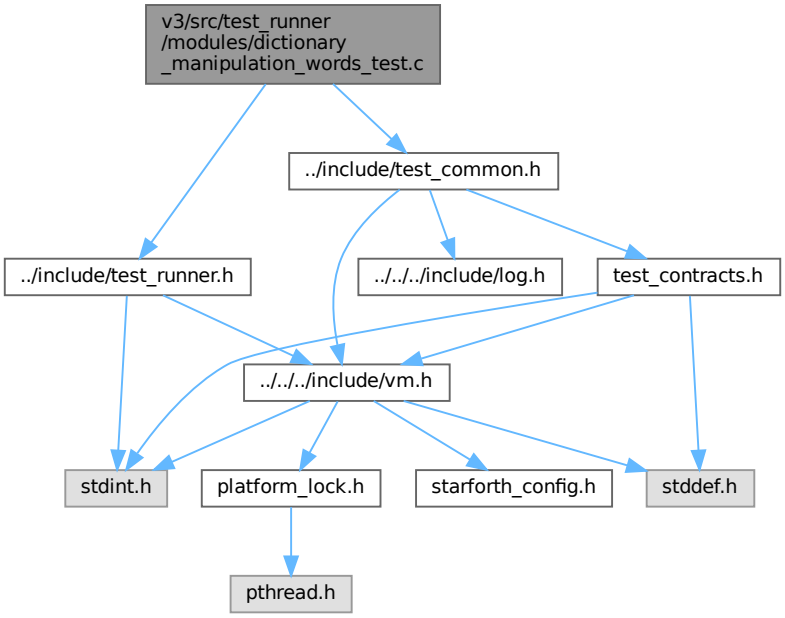
Contains test cases for core defining words including: :, ;, CONSTANT, VARIABLE, CREATE, DOES>, [, and]

Referenced by run_defining_words_tests().

7.318 v3/src/test_runner/modules/dictionary_manipulation_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for dictionary_manipulation_words_test.c:



Functions

- void run_dictionary_manipulation_words_tests (VM *vm)

  Executes all dictionary manipulation words test suites.

Variables

- static WordTestSuite dict_manip_word_suites [ ]

7.318.1 Function Documentation

7.318.1.1 run_dictionary_manipulation_words_tests()

```
void run_dictionary_manipulation_words_tests (
            VM * vm )
```

Executes all dictionary manipulation words test suites.

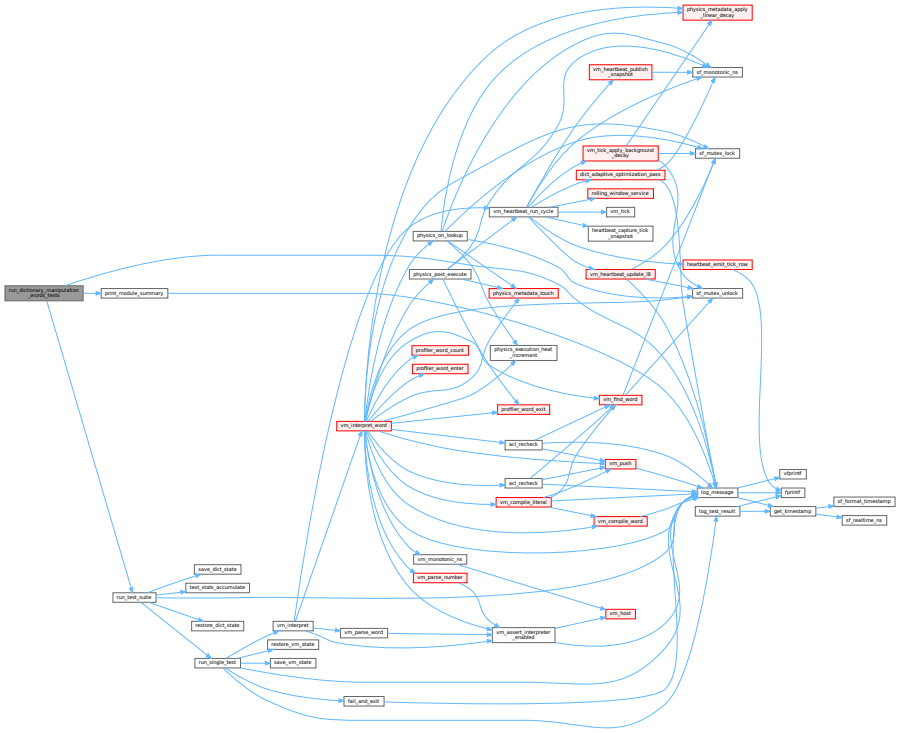
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all test cases for CREATE, FORGET, IMMEDIATE, FIND, DEFINITIONS, SMUDGE, LATEST, >BODY, and HIDDEN words

References dict_manip_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.318.2 Variable Documentation

7.318.2.1 dict_manip_word_suites

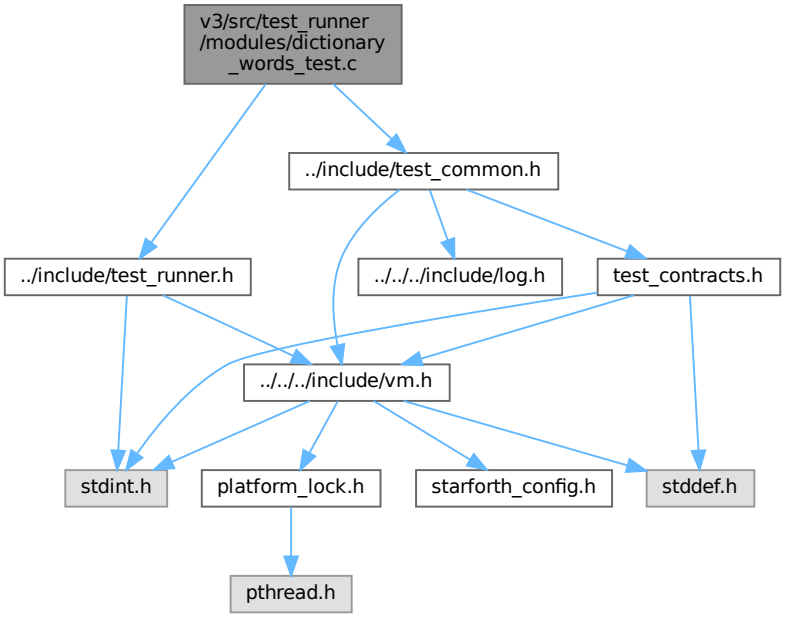
```
WordTestSuite dict_manip_word_suites[]  [static]
```

Referenced by run_dictionary_manipulation_words_tests().

7.319 v3/src/test_runner/modules/dictionary_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for dictionary_words_test.c:



Functions

- void run_dictionary_words_tests (VM *vm)

  Executes all dictionary word test suites.

Variables

- static WordTestSuite dictionary_word_suites [ ]

7.319.1 Function Documentation

7.319.1.1 run_dictionary_words_tests()

```
void run_dictionary_words_tests (
            VM * vm )
```

Executes all dictionary word test suites.

Parameters

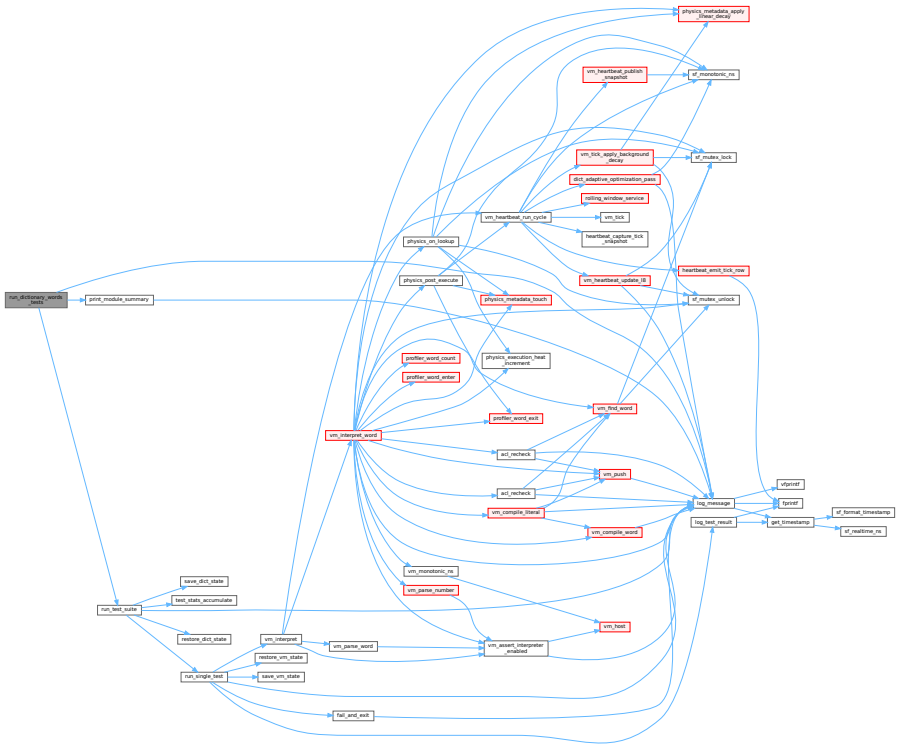
vm	Pointer to the FORTH virtual machine instance

This function runs through all dictionary word test suites, including tests for:

- HERE (dictionary pointer)
- ALLOT (memory allocation)
- Comma operations (, C, 2,)
- PAD (scratch pad area)
- Stack pointer operations (SP@ SP!)
- LATEST (most recent dictionary entry)
- Compilation sequences
- Stack management operations

References dictionary_word_suites, LOG_INFO, log_message(), print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.319.2 Variable Documentation

7.319.2.1 dictionary_word_suites

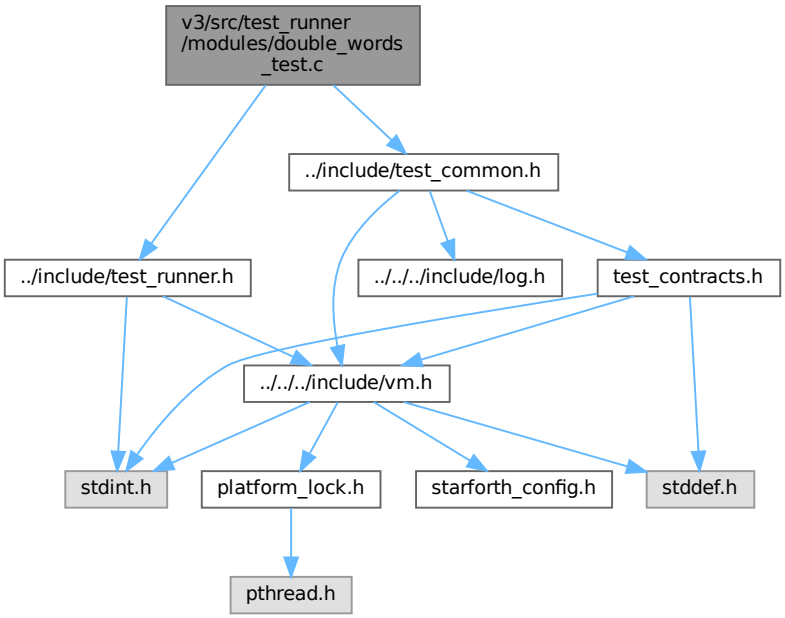
```
WordTestSuite dictionary_word_suites[]  [static]
```

Referenced by run_dictionary_words_tests().

7.320 v3/src/test_runner/modules/double_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for double_words_test.c:



Functions

- void run_double_words_tests (VM *vm)

  Executes all test suites for double number words.

Variables

- static WordTestSuite double_word_suites [ ]

7.320.1 Function Documentation

7.320.1.1 run_double_words_tests()

```
void run_double_words_tests (
            VM * vm )
```

Executes all test suites for double number words.

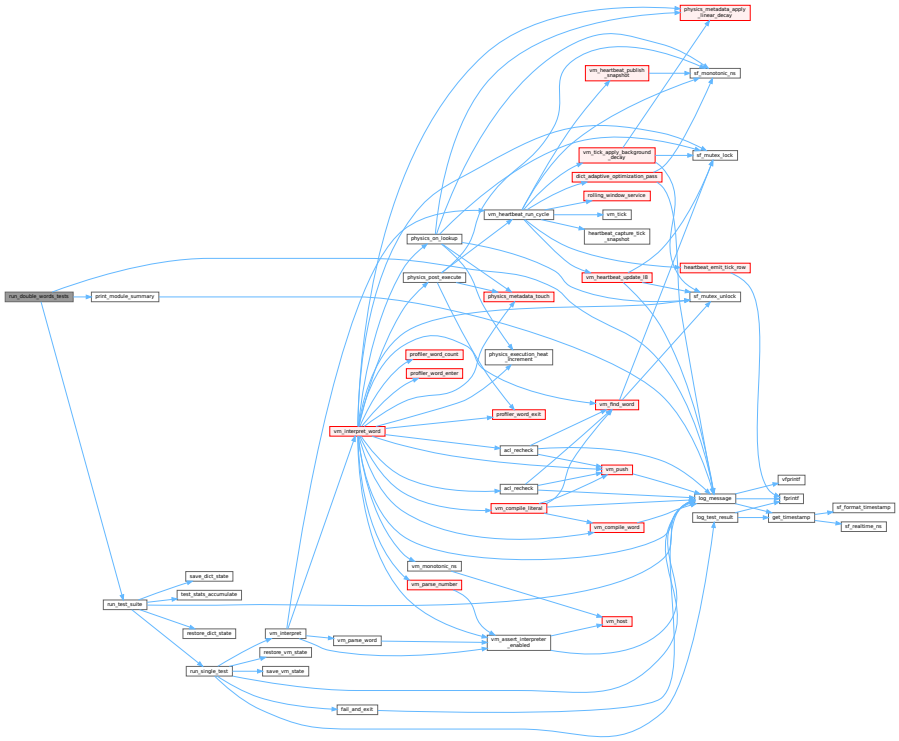
This function runs comprehensive tests for all double number word operations, including stack manipulation (2DROP, 2DUP, etc.), arithmetic operations (D+, D-, etc.), and comparison operations (D=, D<, etc.).

Parameters

vm	Pointer to the virtual machine instance where tests will be executed

References double_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.320.2 Variable Documentation

7.320.2.1 double_word_suites

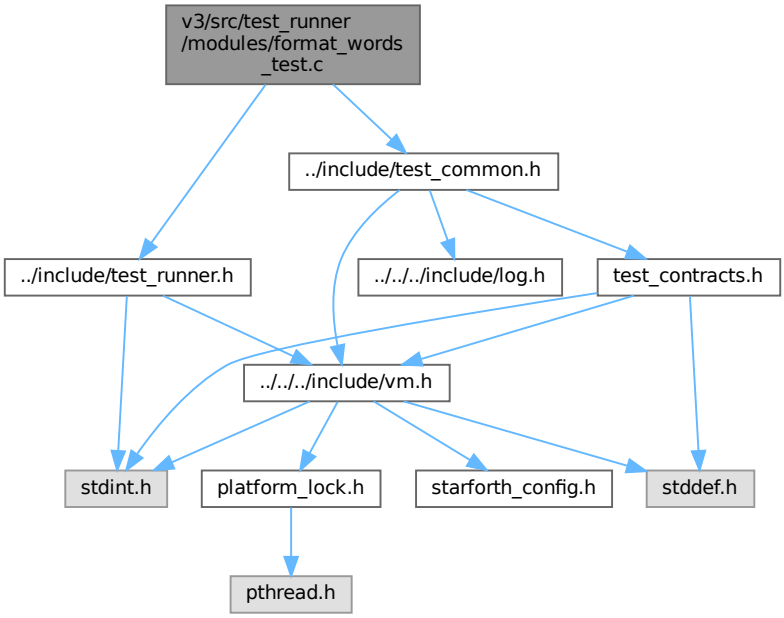
```
WordTestSuite double_word_suites[]  [static]
```

Referenced by run_double_words_tests().

7.321 v3/src/test_runner/modules/format_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for format_words_test.c:



Functions

- void run_format_words_tests (VM *vm)

  Executes all format words test suites.

Variables

- static WordTestSuite format_word_suites [ ]

  Test suites for format words (Module 10) Contains test cases for number base conversion and formatting operations.

7.321.1 Function Documentation

7.321.1.1 run_format_words_tests()

```
void run_format_words_tests (
            VM * vm )
```

Executes all format words test suites.

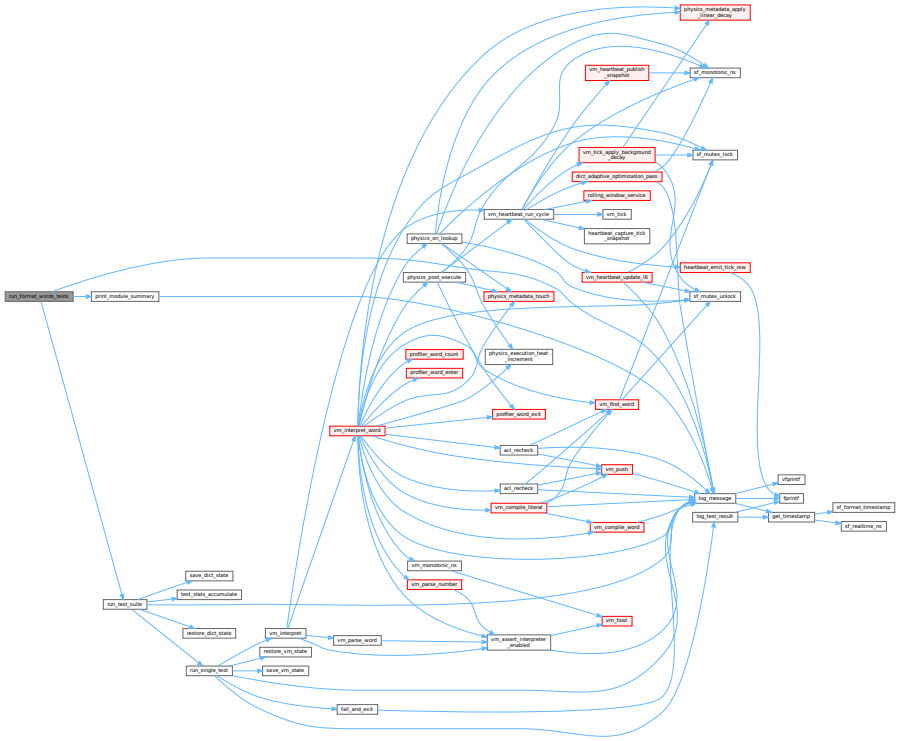
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all test cases for number base manipulation and formatting words

References format_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.321.2 Variable Documentation

7.321.2.1 format_word_suites

```
WordTestSuite format_word_suites[]  [static]
```

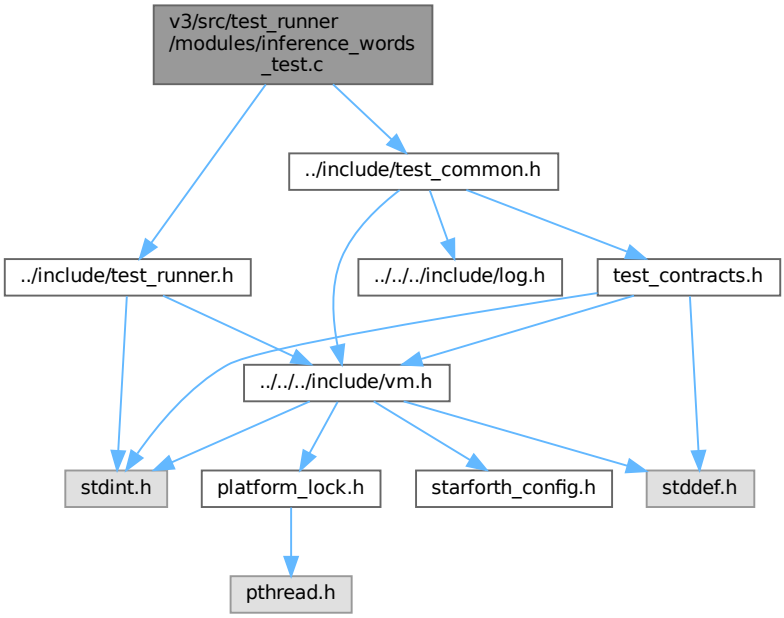
Test suites for format words (Module 10) Contains test cases for number base conversion and formatting operations.

Referenced by run_format_words_tests().

7.322 v3/src/test_runner/modules/inference_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for inference_words_test.c:



Functions

- void run_inference_words_tests (VM *vm)

  Executes the proof-covered subset of inference-engine word test suites.

Variables

- static WordTestSuite inference_word_suites [ ]

7.322.1 Function Documentation

7.322.1.1 run_inference_words_tests()

```
void run_inference_words_tests (
            VM * vm )
```

Executes the proof-covered subset of inference-engine word test suites.

Parameters

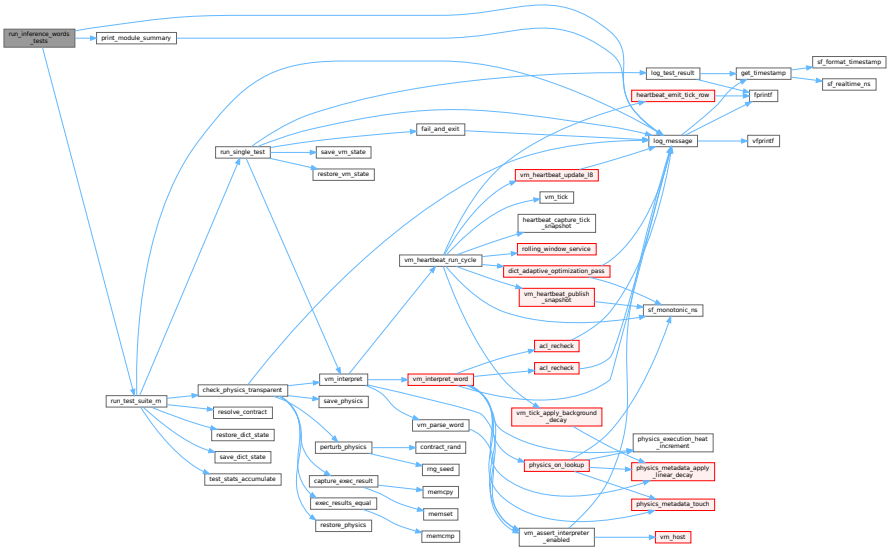
vm	Pointer to the Forth virtual machine instance

Covers the 8 words proof/COVERAGE.md marks as proof-covered in inference_words.c: the three array-based primitives (Q.VARIANCE, INFER-DECAY-SLOPE, INFER-WINDOW-WIDTH), INFER-RUN, and the four *-@ output

accessors (INFER-WINDOW@/DECAY@/VARIANCE@/FIT@) plus INFER-EARLY-EXIT. Does not cover the L8 Jacquard or Bayesian posterior words in the same file — not claimed as proof-covered.

References CONTRACT_NONE, inference_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.322.2 Variable Documentation

7.322.2.1 inference_word_suites

```
WordTestSuite inference_word_suites[]  [static]
```

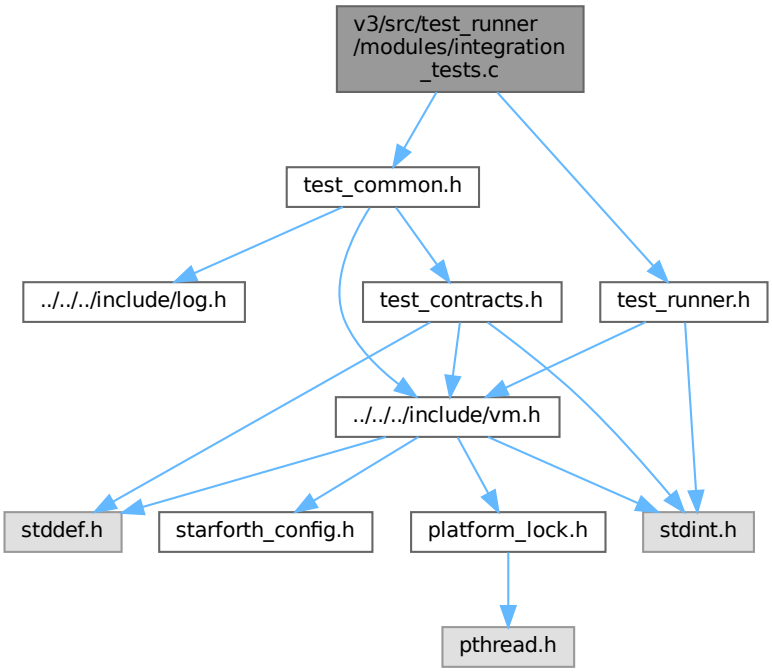
Referenced by run_inference_words_tests().

7.323 v3/src/test_runner/modules/integration_tests.c File Reference

Full program integration tests for StarForth.

```
#include "test_common.h"
#include "test_runner.h"
```

Include dependency graph for integration_tests.c:



Functions

- void run_integration_tests (VM *vm)

  Executes all integration test suites for StarForth.

Variables

- static WordTestSuite integration_suites [ ]

7.323.1 Detailed Description

Full program integration tests for StarForth.

This file contains integration tests that verify complete Forth programs as regression tests. It includes test suites for complete programs, data structures, and algorithms implementations.

7.323.2 Function Documentation

7.323.2.1 run_integration_tests()

```
void run_integration_tests (
            VM * vm )
```

Executes all integration test suites for StarForth.

Parameters

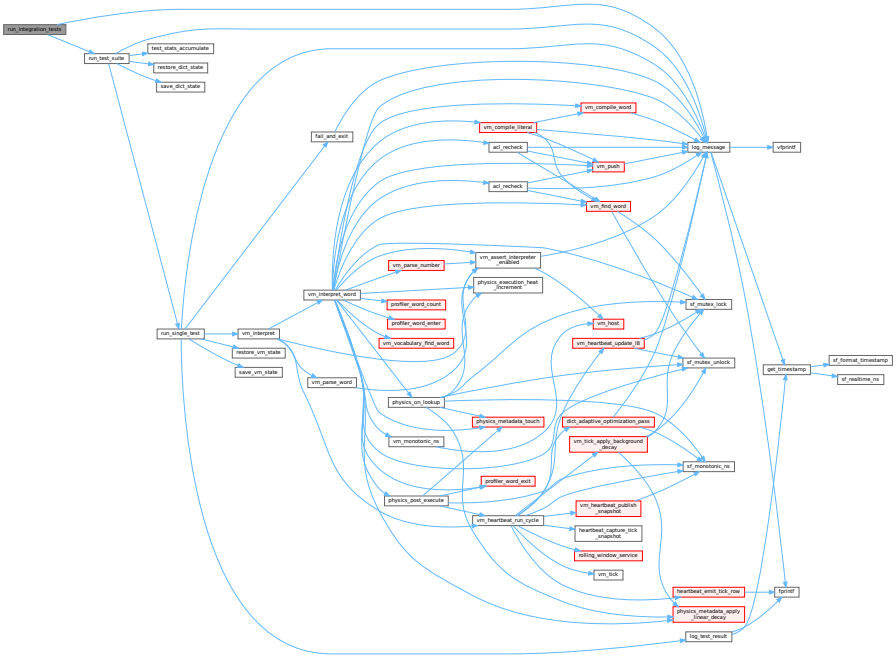
vm	Pointer to the Forth virtual machine instance

This function runs through all defined integration test suites, including:

- Complete Programs suite (prime checker, array operations, etc)
- Data Structures suite (linked list, min/max tracker)
- Algorithms suite (sorting, GCD)

References integration_suites, LOG_INFO, log_message(), and run_test_suite().

Here is the call graph for this function:



7.323.3 Variable Documentation

7.323.3.1 integration_suites

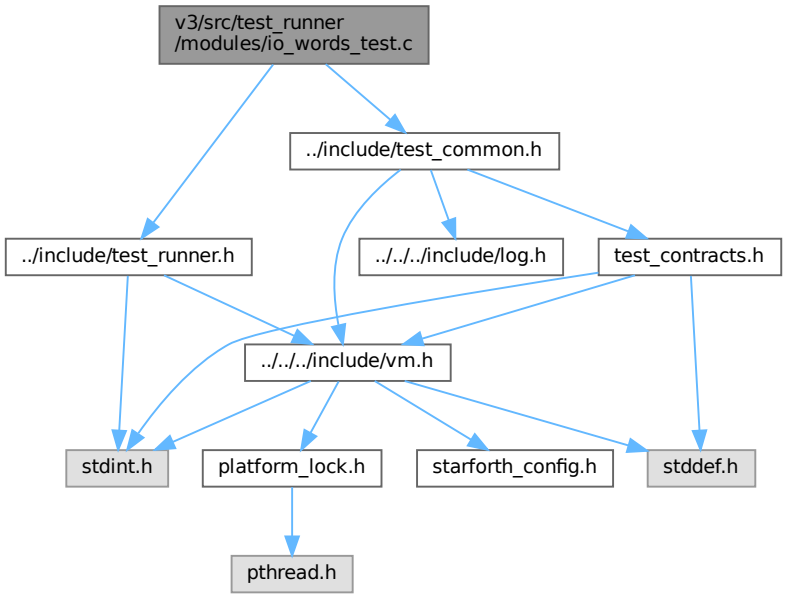
```
WordTestSuite integration_suites[]  [static]
```

Referenced by run_integration_tests().

7.324 v3/src/test_runner/modules/io_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for io_words_test.c:



Functions

- void run_io_words_tests (VM *vm)

  Executes all I/O words test suites.

Variables

- static WordTestSuite io_word_suites [ ]

  Test suites for I/O words (Module 9)

7.324.1 Function Documentation

7.324.1.1 run_io_words_tests()

```
void run_io_words_tests (
            VM * vm )
```

Executes all I/O words test suites.

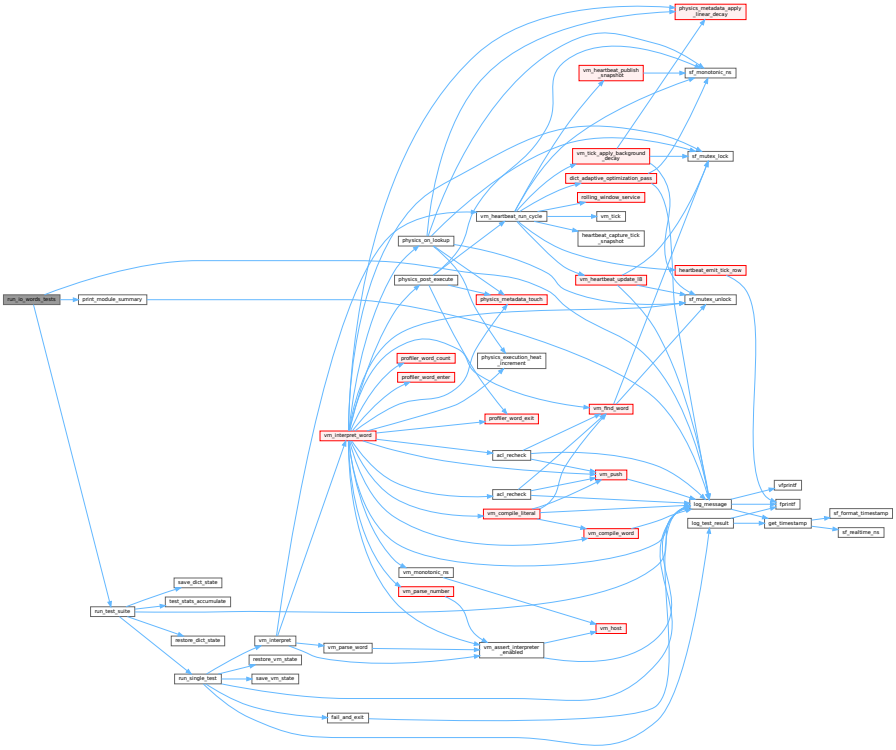
Parameters

vm	Pointer to the FORTH virtual machine instance

Runs through all defined test suites for I/O words, logging results_run_01_2025_12_08 and providing a summary of the test execution

References io_word_suites, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.324.2 Variable Documentation

7.324.2.1 io_word_suites

```
WordTestSuite io_word_suites[]  [static]
```

Test suites for I/O words (Module 9)

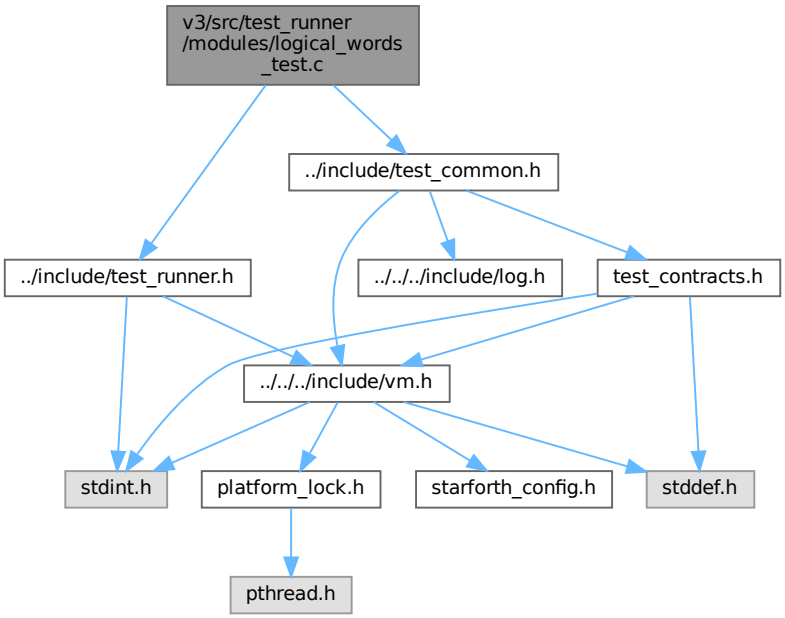
Contains comprehensive test cases for all I/O related FORTH words including output formatting, character I/O, and base conversion operations

Referenced by run_io_words_tests().

7.325 v3/src/test_runner/modules/logical_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for logical_words_test.c:



Functions

- void run_logical_words_tests (VM *vm)

  Executes all logical words test suites.

Variables

- static WordTestSuite logical_word_suites [ ]

7.325.1 Function Documentation

7.325.1.1 run_logical_words_tests()

```
void run_logical_words_tests (
            VM * vm )
```

Executes all logical words test suites.

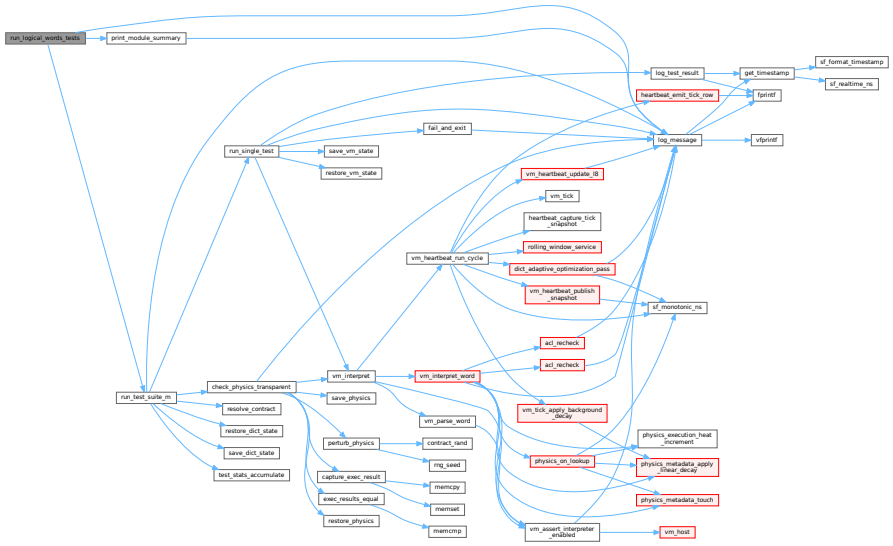
This function runs comprehensive tests for all FORTH logical and comparison operators, including stack underflow checks and edge cases.

Parameters

vm	Pointer to the FORTH virtual machine instance

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, logical_word_suites, print_module_summary(), run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.325.2 Variable Documentation

7.325.2.1 logical_word_suites

```
WordTestSuite logical_word_suites[]  [static]
```

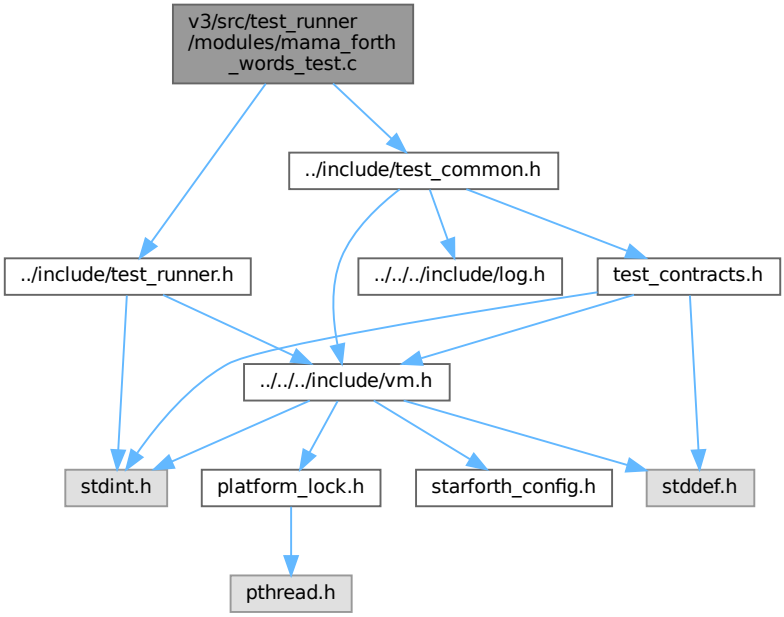
Referenced by run_logical_words_tests().

7.326 v3/src/test_runner/modules/mama_forth_words_test.c File Reference

Test module for Mama FORTH vocabulary (M7.1 Capsule System)

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for mama_forth_words_test.c:



Macros

- #define MAMA_IMPLEMENTED 0

Functions

- void run_mama_forth_words_tests (VM *vm)

  Executes all Mama FORTH vocabulary test suites.

Variables

- static WordTestSuite mama_word_suites [ ]

7.326.1 Detailed Description

Test module for Mama FORTH vocabulary (M7.1 Capsule System)

Tests for the MAMA vocabulary words. These words are kernel-only (**STARKERNEL**), so in hosted builds all tests are marked as not implemented and will be skipped.

In kernel builds, these tests validate:

- Capsule directory enumeration
- Descriptor field access
- VM birth protocol
- Experiment execution

7.326.2 Macro Definition Documentation

7.326.2.1 MAMA_IMPLEMENTED

```
#define MAMA_IMPLEMENTED 0
```

7.326.3 Function Documentation

7.326.3.1 run_mama_forth_words_tests()

```
void run_mama_forth_words_tests (
            VM * vm )
```

Executes all Mama FORTH vocabulary test suites.

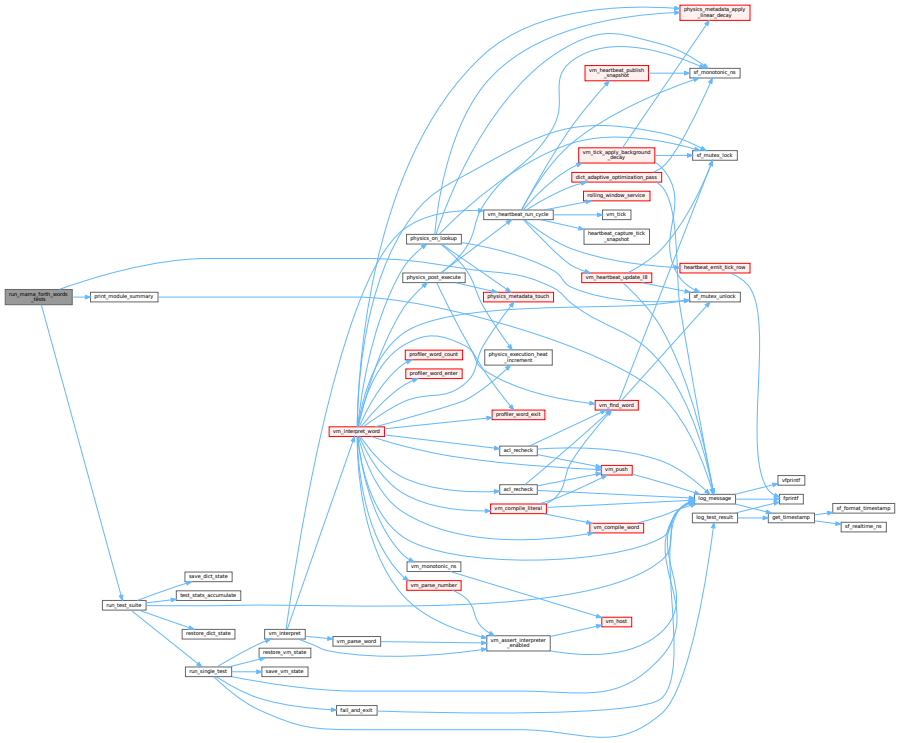
In hosted builds, all tests will be skipped since MAMA_IMPLEMENTED=0. In kernel builds, tests will execute against the capsule system.

Parameters

vm	Pointer to the Forth virtual machine instance

References LOG_INFO, log_message(), mama_word_suites, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.326.4 Variable Documentation

7.326.4.1 mama_word_suites

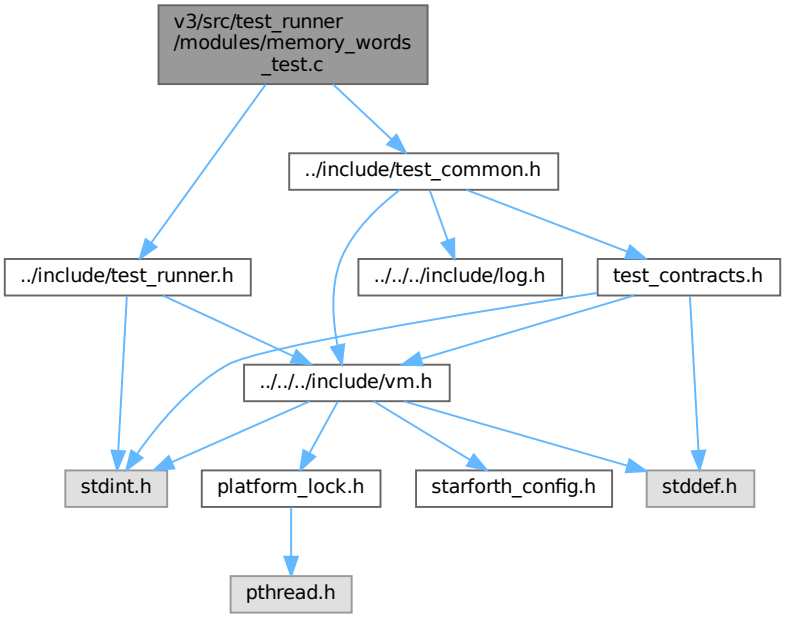
```
WordTestSuite mama_word_suites[]  [static]
```

Referenced by run_mama_forth_words_tests().

7.327 v3/src/test_runner/modules/memory_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for memory_words_test.c:



Functions

- void run_memory_words_tests (VM *vm)

  Executes the test suite for memory manipulation words.

Variables

- static WordTestSuite memory_word_suites [ ]

7.327.1 Function Documentation

7.327.1.1 run_memory_words_tests()

```
void run_memory_words_tests (
            VM * vm )
```

Executes the test suite for memory manipulation words.

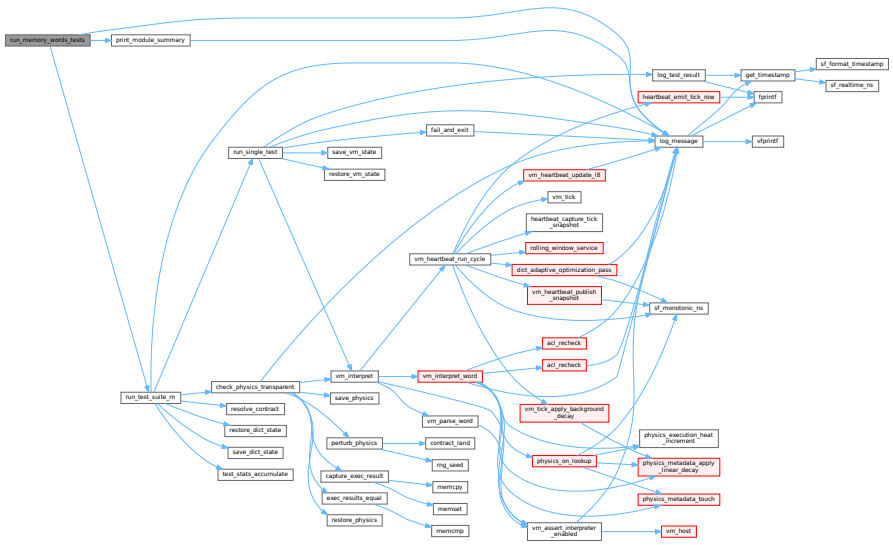
Runs through a series of test cases for Forth memory operations including store, fetch, allot, and other memory-related words

Parameters

vm	Pointer to the Forth virtual machine instance

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, memory_word_suites, print_module_summary(), run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.327.2 Variable Documentation

7.327.2.1 memory_word_suites

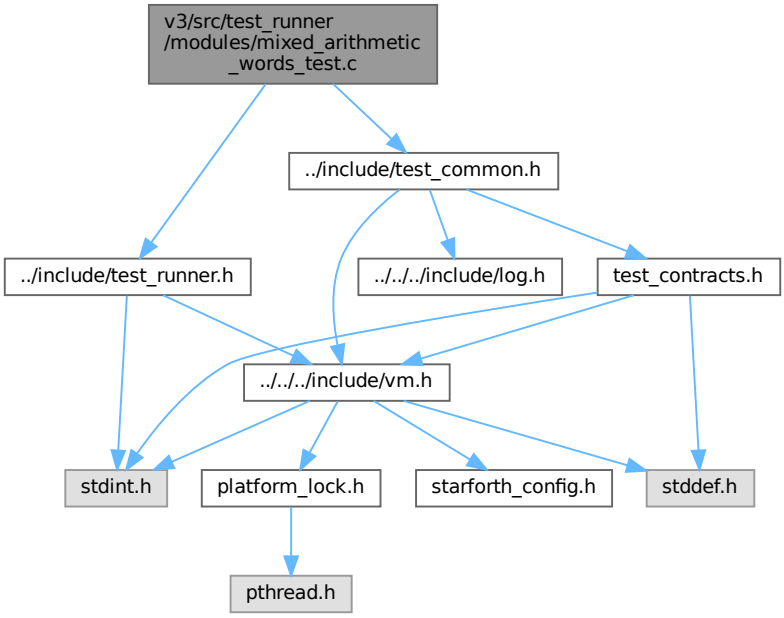
```
WordTestSuite memory_word_suites[]  [static]
```

Referenced by run_memory_words_tests().

7.328 v3/src/test_runner/modules/mixed_arithmetic_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for mixed_arithmetic_words_test.c:



Functions

- void run_mixed_arithmetic_words_tests (VM *vm)

  Module 7 entry point — run all mixed-arithmetic word POST tests.

Variables

- static WordTestSuite mixed_arithmetic_word_suites [ ]

  Module 7 test-suite table for mixed-arithmetic FORTH-79 words.

7.328.1 Function Documentation

7.328.1.1 run_mixed_arithmetic_words_tests()

```
void run_mixed_arithmetic_words_tests (
            VM * vm )
```

Module 7 entry point — run all mixed-arithmetic word POST tests.

Iterates `mixed_arithmetic_word_suites` and delegates each to `run_test_suite()`. Covers five word families:

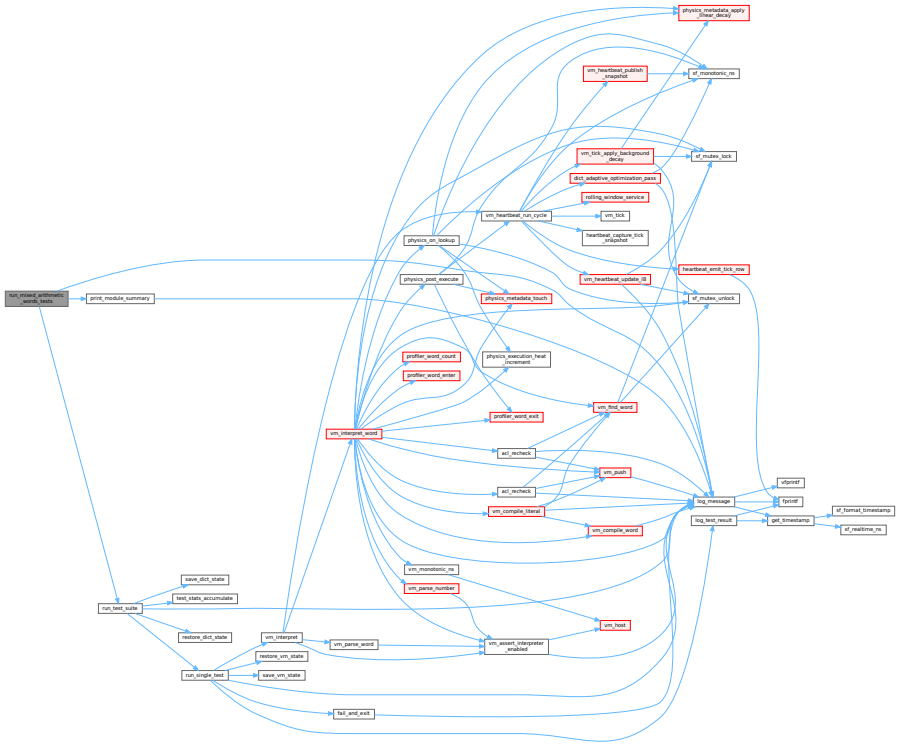
- `*-/` (n1 n2 n3 – n4) — scale: n1*n2/n3 in 64-bit intermediate
- `*-/MOD` (n1 n2 n3 – rem quot) — scale with remainder
- `M+` (d n – d') — add single to double
- `M*` (n1 n2 – d) — signed multiply to double
- `M/MOD` (d n – rem quot) — double divided by single

Parameters

vm	VM instance; must be initialised with the standard vocabulary.

References LOG_INFO, log_message(), LOG_TEST, mixed_arithmetic_word_suites, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.328.2 Variable Documentation

7.328.2.1 mixed_arithmetic_word_suites

```
WordTestSuite mixed_arithmetic_word_suites[]  [static]
```

Module 7 test-suite table for mixed-arithmetic FORTH-79 words.

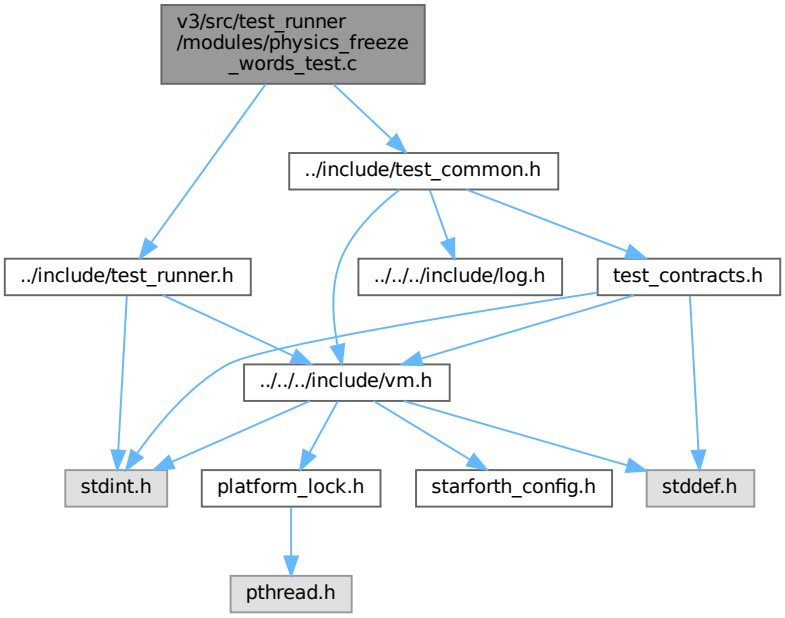
Covers the five combined multiply-divide words: `*-/` , `*-/MOD` , `M+` , `M*` , and `M/MOD` . Each suite exercises normal cases, edge cases (overflow, large products), and error cases (division by zero, stack underflow). The table is terminated with a NULL sentinel.

Referenced by run_mixed_arithmetic_words_tests().

7.329 v3/src/test_runner/modules/physics_freeze_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for physics_freeze_words_test.c:



Functions

- void run_physics_freeze_words_tests (VM *vm)

  Executes the proof-covered subset of physics-freeze word test suites.

Variables

- static WordTestSuite physics_freeze_word_suites [ ]

7.329.1 Function Documentation

7.329.1.1 run_physics_freeze_words_tests()

```
void run_physics_freeze_words_tests (
            VM * vm )
```

Executes the proof-covered subset of physics-freeze word test suites.

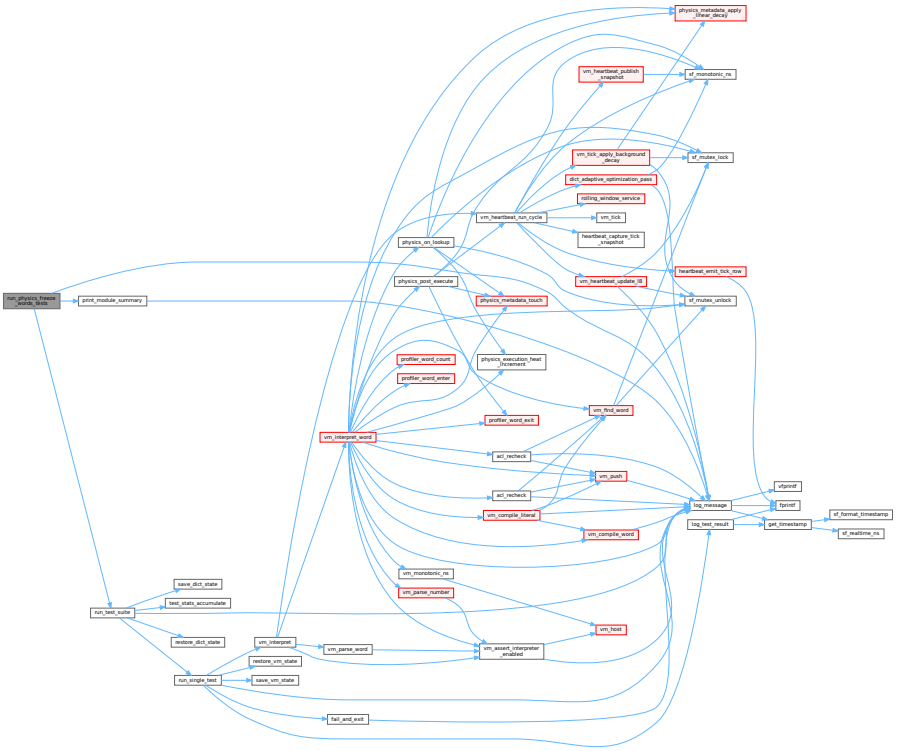
Parameters

vm	Pointer to the Forth virtual machine instance

Covers the 6 words proof/StarForth_Physics_Freeze_Words.thy gives real lemmas for: FREEZE-WORD, UNFREEZE-WORD, FROZEN?, HEAT!, HEAT@, DECAY-RATE. Does not cover FREEZE-CRITICAL, SHOW-HEAT, or ALL-HEATS — those carry only trivial/not-modelled lemmas in the same theory file.

References LOG_INFO, log_message(), LOG_TEST, physics_freeze_word_suites, print_module_summary(), run_test_suite(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.329.2 Variable Documentation

7.329.2.1 physics_freeze_word_suites

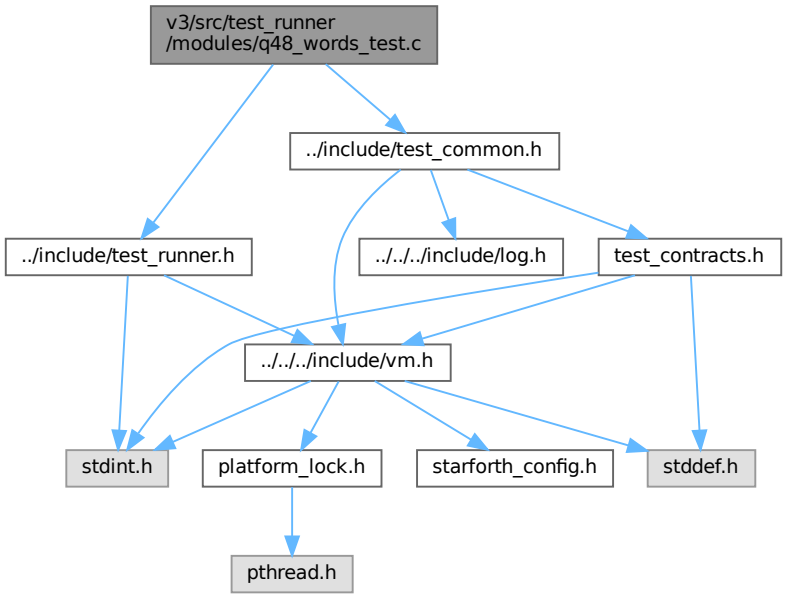
```
WordTestSuite physics_freeze_word_suites[]  [static]
```

Referenced by run_physics_freeze_words_tests().

7.330 v3/src/test_runner/modules/q48_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for q48_words_test.c:



Functions

- void run_q48_words_tests (VM *vm)

  Executes all Q48.16 fixed-point word test suites.

Variables

- static WordTestSuite q48_word_suites [ ]

7.330.1 Function Documentation

7.330.1.1 run_q48_words_tests()

```
void run_q48_words_tests (
            VM * vm )
```

Executes all Q48.16 fixed-point word test suites.

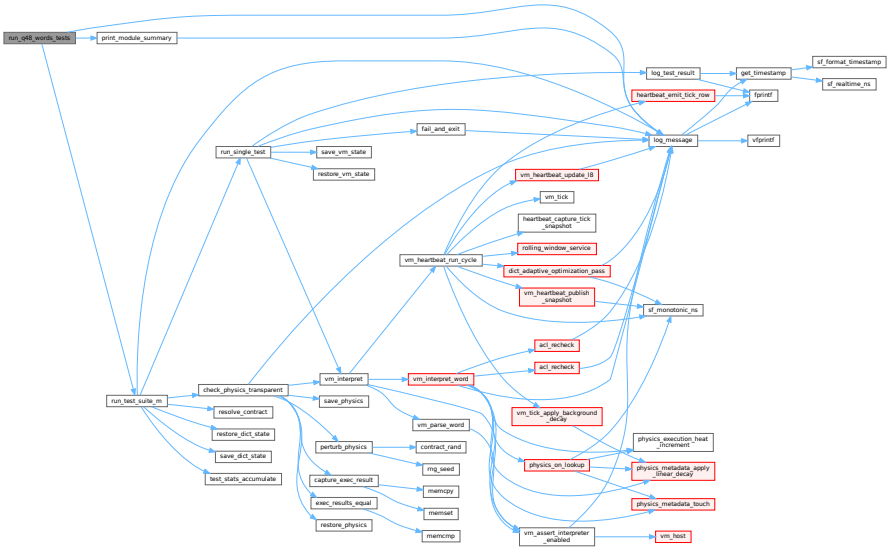
Parameters

vm	Pointer to the Forth virtual machine instance

Covers all 23 words registered by register_q48_words() (q48_words.c): arithmetic (+, -, ∗, /), sign (ABS, NEG), transcendental approximations (LOG, EXP, SQRT, SIN, COS), conversions (FROM-INT, TO-INT), constants (1, 0, SCALE), comparisons (=, <, >, 0=, MAX, MIN), and PRINT.

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), q48_word_suites, run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.330.2 Variable Documentation

7.330.2.1 q48_word_suites

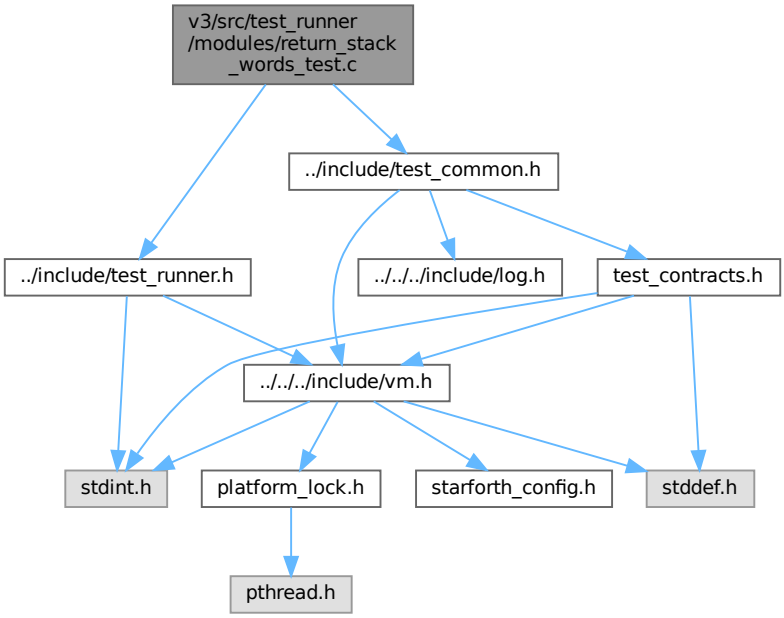
```
WordTestSuite q48_word_suites[]  [static]
```

Referenced by run_q48_words_tests().

7.331 v3/src/test_runner/modules/return_stack_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for return_stack_words_test.c:



Functions

- void run_return_stack_words_tests (VM *vm)

  Executes all test suites for return stack manipulation words.

Variables

- static WordTestSuite return_stack_word_suites [ ]

  Test suites for return stack manipulation words.

7.331.1 Function Documentation

7.331.1.1 run_return_stack_words_tests()

```
void run_return_stack_words_tests (
            VM * vm )
```

Executes all test suites for return stack manipulation words.

Parameters

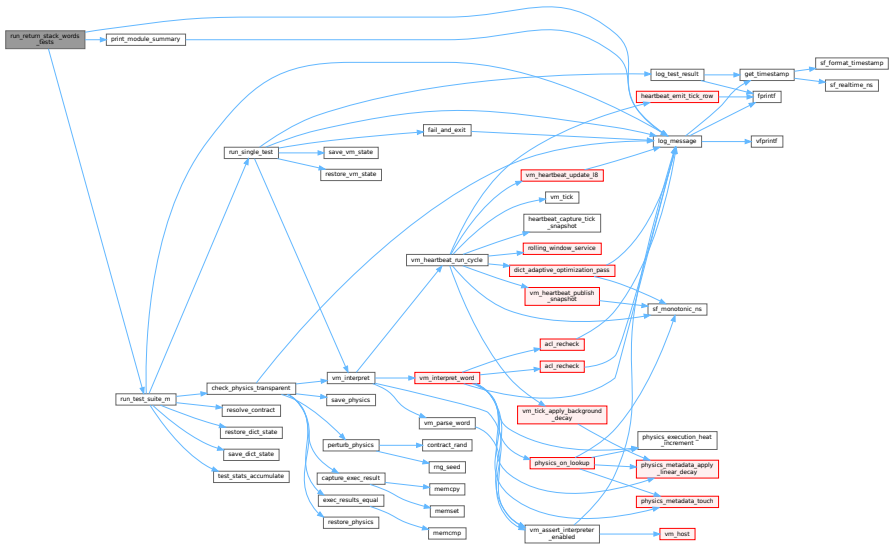
vm	Pointer to the Forth virtual machine instance

Runs through all defined test cases for >R, R>, and R@ words and reports results_run_01_2025_12_08 through

the logging system

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), return_stack_word_suites, run_test_suite_m(), and WordTestSuite::word_name.

Here is the call graph for this function:



7.331.2 Variable Documentation

7.331.2.1 return_stack_word_suites

```
WordTestSuite return_stack_word_suites[]  [static]
```

Test suites for return stack manipulation words.

Contains test cases for >R (push to return stack), R> (pop from return stack), and R@ (copy top of return stack)

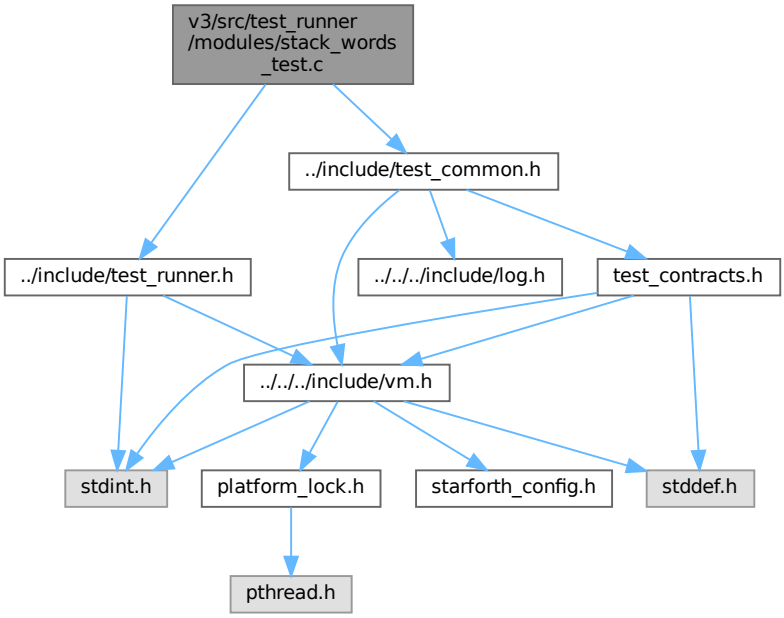
Referenced by run_return_stack_words_tests().

7.332 v3/src/test_runner/modules/stack_words_test.c File Reference

Test module for Forth stack manipulation words.

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for stack_words_test.c:



Functions

- void run_stack_words_tests (VM *vm)

  Executes all test suites for stack manipulation words.

Variables

- static WordTestSuite stack_word_suites [ ]

  Test suites for stack manipulation words.

7.332.1 Detailed Description

Test module for Forth stack manipulation words.

Contains test suites for basic stack operations including DUP, DROP, SWAP, OVER, ROT, DEPTH, PICK, and ROLL

7.332.2 Function Documentation

7.332.2.1 run_stack_words_tests()

```
void run_stack_words_tests (
            VM * vm )
```

Executes all test suites for stack manipulation words.

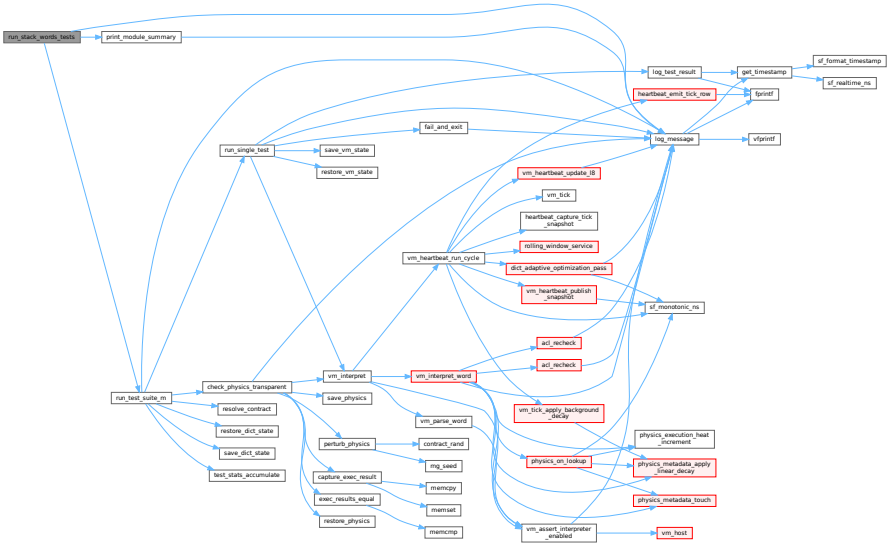
Parameters

vm	Pointer to the Forth virtual machine instance

Runs through all defined test cases for stack manipulation words and reports results_run_01_2025_12_08 through the logging system

References CONTRACT_PHYSICS_TRANSPARENT, LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite_m(), stack_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.332.3 Variable Documentation

7.332.3.1 stack_word_suites

```
WordTestSuite stack_word_suites[]  [static]
```

Test suites for stack manipulation words.

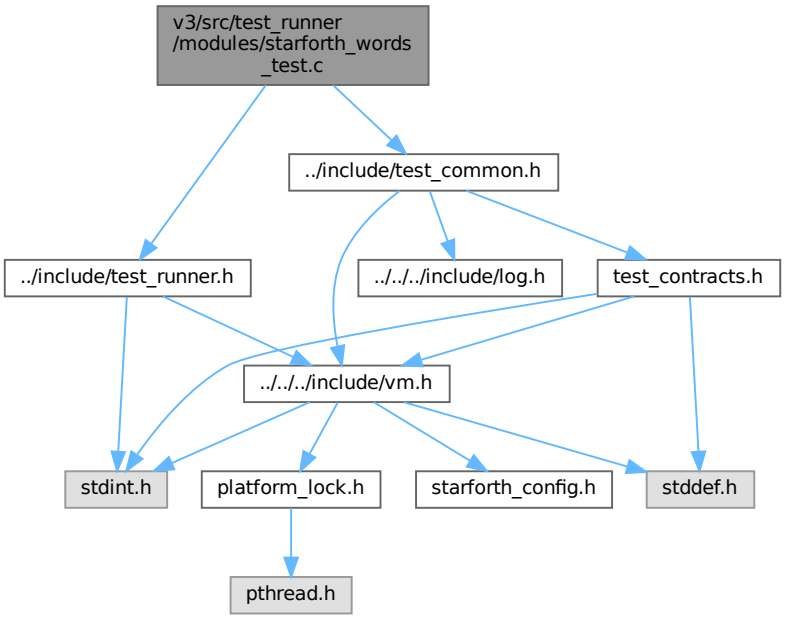
Contains comprehensive test cases for all standard stack operations, including normal cases, edge cases, and error conditions

Referenced by run_stack_words_tests().

7.333 v3/src/test_runner/modules/starforth_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for starforth_words_test.c:



Functions

- void run_starforth_words_tests (VM *vm)

  Executes all StarForth words test suites.

Variables

- static WordTestSuite starforth_word_suites [ ]

  Test suites for StarForth-specific words.

7.333.1 Function Documentation

7.333.1.1 run_starforth_words_tests()

```
void run_starforth_words_tests (
            VM * vm )
```

Executes all StarForth words test suites.

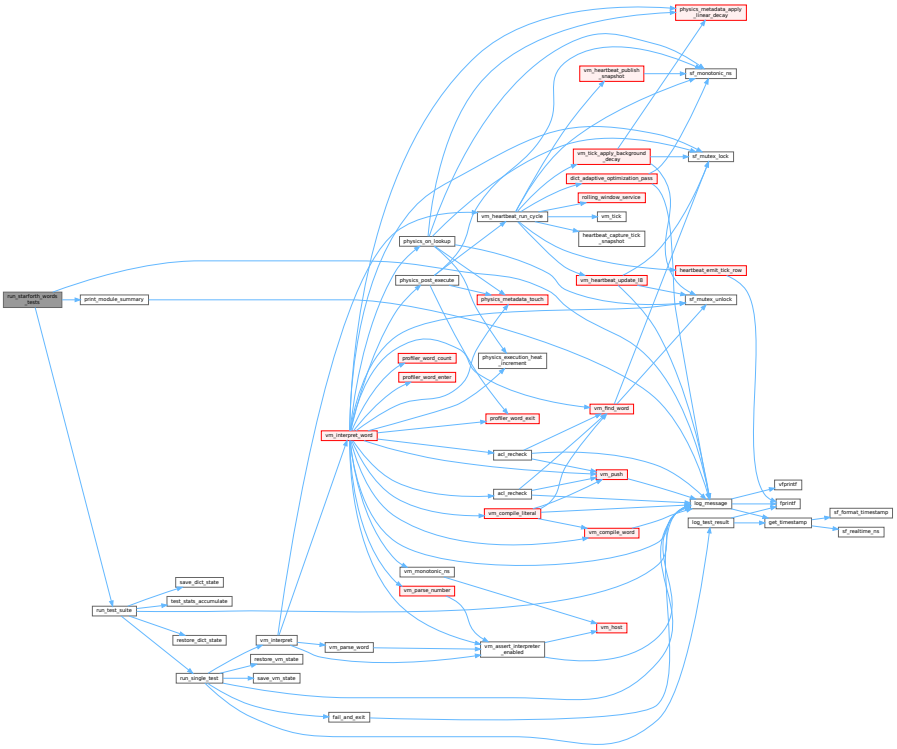
Parameters

vm	Pointer to the FORTH virtual machine instance

Runs through all test cases for StarForth-specific implementation words, including entropy tracking, comments, and initialization.

References LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), starforth_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.333.2 Variable Documentation

7.333.2.1 starforth_word_suites

```
WordTestSuite starforth_word_suites[]  [static]
```

Test suites for StarForth-specific words.

Contains test cases for the following StarForth implementation words:

- ENTROPY@: Fetch word execution count
- ENTROPY!: Store word execution count
- WORD-ENTROPY: Get execution count for a word by name
- RESET-ENTROPY: Reset all word execution counts
- TOP-WORDS: Display most frequently executed words
- (-: Comment to end of line
- INIT: System initialization

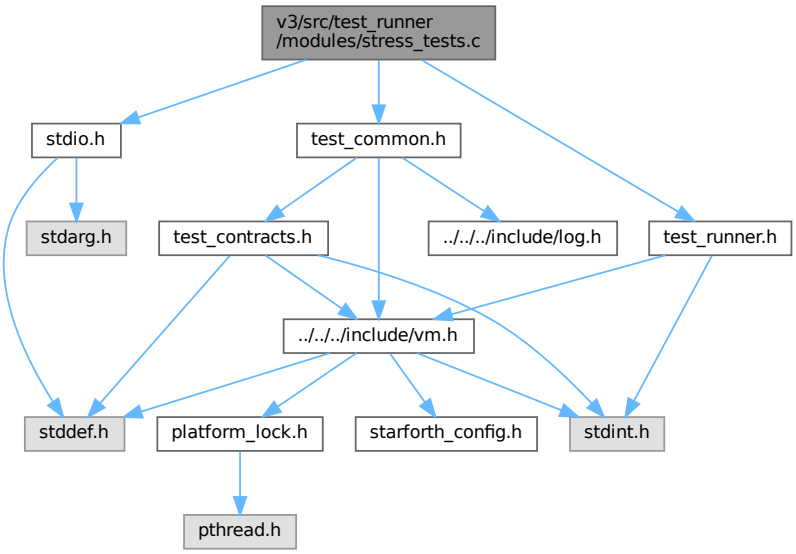
Referenced by run_starforth_words_tests().

7.334 v3/src/test_runner/modules/stress_tests.c File Reference

FORTH-79 Standard Comprehensive Stress Testing Module.

```
#include <stdio.h>
#include "test_common.h"
#include "test_runner.h"
```

Include dependency graph for stress_tests.c:



Functions

- void run_stress_tests (VM *vm)

  Execute comprehensive stress test suites.

Variables

- static WordTestSuite stress_suites [ ]

7.334.1 Detailed Description

FORTH-79 Standard Comprehensive Stress Testing Module.

This module contains extensive stress tests that verify the interpreter's behavior under extreme conditions while remaining within FORTH-79 standard compliance.

Test categories (expanded):

- Deep nesting of control structures (up to 5 levels)

- Stack exhaustion and boundary tests (near-limit operations)
- Numeric boundary conditions (cell_t min/max, overflow)
- Dictionary pressure (many definitions, long names)
- Word execution chains (deep call stacks)
- Control flow edge cases (empty loops, large counts, EXIT)
- Memory allocation patterns (fragmentation, large blocks)
- Error recovery (ABORT, stack underflow recovery)
- Large definitions and word chains

IMPORTANT: These tests only use FORTH-79 standard words. Features from later standards (ANS Forth 1994, Forth 2012) are NOT tested here:

- RECURSE (ANS Forth) - not in FORTH-79
- Self-referential definitions - not supported in FORTH-79
- Extended VARIABLE usage - only standard semantics tested

For testing extended/non-standard features, see integration_tests.c

7.334.2 Function Documentation

7.334.2.1 run_stress_tests()

```
void run_stress_tests (
            VM * vm )
```

Execute comprehensive stress test suites.

Runs an extensive set of stress tests to verify the VM's behavior under extreme conditions. Tests include:

- Stack operations and boundaries
- Deep nesting and control flow
- Numeric boundaries and overflow
- Dictionary pressure
- Memory allocation patterns
- Error recovery
- Mixed complexity scenarios

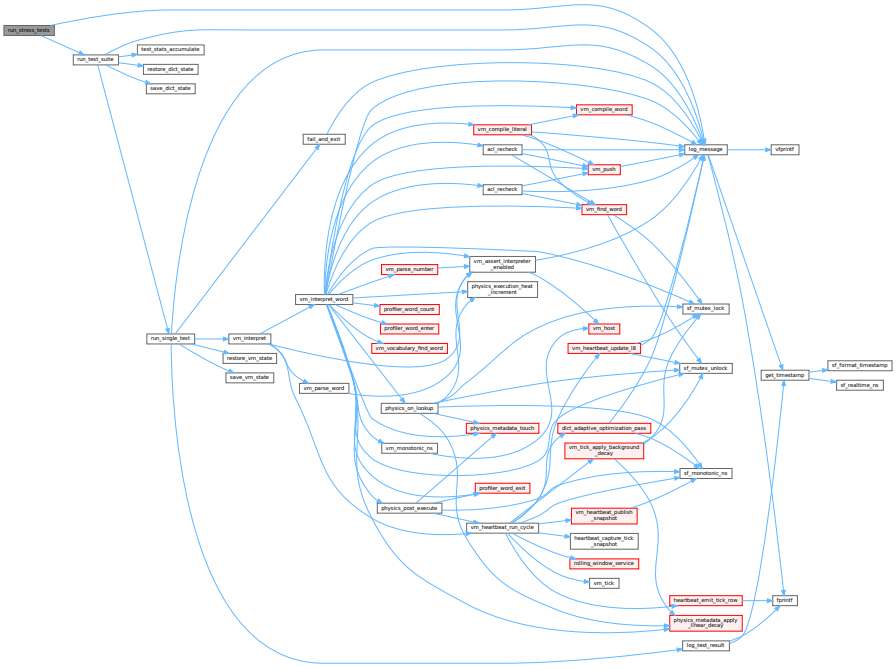
All tests are FORTH-79 compliant.

Parameters

vm	Pointer to the Forth virtual machine instance

References global_test_stats, LOG_INFO, log_message(), LOG_TEST, run_test_suite(), stress_suites, TestStats::total_error, TestStats::total_fail, TestStats::total_pass, TestStats::total_skip, and TestStats::total_tests.

Here is the call graph for this function:



7.334.3 Variable Documentation

7.334.3.1 stress_suites

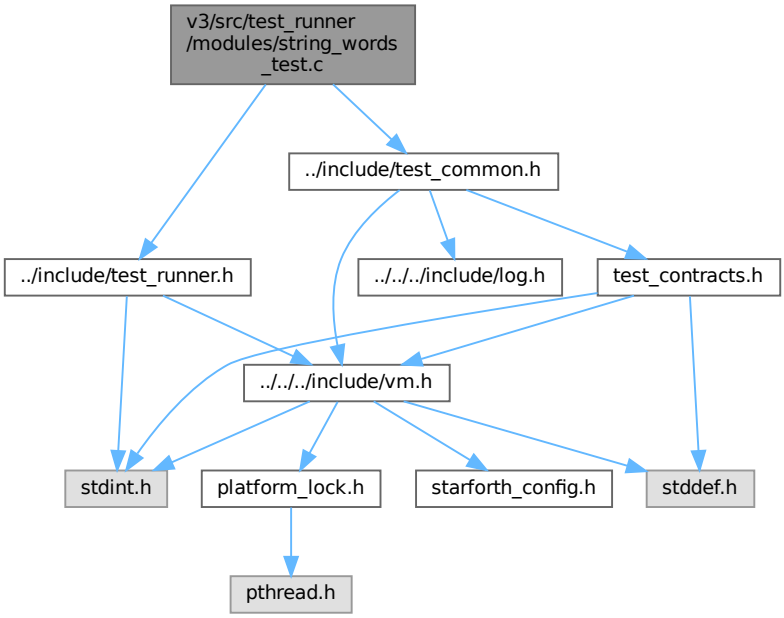
```
WordTestSuite stress_suites[]  [static]
```

Referenced by run_stress_tests().

7.335 v3/src/test_runner/modules/string_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for string_words_test.c:



Functions

- void run_string_words_tests (VM *vm)

  Executes all string manipulation word tests.

Variables

- static WordTestSuite string_word_suites [ ]

7.335.1 Function Documentation

7.335.1.1 run_string_words_tests()

```
void run_string_words_tests (
            VM * vm )
```

Executes all string manipulation word tests.

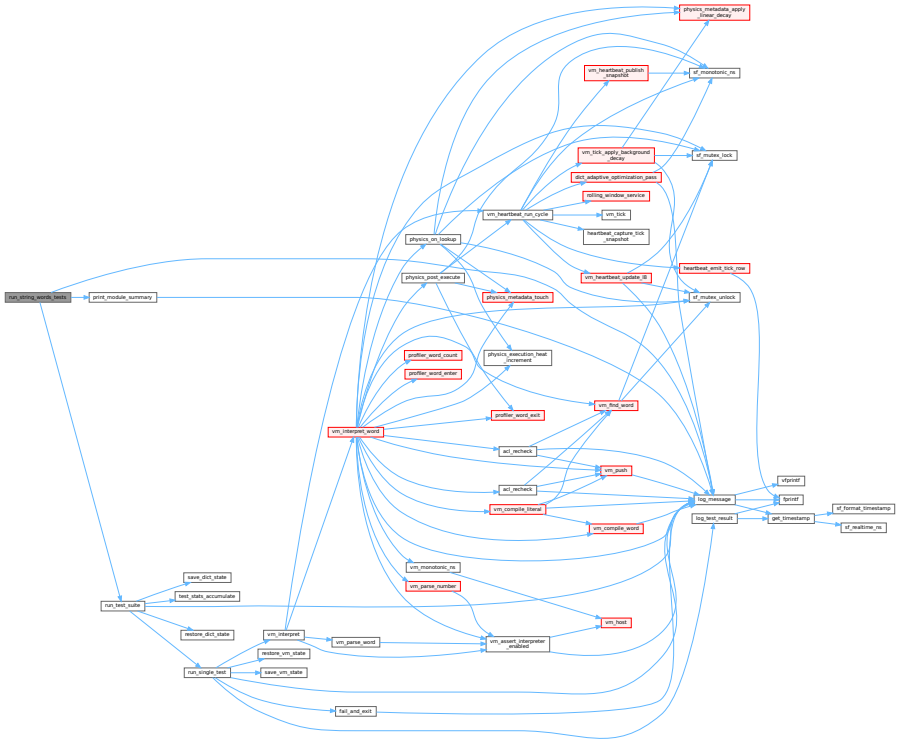
Runs test suites for FORTH string operations including COUNT, -TRAILING, CMOVE, CMOVE>, COMPARE, SEARCH, SCAN, SKIP, and BLANK

Parameters

vm	Pointer to the Forth virtual machine instance

References LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), string_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.335.2 Variable Documentation

7.335.2.1 string_word_suites

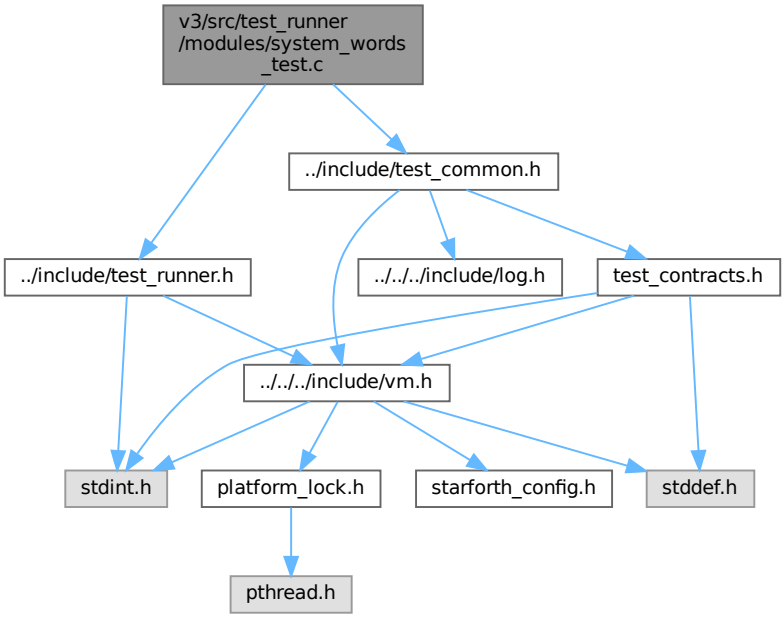
```
WordTestSuite string_word_suites[]  [static]
```

Referenced by run_string_words_tests().

7.336 v3/src/test_runner/modules/system_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for system_words_test.c:



Functions

- void run_system_words_tests (VM *vm)

  Executes all system word tests.

Variables

- static WordTestSuite system_word_suites [ ]

  Test suite definitions for system control words.

7.336.1 Function Documentation

7.336.1.1 run_system_words_tests()

```
void run_system_words_tests (
            VM * vm )
```

Executes all system word tests.

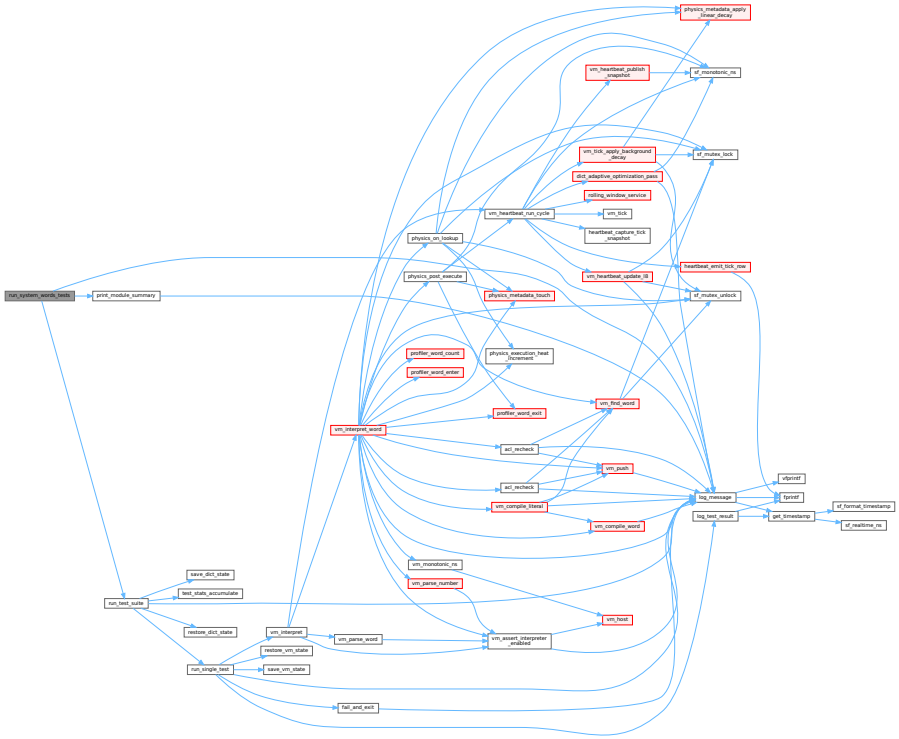
Runs test suites for FORTH system control operations including QUIT, ABORT, ABORT", BYE, COLD, and WARM. Some tests are marked as unimplemented due to their system-level effects

Parameters

vm	Pointer to the Forth virtual machine instance

References LOG_INFO, log_message(), LOG_TEST, LOG_WARN, print_module_summary(), run_test_suite(), system_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.336.2 Variable Documentation

7.336.2.1 system_word_suites

```
WordTestSuite system_word_suites[]  [static]
```

Test suite definitions for system control words.

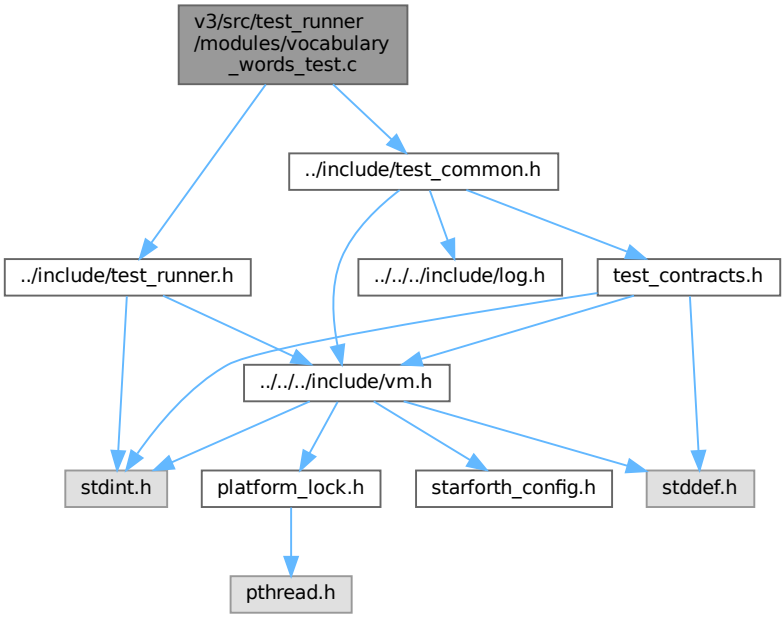
Contains test cases for each system word including normal operations and error conditions

Referenced by run_system_words_tests().

7.337 v3/src/test_runner/modules/vocabulary_words_test.c File Reference

```
#include "../include/test_runner.h"
#include "../include/test_common.h"
```

Include dependency graph for vocabulary_words_test.c:



Functions

- void run_vocabulary_words_tests (VM *vm)

  Executes all vocabulary word test suites.

Variables

- static WordTestSuite vocabulary_word_suites [ ]

  Test suites for vocabulary-related FORTH words (Module 15)

7.337.1 Function Documentation

7.337.1.1 run_vocabulary_words_tests()

```
void run_vocabulary_words_tests (
            VM * vm )
```

Executes all vocabulary word test suites.

Parameters

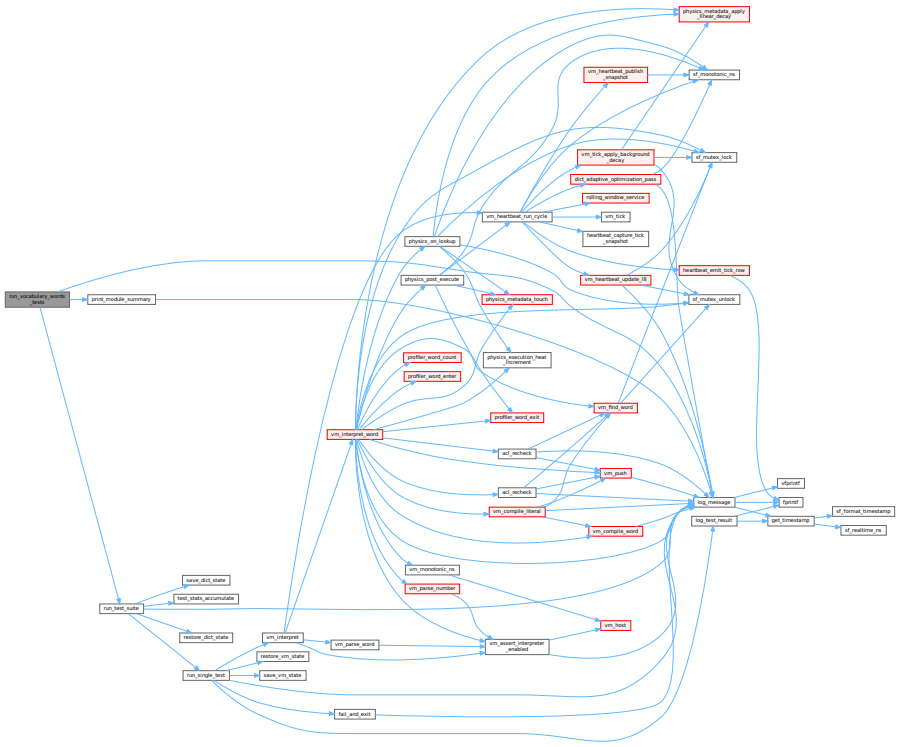
vm	Pointer to the Forth virtual machine instance

Runs through all vocabulary-related test suites, executing each test case and logging the results_run_01_2025_↩

12_08. Tests vocabulary creation, selection, and manipulation.

References LOG_INFO, log_message(), LOG_TEST, print_module_summary(), run_test_suite(), vocabulary_word_suites, and WordTestSuite::word_name.

Here is the call graph for this function:



7.337.2 Variable Documentation

7.337.2.1 vocabulary_word_suites

```
WordTestSuite vocabulary_word_suites[]  [static]
```

Test suites for vocabulary-related FORTH words (Module 15)

Contains test cases for each vocabulary word, including normal operations and error cases. Each suite tests specific functionality of the vocabulary system.

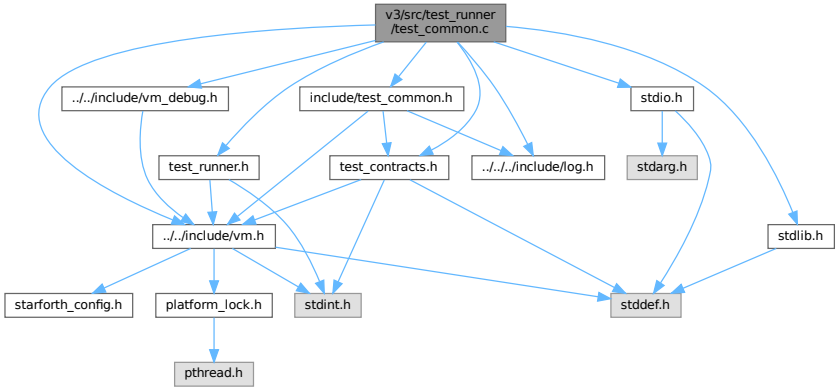
Referenced by run_vocabulary_words_tests().

7.338 v3/src/test_runner/test_common.c File Reference

```
#include "../../include/vm.h"
#include "../../include/vm_debug.h"
#include "include/test_common.h"
#include "include/test_contracts.h"
#include "../../include/log.h"
```

```
#include "test_runner.h"
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for test_common.c:



Macros

- #define test_fatal_exit() exit(1)
- #define DIE(vm, why) dump_and_die((vm), (why), __FILE__, __LINE__)

Functions

- static void dump_and_die (VM *vm, const char *reason, const char *file, int line)

  Dumps VM state and terminates execution.
- static void fail_and_exit (VM *vm, const char *module, const char *case_name, const char *input, const char *reason)

  Handles test failure and exits.
- void save_vm_state (VM *vm, int *dsp, int *rsp, int *error, vm_mode_t *mode)

  Saves current VM state.
- void restore_vm_state (VM *vm, int dsp, int rsp, int error, vm_mode_t mode)

  Restores previously saved VM state.
- int assert_stack_depth (VM *vm, int expected_depth)

  Verifies VM stack depth matches expected value.
- int assert_stack_top (VM *vm, int expected_value)

  Verifies top value on VM stack matches expected value.
- int assert_vm_error (VM *vm, int should_have_error)

  Verifies VM error state matches expectation.
- TestResult run_single_test (VM *vm, const char *word_name, const TestCase *test)

  Executes a single test case.
- void run_test_suite (VM *vm, const WordTestSuite *suite)

  Executes a complete test suite for a Forth word.
- static WordContract resolve_contract (WordContract test_c, WordContract suite_c, WordContract module_c)

  Resolves the effective contract for a test case.
- void run_test_suite_m (VM *vm, const WordTestSuite *suite, WordContract module_contract)

  Runs all tests in a suite with an inherited module-level contract.

- void print_module_summary (const char *module_name, int pass, int fail, int skip, int error)

  Prints test module execution summary.
- void save_dict_state (VM *vm, DictEntry **latest, size_t *here)

  Saves current dictionary state before test execution.
- void restore_dict_state (VM *vm, DictEntry *latest, size_t here)

  Restores dictionary state after test execution.

Variables

- int fail_fast = 0

  Global flag to control test execution behavior.

7.338.1 Macro Definition Documentation

7.338.1.1 DIE

```
#define DIE(
              vm,
              why ) dump_and_die((vm), (why), __FILE__, __LINE__)
```

7.338.1.2 test_fatal_exit

```
#define test_fatal_exit( ) exit(1)
```

7.338.2 Function Documentation

7.338.2.1 assert_stack_depth()

```
int assert_stack_depth (
            VM * vm,
            int expected_depth )
```

Verifies VM stack depth matches expected value.

Parameters

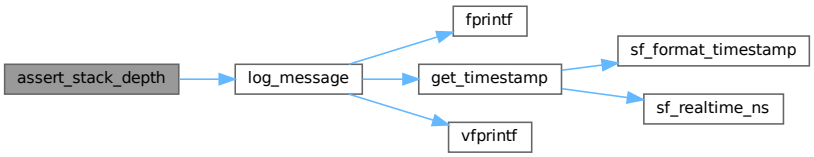
vm	Pointer to VM instance
expected_depth	Expected stack depth

Returns

1 if assertion passes, 0 if it fails

References VM::dsp, LOG_ERROR, and log_message().

Here is the call graph for this function:



7.338.2.2 assert_stack_top()

```
int assert_stack_top (
            VM * vm,
            int expected_value )
```

Verifies top value on VM stack matches expected value.

Parameters

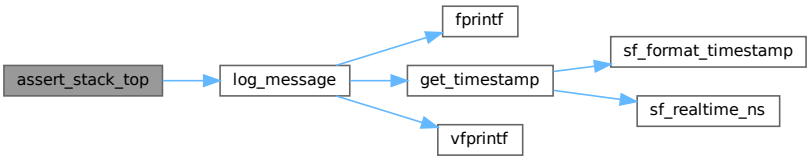
vm	Pointer to VM instance
expected_value	Expected value on top of stack

Returns

1 if assertion passes, 0 if it fails

References VM::data_stack, VM::dsp, LOG_ERROR, and log_message().

Here is the call graph for this function:



7.338.2.3 assert_vm_error()

```
int assert_vm_error (
            VM * vm,
            int should_have_error )
```

Verifies VM error state matches expectation.

Parameters

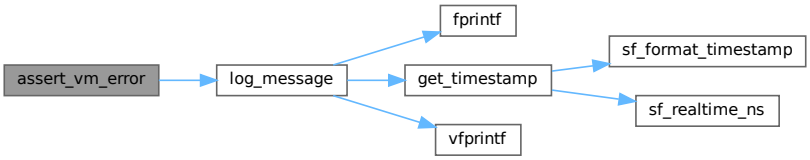
vm	Pointer to VM instance
should_have_error	Expected error state (1 for error, 0 for no error)

Returns

1 if assertion passes, 0 if it fails

References VM::error, LOG_ERROR, and log_message().

Here is the call graph for this function:



7.338.2.4 dump_and_die()

```
static void dump_and_die (
            VM * vm,
            const char * reason,
            const char * file,
            int line )  [static]
```

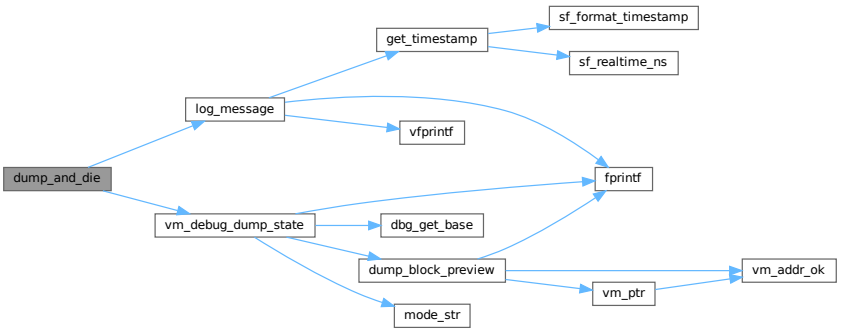
Dumps VM state and terminates execution.

Parameters

vm	Pointer to VM instance
reason	String describing the failure reason
file	Source file where failure occurred
line	Line number where failure occurred

References LOG_ERROR, log_message(), test_fatal_exit, and vm_debug_dump_state().

Here is the call graph for this function:



7.338.2.5 fail_and_exit()

```
static void fail_and_exit (
            VM * vm,
            const char * module,
            const char * case_name,
            const char * input,
            const char * reason )  [static]
```

Handles test failure and exits.

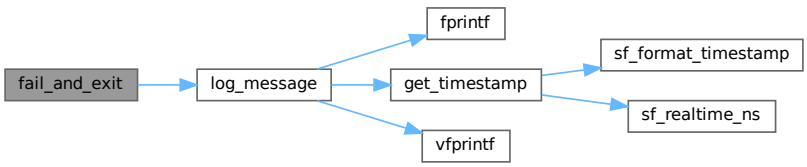
Parameters

vm	Pointer to VM instance
module	Name of the test module
case_name	Name of the test case
input	Test input that caused the failure
reason	Description of failure reason

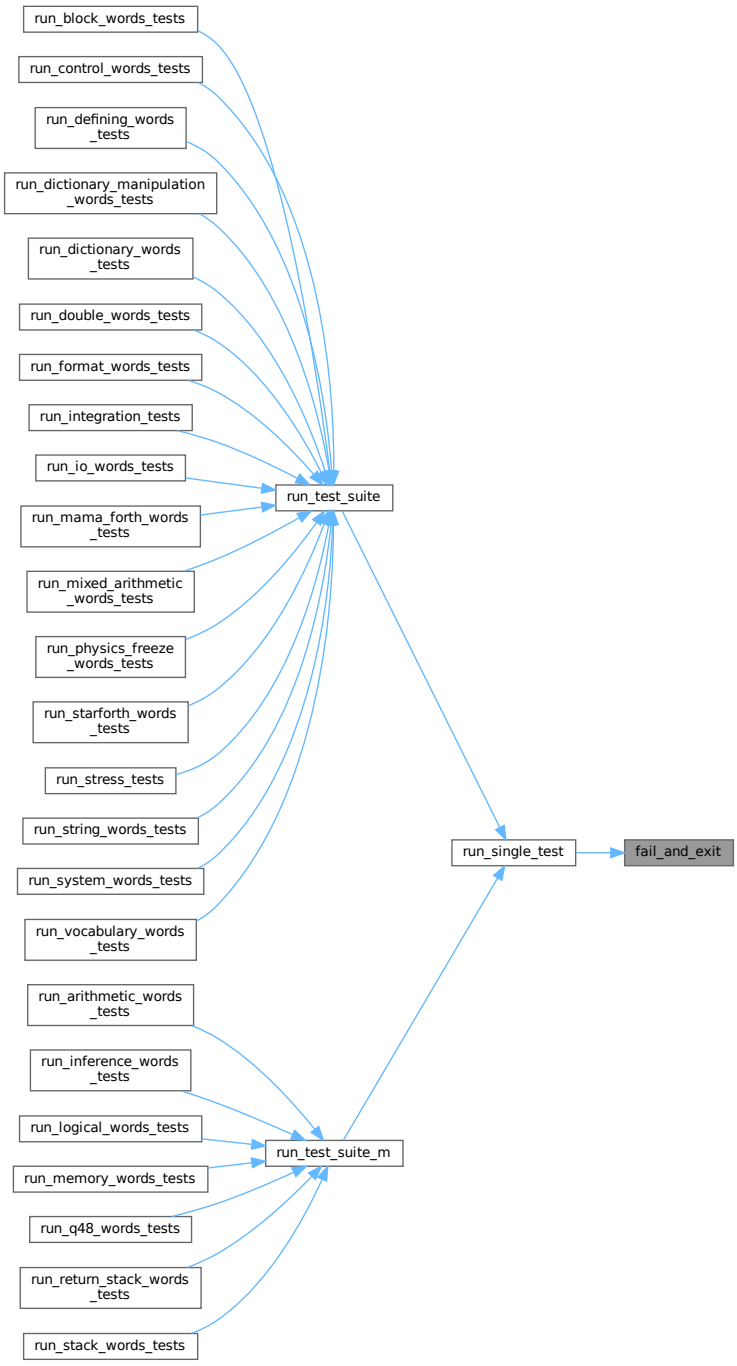
References DIE, LOG_ERROR, and log_message().

Referenced by run_single_test().

Here is the call graph for this function:



Here is the caller graph for this function:



7.338.2.6 print_module_summary()

```
void print_module_summary (
            const char * module_name,
            int pass,
            int fail,
```

```
int skip,
int error )
```

Prints test module execution summary.

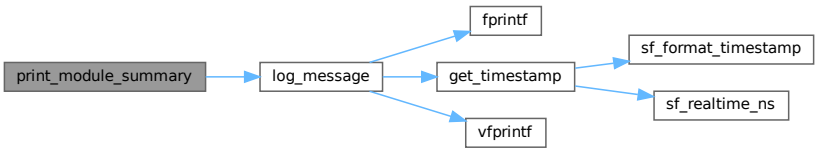
Parameters

module_name	Name of the test module
pass	Number of passed tests
fail	Number of failed tests
skip	Number of skipped tests
error	Number of tests with errors

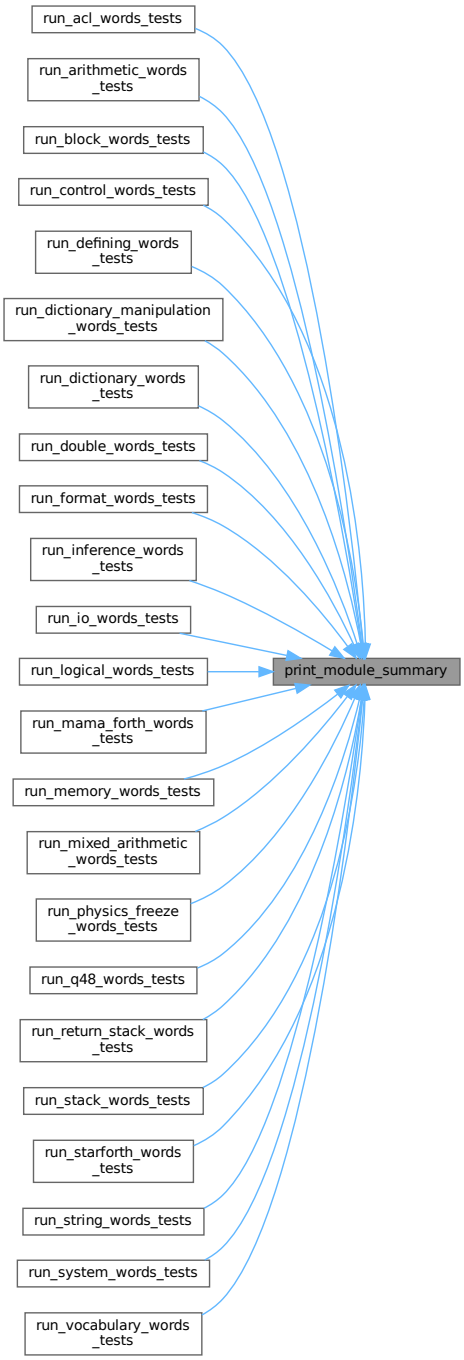
References log_message(), and LOG_TEST.

Referenced by run_acl_words_tests(), run_arithmetic_words_tests(), run_block_words_tests(), run_control_words_tests(), run_defining_words_tests(), run_dictionary_manipulation_words_tests(), run_dictionary_words_tests(), run_double_words_tests(), run_format_words_tests(), run_inference_words_tests(), run_io_words_tests(), run_logical_words_tests(), run_mama_forth_words_tests(), run_memory_words_tests(), run_mixed_arithmetic_words_tests(), run_physics_freeze_words_tests(), run_q48_words_tests(), run_return_stack_words_tests(), run_stack_words_tests(), run_starforth_words_tests(), run_string_words_tests(), run_system_words_tests(), and run_vocabulary_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.338.2.7 resolve_contract()

```
static WordContract resolve_contract (
            WordContract test_c,
            WordContract suite_c,
            WordContract module_c )  [static]
```

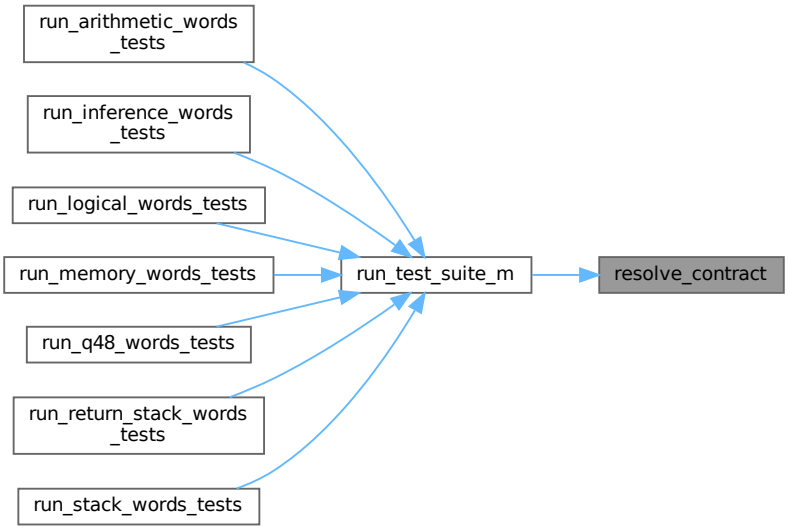
Resolves the effective contract for a test case.

First non-NONE source wins: per-test > per-suite > module.

References WordContract::flags.

Referenced by run_test_suite_m().

Here is the caller graph for this function:



7.338.2.8 restore_dict_state()

```
void restore_dict_state (
            VM * vm,
            DictEntry * latest,
            size_t here )
```

Restores dictionary state after test execution.

Parameters

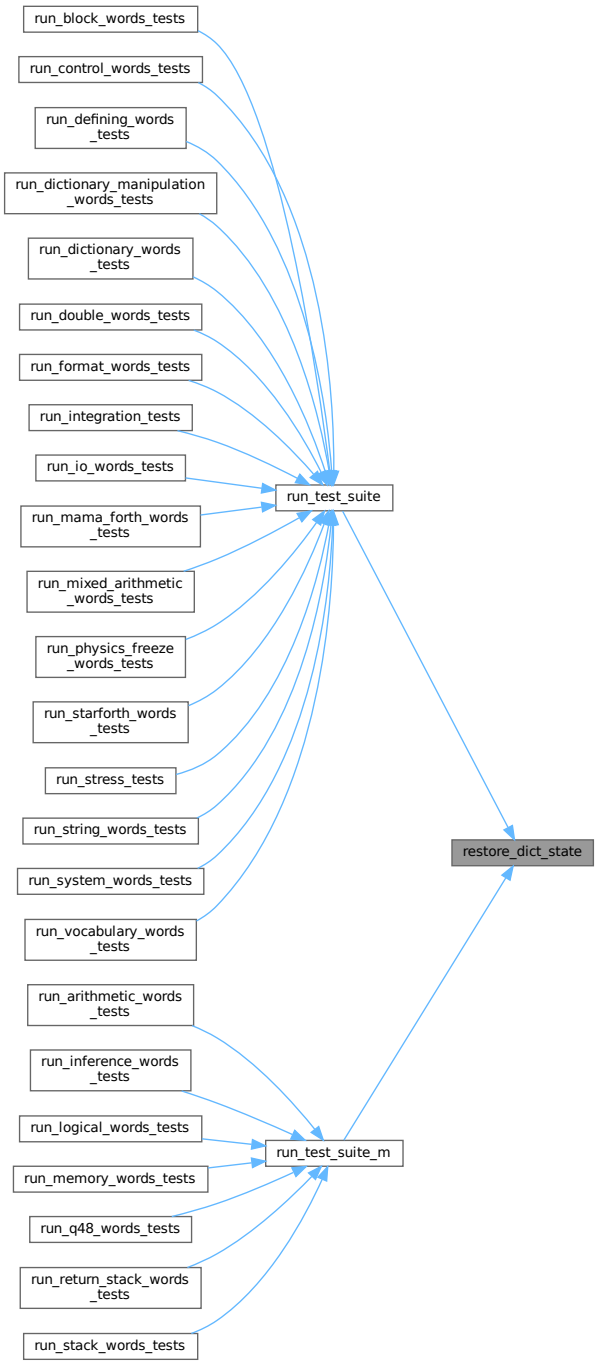
vm	Pointer to VM instance
latest	Latest dictionary entry to restore
here	HERE pointer to restore

Removes all words added during test by restoring dictionary pointers. This prevents test pollution of the dictionary and ensures each test starts with a clean slate.

References VM::here, and VM::latest.

Referenced by run_test_suite(), and run_test_suite_m().

Here is the caller graph for this function:



7.338.2.9 restore_vm_state()

```
void restore_vm_state (
            VM * vm,
            int dsp,
            int rsp,
```

```
    int error,
    vm_mode_t mode )
```

Restores previously saved VM state.

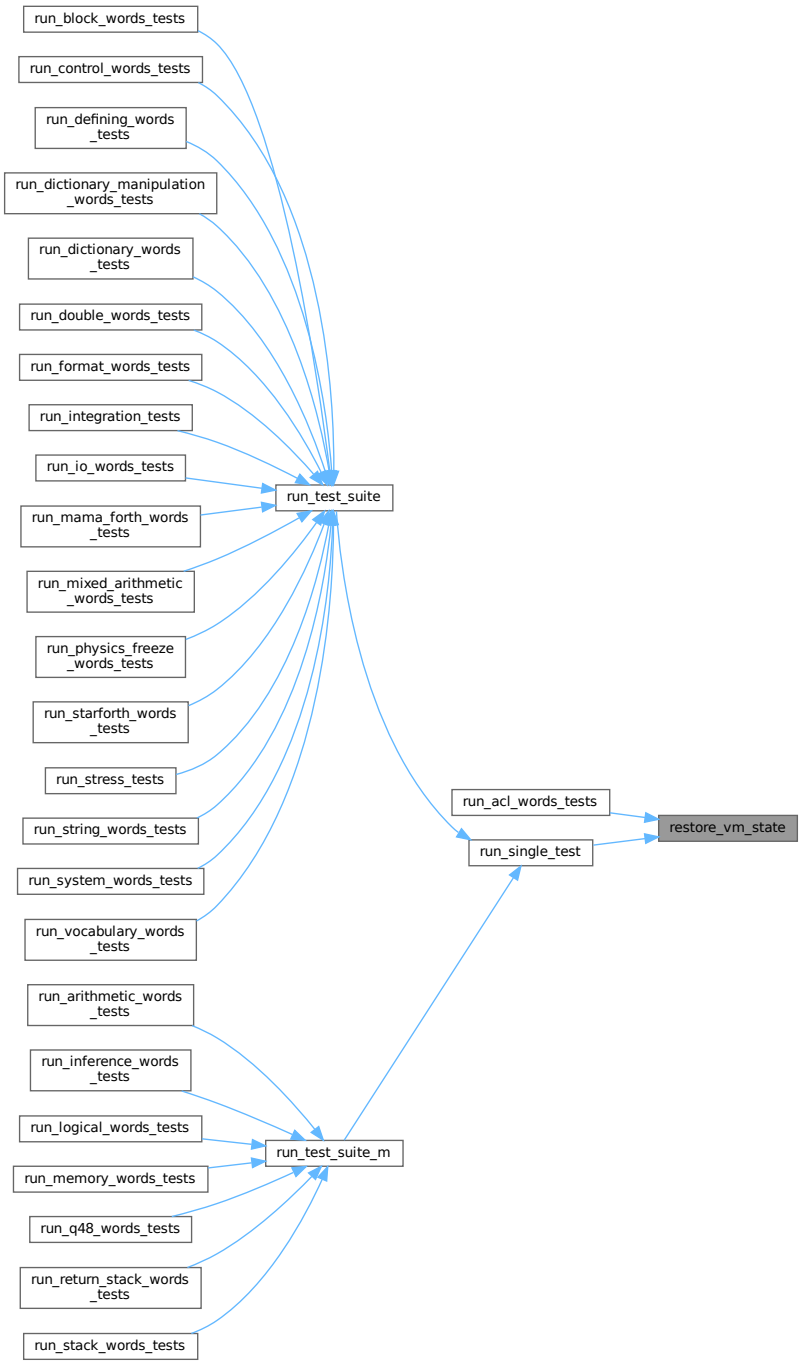
Parameters

vm	Pointer to VM instance
dsp	Data stack pointer to restore
rsp	Return stack pointer to restore
error	Error state to restore
mode	VM mode to restore

References VM::abort_requested, VM::compiling_word, VM::current_executing_entry, VM::dsp, VM::error, VM::exit_colon, VM::mode, and VM::rsp.

Referenced by run_acl_words_tests(), and run_single_test().

Here is the caller graph for this function:



7.338.2.10 run_single_test()

```
TestResult run_single_test (
            VM * vm,
            const char * word_name,
            const TestCase * test )
```

Executes a single test case.

Parameters

vm	Pointer to VM instance
word_name	Name of the word being tested
test	Pointer to test case definition

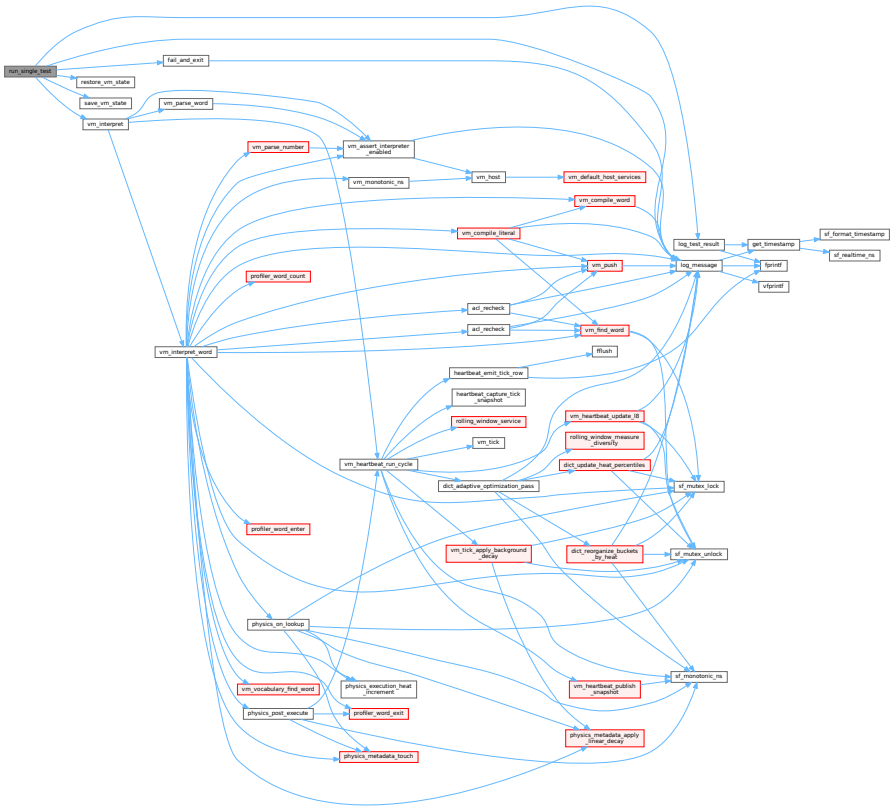
Returns

Test result (PASS, FAIL, SKIP, or ERROR)

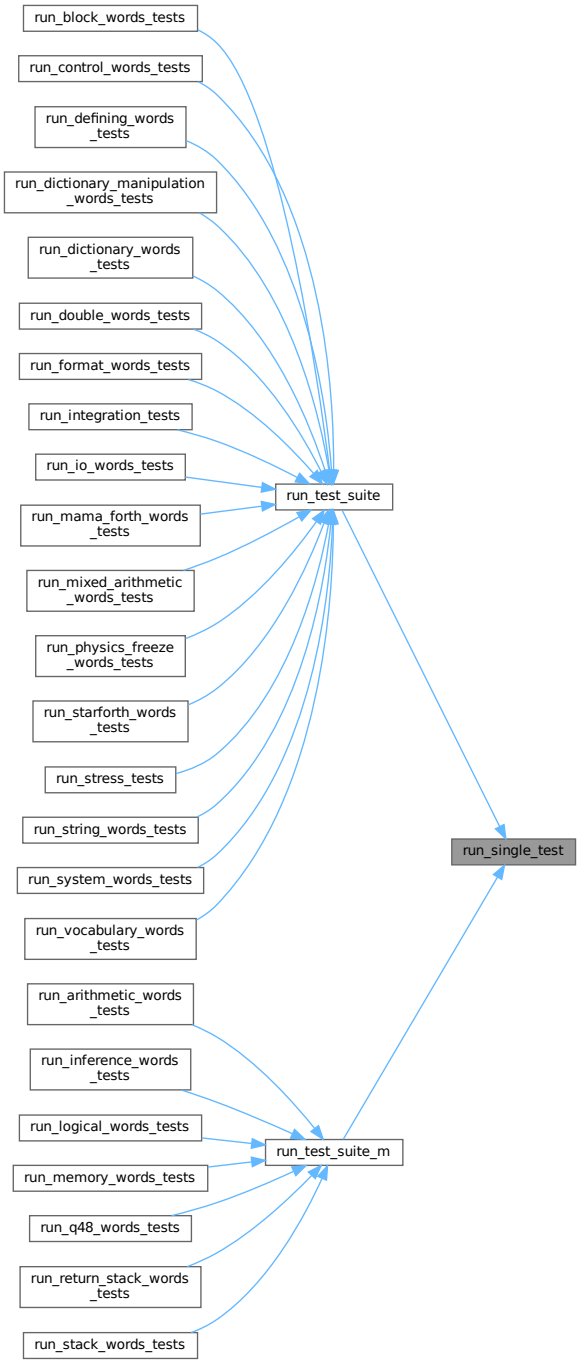
References VM::error, TestCase::expected, fail_and_exit(), fail_fast, TestCase::implemented, TestCase::input, LOG_ERROR, log_message(), LOG_TEST, log_test_result(), TestCase::name, restore_vm_state(), save_vm_state(), TestCase::should_error, TEST_FAIL, TEST_PASS, TEST_SKIP, and vm_interpret().

Referenced by run_test_suite(), and run_test_suite_m().

Here is the call graph for this function:



Here is the caller graph for this function:



7.338.2.11 run_test_suite()

```
void run_test_suite (
            VM * vm,
            const WordTestSuite * suite )
```

Executes a complete test suite for a Forth word.

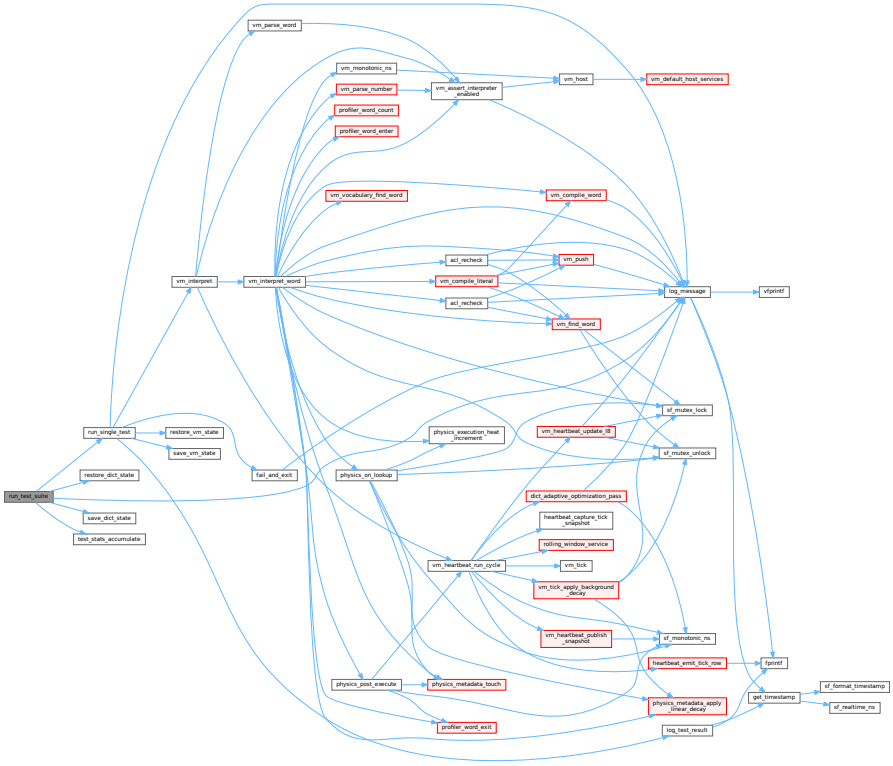
Parameters

vm	Pointer to VM instance
suite	Pointer to test suite definition

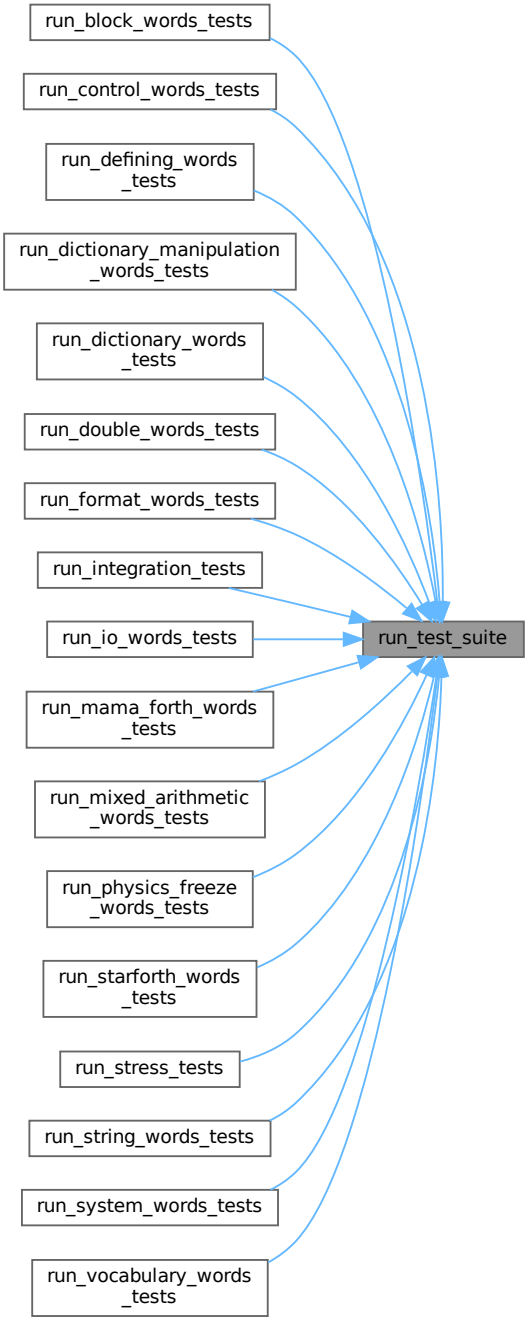
References TestCase::implemented, log_message(), LOG_TEST, TestCase::name, restore_dict_state(), run_single_test(), save_dict_state(), WordTestSuite::test_count, TEST_ERROR, TEST_FAIL, TEST_PASS, TEST_SKIP, test_stats_accumulate(), WordTestSuite::tests, and WordTestSuite::word_name.

Referenced by run_block_words_tests(), run_control_words_tests(), run_defining_words_tests(), run_dictionary_manipulation_words_ run_dictionary_words_tests(), run_double_words_tests(), run_format_words_tests(), run_integration_tests(), run_io_words_tests(), run_mama_forth_words_tests(), run_mixed_arithmetic_words_tests(), run_physics_freeze_words_tests(), run_starforth_words_tests(), run_stress_tests(), run_string_words_tests(), run_system_words_tests(), and run_vocabulary_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.338.2.12 run_test_suite_m()

```
void run_test_suite_m (
            VM * vm,
            const WordTestSuite * suite,
            WordContract module_contract )
```

Runs all tests in a suite with an inherited module-level contract.

Contract resolution order (first non-NONE wins): test->contract > suite->suite_contract > module_contract

When CONTRACTS_ENABLED=1, each passing TEST_NORMAL test is re-run under physics perturbation to witness A4' (word_physics_transparent).

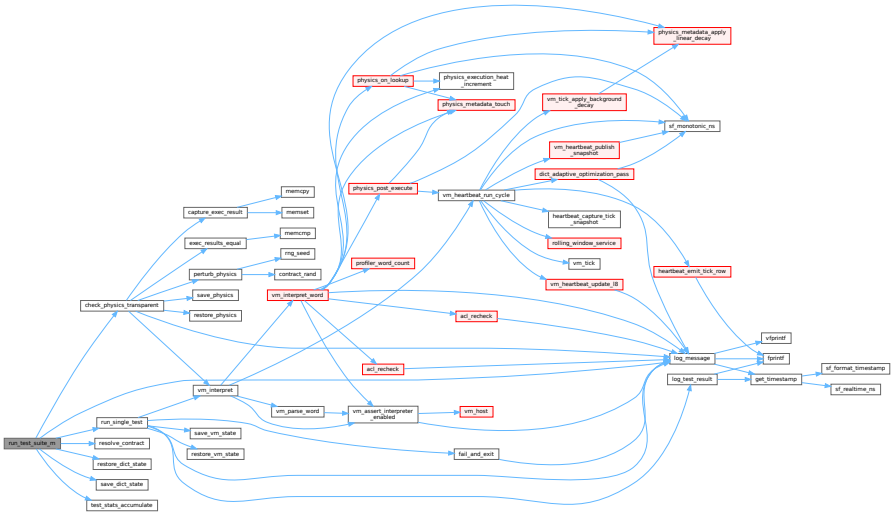
Parameters

vm	Pointer to the VM instance
suite	Test suite to execute
module_contract	Default contract for tests that don't specify one

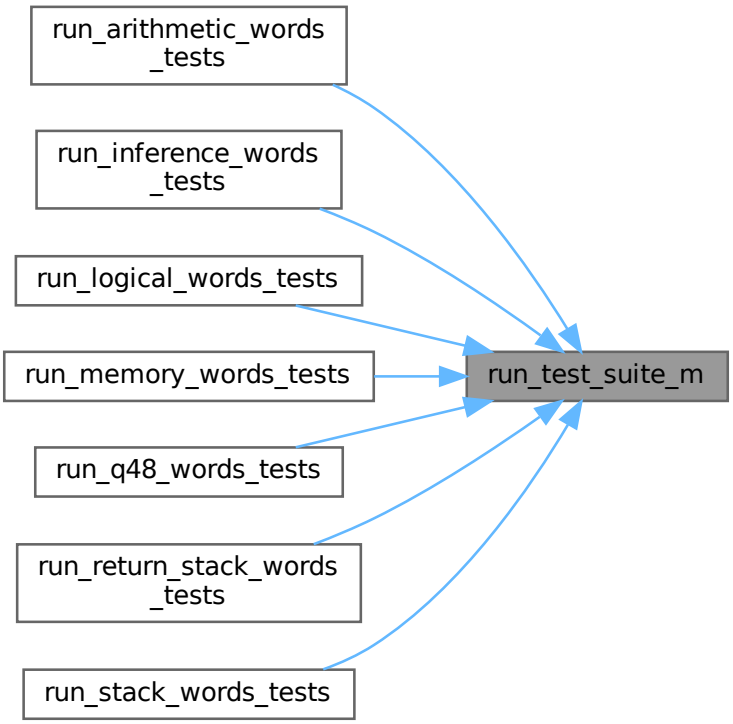
References check_physics_transparent(), TestCase::contract, CONTRACT_PHYSICS_TRANSPARENT, WordContract::flags, WordContract::fuzz_rounds, TestCase::implemented, TestCase::input, log_message(), LOG_TEST, TestCase::name, resolve_contract(), restore_dict_state(), run_single_test(), save_dict_state(), TestCase::should_error, WordTestSuite::suite_contract, WordTestSuite::test_count, TEST_ERROR, TEST_ERROR_CASE, TEST_FAIL, TEST_PASS, TEST_SKIP, test_stats_accumulate(), WordTestSuite::tests, TestCase::type, and WordTestSuite::word_name.

Referenced by run_arithmetic_words_tests(), run_inference_words_tests(), run_logical_words_tests(), run_memory_words_tests(), run_q48_words_tests(), run_return_stack_words_tests(), and run_stack_words_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.338.2.13 save_dict_state()

```
void save_dict_state (
            VM * vm,
            DictEntry ** latest,
            size_t * here )
```

Saves current dictionary state before test execution.

Parameters

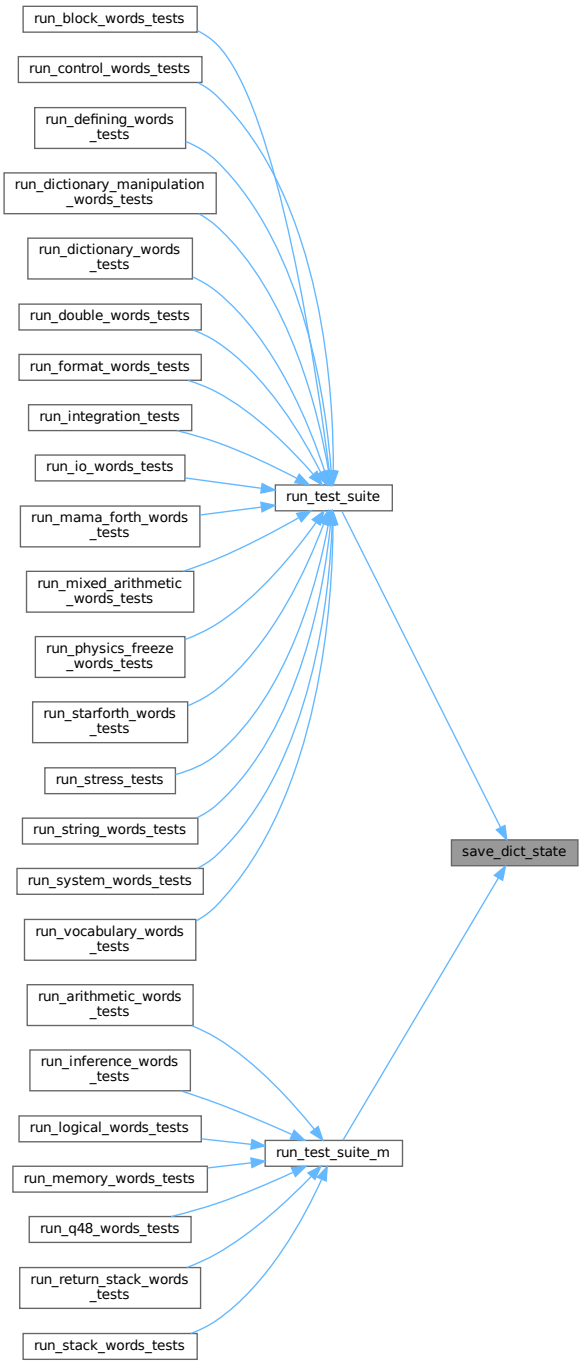
vm	Pointer to VM instance
latest	Pointer to store latest dictionary entry
here	Pointer to store HERE pointer

Captures dictionary state so tests can be cleaned up after execution

References VM::here, and VM::latest.

Referenced by run_test_suite(), and run_test_suite_m().

Here is the caller graph for this function:



7.338.2.14 save_vm_state()

```
void save_vm_state (
            VM * vm,
            int * dsp,
            int * rsp,
```

```
    int * error,
    vm_mode_t * mode )
```

Saves current VM state.

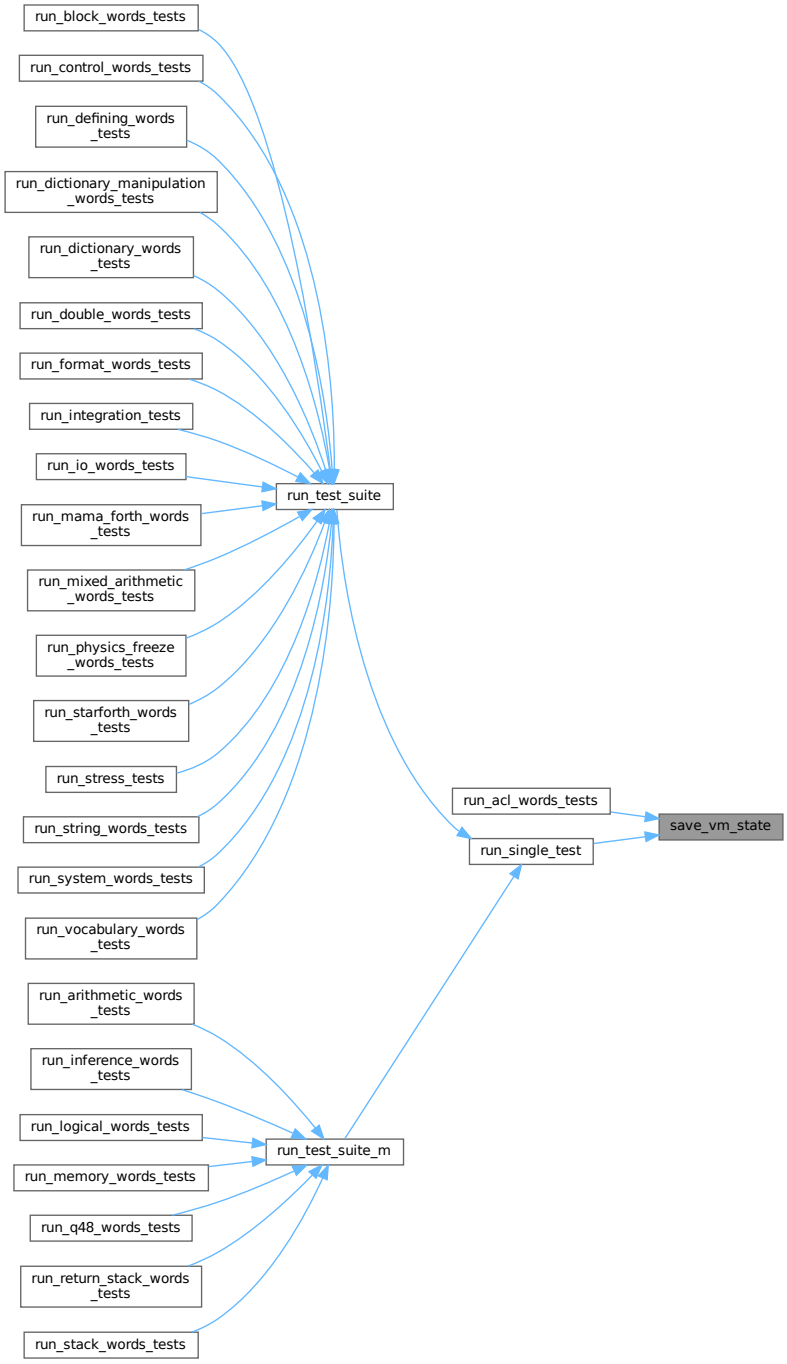
Parameters

vm	Pointer to VM instance
dsp	Pointer to store data stack pointer
rsp	Pointer to store return stack pointer
error	Pointer to store error state
mode	Pointer to store VM mode

References VM::dsp, VM::error, VM::mode, and VM::rsp.

Referenced by run_acl_words_tests(), and run_single_test().

Here is the caller graph for this function:



7.338.3 Variable Documentation

7.338.3.1 fail_fast

```
int fail_fast = 0
```

Global flag to control test execution behavior.

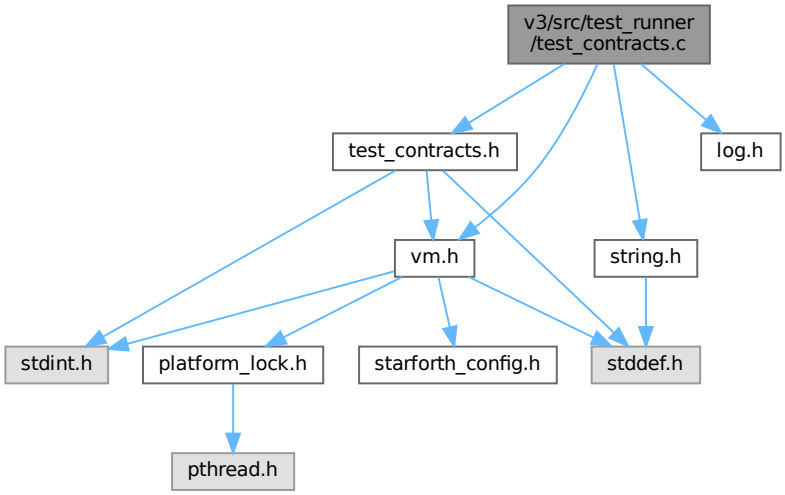
When set to non-zero, causes test runner to exit immediately on first failure

Referenced by main(), and run_single_test().

7.339 v3/src/test_runner/test_contracts.c File Reference

```
#include "test_contracts.h"
#include "log.h"
#include "vm.h"
#include <string.h>
```

Include dependency graph for test_contracts.c:



Functions

- int contract_get_violation_count (void)

  Return the current count of accumulated contract violations.
- void contract_reset_violation_count (void)

  Reset the contract violation counter to zero.
- static uint64_t contract_rand (void)

  Generate the next pseudo-random 64-bit value via xorshift64.
- static void rng_seed (int round)

  Seed the RNG for a given fuzz round.
- void capture_exec_result (VM *vm, ExecResult *r)

  Snapshot the execution-domain fields of a VM into an `ExecResult`.
- int exec_results_equal (const ExecResult *a, const ExecResult *b)

  Compare two `ExecResult` snapshots for execution-domain equality.
- void save_physics (VM *vm, PhysicsSnapshot *s)

  Save the key physics scalars touched by the seven feedback loops.
- void restore_physics (VM *vm, const PhysicsSnapshot *s)

  Restore the physics scalars from a previously saved snapshot.

- void perturb_physics (VM *vm, int round)

  Randomly perturb key physics scalars to expose A4' violations.
- int check_physics_transparent (VM *vm, const char *word_name, const char *input, int should_error, int implemented, int fuzz_rounds)

  A4' contract check — verify a word is transparent to physics state.
- int assert_heartbeat_safe (VM *vm)

  A1 contract check — verify `vm_tick()` does not modify the execution domain.

Variables

- int global_contract_violations = 0
- static uint64_t rng_state = 0xDEADBEEFCAFEBABEULL

7.339.1 Function Documentation

7.339.1.1 assert_heartbeat_safe()

```
int assert_heartbeat_safe (
            VM * vm )
```

A1 contract check — verify `vm_tick()` does not modify the execution domain.

Axiom A1 (heartbeat isolation): the background heartbeat tick must never alter stacks, error state, or any other execution-domain field observable by FORTH words. This function captures an `ExecResult` before and after one `vm_tick()` call and compares them with `exec_results_equal()`.

On violation, logs [A1 CONTRACT] VIOLATION with before/after dsp and rsp values, and increments `global_↩ contract_violations`.

Parameters

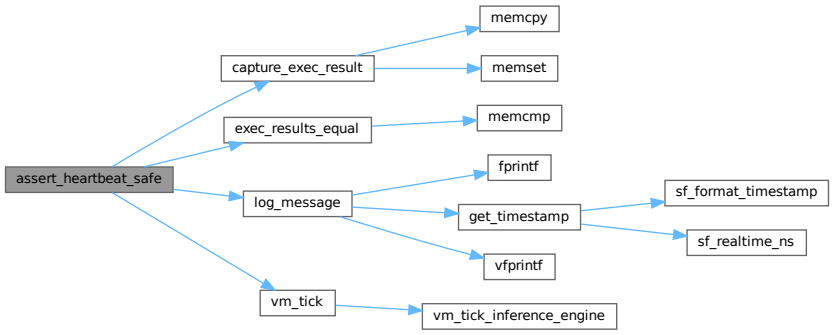
vm	VM instance; stacks and error state must already be in a known reference state before calling.

Returns

1 if the heartbeat left the execution domain unchanged, 0 on violation.

References capture_exec_result(), ExecResult::dsp, exec_results_equal(), global_contract_violations, LOG_ERROR, log_message(), ExecResult::rsp, and vm_tick().

Here is the call graph for this function:



7.339.1.2 capture_exec_result()

```
void capture_exec_result (
            VM * vm,
            ExecResult * r )
```

Snapshot the execution-domain fields of a VM into an `ExecResult`.

Captures `dsp`, `rsp`, `error`, and the live portion of the data stack (indices 0 ... dsp?1). The remainder of `r->stack` is zeroed for deterministic `memcmp()` comparisons in `exec_results_equal()`.

Only execution-domain fields are captured — physics scalars, heartbeat state, and rolling-window counters are deliberately excluded.

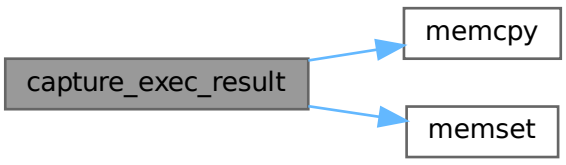
Parameters

vm	VM instance to snapshot.
r	Output `ExecResult` struct to populate.

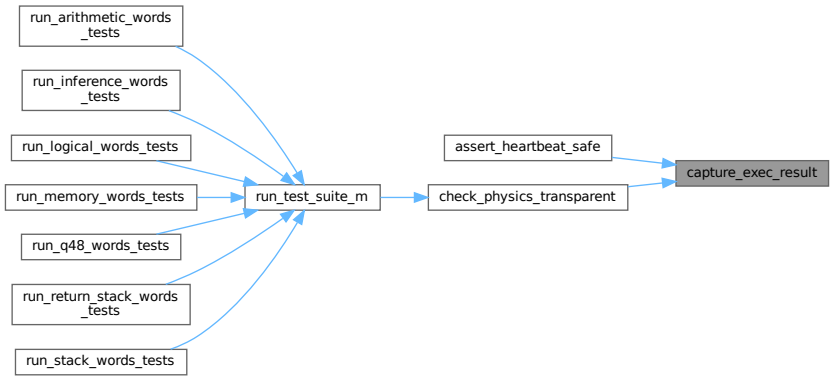
References VM::data_stack, VM::dsp, ExecResult::dsp, VM::error, ExecResult::error, memcpy(), memset(), VM::rsp, ExecResult::rsp, ExecResult::stack, and STACK_SIZE.

Referenced by assert_heartbeat_safe(), and check_physics_transparent().

Here is the call graph for this function:



Here is the caller graph for this function:



7.339.1.3 check_physics_transparent()

```
int check_physics_transparent (
            VM * vm,
            const char * word_name,
            const char * input,
            int should_error,
            int implemented,
            int fuzz_rounds )
```

A4' contract check — verify a word is transparent to physics state.

Axiom A4' (physics transparency): a FORTH word's execution-domain result (`ExecResult`) must be identical regardless of the values of the seven physics feedback-loop scalars.

Protocol:

1. Run `input` with current (unperturbed) physics; capture baseline.
2. For each of `fuzz_rounds` rounds: seed RNG, perturb physics, re-run `input`, capture result, compare to baseline.
3. On divergence, log the violation and increment `global_contract_violations`.
4. Always restore original physics state before returning.

Error-expected tests (`should_error`) and unimplemented words (`implemented` == 0) are skipped: exec-equivalent divergence is legitimate for error paths.

Parameters

vm	VM instance.
word_name	Human-readable word name for diagnostic messages.
input	FORTH source string to execute (same each round).
should_error	Non-zero if the test is expected to raise an error.
implemented	Non-zero if the word is implemented and should be checked.
fuzz_rounds	Number of perturbation rounds; 0 → CONTRACT_DEFAULT_FUZZ_ROUNDS.

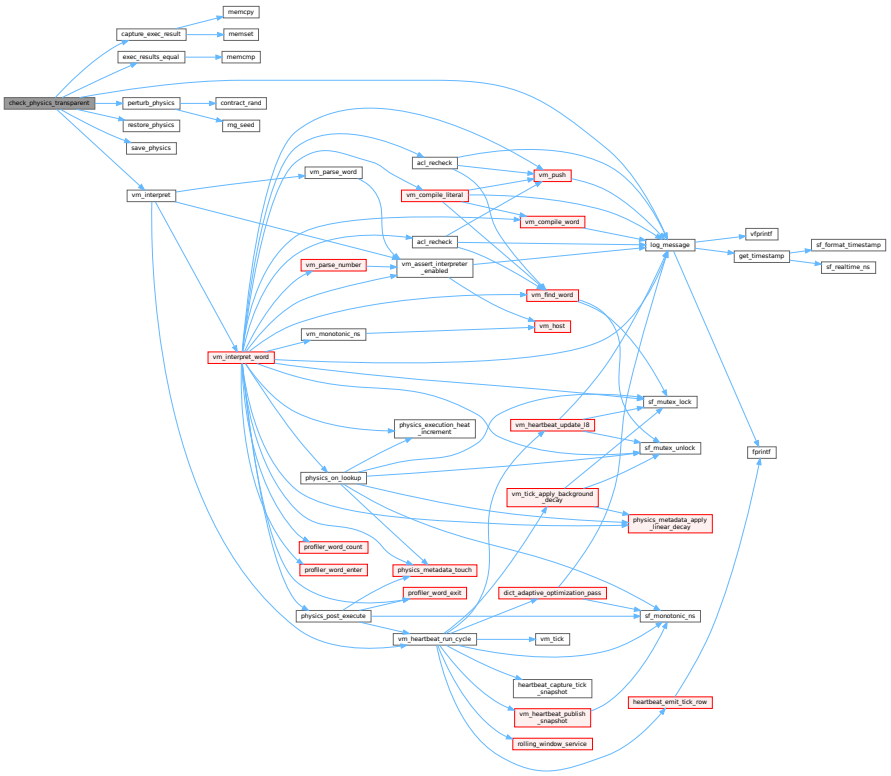
Returns

1 if all rounds passed (or test skipped), 0 on any violation.

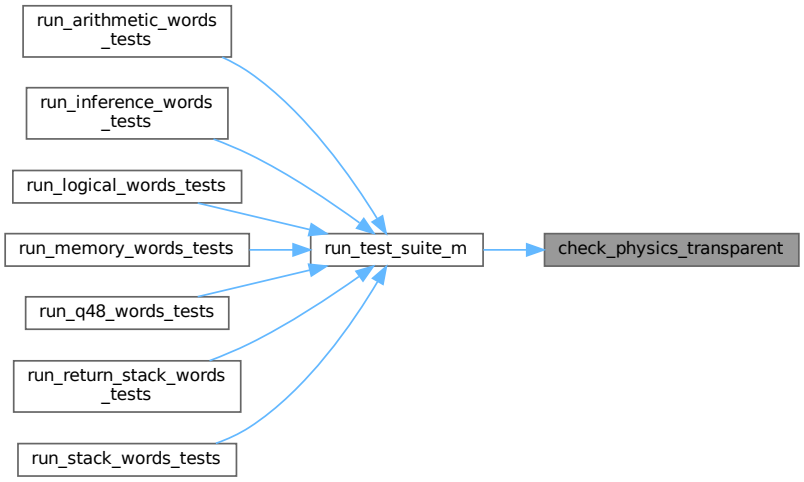
References VM::abort_requested, capture_exec_result(), CONTRACT_DEFAULT_FUZZ_ROUNDS, VM::dsp, ExecResult::dsp, VM::error, exec_results_equal(), VM::exit_colon, global_contract_violations, LOG_ERROR, log_message(), perturb_physics(), restore_physics(), VM::rsp, save_physics(), and vm_interpret().

Referenced by run_test_suite_m().

Here is the call graph for this function:



Here is the caller graph for this function:



7.339.1.4 contract_get_violation_count()

```
int contract_get_violation_count (
            void  )
```

Return the current count of accumulated contract violations.

Reads `global_contract_violations` directly; same-TU access avoids GOT double-dereference in `-fPIC` PE32+ builds where `R_X86_64_REX_GOTPCRELX` is not relaxed to a LEA instruction.

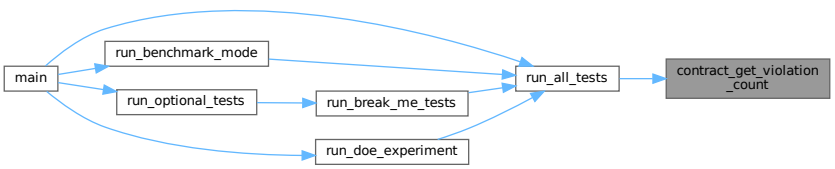
Returns

Number of contract violations recorded since last reset.

References global_contract_violations.

Referenced by run_all_tests().

Here is the caller graph for this function:



7.339.1.5 contract_rand()

```
static uint64_t contract_rand (
            void  )  [static]
```

Generate the next pseudo-random 64-bit value via xorshift64.

Uses the global `rng_state`. The xorshift64 parameters (13, 7, 17) produce a full-period sequence over all non-zero 64-bit states. Caller must seed via `rng_seed()` before starting a new fuzz round to ensure different perturbation sequences across rounds.

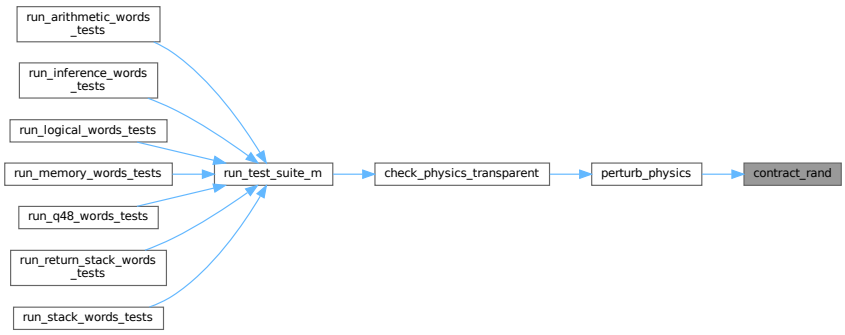
Returns

Next pseudo-random 64-bit integer.

References rng_state.

Referenced by perturb_physics().

Here is the caller graph for this function:



7.339.1.6 contract_reset_violation_count()

```
void contract_reset_violation_count (
            void  )
```

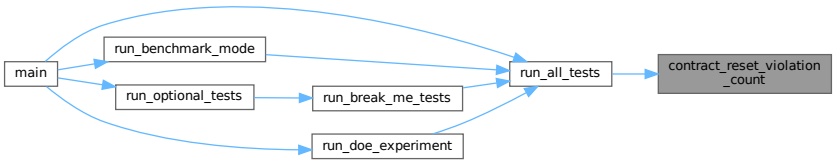
Reset the contract violation counter to zero.

Call before a fresh round of contract checks so that counts from earlier test phases do not accumulate across module boundaries.

References global_contract_violations.

Referenced by run_all_tests().

Here is the caller graph for this function:



7.339.1.7 exec_results_equal()

```
int exec_results_equal (
            const ExecResult * a,
            const ExecResult * b )
```

Compare two `ExecResult` snapshots for execution-domain equality.

Returns 1 only if `dsp`, `rsp`, `error`, and the live stack contents are all identical. Stack comparison uses `memcmp()` over the live portion (0 . . . dsp?1 cells); the zeroed tail guarantees no false positives from uninitialised padding.

Parameters

a	First snapshot.
b	Second snapshot.

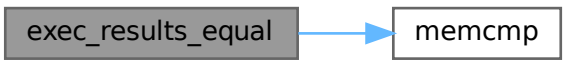
Returns

1 if execution-domain is identical, 0 if any field differs.

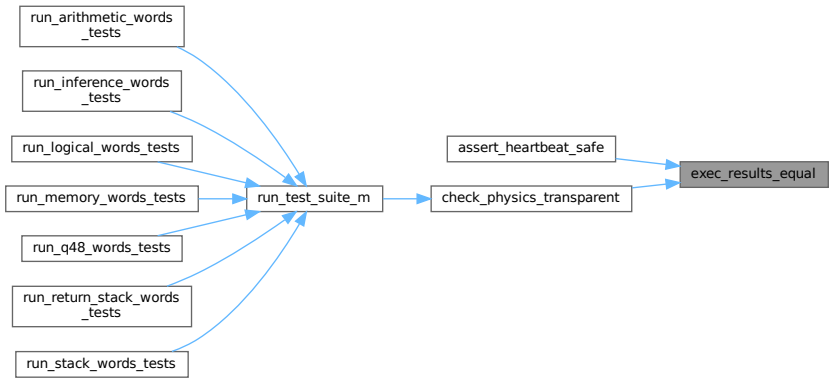
References ExecResult::dsp, ExecResult::error, memcmp(), ExecResult::rsp, and ExecResult::stack.

Referenced by assert_heartbeat_safe(), and check_physics_transparent().

Here is the call graph for this function:



Here is the caller graph for this function:



7.339.1.8 perturb_physics()

```
void perturb_physics (
            VM * vm,
            int round )
```

Randomly perturb key physics scalars to expose A4' violations.

Seeds the RNG via `rng_seed(round)` for deterministic but distinct perturbations per round, then overwrites the eight tracked physics fields with random values that still satisfy their well-formedness invariants:

- `decay_slope_q48`: LSB forced to 1 (must be $>$ 0 per `slope_wf`)
- `effective_window_size`: [1, ROLLING_WINDOW_SIZE] (never 0)
- `tick_target_ns`: LSB forced to 1 (must be $>$ 0 per `hb_state_wf`)
- Heat thresholds: arbitrary positive values (no strict invariant)
- `prefetch_attempts` / `total_executions`: arbitrary counts

After perturbation, running the same FORTH input must produce an identical `ExecResult`; divergence is an A4' violation.

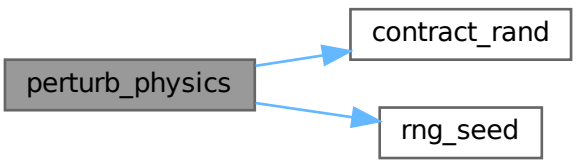
Parameters

vm	VM instance to perturb.
round	Fuzz round index (used to seed the RNG).

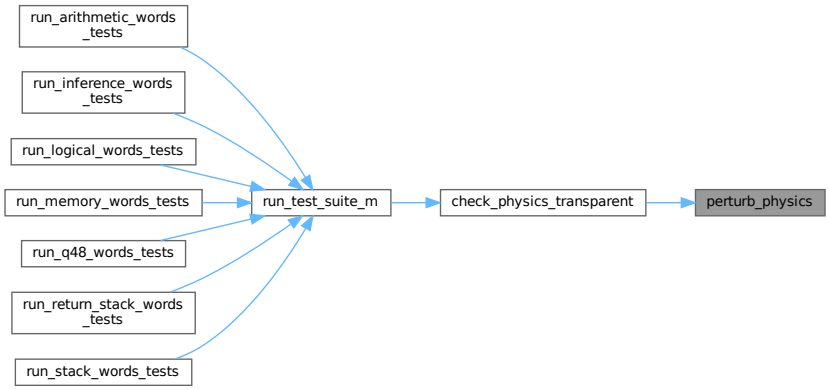
References contract_rand(), VM::decay_slope_q48, RollingWindowOfTruth::effective_window_size, VM::heartbeat, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch rng_seed(), VM::rolling_window, ROLLING_WINDOW_SIZE, HeartbeatState::tick_target_ns, and RollingWindowOfTruth::total_execu

Referenced by check_physics_transparent().

Here is the call graph for this function:



Here is the caller graph for this function:



7.339.1.9 restore_physics()

```
void restore_physics (
            VM * vm,
            const PhysicsSnapshot * s )
```

Restore the physics scalars from a previously saved snapshot.

Writes back the eight fields saved by `save_physics()`, undoing any perturbations made by `perturb_physics()`. Called at the end of each fuzz round and after the full `check_physics_transparent()` loop so the VM state is clean for the next contract check.

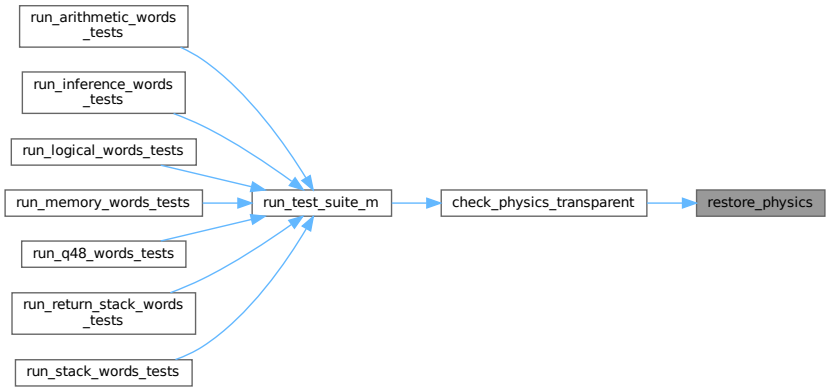
Parameters

vm	VM instance to restore.
s	Source `PhysicsSnapshot` to restore from.

References VM::decay_slope_q48, PhysicsSnapshot::decay_slope_q48, PhysicsSnapshot::eff_window_size, RollingWindowOfTruth::effective_window_size, VM::heartbeat, PhysicsSnapshot::heat_25th, PhysicsSnapshot::heat_50th, PhysicsSnapshot::heat_75th, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PhysicsSnapshot::prefetch_attempts, VM::rolling_window, HeartbeatState::tick_target_ns, PhysicsSnapshot::tick_target_ns, RollingWindowOfTruth::total_executions, and PhysicsSnapshot::total_executions.

Referenced by check_physics_transparent().

Here is the caller graph for this function:



7.339.1.10 rng_seed()

```
static void rng_seed (
            int round )  [static]
```

Seed the RNG for a given fuzz round.

Derives a deterministic seed from the round index mixed with a Fibonacci multiplier (`0x9E3779B97F4A7C15`) so each round produces a distinct, reproducible perturbation sequence regardless of how many rounds have previously run.

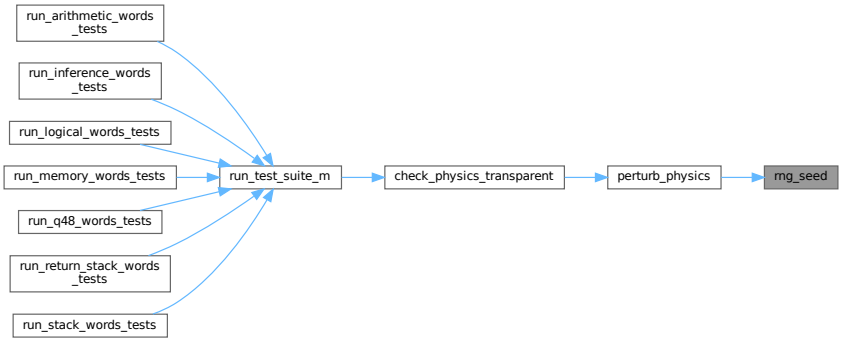
Parameters

round	Zero-based fuzz round index.

References rng_state.

Referenced by perturb_physics().

Here is the caller graph for this function:



7.339.1.11 save_physics()

```
void save_physics (
            VM * vm,
            PhysicsSnapshot * s )
```

Save the key physics scalars touched by the seven feedback loops.

Captures the eight fields that perturb_physics() will randomise: decay_slope_q48, effective_window_size, tick_target_ns, the three heat percentile thresholds, prefetch_attempts, and total_executions. Used as a bracket pair with restore_physics() around each fuzz round so the VM returns to a known state.

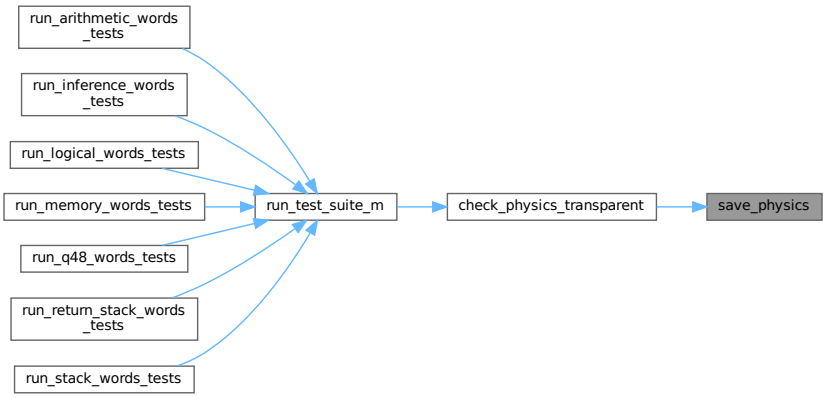
Parameters

vm	VM instance to snapshot.
s	Output PhysicsSnapshot to populate.

References VM::decay_slope_q48, PhysicsSnapshot::decay_slope_q48, PhysicsSnapshot::eff_window_size, RollingWindowOfTruth::effective_window_size, VM::heartbeat, PhysicsSnapshot::heat_25th, PhysicsSnapshot::heat_50th, PhysicsSnapshot::heat_75th, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PhysicsSnapshot::prefetch_attempts, VM::rolling_window, HeartbeatState::tick_target_ns, PhysicsSnapshot::tick_target_ns, RollingWindowOfTruth::total_executions, and PhysicsSnapshot::total_executions.

Referenced by check_physics_transparent().

Here is the caller graph for this function:



7.339.2 Variable Documentation

7.339.2.1 global_contract_violations

```
int global_contract_violations = 0
```

Referenced by assert_heartbeat_safe(), check_physics_transparent(), contract_get_violation_count(), and contract_reset_violation_count().

7.339.2.2 rng_state

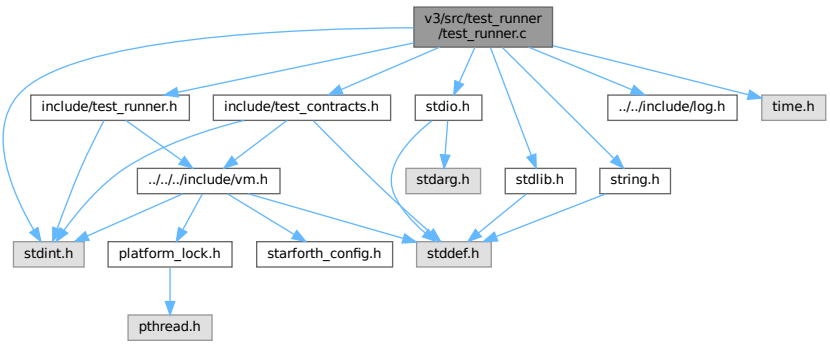
```
uint64_t rng_state = 0xDEADBEEFCAFEBABEULL  [static]
```

Referenced by contract_rand(), and rng_seed().

7.340 v3/src/test_runner/test_runner.c File Reference

```
#include "include/test_runner.h"
#include "include/test_contracts.h"
#include "../../include/log.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Include dependency graph for test_runner.c:



Functions

- void test_stats_accumulate (int tests, int pass, int fail, int skip, int error)
- static uint64_t get_time_ns (void)

  Get current time in nanoseconds.
- void enable_benchmark_mode (int iterations)

  Enable benchmark mode for test execution.
- void run_all_tests (VM *vm)

  Run all test modules with optional benchmarking.
- void run_module_tests (VM *vm, const char *module_name)

  Run tests for a specific module.
- void run_word_tests (VM *vm, const char *word_name)

  Run tests for a specific Forth word.

Variables

- TestStats global_test_stats = {0, 0, 0, 0, 0}

  Global statistics for test execution.
- static int benchmark_mode = 0

  Flag indicating if benchmark mode is enabled.
- static int benchmark_iterations = 1000

  Number of iterations for benchmark tests.
- static TestModule test_modules [ ]

7.340.1 Function Documentation

7.340.1.1 enable_benchmark_mode()

```
void enable_benchmark_mode (
            int iterations )
```

Enable benchmark mode for test execution.

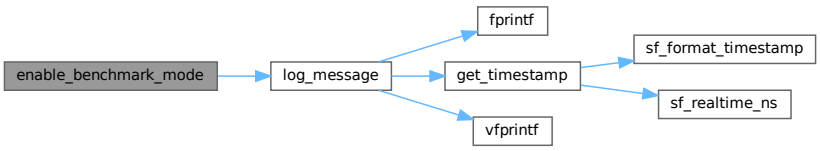
Parameters

iterations	Number of times each test should be repeated

References benchmark_iterations, benchmark_mode, LOG_INFO, and log_message().

Referenced by run_benchmark_mode().

Here is the call graph for this function:



Here is the caller graph for this function:

7.340.1.2 get_time_ns()

```
static uint64_t get_time_ns (
            void  )  [static]
```

Get current time in nanoseconds.

Uses platform-specific timing: sf_monotonic_ns() on kernel, clock() on hosted.

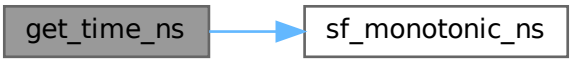
Returns

Current time in nanoseconds, or 0 on error

References sf_monotonic_ns().

Referenced by run_module_tests().

Here is the call graph for this function:



Here is the caller graph for this function:

7.340.1.3 run_all_tests()

```
void run_all_tests (
            VM * vm )
```

Run all test modules with optional benchmarking.

Executes all registered test modules sequentially, collecting statistics and/or benchmark data depending on the current mode.

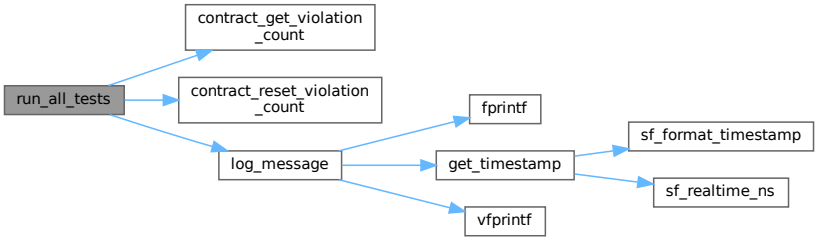
Parameters

vm	Pointer to the Forth virtual machine instance

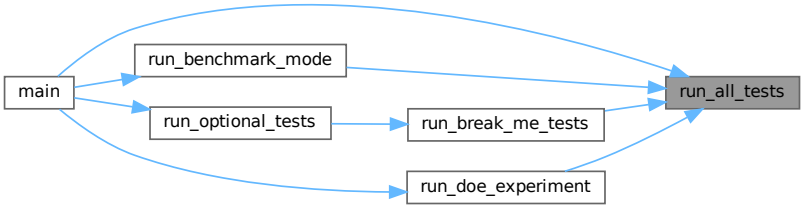
References contract_get_violation_count(), contract_reset_violation_count(), global_test_stats, LOG_ERROR, LOG_INFO, log_message(), LOG_WARN, TestModule::module_name, test_modules, TestStats::total_error, TestStats::total_fail, TestStats::total_pass, TestStats::total_skip, and TestStats::total_tests.

Referenced by main(), run_benchmark_mode(), run_break_me_tests(), and run_doe_experiment().

Here is the call graph for this function:



Here is the caller graph for this function:



7.340.1.4 run_module_tests()

```
void run_module_tests (
            VM * vm,
            const char * module_name )
```

Run tests for a specific module.

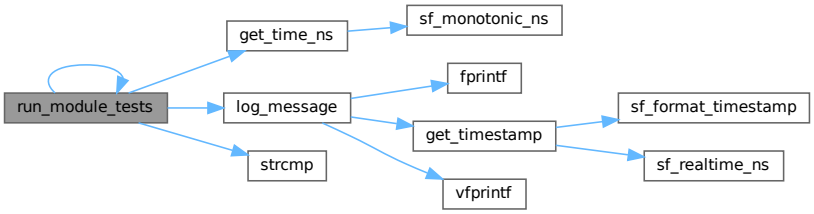
Parameters

vm	Pointer to the Forth virtual machine instance
module_name	Name of the module to test

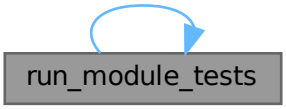
References benchmark_iterations, benchmark_mode, get_time_ns(), LOG_ERROR, LOG_INFO, log_message(), LOG_WARN, TestModule::module_name, run_module_tests(), TestModule::run_module_tests, strcmp(), and test_modules.

Referenced by run_module_tests().

Here is the call graph for this function:



Here is the caller graph for this function:



7.340.1.5 run_word_tests()

```
void run_word_tests (
            VM * vm,
            const char * word_name )
```

Run tests for a specific Forth word.

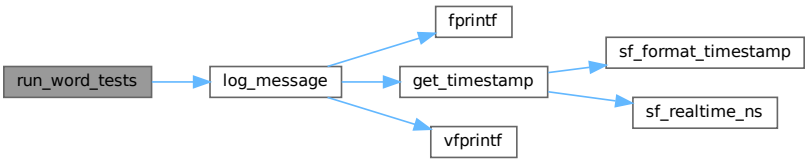
Searches all modules for tests related to the specified word and executes them.

Parameters

vm	Pointer to the Forth virtual machine instance
word_name	Name of the Forth word to test

References LOG_INFO, log_message(), LOG_WARN, TestModule::module_name, and test_modules.

Here is the call graph for this function:



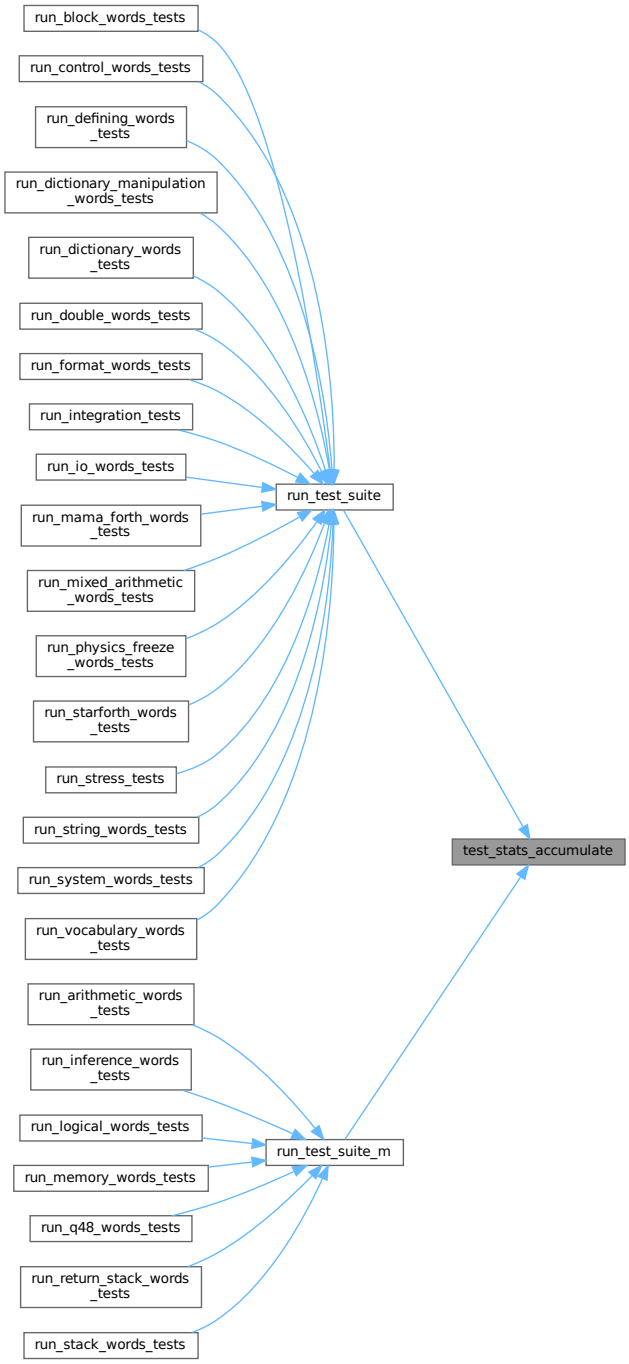
7.340.1.6 test_stats_accumulate()

```
void test_stats_accumulate (
            int tests,
            int pass,
            int fail,
            int skip,
            int error )
```

References global_test_stats, TestStats::total_error, TestStats::total_fail, TestStats::total_pass, TestStats::total_skip, and TestStats::total_tests.

Referenced by run_test_suite(), and run_test_suite_m().

Here is the caller graph for this function:



7.340.2 Variable Documentation

7.340.2.1 benchmark_iterations

```
int benchmark_iterations = 1000  [static]
```

Number of iterations for benchmark tests.

Referenced by enable_benchmark_mode(), and run_module_tests().

7.340.2.2 benchmark_mode

```
int benchmark_mode = 0  [static]
```

Flag indicating if benchmark mode is enabled.

Referenced by enable_benchmark_mode(), and run_module_tests().

7.340.2.3 global_test_stats

```
TestStats global_test_stats = {0, 0, 0, 0, 0}
```

Global statistics for test execution.

Referenced by run_all_tests(), run_break_me_tests(), run_stress_tests(), sk_parity_collect(), and test_stats_accumulate().

7.340.2.4 test_modules

```
TestModule test_modules[]  [static]
```

Initial value:

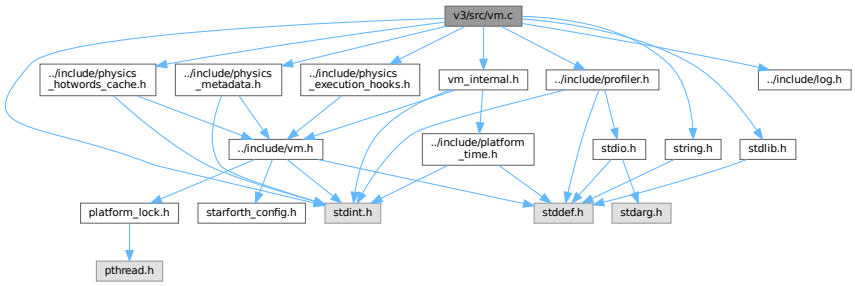
```
= {

    {"Stack Words", NULL, 0, run_stack_words_tests},
    {"Return Stack Words", NULL, 0, run_return_stack_words_tests},
    {"Memory Words", NULL, 0, run_memory_words_tests},
    {"Arithmetic Words", NULL, 0, run_arithmetic_words_tests},
    {"Logical Words", NULL, 0, run_logical_words_tests},
    {"Mixed Arithmetic Words", NULL, 0, run_mixed_arithmetic_words_tests},
    {"Double Words", NULL, 0, run_double_words_tests},
    {"Format Words", NULL, 0, run_format_words_tests},
    {"String Words", NULL, 0, run_string_words_tests},
    {"I/O Words", NULL, 0, run_io_words_tests},
    {"Block Words", NULL, 0, run_block_words_tests},
    {"Dictionary Words", NULL, 0, run_dictionary_words_tests},
    {"Dictionary Manipulation Words", NULL, 0, run_dictionary_manipulation_words_tests},
    {"Vocabulary Words", NULL, 0, run_vocabulary_words_tests},
    {"System Words", NULL, 0, run_system_words_tests},
    {"Defining Words", NULL, 0, run_defining_words_tests},
    {"Control Words", NULL, 0, run_control_words_tests},
    {"StarForth Words", NULL, 0, run_starforth_words_tests},
    {"ACL Words", NULL, 0, run_acl_words_tests},
    {"Mama FORTH Words", NULL, 0, run_mama_forth_words_tests},
    {"Q48.16 Words", NULL, 0, run_q48_words_tests},
    {"Inference Engine Words", NULL, 0, run_inference_words_tests},
    {"Physics Freeze Words", NULL, 0, run_physics_freeze_words_tests},
    {NULL, NULL, 0, NULL}
}
```

Referenced by run_all_tests(), run_module_tests(), and run_word_tests().

7.341 v3/src/vm.c File Reference

```
#include "vm_internal.h"
#include "../include/log.h"
#include "../include/profiler.h"
#include "../include/physics_metadata.h"
#include "../include/physics_execution_hooks.h"
#include "../include/physics_hotwords_cache.h"
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
```

Include dependency graph for vm.c:



Functions

- unsigned vm_get_base (const VM *vm)

  Read the current number base from the VM.
- void vm_set_base (VM *vm, unsigned b)

  Set the VM number base and update both the cell and the host mirror.
- int vm_zuse_cert_install (VM *vm, const uint8_t seed[32], const uint8_t pubkey[32])

  One-time write of the Zuse cert (seed + derived pubkey) – blows the fuse.
- int vm_parse_word (VM *vm, char *word, size_t max_len)

  Parse the next whitespace-delimited token from the input buffer.
- int vm_parse_number (VM *vm, const char *s, cell_t *out)

  Attempt to parse a string as an integer in the current number base.
- void vm_enter_compile_mode (VM *vm, const char *name, size_t len)

  Begin a colon definition and switch the VM to compile mode.
- void vm_compile_word (VM *vm, DictEntry *entry)

  Append a word reference (DictEntry) to the current threaded body.*
- void vm_compile_literal (VM *vm, cell_t value)

  Compile a literal value into the current threaded body (or push it).
- void vm_compile_call (VM *vm, word_func_t func)

  Compile a call to a word identified by its C function pointer.
- void vm_compile_exit (VM *vm)

  Compile an `EXIT` reference into the current threaded body.
- void vm_exit_compile_mode (VM *vm)

  Finalise a colon definition and return to interpret mode.
- static void acl_recheck (VM *vm, DictEntry *entry)

  Cold-path ACL enforcement called when a word's TTL counter hits zero.

- void execute_colon_word (VM *vm)

  Inner interpreter for threaded colon definitions.
- void vm_interpret_word (VM *vm, const char *word_str, size_t len)

  Interpret or compile a single word token.
- void vm_interpret (VM *vm, const char *input)
- int vm_addr_ok (struct VM *vm, vaddr_t addr, size_t len)

  Validate that a VM address range lies within `vm->memory`.
- uint8_t * vm_ptr (struct VM *vm, vaddr_t addr)

  Return a host C pointer into VM memory at `addr`.
- uint8_t vm_load_u8 (struct VM *vm, vaddr_t addr)

  Load an unsigned byte from VM memory.
- void vm_store_u8 (struct VM *vm, vaddr_t addr, uint8_t v)

  Store an unsigned byte to VM memory.
- cell_t vm_load_cell (struct VM *vm, vaddr_t addr)

  Load an aligned cell from VM memory.
- void vm_store_cell (struct VM *vm, vaddr_t addr, cell_t v)

  Store an aligned cell to VM memory.
- void vm_make_immediate (VM *vm)

  Mark the most recently created word as IMMEDIATE (FORTH-79).

Variables

- static int g_vm_interpret_depth

  Interpret a NUL-terminated string of FORTH source code.

7.341.1 Function Documentation

7.341.1.1 acl_recheck()

```
static void acl_recheck (
            VM * vm,
            DictEntry * entry )  [static]
```

Cold-path ACL enforcement called when a word's TTL counter hits zero.

Looks up the FORTH word `ACL-RECHECK` in the dictionary. If found, calls it with `emergency_console` temporarily set to 1 so the recheck itself cannot be denied by the ACL system. The FORTH word is expected to update `entry->acl_allow` and `entry->acl_ttl` before returning.

If `ACL-RECHECK` is not yet loaded (early boot before `ACL.4th` runs), the function **fails open**: sets `acl_allow=1` and `acl_ttl=ACL_TTL_OPEN` so execution continues unimpeded. The same fail-open behaviour applies if `ACL-RECHECK` itself raises an error (error flag is cleared).

This function is the cold path; the hot path in `execute_colon_word()` and `vm_interpret_word()` simply decrements `acl_ttl`.

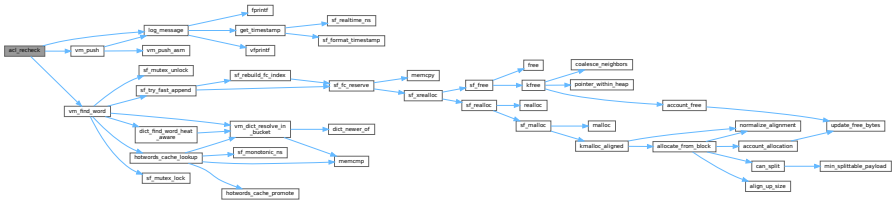
Parameters

vm	VM instance.
entry	Dictionary entry whose TTL has expired and needs rechecking.

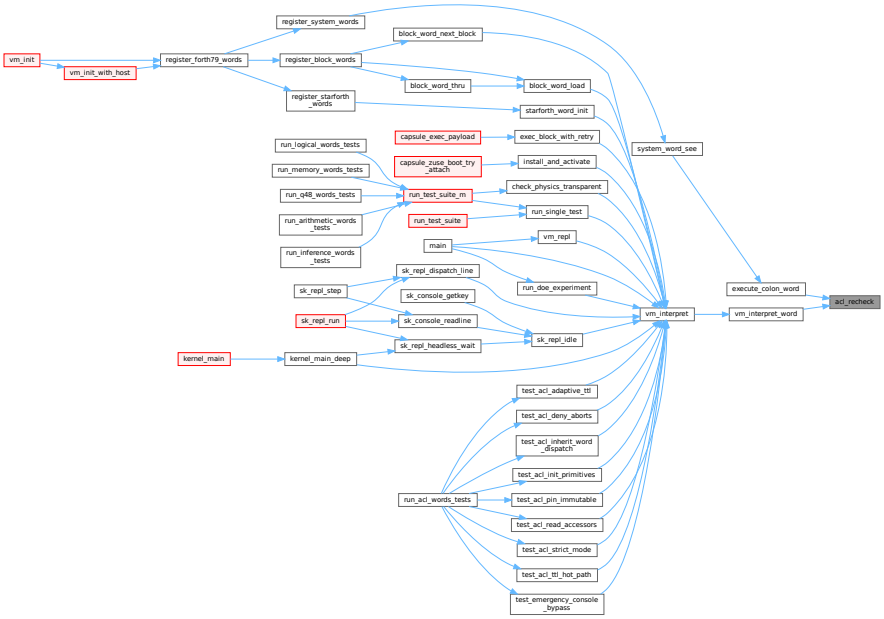
References ACL_TTL_OPEN, VM::current_executing_entry, VM::emergency_console, entry, VM::error, DictEntry::func, log_message(), LOG_WARN, vm_find_word(), and vm_push().

Referenced by execute_colon_word(), and vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.341.1.2 execute_colon_word()

```
void execute_colon_word (
            VM * vm )
```

Inner interpreter for threaded colon definitions.

Runs the body of a colon-defined word whose `DictEntry` points to a sequence of `DictEntry*` cells in VM memory (indirect threaded code). The interpreter pointer (IP) is maintained on the return stack so that runtime control-flow words ((BRANCH), (0BRANCH), (DO), etc.) can redirect execution by modifying the top-of-return-stack IP.

Execution loop per iteration:

1. Read `DictEntry* w` from `*ip`.
2. Run physics pre-execute hooks (heat, decay, pipelining).
3. ACL hot-path check (decrement TTL; call `acl_recheck()` on zero).
4. Push `ip+1` onto the return stack as the resume point.
5. Call `w->func(vm)`.
6. Run physics post-execute hooks.
7. Check `vm->exit_colon` (set by `EXIT`): pop and discard the saved IP, then return to caller.
8. Check `vm->abort_requested`: unwind and return immediately.
9. Pop the (possibly modified) IP from the return stack and repeat.

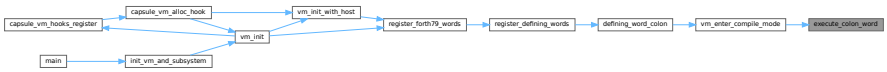
`vm->ecw_nesting` is incremented on entry and decremented on every return path to enable recursive colon calls.

Parameters

vm	VM instance; `vm->current_executing_entry` must point to the `DictEntry` being executed.

Referenced by vm_enter_compile_mode().

Here is the caller graph for this function:



7.341.1.3 vm_addr_ok()

```
int vm_addr_ok (
            struct VM * vm,
            vaddr_t addr,
            size_t len )
```

Validate that a VM address range lies within `vm->memory`.

Checks that [addr, addr+len) is fully contained within the `VM_MEMORY_SIZE`-byte flat address space. The check is done with unsigned arithmetic to avoid UB on overflow: if `len` alone exceeds `VM_MEMORY_SIZE` the range is immediately rejected.

Parameters

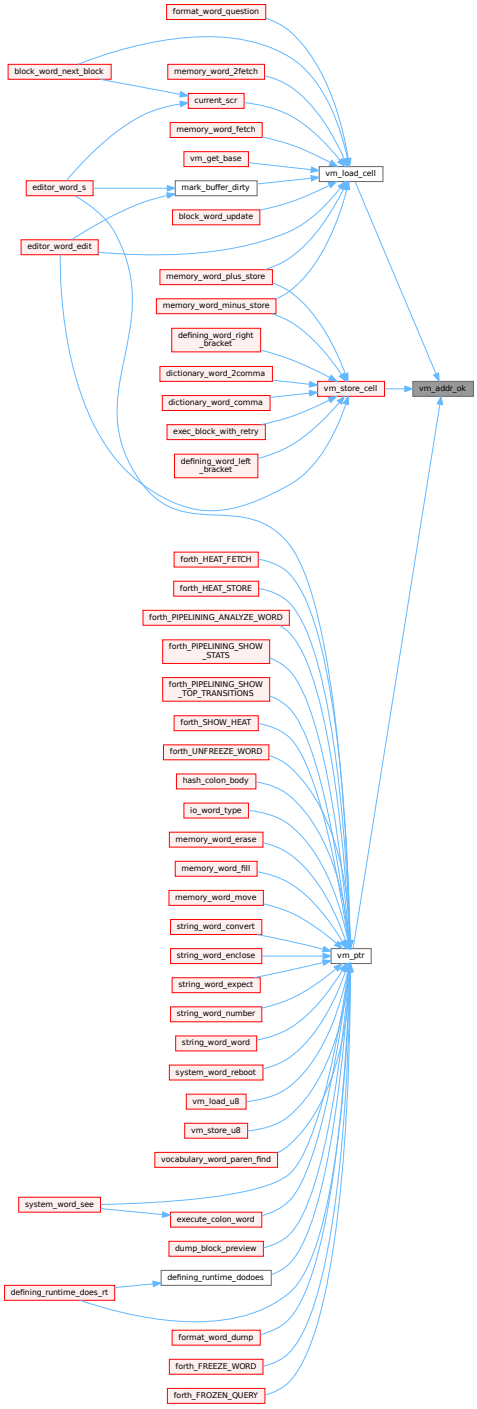
vm	VM instance; returns 0 if NULL or memory not allocated.
addr	Start of the range (inclusive).
len	Number of bytes in the range.

Returns

1 if the full range [addr, addr+len) is in bounds; 0 otherwise.

Referenced by vm_load_cell(), vm_ptr(), and vm_store_cell().

Here is the caller graph for this function:



7.341.1.4 vm_compile_call()

```
void vm_compile_call (
            VM * vm,
            word_func_t func )
```

Compile a call to a word identified by its C function pointer.

Looks up the DictEntry whose func field matches func via vm_dictionary_find_by_func(), then delegates to vm_compile_word(). Used by generating words (e.g., DOES>) that need to embed a specific runtime routine into a compiled definition without knowing its name.

Sets vm->error if not in compile mode or if no matching entry is found.

Parameters

vm	VM instance in compile mode.
func	C function pointer identifying the word to compile.

7.341.1.5 vm_compile_exit()

```
void vm_compile_exit (
            VM * vm )
```

Compile an EXIT reference into the current threaded body.

Appends the EXIT word entry so that when the threaded body is executed, execute_colon_word() sets vm->exit_colon and unwinds the inner loop. Used by control-flow generating words (e.g., IF optimisation paths) that need an explicit early-return.

Sets vm->error if not in compile mode or if EXIT is not found.

Parameters

vm	VM instance in compile mode.

7.341.1.6 vm_compile_literal()

```
void vm_compile_literal (
            VM * vm,
            cell_t value )
```

Compile a literal value into the current threaded body (or push it).

In compile mode, appends the LIT word reference followed by the literal value so the inner interpreter will push value at runtime. In interpret mode, pushes value directly onto the data stack.

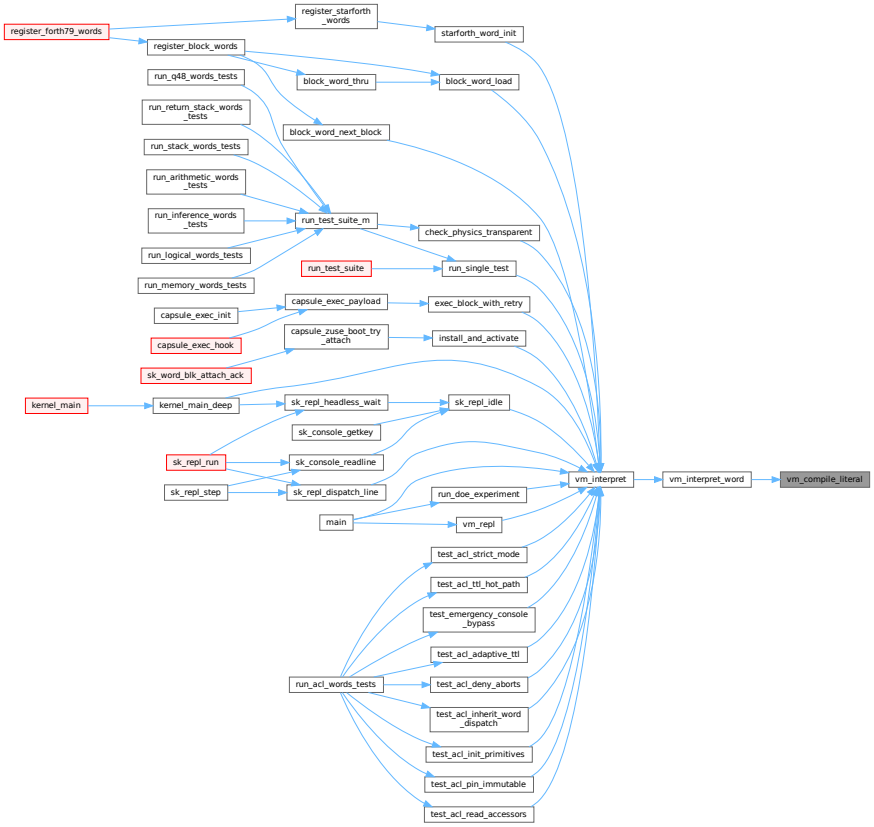
Sets vm->error if LIT cannot be found in the dictionary.

Parameters

vm	VM instance.
value	Cell value to compile or push.

Referenced by vm_interpret_word().

Here is the caller graph for this function:



7.341.1.7 vm_compile_word()

```
void vm_compile_word (
            VM * vm,
            DictEntry * entry )
```

Append a word reference (DictEntry*) to the current threaded body.

Aligns `HERE`, then allots one `cell_t` slot and stores the `DictEntry*` as a cell. This is the fundamental compilation step used by the inner interpreter: each slot in the threaded body is a pointer to the `DictEntry` of the word to call.

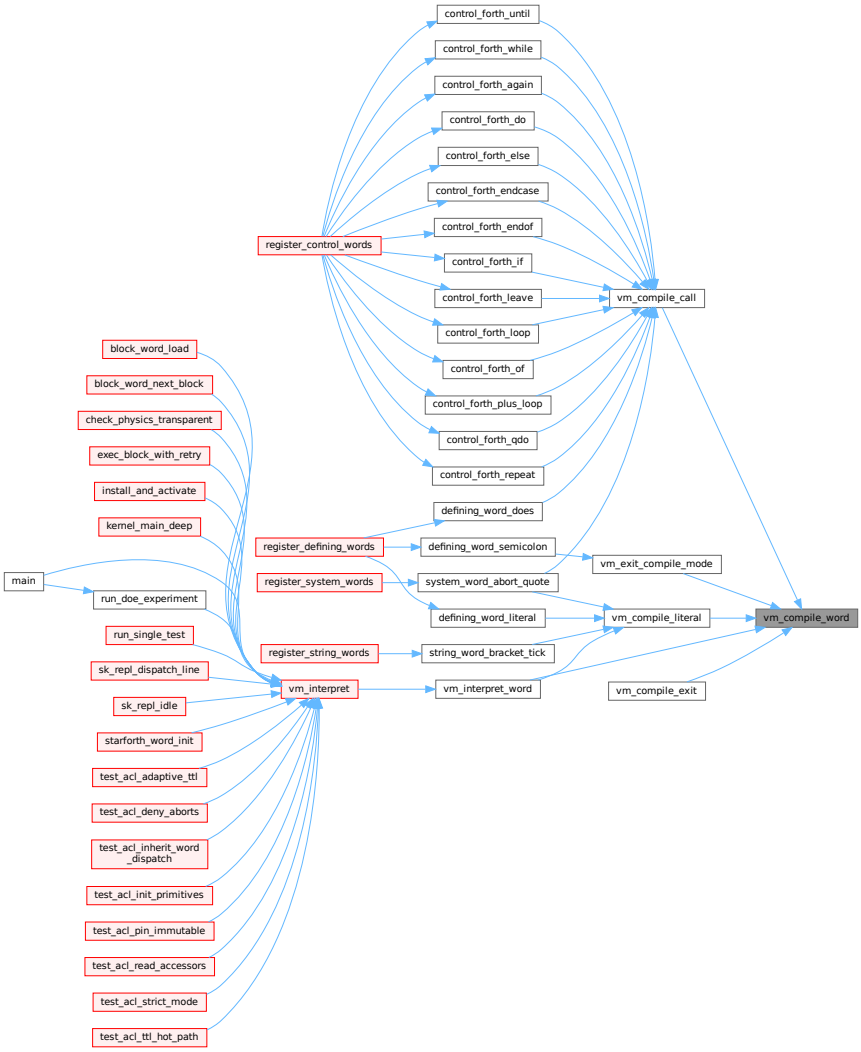
Sets `vm->error` if the VM is not in compile mode, `entry` is NULL, or allotment fails.

Parameters

vm	VM instance in compile mode.
entry	Dictionary entry of the word to compile; must not be NULL.

Referenced by vm_compile_call(), vm_compile_exit(), vm_compile_literal(), vm_exit_compile_mode(), and vm_interpret_word().

Here is the caller graph for this function:



7.341.1.8 vm_enter_compile_mode()

```
void vm_enter_compile_mode (
            VM * vm,
            const char * name,
            size_t len )
```

Begin a colon definition and switch the VM to compile mode.

Creates a new dictionary entry for `name` via `vm_create_word()`, marks it `WORD_SMUDGED` so it is invisible to lookups during compilation, sets the VM mode to `MODE_COMPILE`, and writes the current `HERE` address into the entry's data field as the start of the threaded body.

The SMUDGE flag is cleared by `vm_exit_compile_mode()` when the semicolon is encountered.

Parameters

vm	VM instance.
name	Word name string (does not need to be NUL-terminated).
len	Length of `name`; clamped to `WORD_NAME_MAX`.

7.341.1.9 vm_exit_compile_mode()

```
void vm_exit_compile_mode (
            VM * vm )
```

Finalise a colon definition and return to interpret mode.

Appends a final `EXIT` reference, clears the `WORD_SMUDGED` flag so the new word becomes visible in the dictionary, and sets `WORD_COMPILED`. Also computes the word's mass (header + body bytes) for the physics engine and calls `physics_metadata_refresh_state()`.

Resets `vm->mode` to `MODE_INTERPRET`, clears `vm->state_var`, and NULLs `vm->compiling_word`.

Sets `vm->error` if `vm` has no active `compiling_word` or if `EXIT` cannot be found.

Parameters

vm	VM instance currently in compile mode.

7.341.1.10 vm_get_base()

```
unsigned vm_get_base (
            const VM * vm )
```

Read the current number base from the VM.

Prefers the cell stored at `vm->base_addr` (the canonical FORTH `BASE` variable) if it is aligned and in the valid range [2, 36]. Falls back to the host-mirror `vm->base` if the cell is out of range, and finally returns 10 if both are invalid or `vm` is NULL.

The double-read (cell + mirror) keeps the VM consistent when C code modifies `vm->base` directly without going through `vm_set_base()`.

Parameters

vm	VM instance to query (may be NULL; returns 10 in that case).

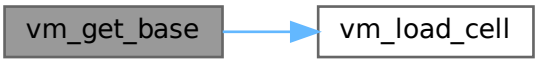
Returns

Current number base in range [2, 36].

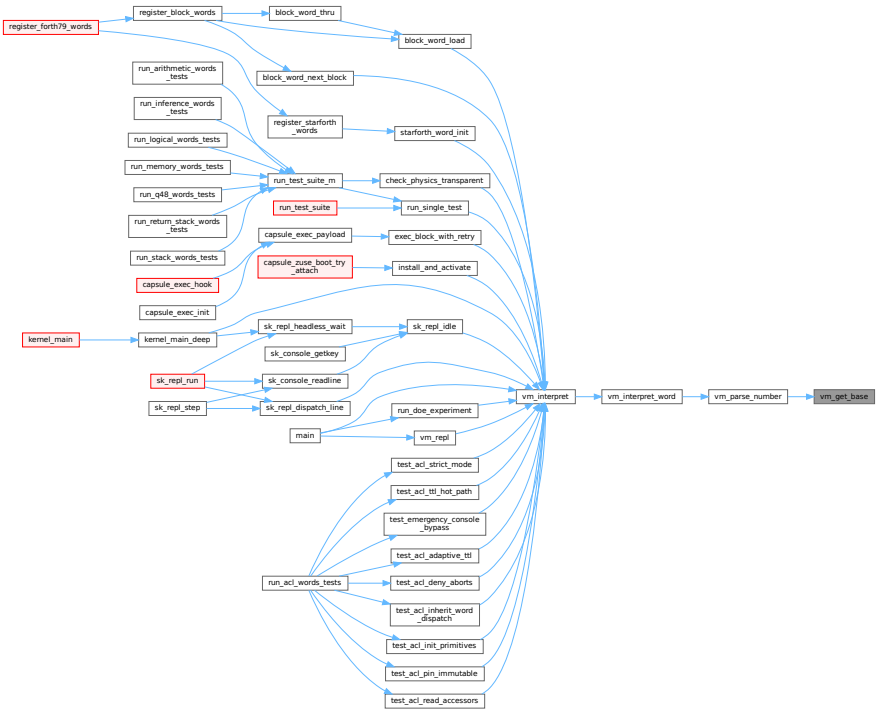
References VM::base, VM::base_addr, vm_load_cell(), and VM_MEMORY_SIZE.

Referenced by vm_parse_number().

Here is the call graph for this function:



Here is the caller graph for this function:



7.341.1.11 vm_interpret()

```
void vm_interpret (
            VM * vm,
            const char * input )
```

7.341.1.12 vm_interpret_word()

```
void vm_interpret_word (
            VM * vm,
            const char * word_str,
            size_t len )
```

Interpret or compile a single word token.

This is the outer interpreter's per-token dispatch:

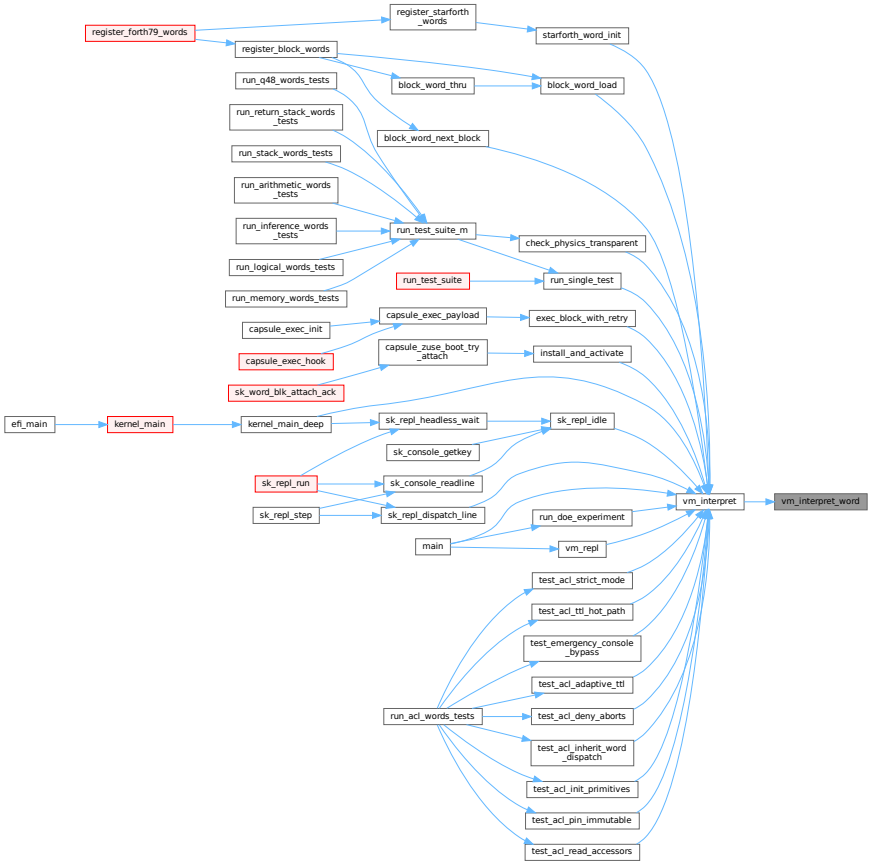
1. **Dictionary lookup**: vocabulary-aware search, falling back to the canonical flat dictionary. Physics heat is recorded via `physics_on_lookup()`.
2. **Compile mode, non-immediate**: append the entry to the current threaded body via `vm_compile_word()`.
3. **Interpret mode, or immediate in compile mode**: ACL check (hot-path TTL decrement / cold-path `acl_recheck()`), then call `entry->func()`.
4. **Number literal**: if no dictionary match, attempt `vm_parse_number()`. In compile mode, compile as a literal; in interpret mode, push directly.
5. **Error**: unknown token — sets `vm->error`.

Parameters

vm	VM instance.
word_str	Token string (need not be NUL-terminated).
len	Length of `word_str` in bytes.

Referenced by vm_interpret().

Here is the caller graph for this function:



7.341.1.13 vm_load_cell()

```
cell_t vm_load_cell (
            struct VM * vm,
            vaddr_t addr )
```

Load an aligned cell from VM memory.

Validates that `addr` is within bounds for a full `cell_t` read and that it is naturally aligned (addr % sizeof(cell_t) == 0). Uses `memcpy()` rather than a direct pointer dereference to avoid strict aliasing and alignment UB. Sets `vm->error` and returns 0 on failure.

Parameters

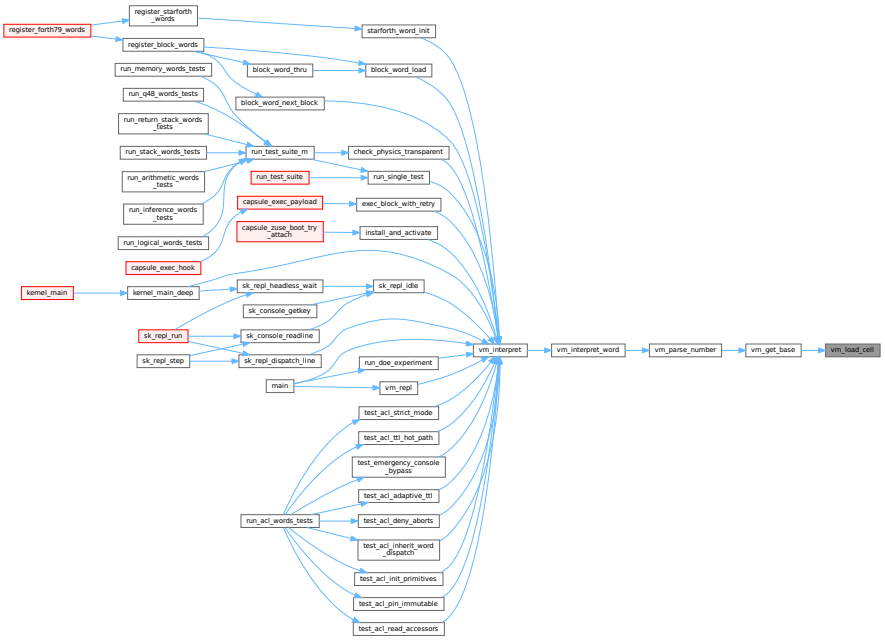
vm	VM instance.
addr	Aligned VM byte offset to read.

Returns

Cell value at `addr`, or 0 on alignment or bounds error.

Referenced by vm_get_base().

Here is the caller graph for this function:



7.341.1.14 vm_load_u8()

```
uint8_t vm_load_u8 (
            struct VM * vm,
            vaddr_t addr )
```

Load an unsigned byte from VM memory.

Resolves `addr` via `vm_ptr()`. Sets `vm->error` and returns 0 if the address is out of bounds.

Parameters

vm	VM instance.
addr	VM byte offset to read.

Returns

Byte value at `addr`, or 0 on error.

7.341.1.15 vm_make_immediate()

```
void vm_make_immediate (
            VM * vm )
```

Mark the most recently created word as IMMEDIATE (FORTH-79).

Sets `WORD_IMMEDIATE` on `vm->latest` so the word executes during compilation rather than being compiled into the current definition. Also calls `physics_metadata_refresh_state()` to update the physics engine's record of the entry's flags.

Sets `vm->error` if `vm` has no `latest` entry (i.e., the dictionary is empty or no word has been created yet).

Parameters

vm	VM instance.

7.341.1.16 vm_parse_number()

```
int vm_parse_number (
            VM * vm,
            const char * s,
            cell_t * out )
```

Attempt to parse a string as an integer in the current number base.

Reads the current base from `vm_get_base()`, then converts each character of `s` (case-insensitive, digits 0–9 and letters A–Z/a–z) to a digit value. Returns 0 immediately on any character that is not a valid digit in the current base, or on an empty string after the optional sign.

Handles an optional leading `'+'` or `'-'` sign. The result is a full-width `cell_t` (sign extended for negative values).

Parameters

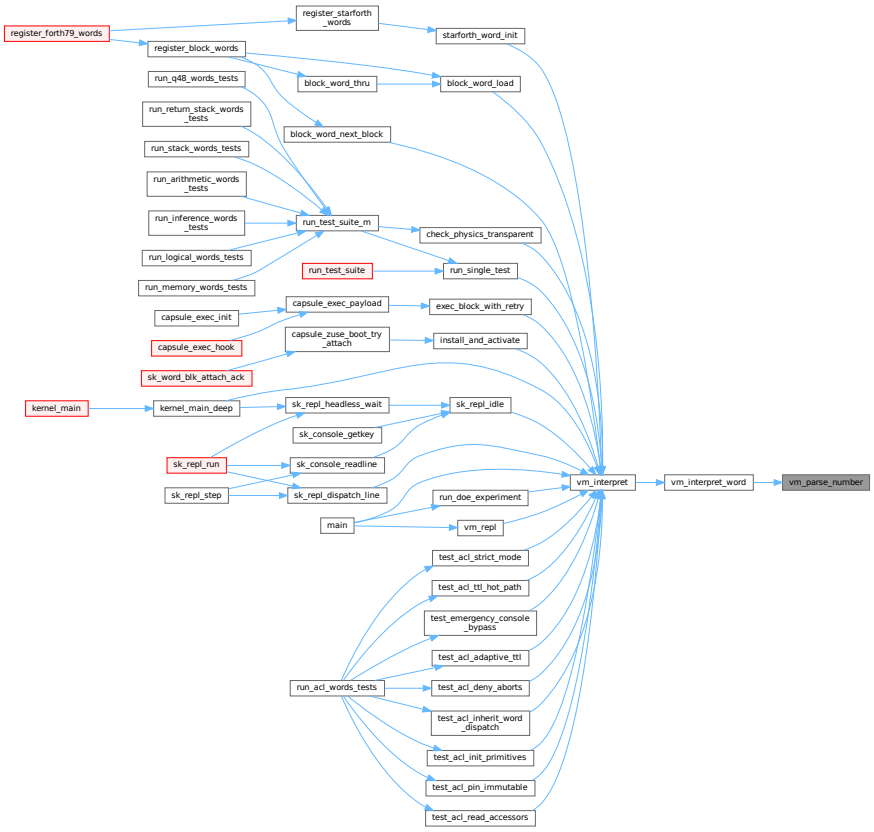
vm	VM instance (used for base only; may be NULL, in which case base defaults to 10).
s	NUL-terminated string to parse.
out	Pointer to receive the parsed value on success.

Returns

1 on successful parse, 0 on any parse failure.

Referenced by vm_interpret_word().

Here is the caller graph for this function:



7.341.1.17 vm_parse_word()

```
int vm_parse_word (
            VM * vm,
            char * word,
            size_t max_len )
```

Parse the next whitespace-delimited token from the input buffer.

Skips leading whitespace (space, tab, CR, LF), then extracts characters until the next whitespace or end of buffer, storing up to `max_len` ? 1 characters in `word` followed by a NUL terminator. Advances `vm->input_pos` past the consumed token.

Parameters

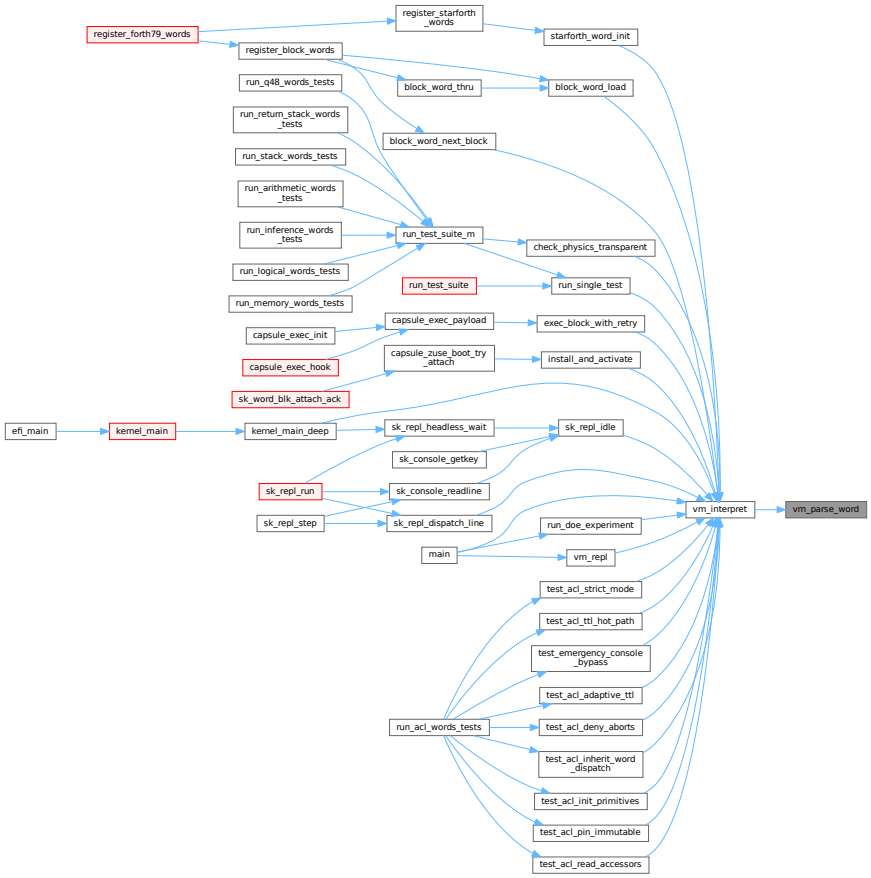
vm	VM instance; `vm->input_buffer` / `input_pos` / `input_length` must be populated (typically by `vm_interpret()`).
word	Output buffer for the parsed token.
max_len	Size of `word` in bytes (including NUL terminator).

Returns

Number of characters written (excluding NUL), or 0 if the buffer is exhausted or `vm` / `word` is NULL.

Referenced by vm_interpret().

Here is the caller graph for this function:



7.341.1.18 vm_ptr()

```
uint8_t * vm_ptr (
            struct VM * vm,
            vaddr_t addr )
```

Return a host C pointer into VM memory at `addr`.

Validates the address with `vm_addr_ok()` (minimum 1-byte range) and returns `NULL` if the address is out of bounds or `vm` is uninitialised. The returned pointer is valid for at least one byte; callers that need multi-byte access should perform their own range check first.

Parameters

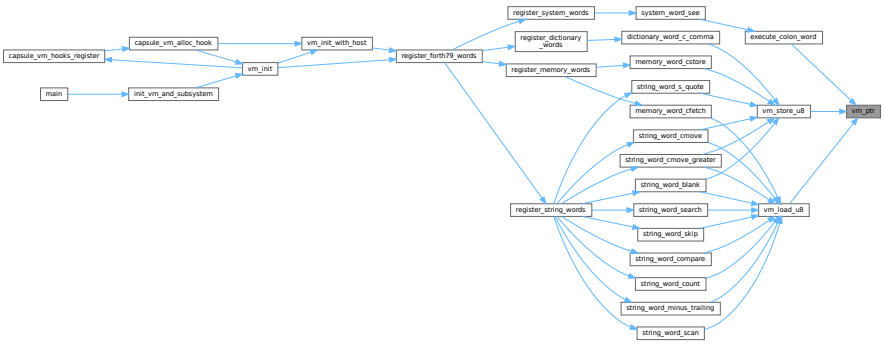
vm	VM instance.
addr	VM byte offset to dereference.

Returns

Host pointer into `vm->memory`, or NULL on bounds failure.

Referenced by execute_colon_word(), vm_load_u8(), and vm_store_u8().

Here is the caller graph for this function:



7.341.1.19 vm_set_base()

```
void vm_set_base (
            VM * vm,
            unsigned b )
```

Set the VM number base and update both the cell and the host mirror.

Clamps `b` to [2, 36] (replacing out-of-range values with 10), then writes to the FORTH `BASE` variable cell at `vm->base_addr` and to the host-mirror field `vm->base` so both sources agree.

Parameters

vm	VM instance to update.
b	Desired number base; clamped to [2, 36].

7.341.1.20 vm_store_cell()

```
void vm_store_cell (
            struct VM * vm,
            vaddr_t addr,
            cell_t v )
```

Store an aligned cell to VM memory.

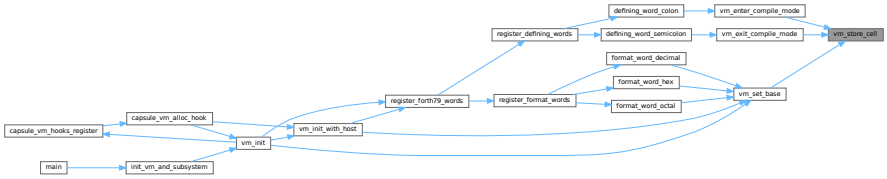
Validates alignment and bounds, then copies `v` into VM memory via `memcpy()`. Sets `vm->error` on alignment or bounds failure.

Parameters

vm	VM instance.
addr	Aligned VM byte offset to write.
v	Cell value to store.

Referenced by vm_enter_compile_mode(), vm_exit_compile_mode(), and vm_set_base().

Here is the caller graph for this function:



7.341.1.21 vm_store_u8()

```
void vm_store_u8 (
            struct VM * vm,
            vaddr_t addr,
            uint8_t v )
```

Store an unsigned byte to VM memory.

Resolves `addr` via `vm_ptr()`. Sets `vm->error` if the address is out of bounds; otherwise writes `v`.

Parameters

vm	VM instance.
addr	VM byte offset to write.
v	Byte value to store.

7.341.1.22 vm_zuse_cert_install()

```
int vm_zuse_cert_install (
            VM * vm,
            const uint8_t seed[32],
            const uint8_t pubkey[32] )
```

One-time write of the Zuse cert (seed + derived pubkey) – blows the fuse.

Deliberately not backed by a dictionary CONSTANT: ACL-PIN only blocks redefinition (vm_create_word shadowing), not a >BODY-then-store on the word's data field, so a pinned CONSTANT is not actually immutable against FORTH-level tampering. Keeping the value in a VM struct field with no corresponding FORTH store word closes that path entirely – see FABRIC-2.md's Milestone 4 mint-then-pin writeup for the finding.

Parameters

vm	VM instance.
seed	Ed25519 seed, 32 bytes (the private identity).
pubkey	Ed25519 public key derived from seed, 32 bytes.

Returns

0 on success; -1 if already installed (fuse already blown).

7.341.2 Variable Documentation

7.341.2.1 g_vm_interpret_depth

```
int g_vm_interpret_depth  [static]
```

Interpret a NUL-terminated string of FORTH source code.

Copies `input` into `vm->input_buffer` (capped at `INPUT_BUFFER_SIZE` ? 1 bytes, NUL-terminated), then drives the word-parse / token-dispatch loop until the buffer is exhausted or `vm->error` is set. Each token is dispatched to `vm_interpret_word()`.

This is the top-level entry point used by the REPL, `INCLUDE`, and inline `-c` execution. It does not reset `vm->error` before running — the caller is responsible for clearing error state if desired.

Parameters

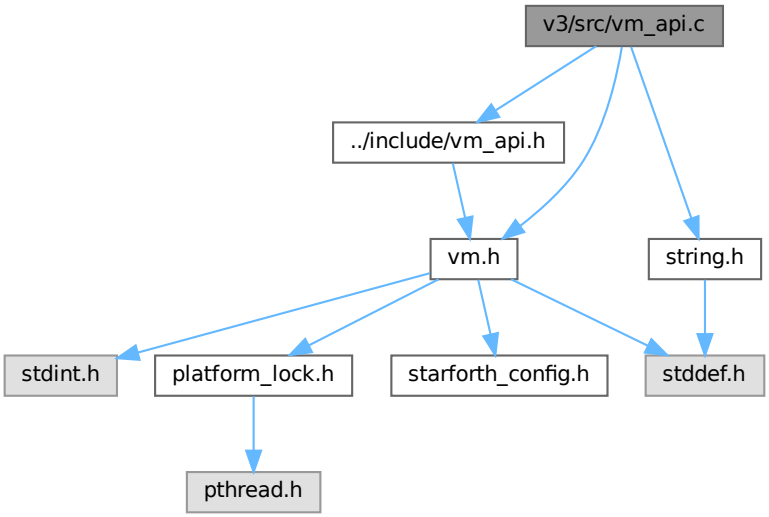
vm	VM instance.
input	NUL-terminated FORTH source string to interpret.

Referenced by vm_interpret().

7.342 v3/src/vm_api.c File Reference

```
#include "../include/vm_api.h"
#include "../include/vm.h"
#include <string.h>
```

Include dependency graph for vm_api.c:



Functions

- int vm_input_ensure (VM *vm)

  Ensure input buffers are allocated.
- unsigned char * vm_input_tib (VM *vm)

  Get pointer to terminal input buffer (TIB)
- cell_t * vm_input_in (VM *vm)

  Get pointer to input offset variable.
- cell_t * vm_input_span (VM *vm)

  Get pointer to input span variable.
- void vm_input_load_line (VM *vm, const char *src, size_t n)

  Load input line into TIB.
- void vm_input_source (VM *vm, cell_t *out_addr, cell_t *out_len)

  Get current input source buffer info.

Stack Operations

- void vm_api_push (VM *vm, intptr_t v)

  Push value onto data stack.
- intptr_t vm_api_pop (VM *vm)

  Pop value from data stack.
- void vm_api_rpush (VM *vm, intptr_t v)

  Push value onto return stack.
- intptr_t vm_api_rpop (VM *vm)

  Pop value from return stack.

Dictionary Operations

- void * vm_api_create_word (VM *vm, const char *n, size_t len, word_func_t fn)

Create new word in dictionary.

- void * vm_api_find_word (VM *vm, const char *n, size_t len)
  Find word in dictionary.
- void vm_api_hide_word (VM *vm)
  Hide last defined word.
- void vm_api_smudge_word (VM *vm)
  Toggle smudge bit of last word.

Memory Operations

- cell_t vm_addr_from_ptr (VM *vm, void *p)
  Convert native pointer to Forth memory address.

7.342.1 Function Documentation

7.342.1.1 vm_api_create_word()

```
void * vm_api_create_word (
            VM * vm,
            const char * n,
            size_t len,
            word_func_t fn )
```

Create new word in dictionary.

Parameters

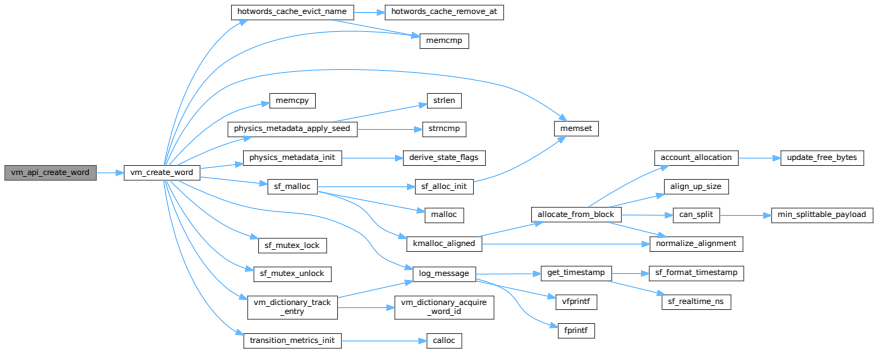
vm	VM instance
n	Word name
len	Name length
fn	Word implementation function

Returns

Pointer to created dictionary entry

References vm_create_word().

Here is the call graph for this function:



7.342.1.2 vm_api_find_word()

```
void * vm_api_find_word (
            VM * vm,
            const char * n,
            size_t len )
```

Find word in dictionary.

Parameters

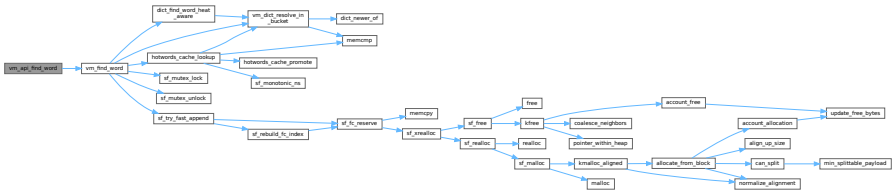
vm	VM instance
n	Word name to find
len	Name length

Returns

Pointer to found dictionary entry or NULL

References vm_find_word().

Here is the call graph for this function:



7.342.1.3 vm_api_hide_word()

```
void vm_api_hide_word (
            VM * vm )
```

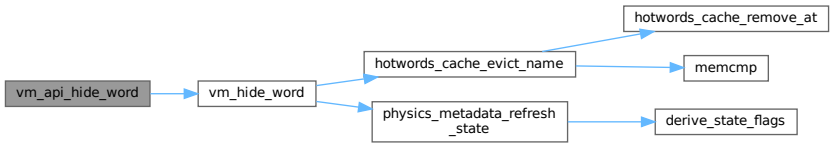
Hide last defined word.

Parameters

vm	VM instance

References vm_hide_word().

Here is the call graph for this function:



7.342.1.4 vm_api_pop()

```
intptr_t vm_api_pop (
            VM * vm )
```

Pop value from data stack.

Parameters

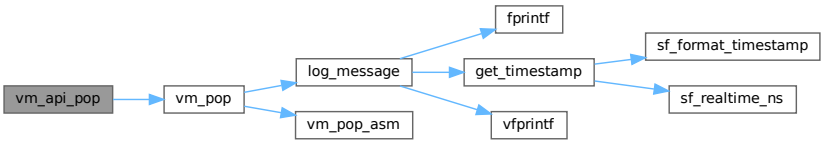
vm	VM instance

Returns

Popped value

References vm_pop().

Here is the call graph for this function:



7.342.1.5 vm_api_push()

```
void vm_api_push (
            VM * vm,
            intptr_t v )
```

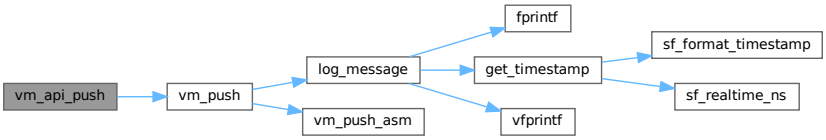
Push value onto data stack.

Parameters

vm	VM instance
v	Value to push

References vm_push().

Here is the call graph for this function:



7.342.1.6 vm_api_rpop()

```
intptr_t vm_api_rpop (
            VM * vm )
```

Pop value from return stack.

Parameters

vm	VM instance

Returns

Popped value

References vm_rpop().

Here is the call graph for this function:

7.342.1.7 vm_api_rpush()

```
void vm_api_rpush (
            VM * vm,
            intptr_t v )
```

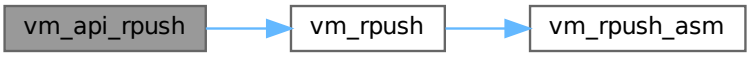
Push value onto return stack.

Parameters

vm	VM instance
v	Value to push

References vm_rpush().

Here is the call graph for this function:



7.342.1.8 vm_api_smudge_word()

```
void vm_api_smudge_word (
            VM * vm )
```

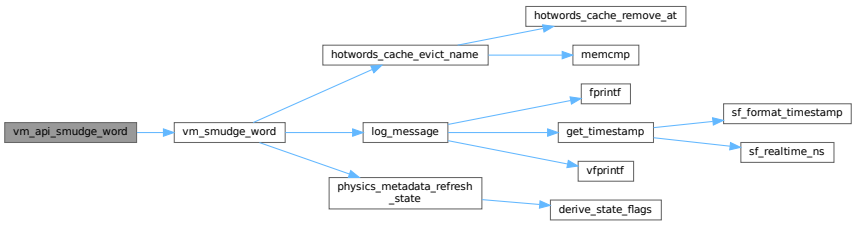
Toggle smudge bit of last word.

Parameters

vm	VM instance

References vm_smudge_word().

Here is the call graph for this function:

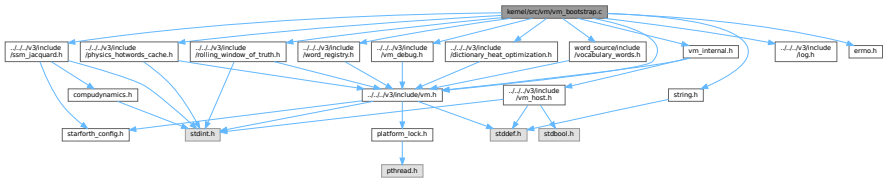


7.343 kernel/src/vm/vm_bootstrap.c File Reference

```
#include "../../../v3/include/vm.h"
#include "../../../v3/include/log.h"
#include "../../../v3/include/word_registry.h"
```

```
#include "../../../v3/include/vm_debug.h"
#include "../../../v3/include/physics_hotwords_cache.h"
#include "../../../v3/include/rolling_window_of_truth.h"
#include "../../../v3/include/dictionary_heat_optimization.h"
#include "../../../v3/include/ssm_jacquard.h"
#include "word_source/include/vocabulary_words.h"
#include "vm_internal.h"
#include <string.h>
#include <errno.h>
```

Include dependency graph for vm_bootstrap.c:



Functions

- static void vm_bootstrap_scr (VM *vm)
- void vm_bootstrap_root_vocabulary (VM *vm, const char *name)

  Bootstrap a root vocabulary for the VM.
- void vm_init (VM *vm)

  Initialize a new virtual machine instance.
- void vm_init_with_host (VM *vm, const VMHostServices *host)

  Initialize VM with explicit host services.

7.343.1 Function Documentation

7.343.1.1 vm_bootstrap_root_vocabulary()

```
void vm_bootstrap_root_vocabulary (
            VM * vm,
            const char * name )
```

Bootstrap a root vocabulary for the VM.

Creates a vocabulary by name, finds its dictionary entry, executes it to make it current, then calls DEFINITIONS.

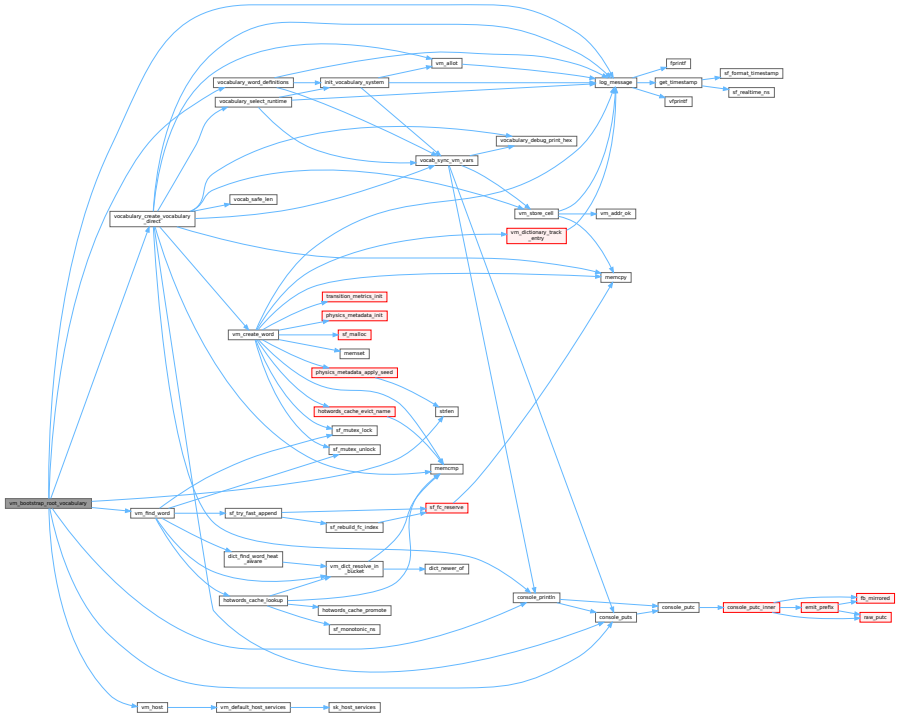
Parameters

vm	Pointer to VM structure
name	Vocabulary name to bootstrap

References console_println(), console_puts(), entry, VM::error, VMHostServices::is_executable_ptr, LOG_ERROR, log_message(), name, VMHostServices::panic, strlen(), vm_find_word(), vm_host(), vocabulary_create_vocabulary_direct(), and vocabulary_word_definitions().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

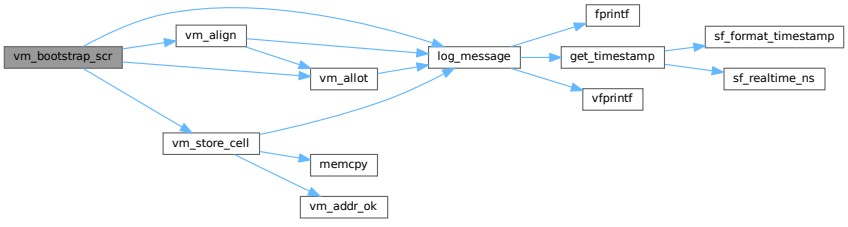
7.343.1.2 vm_bootstrap_scr()

```
static void vm_bootstrap_scr (
            VM * vm )  [static]
```

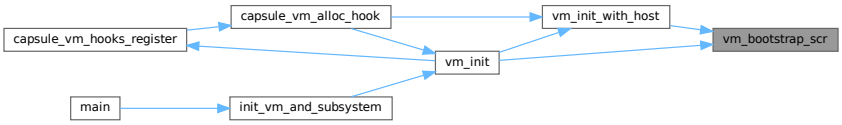
References VM::error, LOG_ERROR, log_message(), VM::memory, VM::scr_addr, vm_align(), vm_allot(), and vm_store_cell().

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.343.1.3 vm_init()

```
void vm_init (
            VM * vm )
```

Initialize a new virtual machine instance.

Allocates memory and initializes all VM structures including:

- Memory array
- Data and return stacks
- System variables (SCR, STATE, BASE)
- Dictionary
- Forth-79 wordset

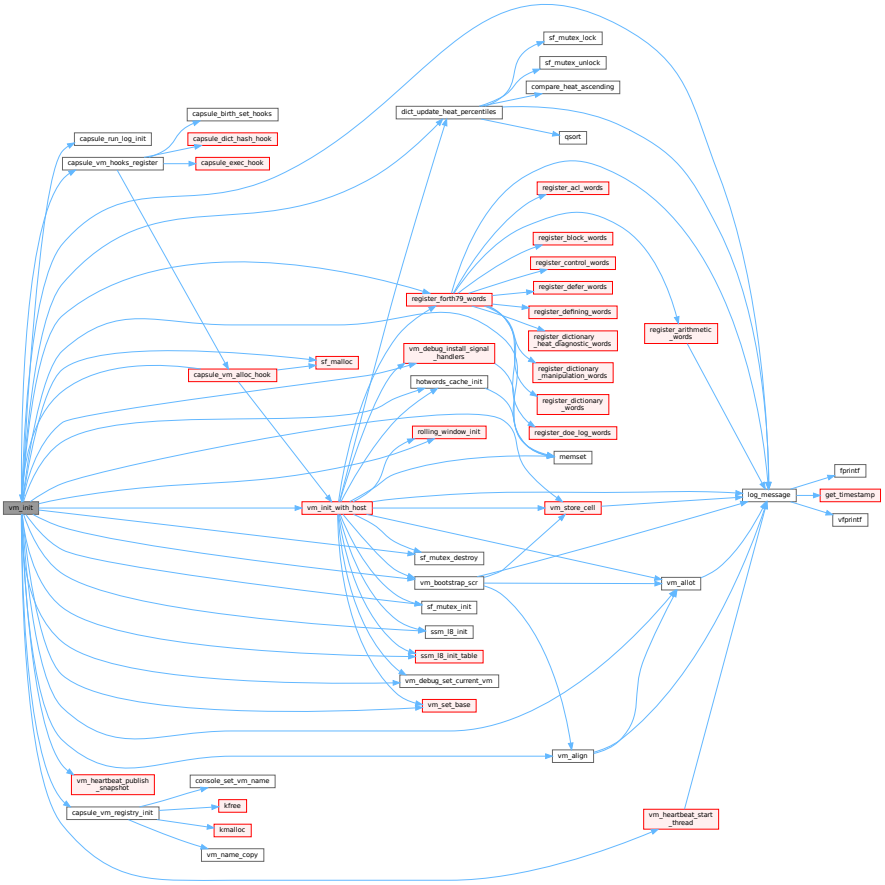
Parameters

vm	Pointer to VM structure to initialize

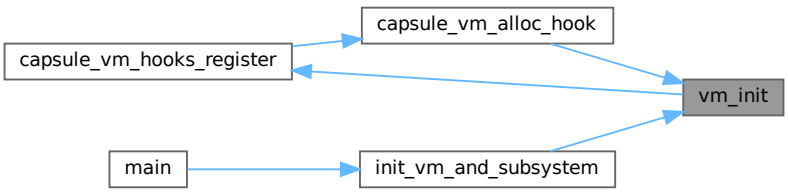
References VM::abort_requested, VM::base_addr, capsule_run_log_init(), capsule_vm_hooks_register(), capsule_vm_registry_init(), VM::cf, ControlFlowState::cf_last_mode, ControlFlowState::cf_sp, HeartbeatState::check_counter, VM::compiling_word, VM::current_executing_entry, VM::decay_slope_direction, VM::decay_slope_q48, VM::dict_fence_here, VM::dict_fence_latest, VM::dict_lock, dict_update_heat_percentiles(), VM::dsp, RollingWindowOfTruth::effective_window_size, ControlFlowState::endof_mark_sp, ControlFlowState::endof_sp, VM::error, VM::exit_colon, VM::halted, VM::heartbeat, VM::heartbeat_decay_cursor_id, HeartbeatState::heartbeat_enabled, HEARTBEAT_TICK_NS, VM::here, VM::hold_addr, VM::hold_pos, VM::hotwords_cache, hotwords_cache_init(), VM::input_length, VM::input_pos, VM::last_bucket_reorg_ns, PipelineGlobalMetrics::last_checked_accuracy, PipelineGlobalMetrics::last_checked_window_size, VM::last_decay_check_ns, HeartbeatState::last_inference_tick, VM::latest, ControlFlowState::leave_mark_sp, ControlFlowState::leave_sp, LOG_ERROR, log_message(), VM::lookup_strategy, VM::memory, memset(), VM::mode, MODE_INTERPRET, VM::next_word_id, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::recycled_word_id_count, register_forth79_words(), VM::rolling_window, rolling_window_init(), VM::rsp, VM::scr_addr, sf_malloc(), sf_mutex_destroy(), sf_mutex_init(), HeartbeatState::snapshot_index, VM::ssm_config, ssm_l8_init(), ssm_l8_init_table(), VM::ssm_l8_state, SSM_MODE_C0, VM::stale_word_count_at_check, VM::state_addr, VM::state_var, PipelineGlobalMetrics::suggested_next_size, HeartbeatState::tick_count, HeartbeatState::tick_target_ns, VM::total_heat_at_last_check, VM::tuning_lock, vm_align(), vm_allot(), vm_bootstrap_scr(), vm_debug_install_signal_handlers(), vm_debug_set_current_vm(), vm_heartbeat_publish_snapshot(), vm_heartbeat_start_thread(), vm_init_with_host(), VM_MEMORY_SIZE, vm_set_base(), vm_store_cell(), PipelineGlobalMetrics::window_tuning_checks, WORD_ID_INVALID, and HeartbeatState::worker.

Referenced by capsule_vm_alloc_hook(), and init_vm_and_subsystem().

Here is the call graph for this function:



Here is the caller graph for this function:



7.343.1.4 vm_init_with_host()

```
void vm_init_with_host (
            VM * vm,
            const VMHostServices * host )
```

Initialize VM with explicit host services.

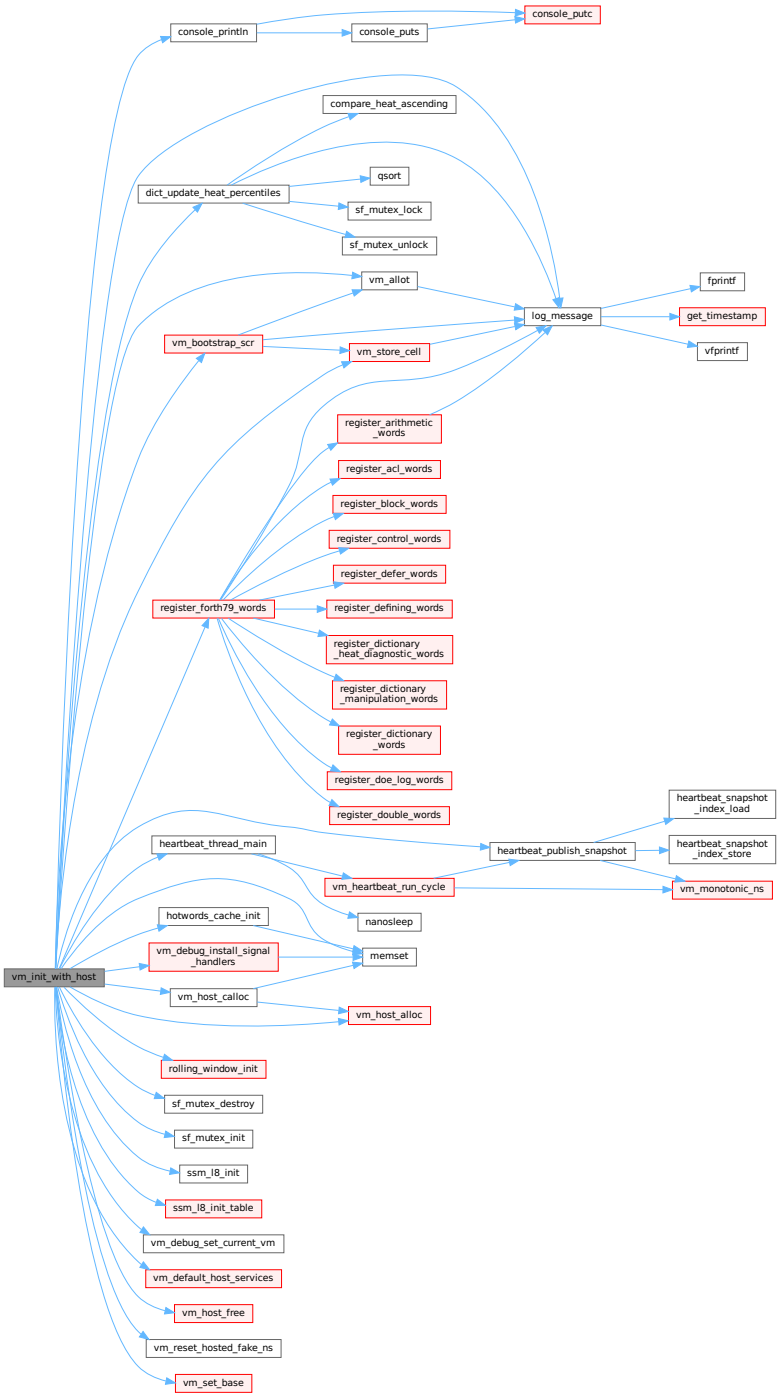
Parameters

vm	Pointer to VM structure to initialize
host	Host services (allocator/time). If NULL, uses build default.

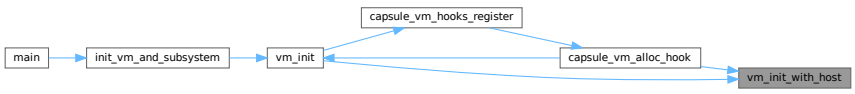
References VM::abort_requested, VM::base_addr, VM::cf, ControlFlowState::cf_last_mode, ControlFlowState::cf_sp, HeartbeatState::check_counter, VM::compiling_word, console_println(), VM::current_executing_entry, VM::decay_slope_direction, VM::decay_slope_q48, VM::dict_fence_here, VM::dict_fence_latest, VM::dict_lock, dict_update_heat_percentiles(), VM::dsp, RollingWindowOfTruth::effective_window_size, ControlFlowState::endof_mark_sp, ControlFlowState::endof_sp, errno, VM::error, VM::exit_colon, VM::halted, VM::heartbeat, VM::heartbeat_decay_cursor_id, HeartbeatState::heartbeat_enabled, heartbeat_publish_snapshot(), heartbeat_thread_main(), HEARTBEAT_TICK_NS, VM::here, VM::hold_addr, VM::hold_pos, VM::host, VM::hotwords_cache, hotwords_cache_init(), VM::input_length, VM::input_pos, VM::interpreter_enabled, VM::last_bucket_reorg_ns, PipelineGlobalMetrics::last_checked_accuracy, PipelineGlobalMetrics::last_chec VM::last_decay_check_ns, HeartbeatState::last_inference_tick, VM::latest, ControlFlowState::leave_mark_sp, ControlFlowState::leave_sp, LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, VM::lookup_strategy, VM::memory, memset(), VM::mode, MODE_INTERPRET, VM::next_word_id, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_ PipelineGlobalMetrics::prefetch_hits, VM::recycled_word_id_count, register_forth79_words(), VM::rolling_window, rolling_window_init(), VM::rsp, sf_mutex_destroy(), sf_mutex_init(), HeartbeatState::snapshot_index, VM::ssm_config, ssm_l8_init(), ssm_l8_init_table(), VM::ssm_l8_state, SSM_MODE_C0, VM::stale_word_count_at_check, VM::state_addr, VM::state_var, PipelineGlobalMetrics::suggested_next_size, HeartbeatState::tick_count, HeartbeatWorker::tick_ns, HeartbeatState::tick_target_ns, VM::total_heat_at_last_check, VM::tuning_lock, vm_allot(), vm_bootstrap_scr(), vm_debug_install_signal_handlers(), vm_debug_set_current_vm(), vm_default_host_services(), vm_host_alloc(), vm_host_calloc(), vm_host_free(), VM_MEMORY_SIZE, vm_reset_hosted_fake_ns(), vm_set_base(), vm_store_cell(), PipelineGlobalMetrics::window_tuning_checks, WORD_ID_INVALID, and HeartbeatState::worker.

Referenced by capsule_vm_alloc_hook(), and vm_init().

Here is the call graph for this function:



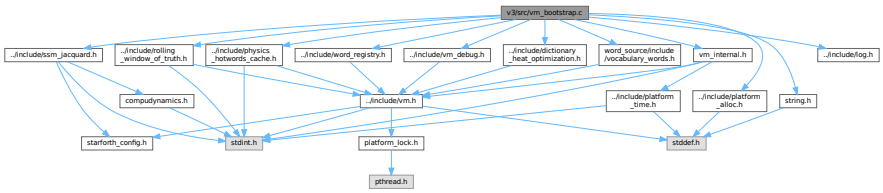
Here is the caller graph for this function:



7.344 v3/src/vm_bootstrap.c File Reference

```
#include "vm_internal.h"
#include "../include/log.h"
#include "../include/word_registry.h"
#include "../include/vm_debug.h"
#include "../include/physics_hotwords_cache.h"
#include "../include/rolling_window_of_truth.h"
#include "../include/dictionary_heat_optimization.h"
#include "../include/ssm_jacquard.h"
#include "../include/platform_alloc.h"
#include "word_source/include/vocabulary_words.h"
#include <string.h>
```

Include dependency graph for vm_bootstrap.c:



Functions

- static void vm_bootstrap_scr (VM *vm)

  Ensure the SCR (Screen) system variable cell is allocated and zeroed.
- void vm_bootstrap_root_vocabulary (VM *vm, const char *name)

  Bootstrap a root vocabulary for the VM.
- void vm_init (VM *vm)

  Initialize a new virtual machine instance.
- void vm_cleanup (VM *vm)

  Clean up and free VM resources.

7.344.1 Function Documentation

7.344.1.1 vm_bootstrap_root_vocabulary()

```
void vm_bootstrap_root_vocabulary (
            VM * vm,
            const char * name )
```

Bootstrap a root vocabulary for the VM.

Creates a vocabulary by name, finds its dictionary entry, executes it to make it current, then calls DEFINITIONS.

Parameters

vm	Pointer to VM structure
name	Vocabulary name to bootstrap

7.344.1.2 vm_bootstrap_scr()

```
static void vm_bootstrap_scr (
            VM * vm )  [static]
```

Ensure the SCR (Screen) system variable cell is allocated and zeroed.

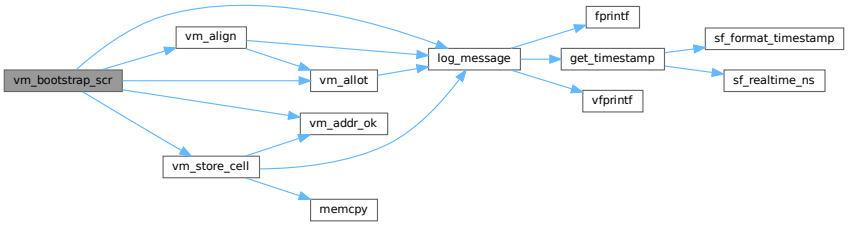
If `vm->scr_addr` does not point to a valid, aligned cell inside the VM memory array, aligns the allocation pointer and allots a new cell. Always writes 0 to the SCR cell on exit. Called at the end of `vm_init()` to guarantee a clean SCR state regardless of how the initial allot sequence progressed.

Parameters

vm	Active VM; sets `vm->error` = 1 and logs if allot fails

References VM::error, LOG_ERROR, log_message(), VM::memory, VM::scr_addr, vm_addr_ok(), vm_align(), vm_allot(), and vm_store_cell().

Here is the call graph for this function:



7.344.1.3 vm_cleanup()

```
void vm_cleanup (
            VM * vm )
```

Clean up and free VM resources.

Frees allocated memory and resets VM state.

Parameters

vm	Pointer to VM structure to clean up

7.344.1.4 vm_init()

```
void vm_init (
            VM * vm )
```

Initialize a new virtual machine instance.

Allocates memory and initializes all VM structures including:

- Memory array
- Data and return stacks
- System variables (SCR, STATE, BASE)
- Dictionary
- Forth-79 wordset

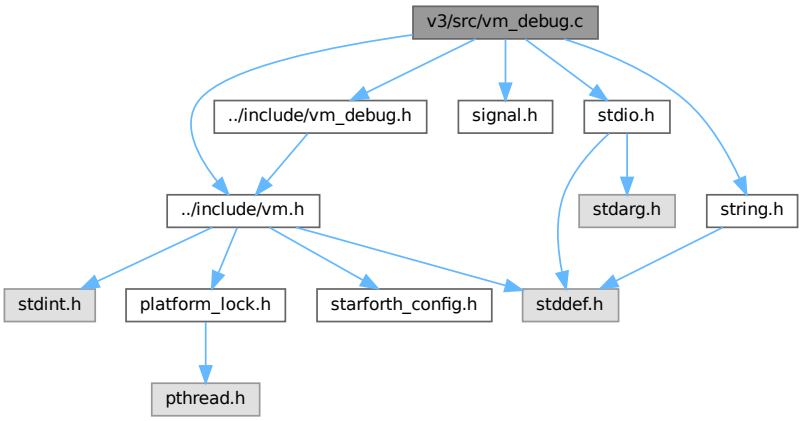
Parameters

vm	Pointer to VM structure to initialize

7.345 v3/src/vm_debug.c File Reference

```
#include "../include/vm.h"
#include "../include/vm_debug.h"
#include <signal.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for vm_debug.c:



Functions

- void vm_debug_set_current_vm (VM *vm)

  Sets the VM instance to dump state for when signals occur.
- static unsigned dbg_get_base (const VM *vm)

  Safely retrieves the current number base from VM.
- static const char * mode_str (int m)

  Converts VM mode to string representation.

- static void dump_block_preview (const VM *vm, cell_t scr)

  Dumps preview of a block's contents.
- void vm_debug_dump_state (const VM *vm, const char *reason)

  Manually dumps the state of a VM instance.
- static void sig_handler (int sig)

  Signal handler for crash debugging.
- void vm_debug_install_signal_handlers (void)

  Installs signal handlers for crash debugging.

Variables

- static VM * g_vm = NULL

7.345.1 Function Documentation

7.345.1.1 dbg_get_base()

```
static unsigned dbg_get_base (
            const VM * vm )  [static]
```

Safely retrieves the current number base from VM.

Parameters

vm	Pointer to the VM instance

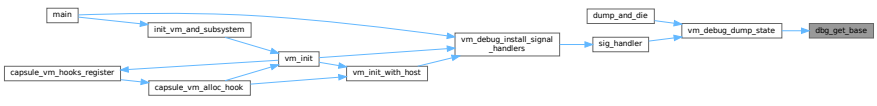
Returns

Current number base (2-36) or 10 if invalid

References VM::base, VM::base_addr, VM::memory, and VM_MEMORY_SIZE.

Referenced by vm_debug_dump_state().

Here is the caller graph for this function:



7.345.1.2 dump_block_preview()

```
static void dump_block_preview (
            const VM * vm,
            cell_t scr )  [static]
```

Dumps preview of a block's contents.

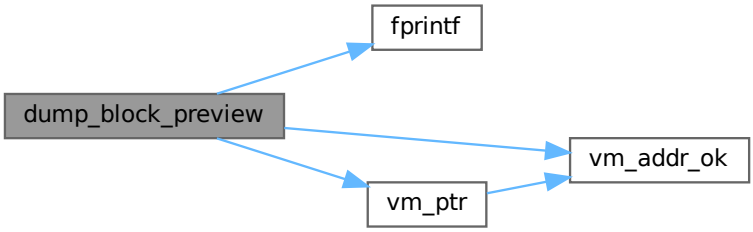
Parameters

vm	Pointer to the VM instance
scr	Block number to preview

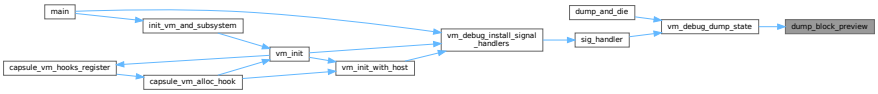
References BLOCK_SIZE, fprintf(), MAX_BLOCKS, stderr, vm_addr_ok(), and vm_ptr().

Referenced by vm_debug_dump_state().

Here is the call graph for this function:



Here is the caller graph for this function:



7.345.1.3 mode_str()

```
static const char * mode_str (
            int m )  [static]
```

Converts VM mode to string representation.

Parameters

m	The VM mode value

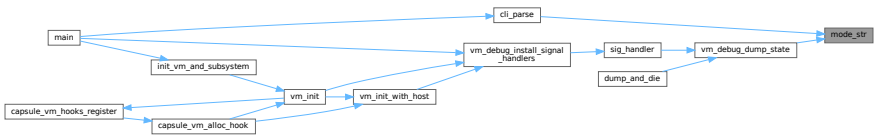
Returns

String representing the mode

References MODE_COMPILE, and MODE_INTERPRET.

Referenced by cli_parse(), and vm_debug_dump_state().

Here is the caller graph for this function:



7.345.1.4 sig_handler()

```
static void sig_handler (
            int sig )  [static]
```

Signal handler for crash debugging.

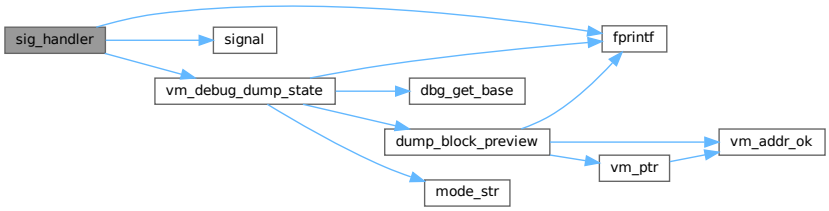
Parameters

sig	Signal number that was caught

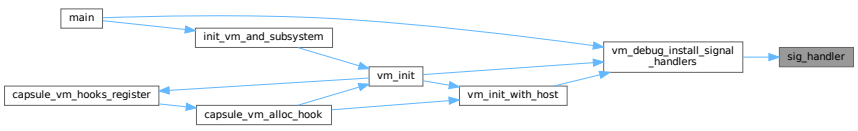
References fprintf(), g_vm, name, SIG_DFL, SIGABRT, signal(), SIGSEGV, stderr, and vm_debug_dump_state().

Referenced by vm_debug_install_signal_handlers().

Here is the call graph for this function:



Here is the caller graph for this function:



7.345.1.5 vm_debug_dump_state()

```
void vm_debug_dump_state (
            const VM * vm,
            const char * reason )
```

Manually dumps the state of a VM instance.

Parameters

vm	The VM instance to dump state for
reason	Description of why the dump was triggered

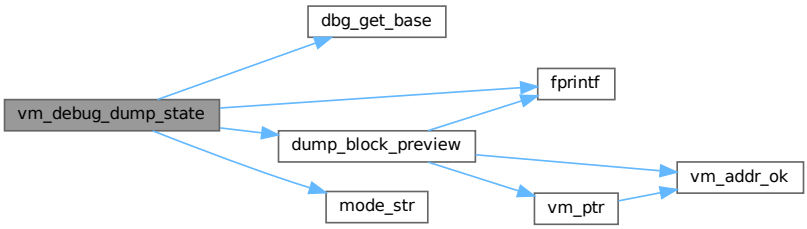
Note

Typically called before exit(1) on test failure

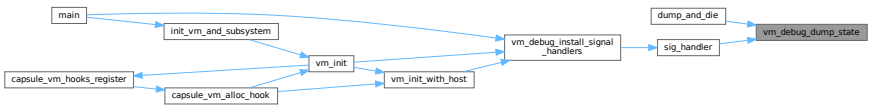
References base, VM::current_executing_entry, dbg_get_base(), VM::dsp, dump_block_preview(), VM::error, fprintf(), VM::here, VM::ip, VM::memory, VM::mode, mode_str(), DictEntry::name, VM::rsp, VM::scr_addr, stderr, and VM_MEMORY_SIZE.

Referenced by dump_and_die(), and sig_handler().

Here is the call graph for this function:



Here is the caller graph for this function:



7.345.1.6 vm_debug_install_signal_handlers()

```
void vm_debug_install_signal_handlers (
            void  )
```

Installs signal handlers for crash debugging.

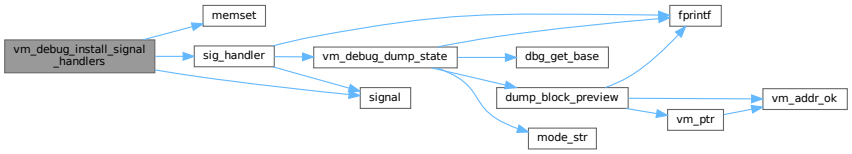
Installs signal handlers for SIGSEGV and SIGABRT.

Sets up handlers for SIGSEGV and SIGABRT to dump VM state on crash

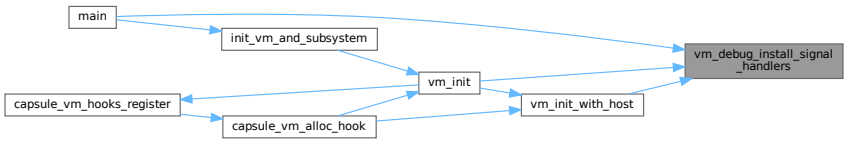
References memset(), sigaction::sa_flags, sigaction::sa_handler, SA_RESETHAND, sig_handler(), SIGABRT, signal(), and SIGSEGV.

Referenced by main(), vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.345.1.7 vm_debug_set_current_vm()

```
void vm_debug_set_current_vm (
            VM * vm )
```

Sets the VM instance to dump state for when signals occur.

Parameters

vm	Pointer to the VM instance to monitor

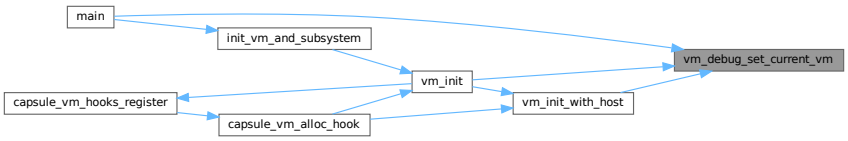
Note

Should be called once after vm_init

References g_vm.

Referenced by main(), vm_init(), and vm_init_with_host().

Here is the caller graph for this function:



7.345.2 Variable Documentation

7.345.2.1 g_vm

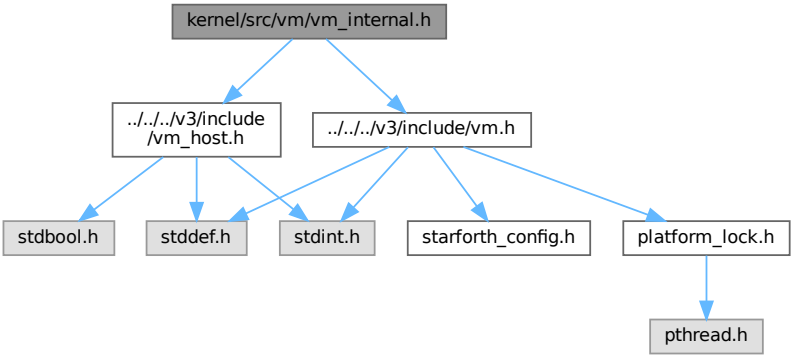
```
VM* g_vm = NULL  [static]
```

Referenced by sig_handler(), and vm_debug_set_current_vm().

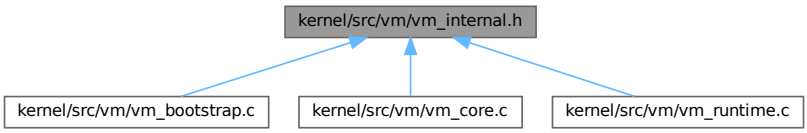
7.346 kernel/src/vm/vm_internal.h File Reference

```
#include "../../../v3/include/vm.h"
#include "../../../v3/include/vm_host.h"
```

Include dependency graph for vm_internal.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct HeartbeatWorker

Macros

- #define PARITY_MODE 0
- #define SK_PARITY_DEBUG 0
- #define HEARTBEAT_HAS_THREADS 0
- #define HEARTBEAT_DECAY_BATCH 64u

Typedefs

- typedef struct HeartbeatWorker HeartbeatWorker

Functions

- const VMHostServices * vm_default_host_services (void)
- const VMHostServices * vm_host (const VM *vm)
- void * vm_host_alloc (VM *vm, size_t size, size_t align)
- void * vm_host_calloc (VM *vm, size_t n, size_t size)
- void vm_host_free (VM *vm, void *ptr)
- void vm_reset_hosted_fake_ns (VM *vm)
- uint64_t vm_monotonic_ns (VM *vm)
- unsigned vm_get_base (const VM *vm)

  Read the current number base from the VM.
- void vm_set_base (VM *vm, unsigned b)

  Set the VM number base and update both the cell and the host mirror.
- uint32_t heartbeat_snapshot_index_load (const volatile uint32_t *ptr)
- void heartbeat_snapshot_index_store (volatile uint32_t *ptr, uint32_t value)
- void heartbeat_publish_snapshot (VM *vm)
- void vm_heartbeat_run_cycle (VM *vm)

7.346.1 Macro Definition Documentation

7.346.1.1 HEARTBEAT_DECAY_BATCH

```
#define HEARTBEAT_DECAY_BATCH 64u
```

7.346.1.2 HEARTBEAT_HAS_THREADS

```
#define HEARTBEAT_HAS_THREADS 0
```

7.346.1.3 PARITY_MODE

```
#define PARITY_MODE 0
```

7.346.1.4 SK_PARITY_DEBUG

```
#define SK_PARITY_DEBUG 0
```

7.346.2 Typedef Documentation

7.346.2.1 HeartbeatWorker

```
typedef struct HeartbeatWorker HeartbeatWorker
```

7.346.3 Function Documentation

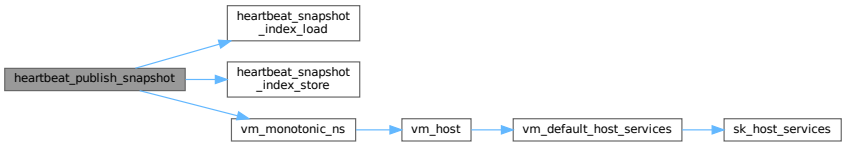
7.346.3.1 heartbeat_publish_snapshot()

```
void heartbeat_publish_snapshot (
            VM * vm )
```

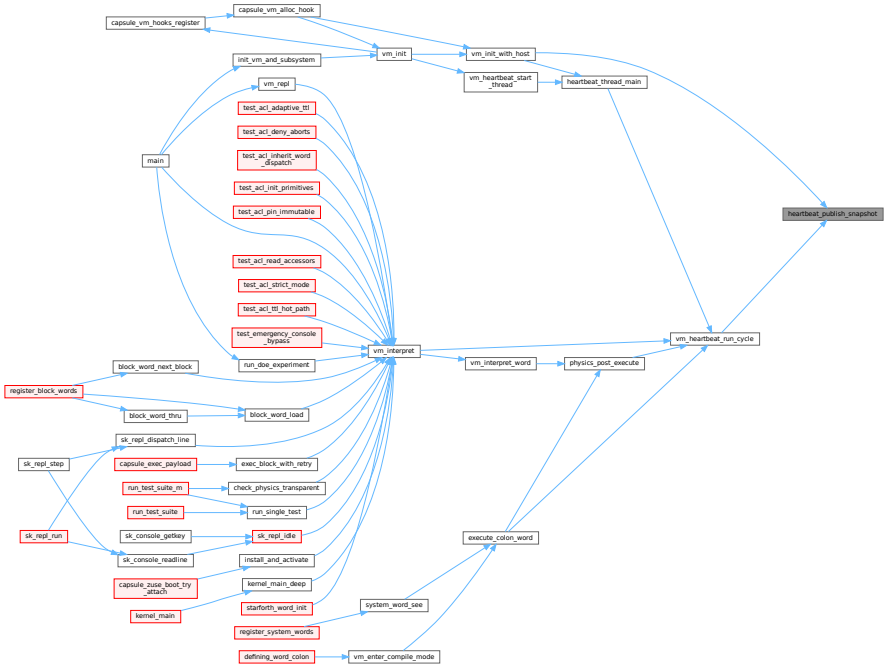
References HeartbeatSnapshot::decay_slope_q48, VM::decay_slope_q48, RollingWindowOfTruth::effective_window_size, VM::heartbeat, heartbeat_snapshot_index_load(), heartbeat_snapshot_index_store(), HeartbeatSnapshot::hot_word_count, VM::hot_word_count_at_check, HeartbeatSnapshot::published_ns, HeartbeatSnapshot::published_tick, VM::rolling_window, HeartbeatState::snapshot_index, HeartbeatState::snapshots, HeartbeatSnapshot::stale_word_count, VM::stale_word_count_at_chec HeartbeatState::tick_count, HeartbeatSnapshot::total_heat, VM::total_heat_at_last_check, vm_monotonic_ns(), and HeartbeatSnapshot::window_width.

Referenced by vm_heartbeat_run_cycle(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:

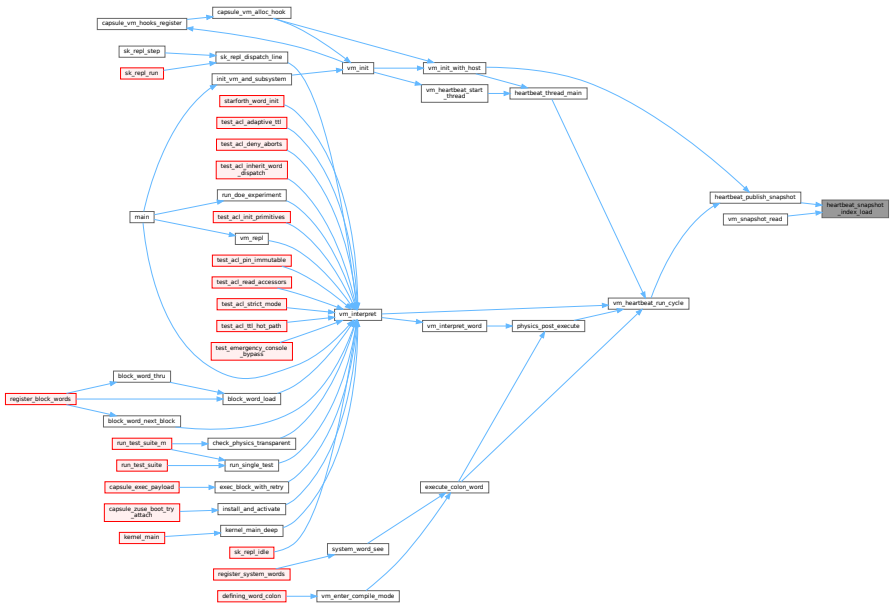


7.346.3.2 heartbeat_snapshot_index_load()

```
uint32_t heartbeat_snapshot_index_load (
            const volatile uint32_t * ptr )
```

Referenced by heartbeat_publish_snapshot(), and vm_snapshot_read().

Here is the caller graph for this function:

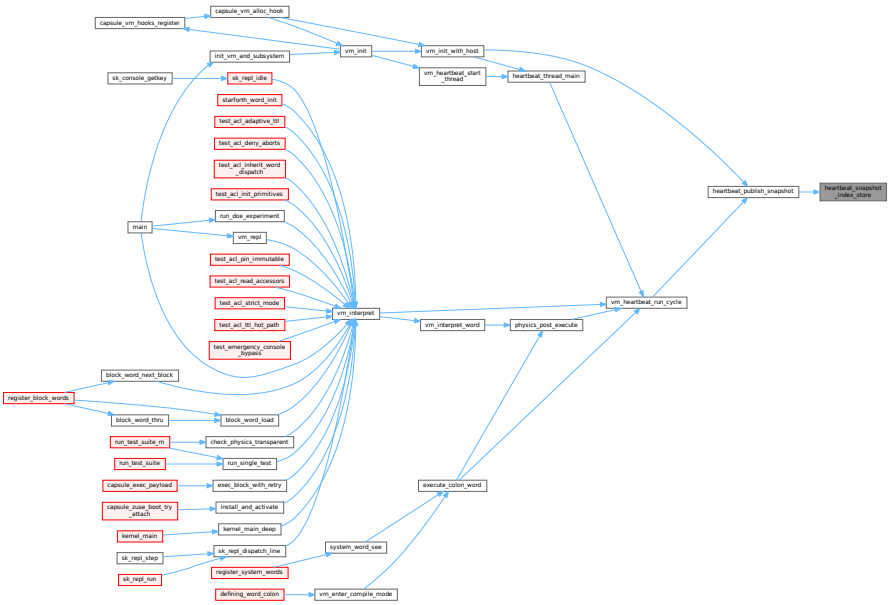


7.346.3.3 heartbeat_snapshot_index_store()

```
void heartbeat_snapshot_index_store (
            volatile uint32_t * ptr,
            uint32_t value )
```

Referenced by heartbeat_publish_snapshot().

Here is the caller graph for this function:



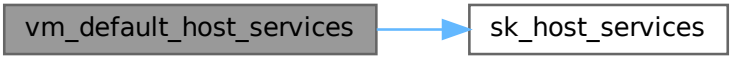
7.346.3.4 vm_default_host_services()

```
const VMHostServices * vm_default_host_services (
            void  )
```

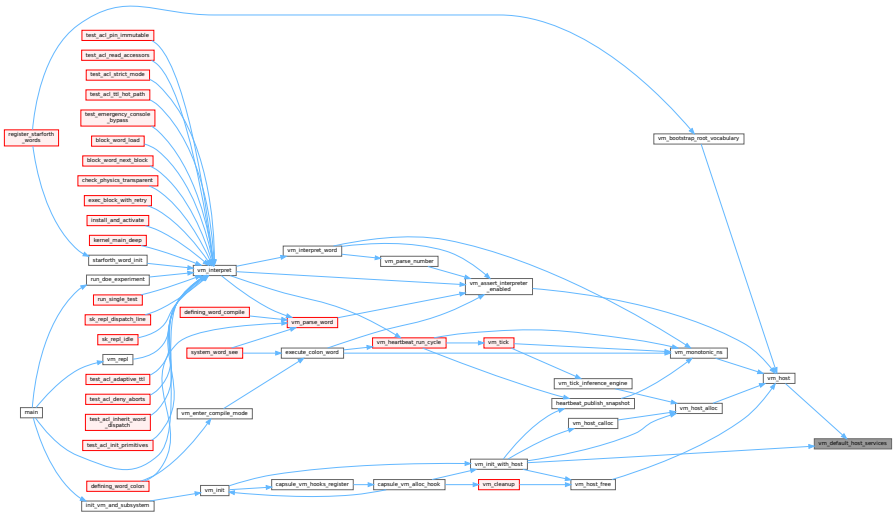
References hosted_services, and sk_host_services().

Referenced by vm_host(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.346.3.5 vm_get_base()

```
unsigned vm_get_base (
            const VM * vm )
```

Read the current number base from the VM.

Prefers the cell stored at `vm->base_addr` (the canonical FORTH `BASE` variable) if it is aligned and in the valid range [2, 36]. Falls back to the host-mirror `vm->base` if the cell is out of range, and finally returns 10 if both are invalid or `vm` is NULL.

The double-read (cell + mirror) keeps the VM consistent when C code modifies `vm->base` directly without going through `vm_set_base()`.

Parameters

vm	VM instance to query (may be NULL; returns 10 in that case).

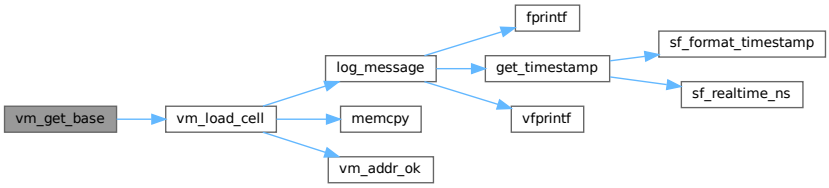
Returns

Current number base in range [2, 36].

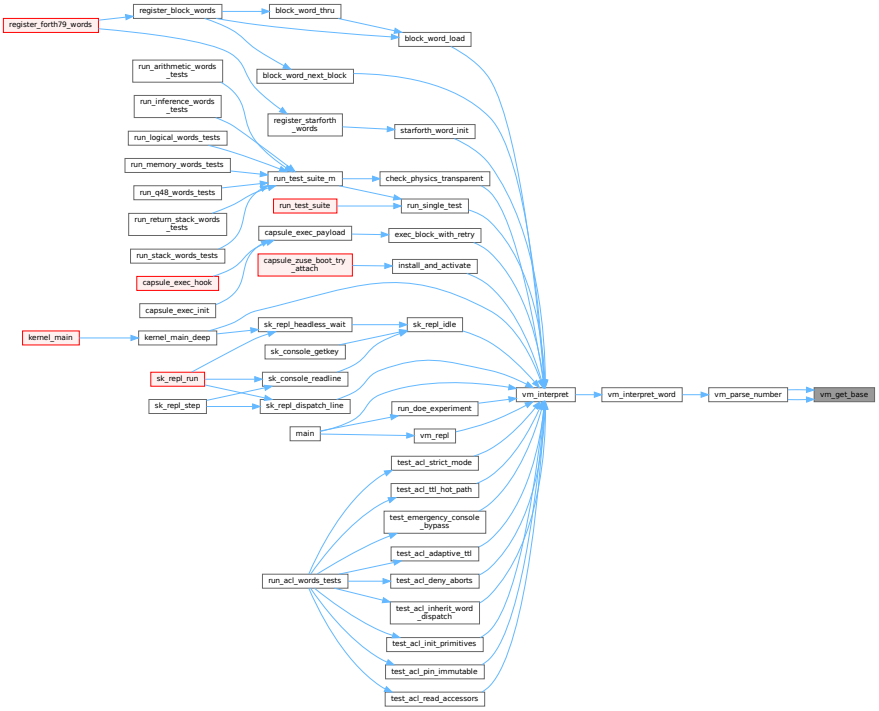
References VM::base, VM::base_addr, vm_load_cell(), and VM_MEMORY_SIZE.

Referenced by vm_parse_number(), and vm_parse_number().

Here is the call graph for this function:



Here is the caller graph for this function:



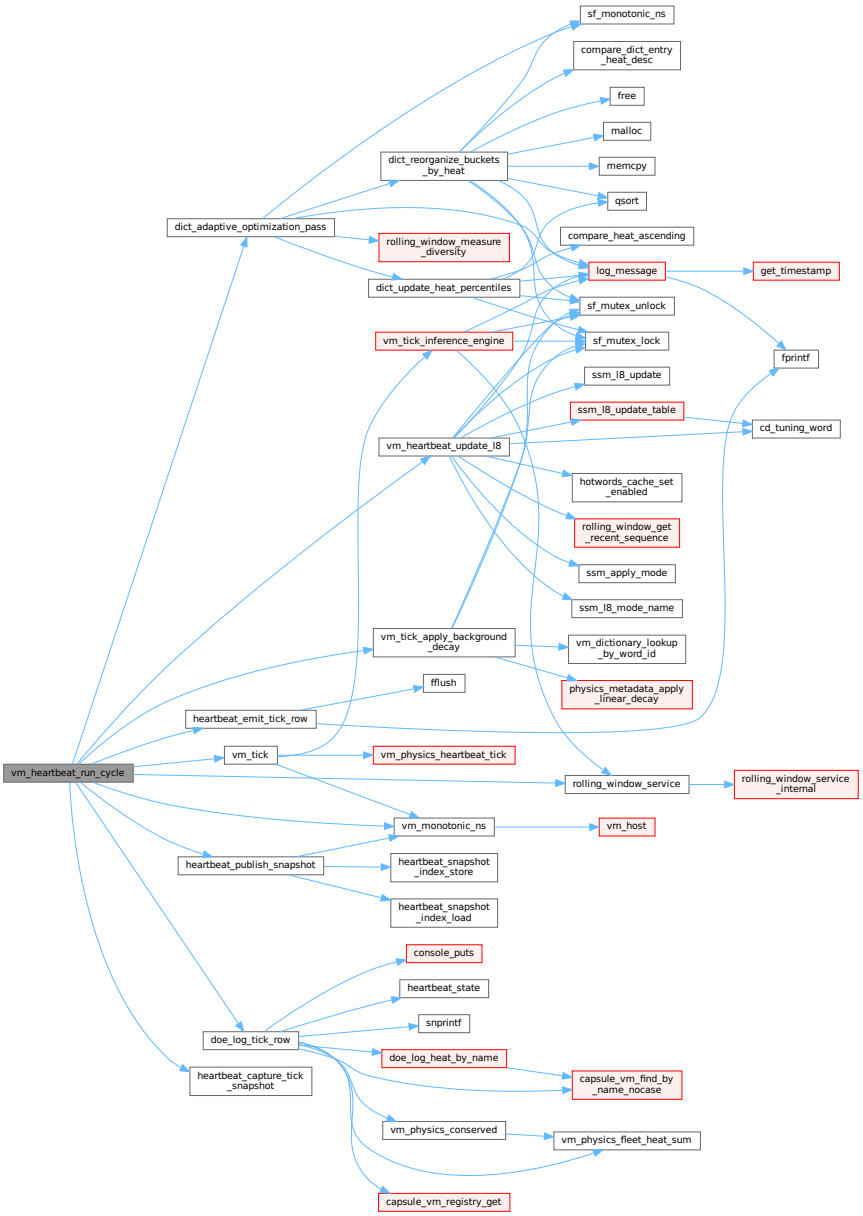
7.346.3.6 vm_heartbeat_run_cycle()

```
void vm_heartbeat_run_cycle (
            VM * vm )
```

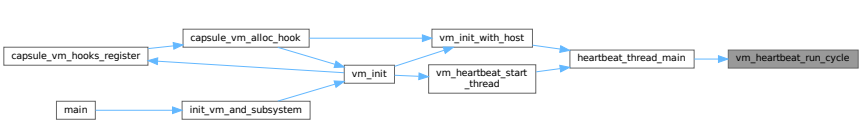
References dict_adaptive_optimization_pass(), doe_log_tick_row(), VM::heartbeat, heartbeat_capture_tick_snapshot(), heartbeat_emit_tick_row(), HeartbeatState::heartbeat_enabled, heartbeat_publish_snapshot(), VM::rolling_window, rolling_window_service(), vm_heartbeat_update_l8(), vm_monotonic_ns(), vm_tick(), and vm_tick_apply_background_decay().

Referenced by heartbeat_thread_main().

Here is the call graph for this function:



Here is the caller graph for this function:



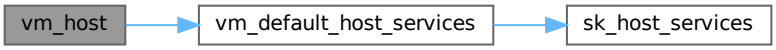
7.346.3.7 vm_host()

```
const VMHostServices * vm_host (
            const VM * vm )
```

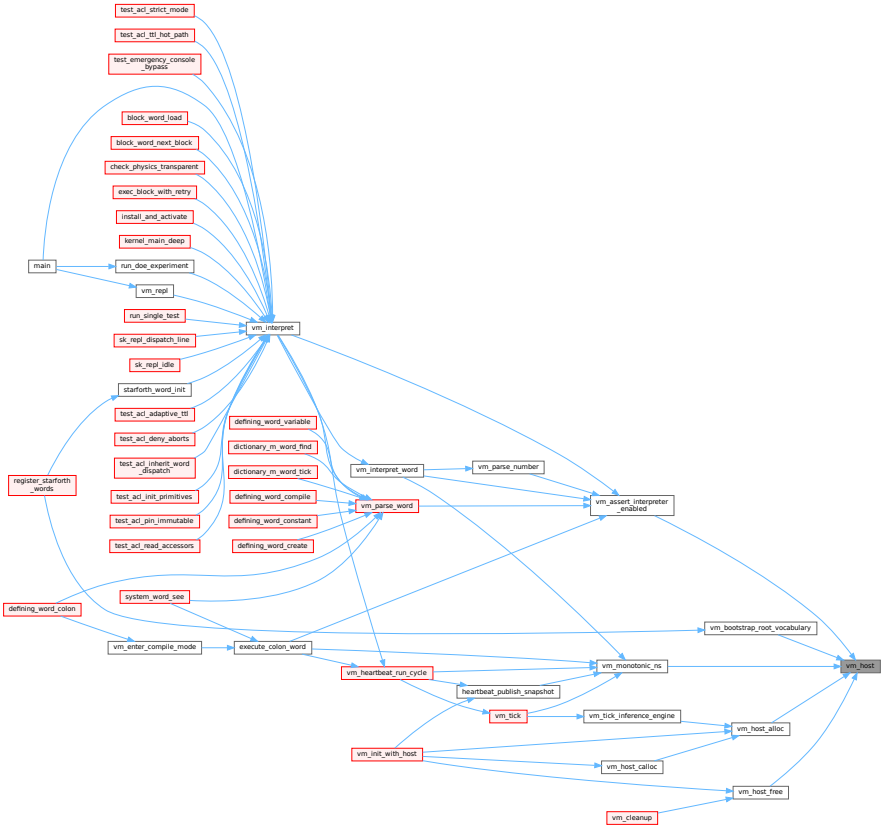
References VM::host, and vm_default_host_services().

Referenced by vm_assert_interpreter_enabled(), vm_bootstrap_root_vocabulary(), vm_host_alloc(), vm_host_free(), and vm_monotonic_ns().

Here is the call graph for this function:



Here is the caller graph for this function:



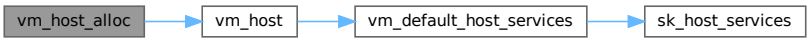
7.346.3.8 vm_host_alloc()

```
void * vm_host_alloc (
            VM * vm,
            size_t size,
            size_t align )
```

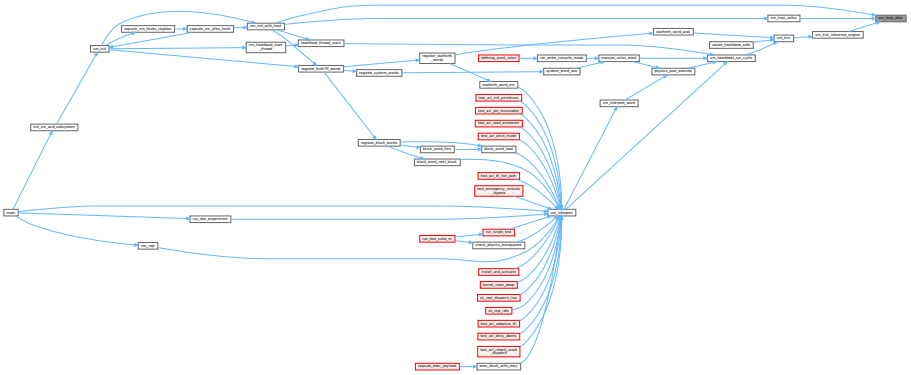
References VMHostServices::alloc, and vm_host().

Referenced by vm_host_calloc(), vm_init_with_host(), and vm_tick_inference_engine().

Here is the call graph for this function:



Here is the caller graph for this function:



7.346.3.9 vm_host_calloc()

```
void * vm_host_calloc (
            VM * vm,
            size_t n,
            size_t size )
```

References memset(), and vm_host_alloc().

Referenced by vm_init_with_host().

Here is the call graph for this function:

Here is the caller graph for this function:

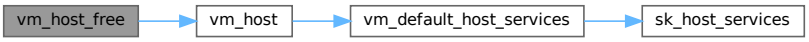
7.346.3.10 vm_host_free()

```
void vm_host_free (
            VM * vm,
            void * ptr )
```

References VMHostServices::free, and vm_host().

Referenced by vm_cleanup(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:

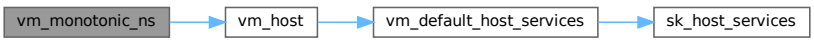
7.346.3.11 vm_monotonic_ns()

```
uint64_t vm_monotonic_ns (
            VM * vm )
```

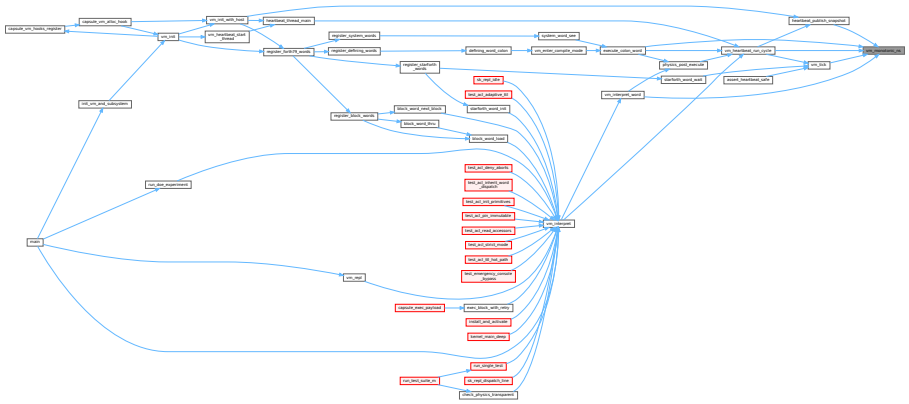
References VMHostServices::monotonic_ns, and vm_host().

Referenced by execute_colon_word(), heartbeat_publish_snapshot(), vm_heartbeat_run_cycle(), vm_interpret_word(), and vm_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.346.3.12 vm_reset_hosted_fake_ns()

```
void vm_reset_hosted_fake_ns (
            VM * vm )
```

References VM::host, hosted_fake_ns, hosted_services, and PARITY_MODE.

Referenced by vm_init_with_host().

Here is the caller graph for this function:

7.346.3.13 vm_set_base()

```
void vm_set_base (
            VM * vm,
            unsigned b )
```

Set the VM number base and update both the cell and the host mirror.

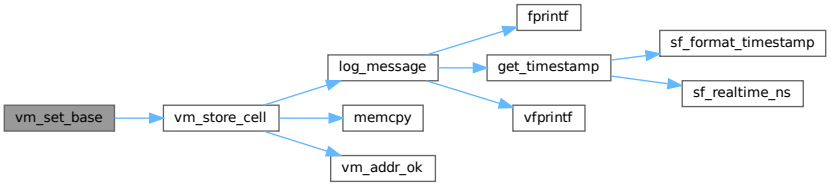
Clamps `b` to [2, 36] (replacing out-of-range values with 10), then writes to the FORTH `BASE` variable cell at `vm->base_addr` and to the host-mirror field `vm->base` so both sources agree.

Parameters

vm	VM instance to update.
b	Desired number base; clamped to [2, 36].

References VM::base, VM::base_addr, and vm_store_cell().

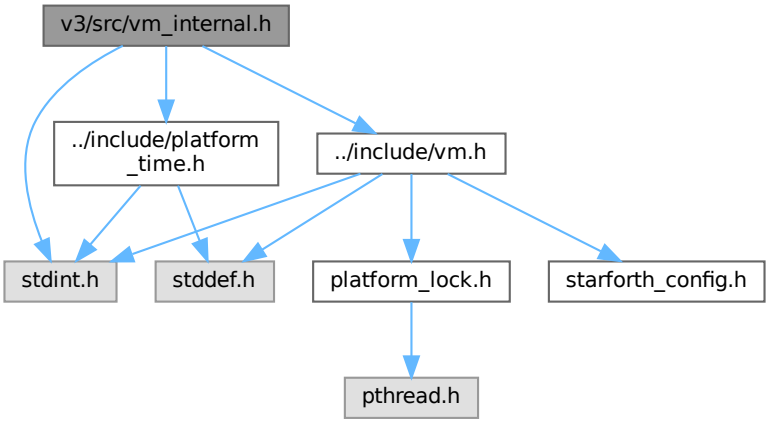
Here is the call graph for this function:



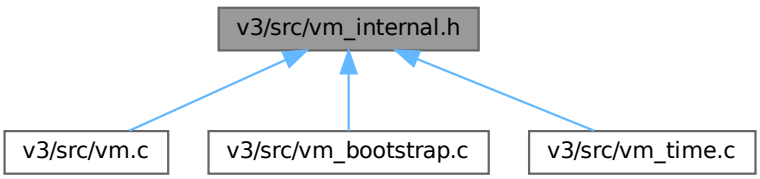
7.347 v3/src/vm_internal.h File Reference

```
#include "../include/vm.h"
#include "../include/platform_time.h"
#include <stdint.h>
```

Include dependency graph for vm_internal.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct HeartbeatWorker

Macros

- #define HEARTBEAT_HAS_THREADS 0
- #define HEARTBEAT_DECAY_BATCH 64u

Typedefs

- typedef struct HeartbeatWorker HeartbeatWorker

Functions

- static uint32_t heartbeat_snapshot_index_load (const volatile uint32_t *ptr)
- static void heartbeat_snapshot_index_store (volatile uint32_t *ptr, uint32_t value)
- void execute_colon_word (VM *vm)

  Inner interpreter for threaded colon definitions.
- void vm_tick (VM *vm)

  Central heartbeat dispatcher for time-driven tuning operations.
- void vm_tick_window_tuner (VM *vm)

  Loop #5: Context-aware window tuning via binary chop search.
- void vm_tick_slope_validator (VM *vm)

  Loop #3: Heat decay slope validation via periodic measurement.
- void vm_tick_inference_engine (VM *vm)

  Phase 2: Unified Inference Engine - Adaptive Window & Decay Slope Tuning.
- void vm_heartbeat_run_cycle (VM *vm)
- void vm_snapshot_read (const VM *vm, HeartbeatSnapshot *out_snapshot)
- void vm_heartbeat_start_thread (VM *vm)
- void vm_heartbeat_stop_thread (VM *vm)
- void vm_heartbeat_publish_snapshot (VM *vm)
- void vm_init (VM *vm)

  Initialize a new virtual machine instance.
- void vm_cleanup (VM *vm)

  Clean up and free VM resources.
- unsigned vm_get_base (const VM *vm)

  Read the current number base from the VM.
- void vm_set_base (VM *vm, unsigned b)

  Set the VM number base and update both the cell and the host mirror.

7.347.1 Macro Definition Documentation

7.347.1.1 HEARTBEAT_DECAY_BATCH

```
#define HEARTBEAT_DECAY_BATCH 64u
```

7.347.1.2 HEARTBEAT_HAS_THREADS

```
#define HEARTBEAT_HAS_THREADS 0
```

7.347.2 Typedef Documentation

7.347.2.1 HeartbeatWorker

```
typedef struct HeartbeatWorker HeartbeatWorker
```

7.347.3 Function Documentation

7.347.3.1 execute_colon_word()

```
void execute_colon_word (
            VM * vm )
```

Inner interpreter for threaded colon definitions.

Runs the body of a colon-defined word whose `DictEntry` points to a sequence of `DictEntry*` cells in VM memory (indirect threaded code). The interpreter pointer (IP) is maintained on the return stack so that runtime control-flow words ((BRANCH), (0BRANCH), (DO), etc.) can redirect execution by modifying the top-of-return-stack IP.

Execution loop per iteration:

1. Read `DictEntry* w` from `*ip`.
2. Run physics pre-execute hooks (heat, decay, pipelining).
3. ACL hot-path check (decrement TTL; call `acl_recheck()` on zero).
4. Push `ip+1` onto the return stack as the resume point.
5. Call `w->func(vm)`.
6. Run physics post-execute hooks.
7. Check `vm->exit_colon` (set by `EXIT`): pop and discard the saved IP, then return to caller.
8. Check `vm->abort_requested`: unwind and return immediately.
9. Pop the (possibly modified) IP from the return stack and repeat.

`vm->ecw_nesting` is incremented on entry and decremented on every return path to enable recursive colon calls.

Parameters

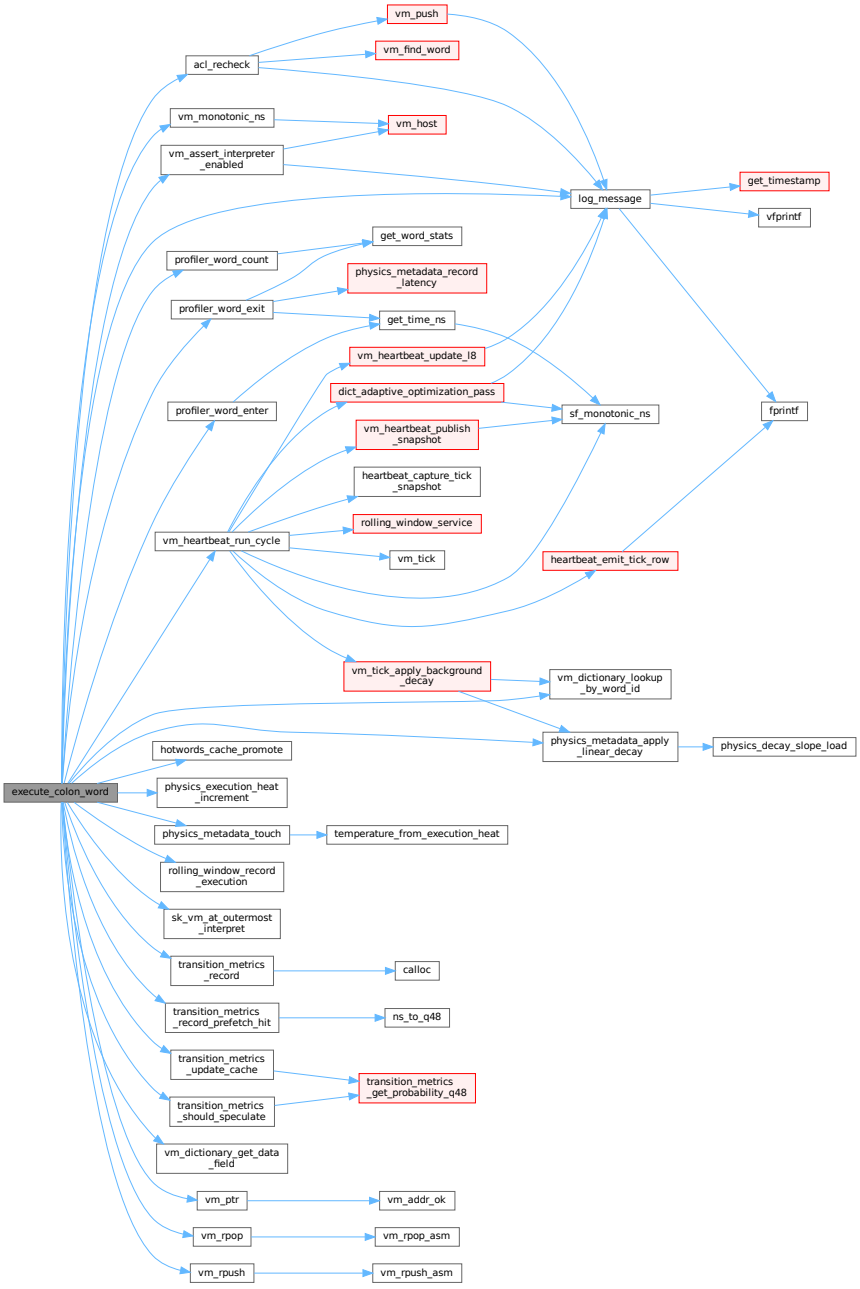
vm	VM instance; `vm->current_executing_entry` must point to the `DictEntry` being executed.

References VM::abort_requested, DictEntry::acl_allow, acl_recheck(), DictEntry::acl_ttl, HeartbeatState::check_counter, VM::current_executing_entry, DICTIONARY_SIZE, VM::dsp, VM::ecw_nesting, VM::emergency_console,

ENABLE_HOTWORDS_CACHE, ENABLE_PIPELINING, entry, VM::error, DictEntry::execution_heat, VM::exit_colon, DictEntry::func, g_sk_fault_word, VM::heartbeat, HEARTBEAT_CHECK_FREQUENCY, VM::hotwords_cache, hotwords_cache_promote(), HOTWORDS_EXECUTION_HEAT_THRESHOLD, DictPhysics::last_active_ns, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, LOG_DEBUG, LOG_ERROR, log_message(), LOG_WARN, WordTransitionMetrics::most_likely_next_word_id, DictEntry::name, DictEntry::name_len, DictEntry::physics, physics_execution_heat_increment(), physics_metadata_apply_linear_decay(), physics_metadata_touch(), VM::pipeline_metrics, WordTransitionMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, profiler_word_count(), profiler_word_enter(), profiler_word_exit(), VM::rolling_window, rolling_window_record_execution(), VM::rsp, sk_vm_at_outermost_interpret(), HeartbeatState::tick_count, DictEntry::transition_metrics, transition_metrics_record(), transition_metrics_record_prefetch_hit(), transition_metrics_should_specula transition_metrics_update_cache(), vm_assert_interpreter_enabled(), vm_dictionary_get_data_field(), vm_dictionary_lookup_by_wor vm_heartbeat_run_cycle(), vm_monotonic_ns(), vm_ptr(), vm_rpop(), vm_rpush(), word_id, DictEntry::word_id, HeartbeatState::words_executed, and HeartbeatState::worker.

Referenced by vm_enter_compile_mode().

Here is the call graph for this function:



Here is the caller graph for this function:

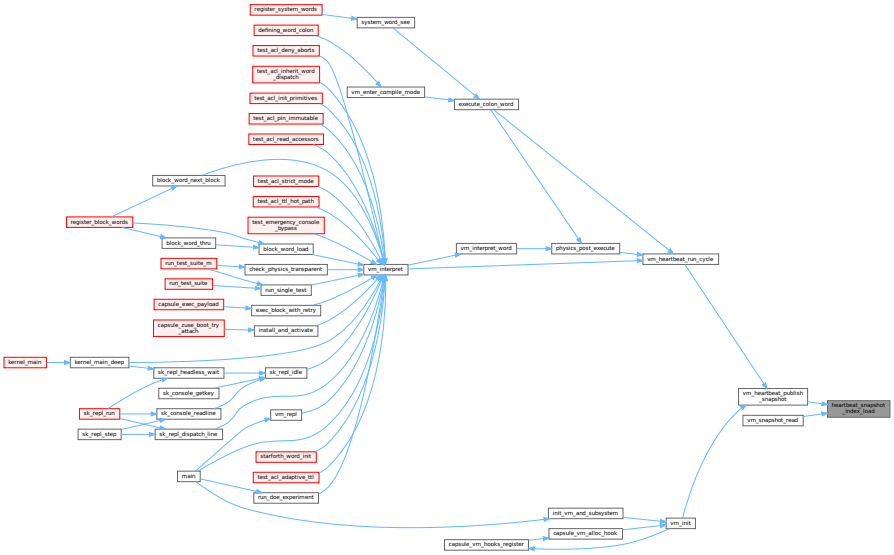
7.347.3.2 heartbeat_snapshot_index_load()

```
static uint32_t heartbeat_snapshot_index_load (
```

```
            const volatile uint32_t * ptr )  [inline], [static]
```

Referenced by vm_heartbeat_publish_snapshot(), and vm_snapshot_read().

Here is the caller graph for this function:

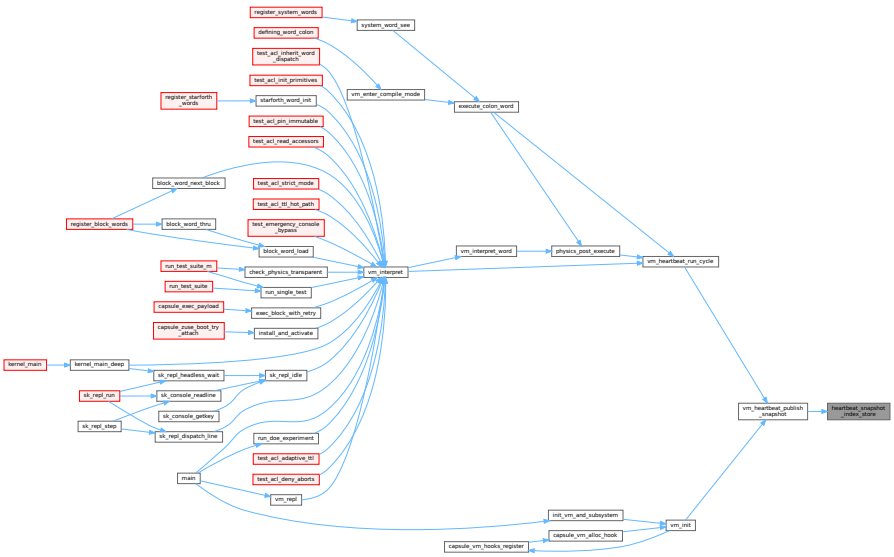


7.347.3.3 heartbeat_snapshot_index_store()

```
static void heartbeat_snapshot_index_store (
            volatile uint32_t * ptr,
            uint32_t value )  [inline], [static]
```

Referenced by vm_heartbeat_publish_snapshot().

Here is the caller graph for this function:



7.347.3.4 vm_cleanup()

```
void vm_cleanup (
            VM * vm )
```

Clean up and free VM resources.

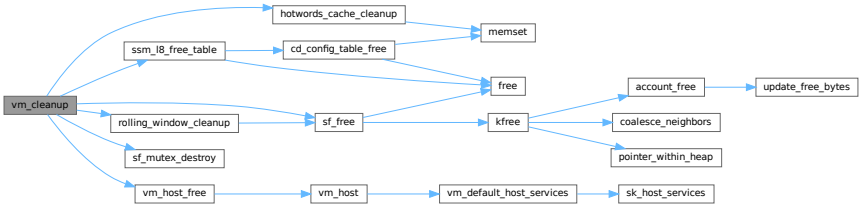
Frees allocated memory and resets VM state.

Parameters

vm	Pointer to VM structure to clean up

References VM::dict_lock, VM::heartbeat, VM::here, VM::hotwords_cache, hotwords_cache_cleanup(), VM::latest, DictEntry::link, VM::memory, VM::rolling_window, rolling_window_cleanup(), sf_free(), sf_mutex_destroy(), VM::ssm_config, ssm_l8_free_table(), VM::ssm_l8_state, HeartbeatWorker::stop_requested, DictEntry::transition_metrics, VM::tuning_lock, vm_host_free(), and HeartbeatState::worker.

Here is the call graph for this function:



7.347.3.5 vm_get_base()

```
unsigned vm_get_base (
            const VM * vm )
```

Read the current number base from the VM.

Prefers the cell stored at `vm->base_addr` (the canonical FORTH `BASE` variable) if it is aligned and in the valid range [2, 36]. Falls back to the host-mirror `vm->base` if the cell is out of range, and finally returns 10 if both are invalid or `vm` is NULL.

The double-read (cell + mirror) keeps the VM consistent when C code modifies `vm->base` directly without going through `vm_set_base()`.

Parameters

vm	VM instance to query (may be NULL; returns 10 in that case).

Returns

Current number base in range [2, 36].

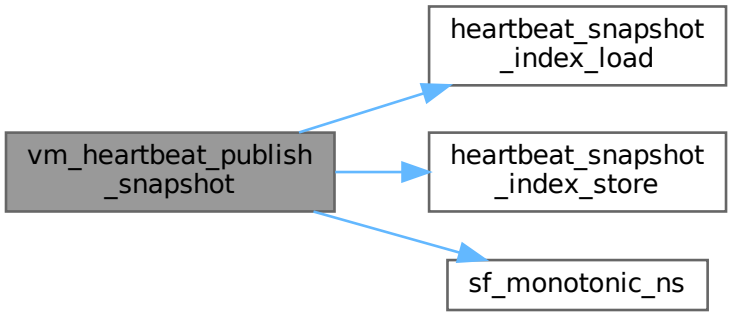
7.347.3.6 vm_heartbeat_publish_snapshot()

```
void vm_heartbeat_publish_snapshot (
            VM * vm )
```

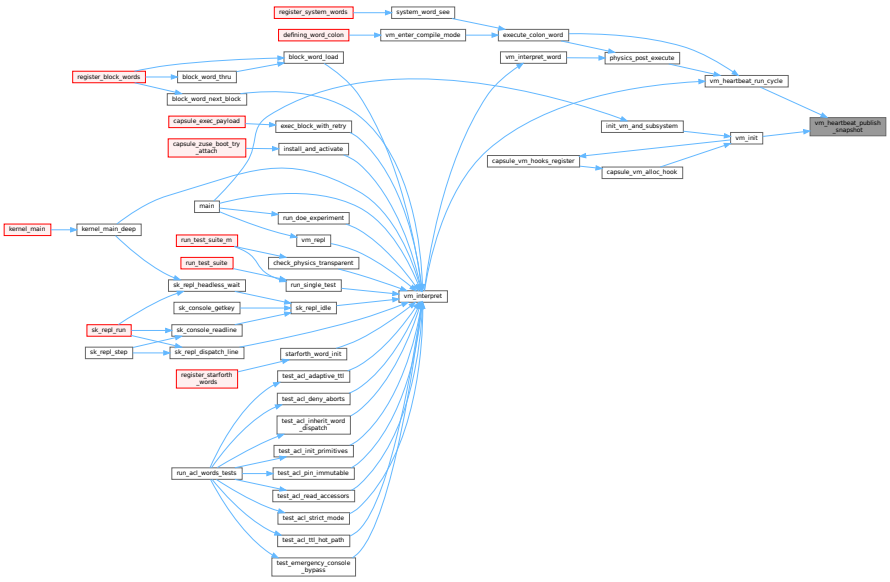
References HeartbeatSnapshot::actual_window_size, HeartbeatSnapshot::decay_slope_q48, VM::decay_slope_q48, RollingWindowOfTruth::effective_window_size, VM::heartbeat, heartbeat_snapshot_index_load(), heartbeat_snapshot_index_store(), HeartbeatSnapshot::hot_word_count, VM::hot_word_count_at_check, HeartbeatSnapshot::published_ns, HeartbeatSnapshot::publis VM::rolling_window, ROLLING_WINDOW_SIZE, sf_monotonic_ns(), HeartbeatState::snapshot_index, HeartbeatState::snapshots, HeartbeatSnapshot::stale_word_count, VM::stale_word_count_at_check, HeartbeatState::tick_count, RollingWindowOfTruth::total_ex HeartbeatSnapshot::total_heat, VM::total_heat_at_last_check, and HeartbeatSnapshot::window_width.

Referenced by vm_heartbeat_run_cycle(), and vm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



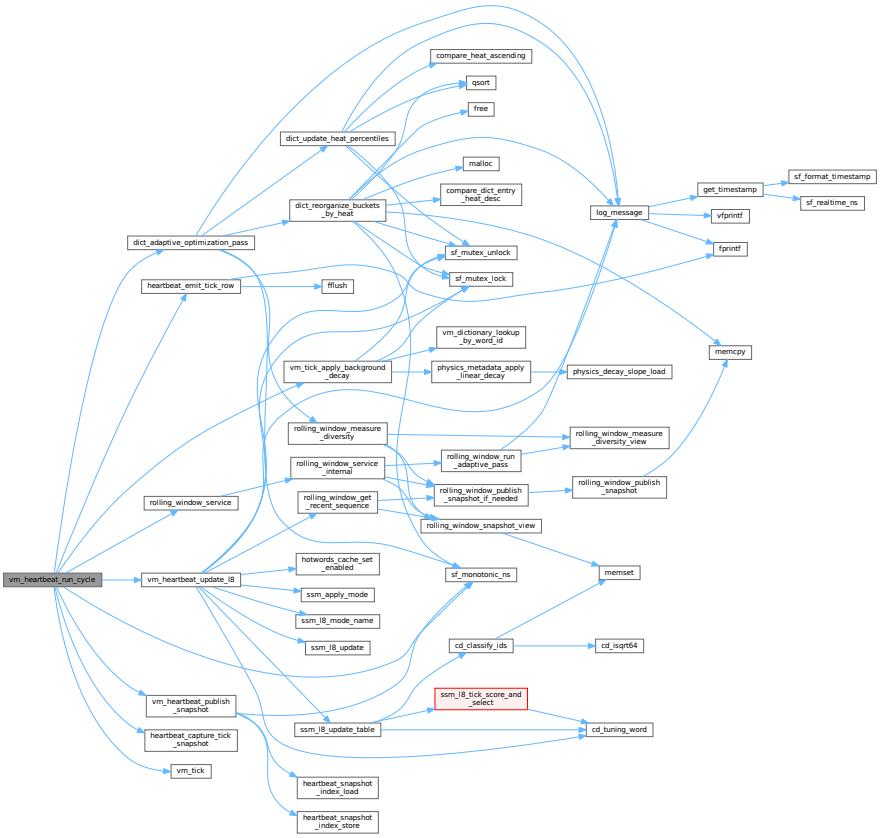
7.347.3.7 vm_heartbeat_run_cycle()

```
void vm_heartbeat_run_cycle (
            VM * vm )
```

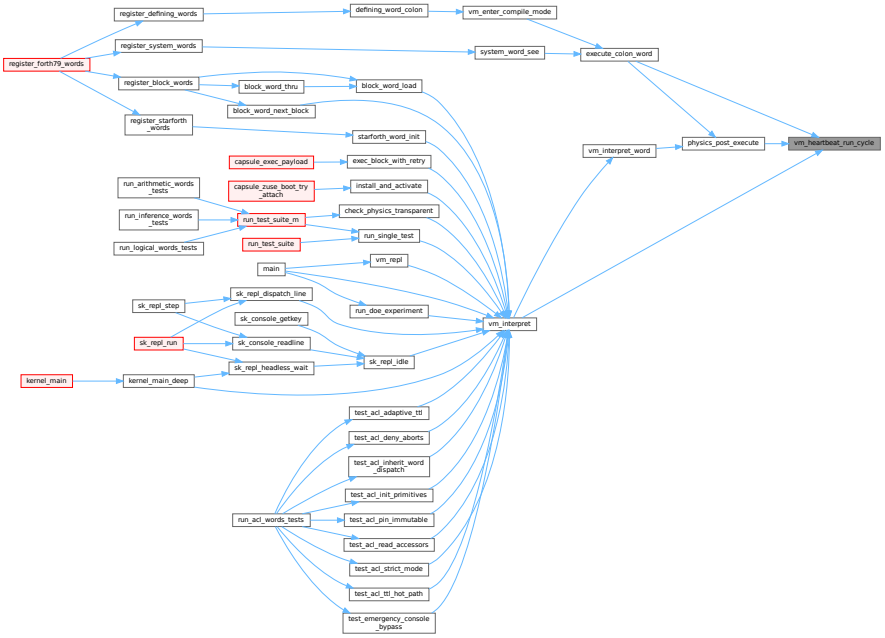
References dict_adaptive_optimization_pass(), VM::heartbeat, heartbeat_capture_tick_snapshot(), heartbeat_emit_tick_row(), HeartbeatState::heartbeat_enabled, VM::rolling_window, rolling_window_service(), sf_monotonic_ns(), vm_heartbeat_publish_snaps vm_heartbeat_update_l8(), vm_tick(), and vm_tick_apply_background_decay().

Referenced by execute_colon_word(), physics_post_execute(), and vm_interpret().

Here is the call graph for this function:



Here is the caller graph for this function:



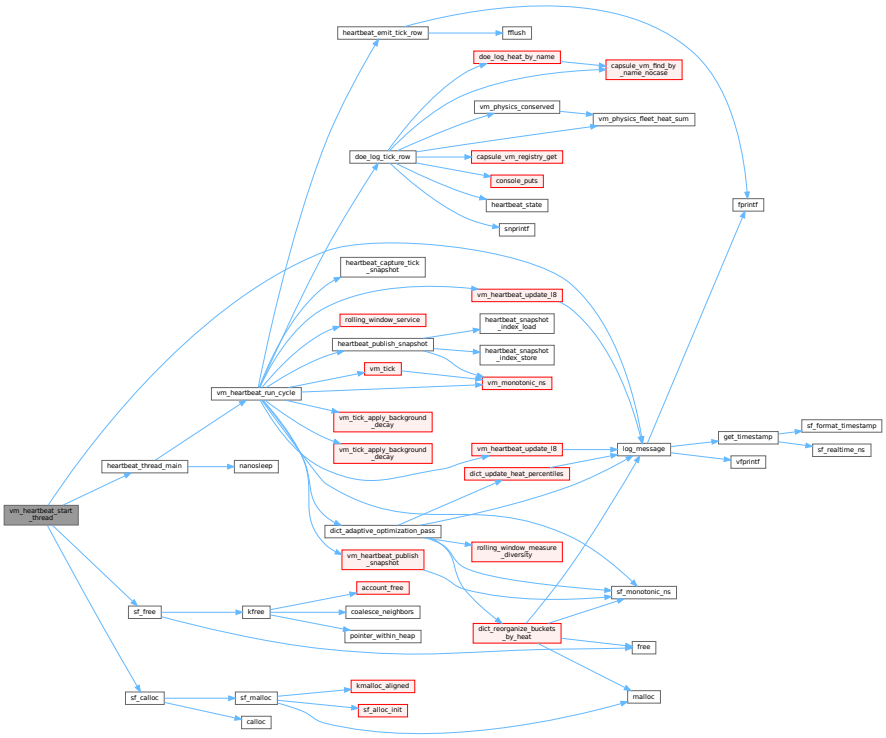
7.347.3.8 vm_heartbeat_start_thread()

```
void vm_heartbeat_start_thread (
            VM * vm )
```

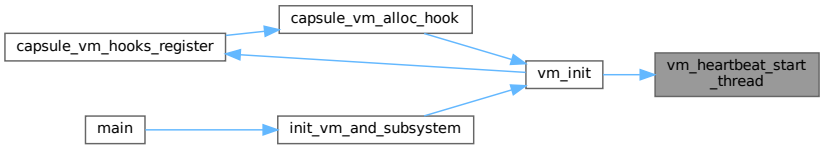
References errno, VM::heartbeat, heartbeat_thread_main(), HEARTBEAT_TICK_NS, log_message(), LOG_WARN, sf_calloc(), sf_free(), HeartbeatWorker::tick_ns, and HeartbeatState::worker.

Referenced by vm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



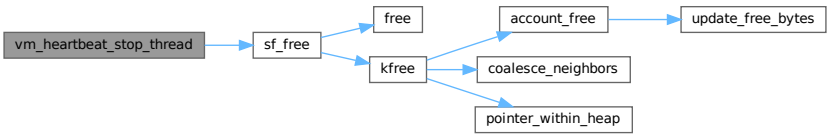
7.347.3.9 vm_heartbeat_stop_thread()

```
void vm_heartbeat_stop_thread (
            VM * vm )
```

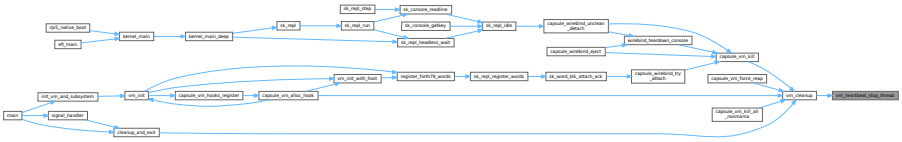
References VM::heartbeat, sf_free(), HeartbeatWorker::stop_requested, and HeartbeatState::worker.

Referenced by vm_cleanup().

Here is the call graph for this function:



Here is the caller graph for this function:



7.347.3.10 vm_init()

```
void vm_init (
            VM * vm )
```

Initialize a new virtual machine instance.

Allocates memory and initializes all VM structures including:

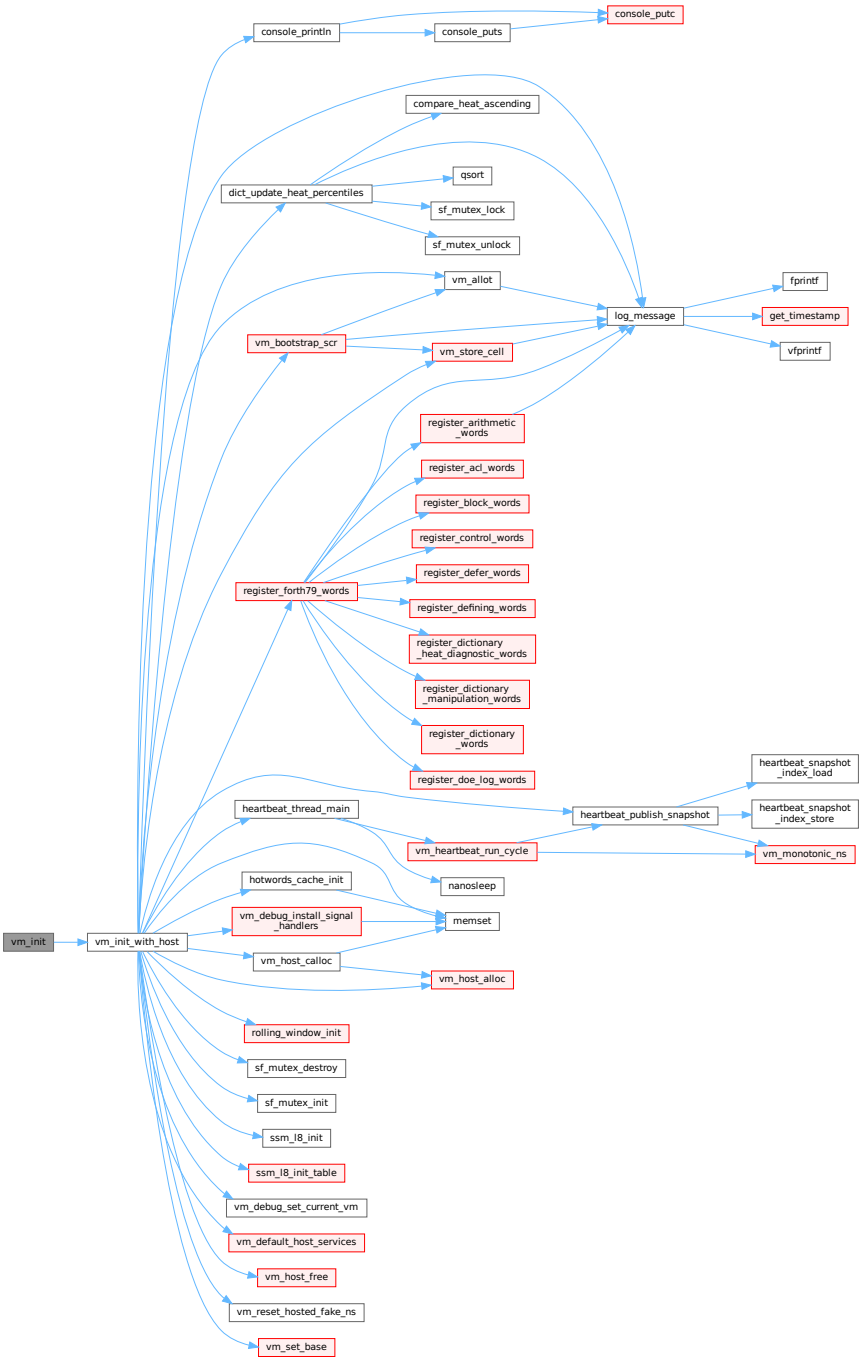
- Memory array
- Data and return stacks
- System variables (SCR, STATE, BASE)
- Dictionary
- Forth-79 wordset

Parameters

vm	Pointer to VM structure to initialize

References vm_init_with_host().

Here is the call graph for this function:



7.347.3.11 vm_set_base()

```
void vm_set_base (
            VM * vm,
            unsigned b )
```

Set the VM number base and update both the cell and the host mirror.

Clamps `b` to [2, 36] (replacing out-of-range values with 10), then writes to the FORTH `BASE` variable cell at `vm->base_addr` and to the host-mirror field `vm->base` so both sources agree.

Parameters

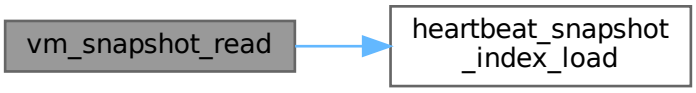
vm	VM instance to update.
b	Desired number base; clamped to [2, 36].

7.347.3.12 vm_snapshot_read()

```
void vm_snapshot_read (
            const VM * vm,
            HeartbeatSnapshot * out_snapshot )
```

References VM::heartbeat, heartbeat_snapshot_index_load(), HeartbeatState::snapshot_index, and HeartbeatState::snapshots.

Here is the call graph for this function:



7.347.3.13 vm_tick()

```
void vm_tick (
            VM * vm )
```

Central heartbeat dispatcher for time-driven tuning operations.

Aggregates all periodic optimization tasks (Loop #3 and Loop #5) into one place. Designed as plugin architecture - new tuning operations can be added as plugins.

Options for integration:

- Synchronous (now): Called from main execution loop, every N executions
- Background thread (future): Runs in separate thread, decoupled from VM execution

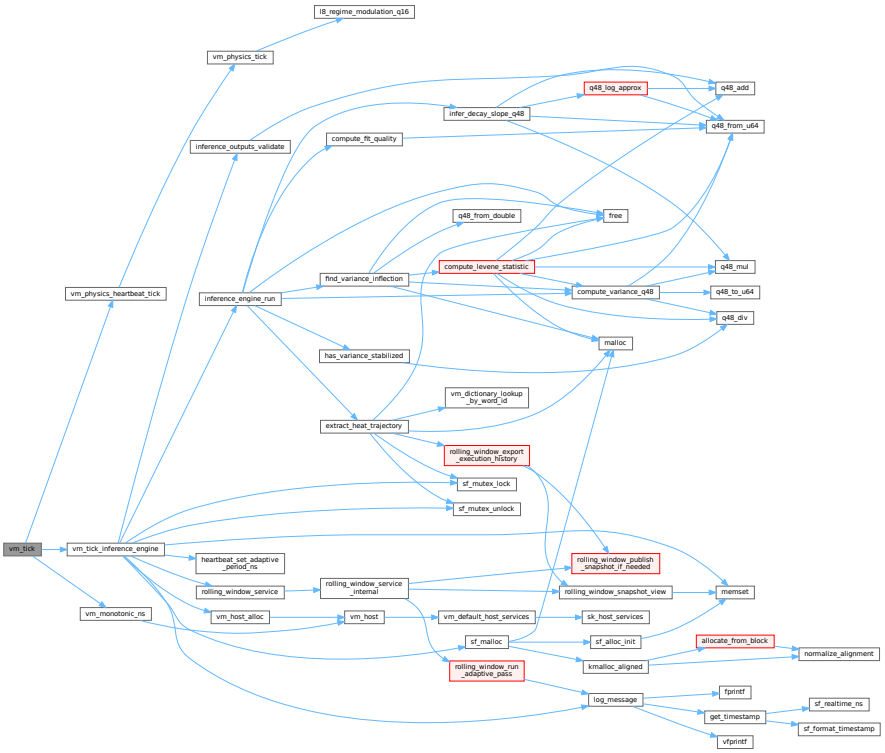
Parameters

vm	Pointer to VM instance

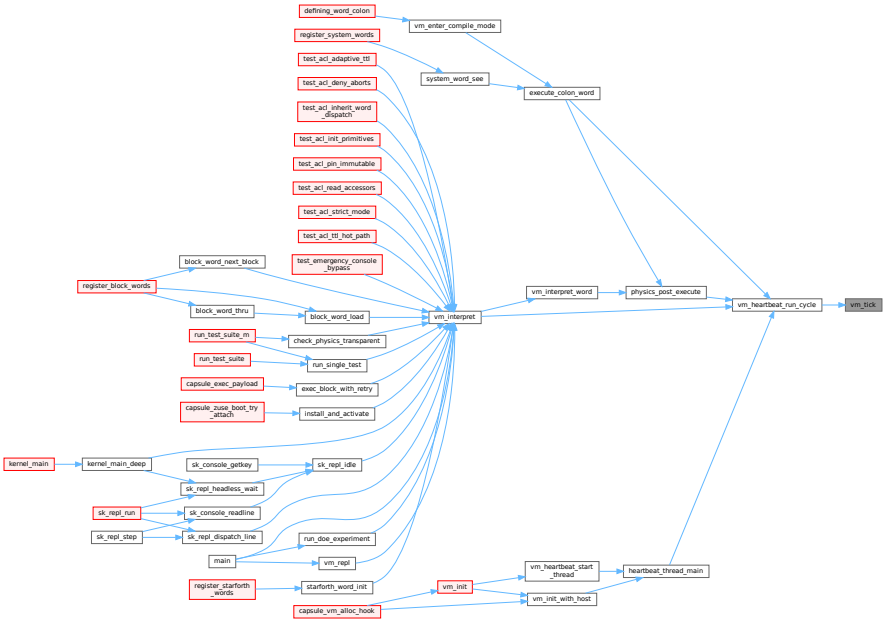
References VM::heartbeat, HeartbeatState::heartbeat_enabled, HEARTBEAT_INFERENCE_FREQUENCY, HeartbeatState::last_inference_tick, HeartbeatState::tick_count, vm_monotonic_ns(), vm_physics_heartbeat_tick(), and vm_tick_inference_engine().

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.347.3.14 vm_tick_inference_engine()

```
void vm_tick_inference_engine (
            VM * vm )
```

Phase 2: Unified Inference Engine - Adaptive Window & Decay Slope Tuning.

Coordinates inference on rolling window of truth to determine:

- Optimal adaptive window width (via variance inflection detection)
- Optimal decay slope (via exponential regression on heat trajectory)

Uses ANOVA early-exit to skip full inference when variance is stable (<5% change). All math uses Q48.16 fixed-point (integer-only, no floating-point).

Replaces legacy vm_tick_window_tuner() and vm_tick_slope_validator().

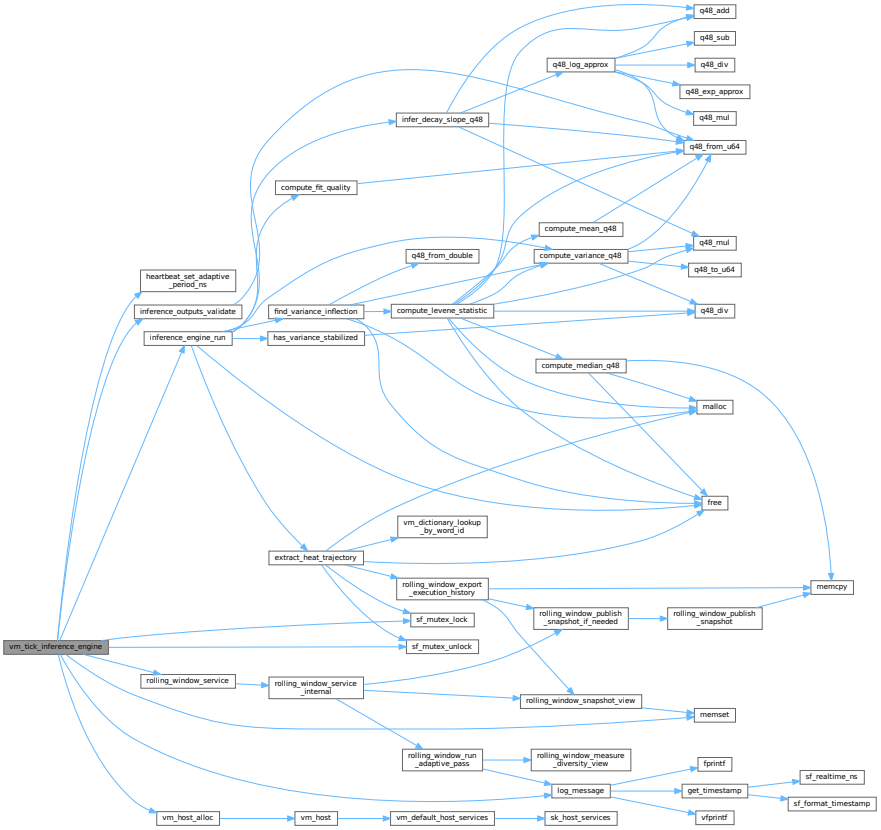
Parameters

vm	Pointer to VM instance

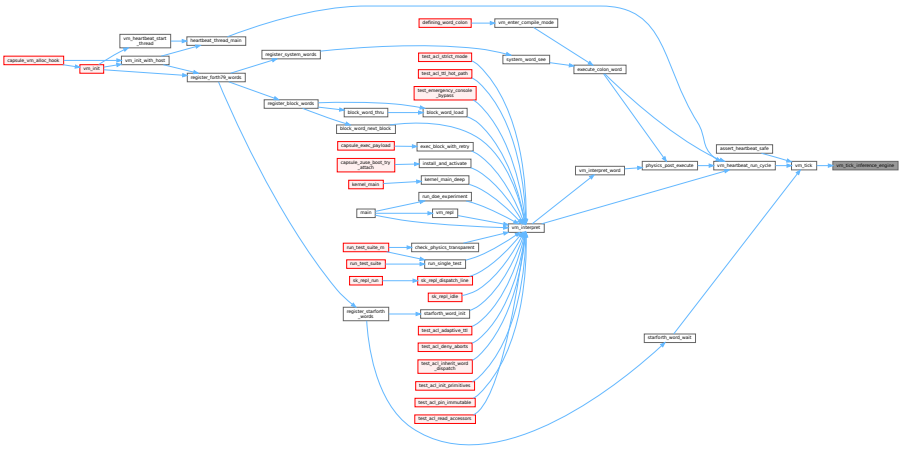
References InferenceOutputs::adaptive_decay_slope, InferenceOutputs::adaptive_window_width, VM::decay_slope_q48, VM::dict_lock, HeartbeatState::early_exit_count, InferenceOutputs::early_exited, RollingWindowOfTruth::effective_window_size, VM::heartbeat, HeartbeatState::heartbeat_enabled, heartbeat_set_adaptive_period_ns(), HEARTBEAT_TICK_NS, HOTWORDS_EXECUTION_HEAT_THRESHOLD, inference_engine_run(), inference_outputs_validate(), HeartbeatState::inference_, RollingWindowOfTruth::is_warm, VM::last_inference_outputs, HeartbeatState::last_inference_tick, VM::latest, LOG_DEBUG, LOG_ERROR, LOG_INFO, log_message(), memset(), VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::rolling_window, rolling_window_service(), sf_mutex_lock(), sf_mutex_unlock(), InferenceOutputs::slope_fit_quality_q48, VM::stale_word_count_at_check, HeartbeatState::tick_count, HeartbeatWorker::tick_ns, HeartbeatState::tick_target_ns, RollingWindowOfTruth::total_executions, VM::total_heat_at_last_check, VM::tuning_lock, vm_host_alloc(), InferenceInputs::window, RollingWindowOfTruth::window_pos, InferenceOutputs::window_variance_q48, word_count, VM::word_count_at_check, and HeartbeatState::worker.

Referenced by vm_tick().

Here is the call graph for this function:



Here is the caller graph for this function:



7.347.3.15 vm_tick_slope_validator()

```
void vm_tick_slope_validator (
            VM * vm )
```

Loop #3: Heat decay slope validation via periodic measurement.

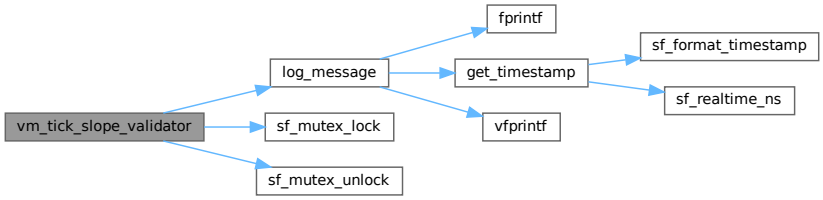
Validates that linear decay is actually helping optimize dictionary caching. Measures stale word ratio, hot word count, and average heat distribution.

Parameters

vm	Pointer to VM instance

References VM::decay_slope_direction, VM::decay_slope_q48, VM::dict_lock, VM::hot_word_count_at_check, HOTWORDS_EXECUTION_HEAT_THRESHOLD, VM::latest, LOG_INFO, log_message(), sf_mutex_lock(), sf_mutex_unlock(), VM::stale_word_count_at_check, VM::total_heat_at_last_check, VM::tuning_lock, word_count, and VM::word_count_at_check.

Here is the call graph for this function:



7.347.3.16 vm_tick_window_tuner()

```
void vm_tick_window_tuner (
            VM * vm )
```

Loop #5: Context-aware window tuning via binary chop search.

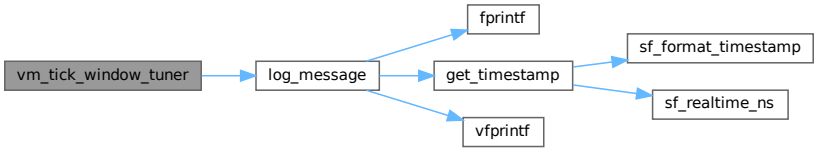
Uses prefetch accuracy to guide window size adaptation. Binary search converges on optimal effective_window_↩ size for current workload.

Parameters

vm	Pointer to VM instance

References ADAPTIVE_MIN_WINDOW_SIZE, RollingWindowOfTruth::effective_window_size, ENABLE_PIPELINING, RollingWindowOfTruth::is_warm, PipelineGlobalMetrics::last_checked_accuracy, PipelineGlobalMetrics::last_checked_window_size, LOG_INFO, log_message(), VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, VM::rolling_window, ROLLING_WINDOW_SIZE, and PipelineGlobalMetrics::window_tuning_checks.

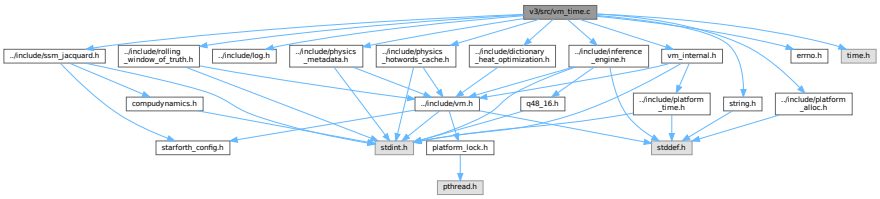
Here is the call graph for this function:



7.348 v3/src/vm_time.c File Reference

```
#include "vm_internal.h"
#include "../include/inference_engine.h"
#include "../include/log.h"
#include "../include/physics_metadata.h"
#include "../include/physics_hotwords_cache.h"
#include "../include/rolling_window_of_truth.h"
#include "../include/dictionary_heat_optimization.h"
#include "../include/ssm_jacquard.h"
#include "../include/platform_alloc.h"
#include <string.h>
#include <errno.h>
#include <time.h>
```

Include dependency graph for vm_time.c:



Functions

- static void vm_tick_apply_background_decay (VM *vm, uint64_t now_ns)
- static void vm_heartbeat_update_l8 (VM *vm)

  L8 FINAL INTEGRATION: Jacquard mode selector heartbeat update.
- void vm_heartbeat_publish_snapshot (VM *vm)
- void vm_tick (VM *vm)

  Central heartbeat dispatcher for time-driven tuning operations.
- void vm_tick_window_tuner (VM *vm)

  Loop #5: Context-aware window tuning via binary chop search.
- void vm_tick_slope_validator (VM *vm)

  Loop #3: Heat decay slope validation via periodic measurement.
- void vm_heartbeat_run_cycle (VM *vm)
- void vm_snapshot_read (const VM *vm, HeartbeatSnapshot *out_snapshot)
- void vm_tick_inference_engine (VM *vm)

  Phase 2: Unified Inference Engine - Adaptive Window & Decay Slope Tuning.
- void vm_heartbeat_start_thread (VM *vm)
- void vm_heartbeat_stop_thread (VM *vm)

7.348.1 Function Documentation

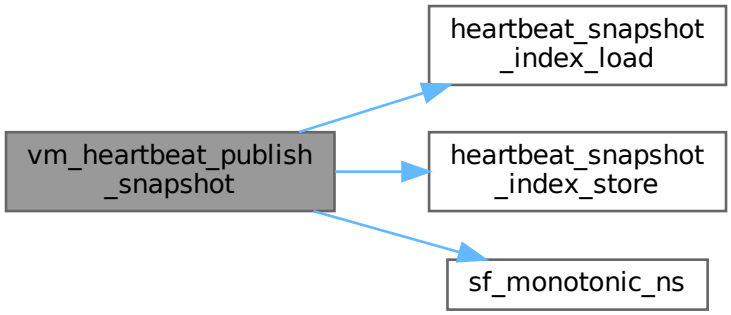
7.348.1.1 vm_heartbeat_publish_snapshot()

```
void vm_heartbeat_publish_snapshot (
            VM * vm )
```

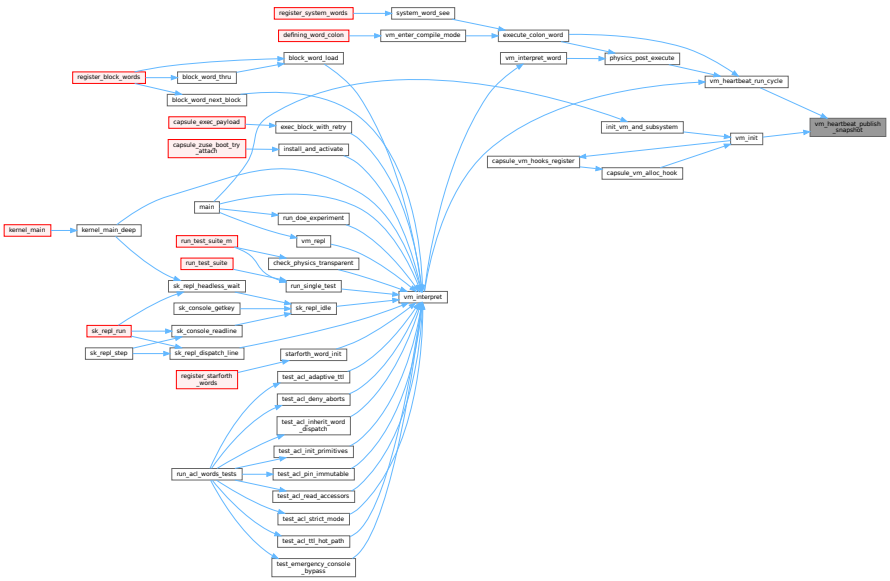
References HeartbeatSnapshot::actual_window_size, HeartbeatSnapshot::decay_slope_q48, VM::decay_slope_q48, RollingWindowOfTruth::effective_window_size, VM::heartbeat, heartbeat_snapshot_index_load(), heartbeat_snapshot_index_store(), HeartbeatSnapshot::hot_word_count, VM::hot_word_count_at_check, HeartbeatSnapshot::published_ns, HeartbeatSnapshot::publish VM::rolling_window, ROLLING_WINDOW_SIZE, sf_monotonic_ns(), HeartbeatState::snapshot_index, HeartbeatState::snapshots, HeartbeatSnapshot::stale_word_count, VM::stale_word_count_at_check, HeartbeatState::tick_count, RollingWindowOfTruth::total_ex HeartbeatSnapshot::total_heat, VM::total_heat_at_last_check, and HeartbeatSnapshot::window_width.

Referenced by vm_heartbeat_run_cycle(), and vm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.348.1.2 vm_heartbeat_run_cycle()

```
void vm_heartbeat_run_cycle (
            VM * vm )
```

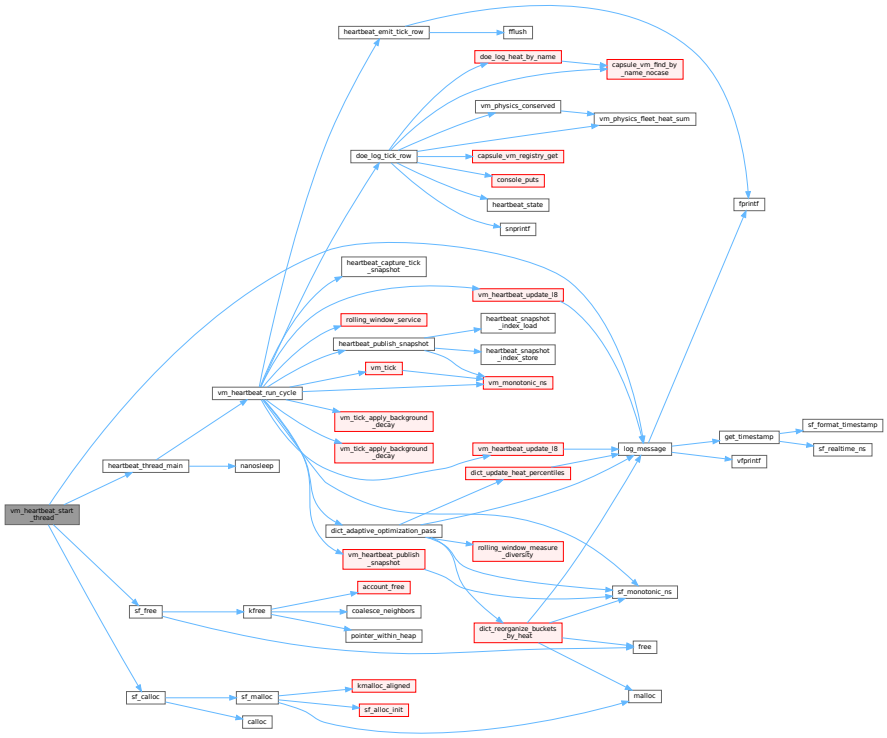
7.348.1.3 vm_heartbeat_start_thread()

```
void vm_heartbeat_start_thread (
            VM * vm )
```

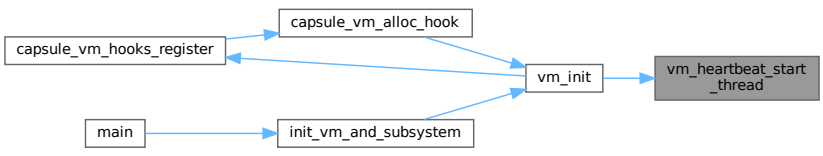
References errno, VM::heartbeat, heartbeat_thread_main(), HEARTBEAT_TICK_NS, log_message(), LOG_WARN, sf_calloc(), sf_free(), HeartbeatWorker::tick_ns, and HeartbeatState::worker.

Referenced by vm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



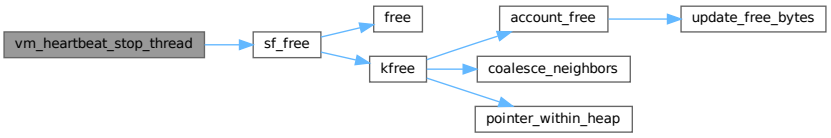
7.348.1.4 vm_heartbeat_stop_thread()

```
void vm_heartbeat_stop_thread (
            VM * vm )
```

References VM::heartbeat, sf_free(), HeartbeatWorker::stop_requested, and HeartbeatState::worker.

Referenced by vm_cleanup().

Here is the call graph for this function:



Here is the caller graph for this function:

7.348.1.5 vm_heartbeat_update_l8()

```
static void vm_heartbeat_update_l8 (
            VM * vm )  [static]
```

L8 FINAL INTEGRATION: Jacquard mode selector heartbeat update.

Collects metrics from L1-L7 physics layers and feeds them to the L8 Jacquard mode selector. L8 then chooses the optimal configuration mode based on workload characteristics and applies it to the runtime.

This is the SOLE policy engine - all adaptive decisions flow through L8.

Parameters

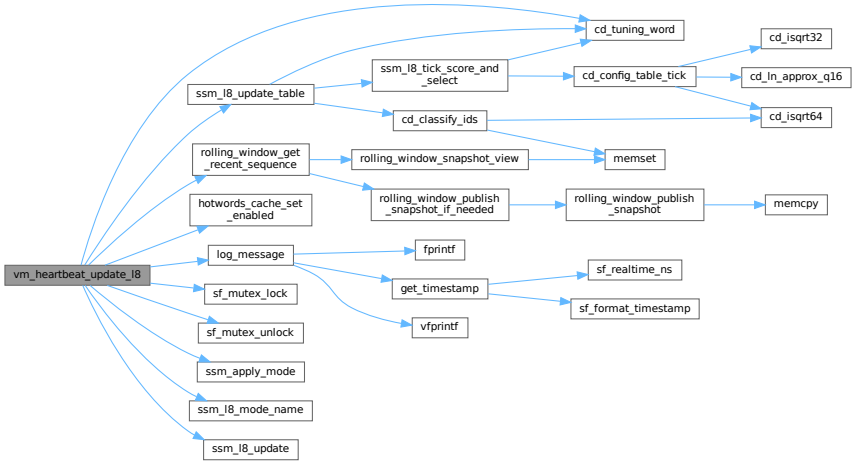
vm	Pointer to VM instance

References CD_MAX_CLASSIFY_DEPTH, cd_tuning_word(), CDConfigEntry::config_bits, CDConfigTable::current_config, ssm_l8_state_t::current_mode, CDConfigTable::current_regime, ssm_l8_metrics_t::cv, VM::decay_slope_q48, InferenceOutputs::early_exited, RollingWindowOfTruth::effective_window_size, CDConfigTable::entries, ssm_l8_metrics_t::entropy, VM::heartbeat, VM::hotwords_cache, hotwords_cache_set_enabled(), CDTuning::id_window_depth, ssm_l8_metrics_t::inference_ea ssm_l8_metrics_t::inference_ran_this_tick, ssm_config_t::L1_heat_tracking, ssm_config_t::L2_rolling_window, ssm_config_t::L3_linear_decay, ssm_config_t::L4_pipelining, ssm_config_t::L5_window_inference, ssm_config_t::L6_decay_inferenc

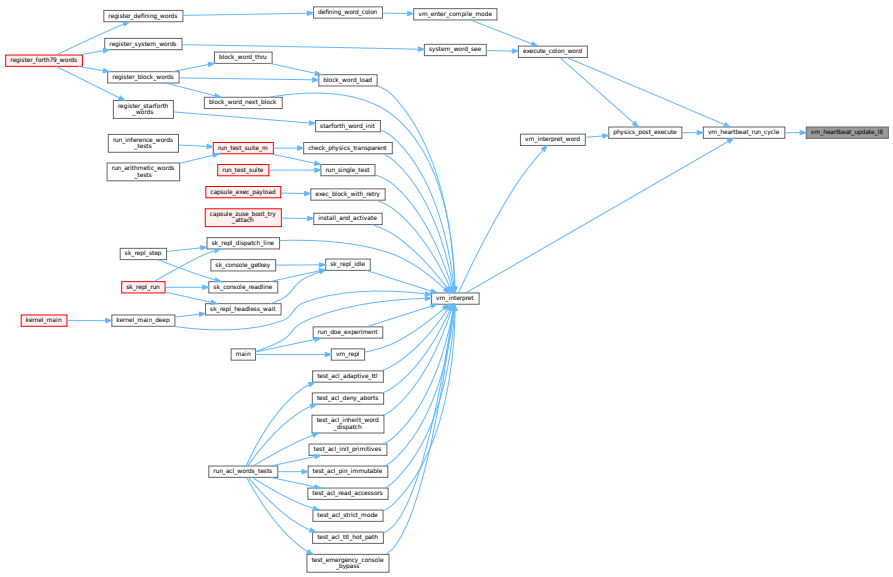
ssm_config_t::L7_adaptive_heartrate, VM::last_inference_outputs, HeartbeatState::last_inference_tick, LOG_DEBUG, log_message(), CDConfigTable::num_configs, VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, CDConfigTable::regime_scores, VM::rolling_window, rolling_window_get_recent_sequence(), ROLLING_WINDOW_SIZE, sf_mutex_lock(), sf_mutex_unlock(), ssm_apply_mode(), VM::ssm_config, ssm_l8_mode_name(), VM::ssm_l8_state, ssm_l8_update(), ssm_l8_update_table(), ssm_l8_metrics_t::stability_score, ssm_l8_state_t::table, ssm_l8_metrics_t::temporal_decay, HeartbeatState::tick_count, RollingWindowOfTruth::total_executions, and VM::tuning_lock.

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.348.1.6 vm_snapshot_read()

```
void vm_snapshot_read (
            const VM * vm,
            HeartbeatSnapshot * out_snapshot )
```

7.348.1.7 vm_tick()

```
void vm_tick (
            VM * vm )
```

Central heartbeat dispatcher for time-driven tuning operations.

Aggregates all periodic optimization tasks (Loop #3 and Loop #5) into one place. Designed as plugin architecture - new tuning operations can be added as plugins.

Options for integration:

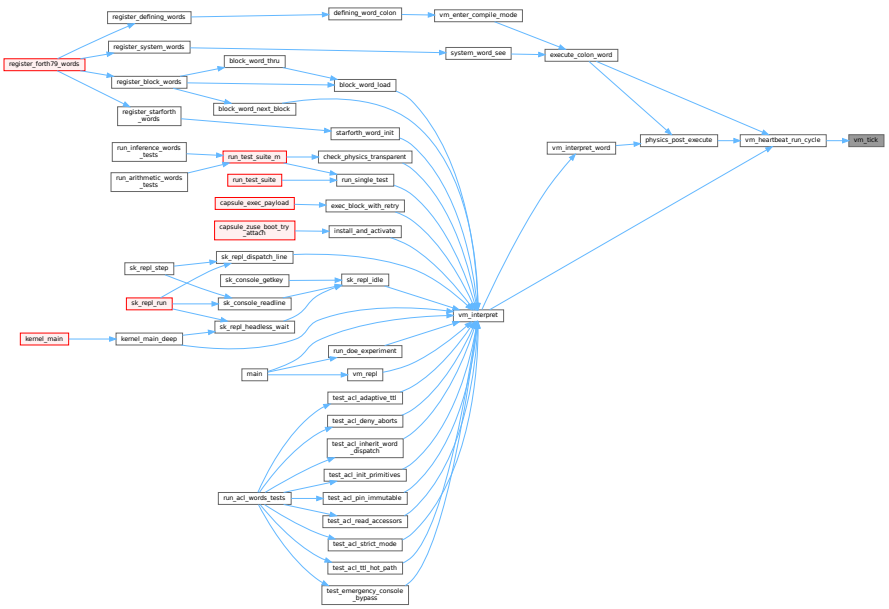
- Synchronous (now): Called from main execution loop, every N executions
- Background thread (future): Runs in separate thread, decoupled from VM execution

Parameters

vm	Pointer to VM instance

Referenced by vm_heartbeat_run_cycle().

Here is the caller graph for this function:



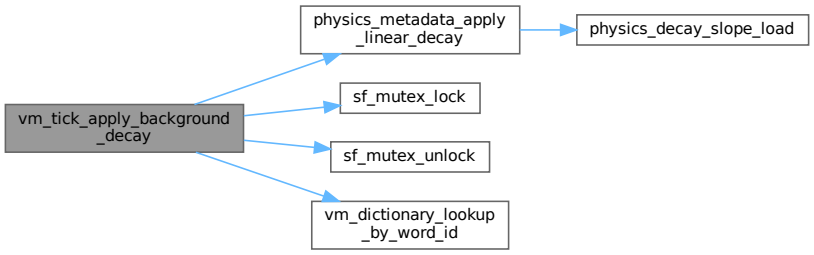
7.348.1.8 vm_tick_apply_background_decay()

```
static void vm_tick_apply_background_decay (
            VM * vm,
            uint64_t now_ns )  [static]
```

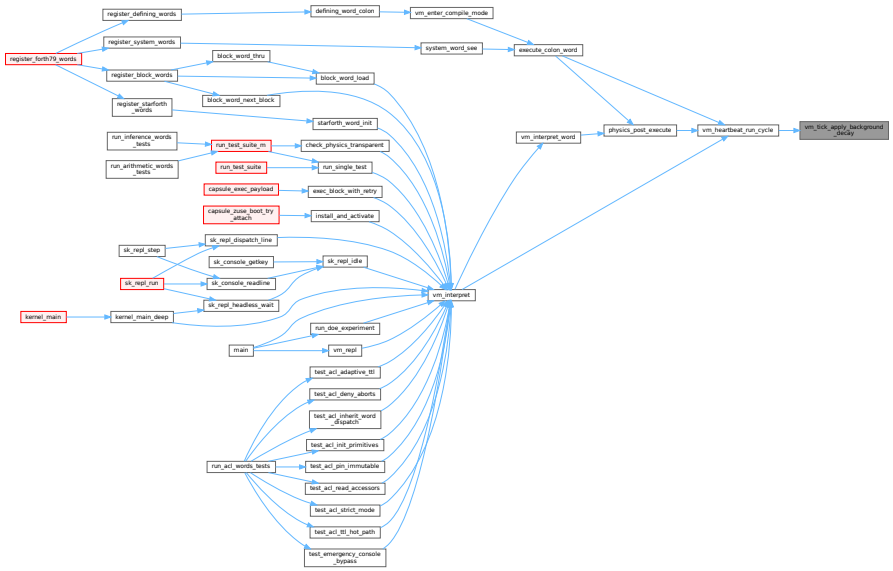
References VM::dict_lock, VM::heartbeat, HEARTBEAT_DECAY_BATCH, VM::heartbeat_decay_cursor_id, DictPhysics::last_decay_ns, DictPhysics::last_decay_tick, VM::latest, DictEntry::link, DictEntry::physics, physics_metadata_apply_line sf_mutex_lock(), sf_mutex_unlock(), HeartbeatState::tick_count, vm_dictionary_lookup_by_word_id(), DictEntry::word_id, and WORD_ID_INVALID.

Referenced by vm_heartbeat_run_cycle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.348.1.9 vm_tick_inference_engine()

```
void vm_tick_inference_engine (
            VM * vm )
```

Phase 2: Unified Inference Engine - Adaptive Window & Decay Slope Tuning.

Coordinates inference on rolling window of truth to determine:

- Optimal adaptive window width (via variance inflection detection)
- Optimal decay slope (via exponential regression on heat trajectory)

Uses ANOVA early-exit to skip full inference when variance is stable (<5% change). All math uses Q48.16 fixed-point (integer-only, no floating-point).

Replaces legacy vm_tick_window_tuner() and vm_tick_slope_validator().

Parameters

vm	Pointer to VM instance

Referenced by vm_tick().

Here is the caller graph for this function:

7.348.1.10 vm_tick_slope_validator()

```
void vm_tick_slope_validator (
            VM * vm )
```

Loop #3: Heat decay slope validation via periodic measurement.

Validates that linear decay is actually helping optimize dictionary caching. Measures stale word ratio, hot word count, and average heat distribution.

Parameters

vm	Pointer to VM instance

7.348.1.11 vm_tick_window_tuner()

```
void vm_tick_window_tuner (
            VM * vm )
```

Loop #5: Context-aware window tuning via binary chop search.

Uses prefetch accuracy to guide window size adaptation. Binary search converges on optimal effective_window_↩ size for current workload.

Parameters

vm	Pointer to VM instance

7.349 v3/src/word_registry.c File Reference

```
#include "../include/word_registry.h"
#include "../include/log.h"
#include <string.h>
#include "word_source/include/acl_words.h"
#include "word_source/include/stack_words.h"
#include "word_source/include/return_stack_words.h"
#include "word_source/include/memory_words.h"
#include "word_source/include/dictionary_words.h"
#include "word_source/include/arithmetic_words.h"
#include "word_source/include/double_words.h"
#include "word_source/include/mixed_arithmetic_words.h"
#include "word_source/include/logical_words.h"
#include "word_source/include/io_words.h"
#include "word_source/include/format_words.h"
#include "word_source/include/string_words.h"
#include "word_source/include/control_words.h"
#include "word_source/include/defining_words.h"
#include "word_source/include/dictionary_manipulation_words.h"
#include "word_source/include/vocabulary_words.h"
#include "word_source/include/block_words.h"
#include "word_source/include/editor_words.h"
#include "word_source/include/system_words.h"
#include "word_source/include/starforth_words.h"
#include "word_source/include/physics_benchmark_words.h"
#include "word_source/include/physics_pipelining_diagnostic_words.h"
#include "word_source/include/physics_freeze_words.h"
#include "word_source/include/dictionary_heat_diagnostic_words.h"
#include "word_source/include/log_words.h"
#include "word_source/include/q48_words.h"
#include "word_source/include/inference_words.h"
#include "word_source/include/defer_words.h"
#include "word_source/include/framebuffer_words.h"
#include "word_source/include/keyboard_words.h"
#include "word_source/include/ttf_words.h"
#include "word_source/include/scroll_words.h"
```

Include dependency graph for word_registry.c:

Functions

- void register_word (VM ∗vm, const char ∗name, word_func_t func)

Registers a single FORTH word in the virtual machine.

- void register_forth79_words (VM *vm)

  Registers all standard FORTH-79 words in the virtual machine.

7.349.1 Function Documentation

7.349.1.1 register_forth79_words()

```
void register_forth79_words (
            VM * vm )
```

Registers all standard FORTH-79 words in the virtual machine.

Parameters

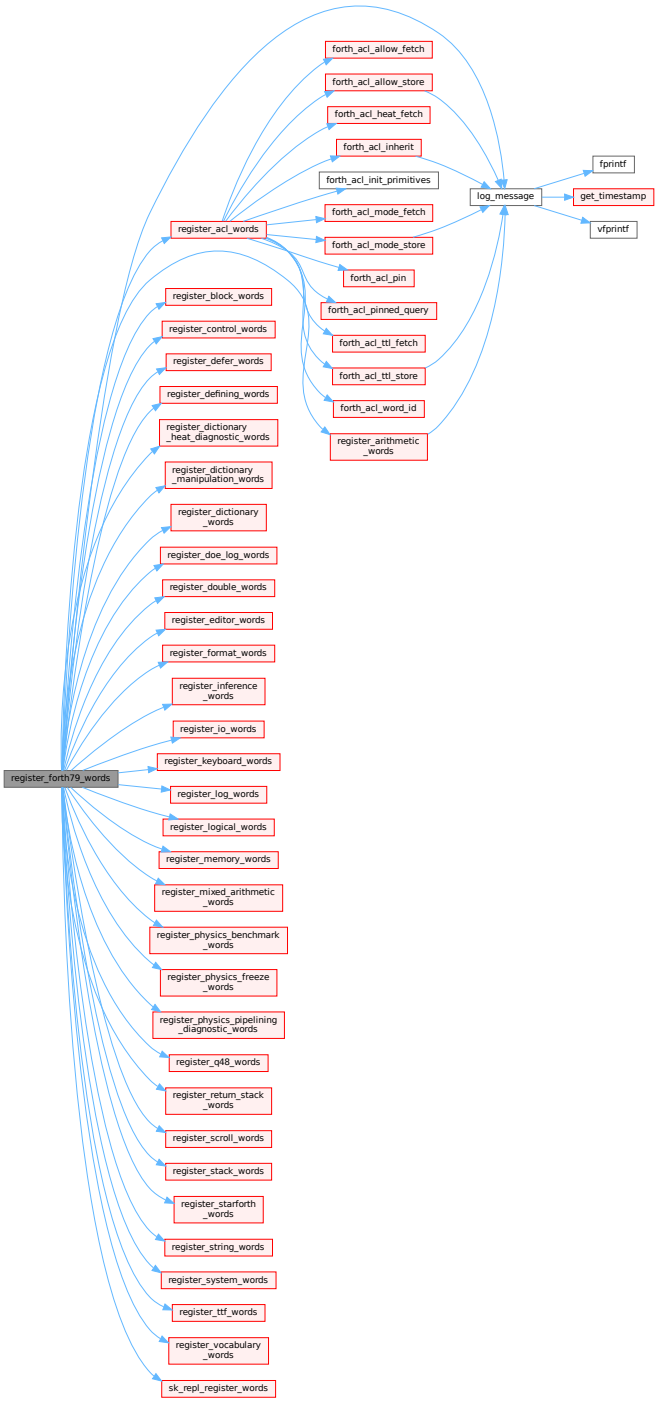
vm	Pointer to the virtual machine instance

Registers all 18 FORTH-79 word modules in dependency order, starting with fundamental stack operations and building up to more complex functionality like control flow and editing.

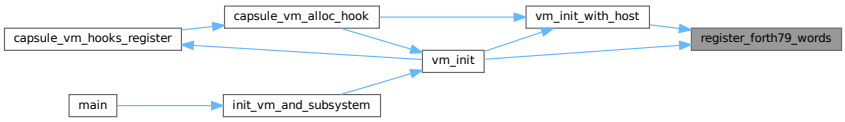
References LOG_INFO, log_message(), register_acl_words(), register_arithmetic_words(), register_block_words(), register_control_words(), register_defer_words(), register_defining_words(), register_dictionary_heat_diagnostic_words(), register_dictionary_manipulation_words(), register_dictionary_words(), register_doe_log_words(), register_double_words(), register_editor_words(), register_format_words(), register_inference_words(), register_io_words(), register_keyboard_words(), register_log_words(), register_logical_words(), register_memory_words(), register_mixed_arithmetic_words(), register_physics_benchmark_words(), register_physics_freeze_words(), register_physics_pipelining_diagnostic_words(), register_q48_words(), register_return_stack_words(), register_scroll_words(), register_stack_words(), register_starforth_words(), register_string_words(), register_system_words(), register_ttf_words(), register_vocabulary_words(), and sk_repl_register_words().

Referenced by vm_init(), and vm_init_with_host().

Here is the call graph for this function:



Here is the caller graph for this function:



7.349.1.2 register_word()

```
void register_word (
            VM * vm,
            const char * name,
            word_func_t func )
```

Registers a single FORTH word in the virtual machine.

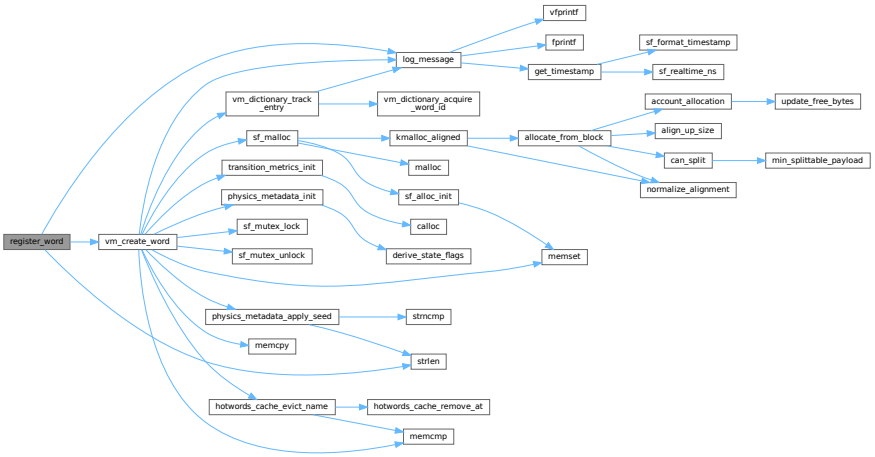
Parameters

vm	Pointer to the virtual machine instance
name	Name of the FORTH word to register
func	Function pointer to the word's implementation

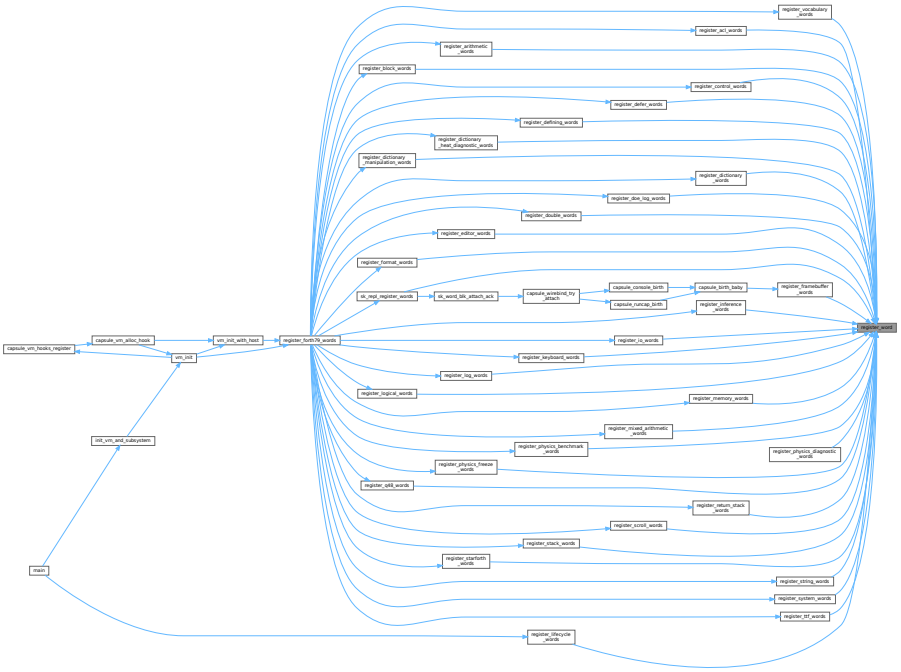
References entry, func, LOG_DEBUG, LOG_ERROR, log_message(), name, strlen(), and vm_create_word().

Referenced by register_acl_words(), register_arithmetic_words(), register_block_words(), register_control_words(), register_defer_words(), register_defining_words(), register_dictionary_heat_diagnostic_words(), register_dictionary_manipulation_wo register_dictionary_words(), register_doe_log_words(), register_double_words(), register_editor_words(), register_format_words(), register_framebuffer_words(), register_inference_words(), register_io_words(), register_keyboard_words(), register_lifecycle_words(), register_log_words(), register_logical_words(), register_memory_words(), register_mixed_arithmetic_word register_physics_benchmark_words(), register_physics_diagnostic_words(), register_physics_freeze_words(), register_q48_words(), register_return_stack_words(), register_scroll_words(), register_stack_words(), register_starforth_words(), register_string_words(), register_system_words(), register_ttf_words(), register_vocabulary_words(), and sk_repl_register_words().

Here is the call graph for this function:



Here is the caller graph for this function:

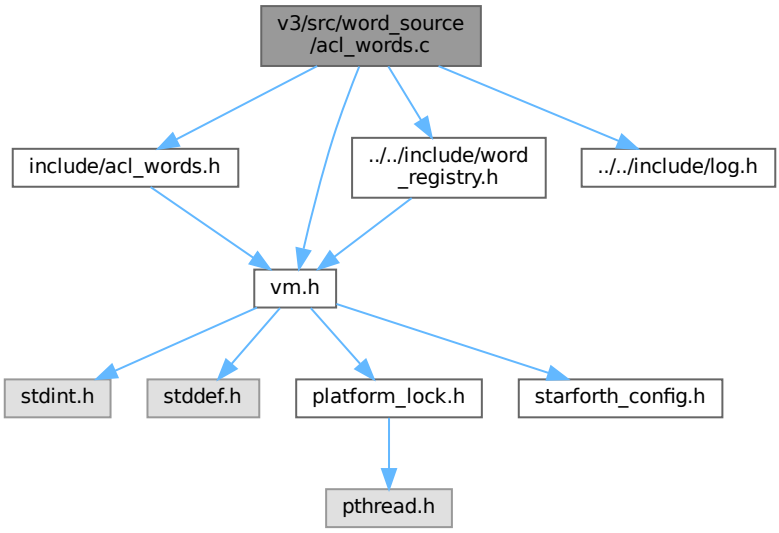


7.350 v3/src/word_source/acl_words.c File Reference

```
#include "include/acl_words.h"
#include "../../include/vm.h"
#include "../../include/word_registry.h"
```

```
#include "../../include/log.h"
```

Include dependency graph for acl_words.c:



Functions

- static DictEntry * pop_xt (VM *vm)

  Pop an execution token (XT) from the data stack and return the corresponding DictEntry pointer.
- static void forth_acl_mode_fetch (VM *vm)

  Implements FORTH word `ACL-MODE@` — fetch ACL enforcement mode.
- static void forth_acl_mode_store (VM *vm)

  Implements FORTH word `ACL-MODE!` — store ACL enforcement mode.
- static void forth_acl_pinned_query (VM *vm)

  Implements FORTH word `ACL-PINNED?` — query pin status.
- static void forth_acl_pin (VM *vm)

  Implements FORTH word `ACL-PIN` — pin a word's ACL entry.
- static void forth_acl_ttl_fetch (VM *vm)

  Implements FORTH word `ACL-TTL@` — fetch TTL countdown value.
- static void forth_acl_ttl_store (VM *vm)

  Implements FORTH word `ACL-TTL!` — store TTL countdown value.
- static void forth_acl_allow_fetch (VM *vm)

  Implements FORTH word `ACL-ALLOW@` — fetch cached allow/deny flag.
- static void forth_acl_allow_store (VM *vm)

  Implements FORTH word `ACL-ALLOW!` — store cached allow/deny flag.
- static void forth_acl_heat_fetch (VM *vm)

  Implements FORTH word `ACL-HEAT@` — fetch execution heat.
- static void forth_acl_word_id (VM *vm)

  Implements FORTH word `ACL-WORD-ID` — fetch stable word identifier.
- static void forth_acl_inherit (VM *vm)

  Implements FORTH word `ACL-INHERIT` — inherit ACL policy from parent.
- static void forth_acl_init_primitives (VM *vm)

Implements FORTH word `ACL-INIT-PRIMITIVES` — reset ACL fields on all unpinned dictionary entries.

- void acl_inherit_entry (DictEntry *src, DictEntry *dst)

  Direct C entry point for ACL policy inheritance on the capsule birth path.

- void register_acl_words (VM *vm)

  Register all ACL C-primitive words with the VM dictionary.

7.350.1 Function Documentation

7.350.1.1 acl_inherit_entry()

```
void acl_inherit_entry (
            DictEntry * src,
            DictEntry * dst )
```

Direct C entry point for ACL policy inheritance on the capsule birth path.

Copies `acl_mode` from `src` to `dst`, clears `acl_pinned`, resets `acl_ttl` to 0, and sets `acl_allow` to 1. Called from capsule birth code (not from FORTH) — no stack manipulation is performed.

Parameters

src	Source DictEntry whose `acl_mode` is inherited
dst	Destination DictEntry whose ACL fields are updated

References DictEntry::acl_allow, DictEntry::acl_mode, DictEntry::acl_pinned, and DictEntry::acl_ttl.

Referenced by forth_acl_inherit(), and test_acl_inherit().

Here is the caller graph for this function:

7.350.1.2 forth_acl_allow_fetch()

```
static void forth_acl_allow_fetch (
            VM * vm )  [static]
```

Implements FORTH word `ACL-ALLOW@` — fetch cached allow/deny flag.

Stack effect: (xt – flag) Pushes the cached ACL decision for the word: -1 = allowed, 0 = denied. This cached value is checked on every execution in TTL mode; the cache is refreshed by `acl_recheck()` when TTL expires.

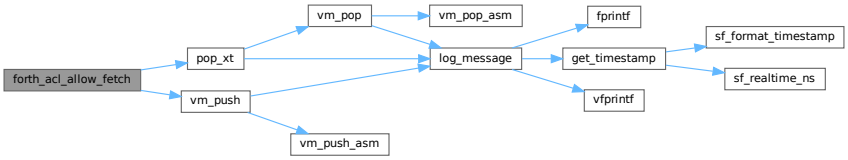
Parameters

vm	Pointer to the VM structure

References DictEntry::acl_allow, pop_xt(), and vm_push().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.3 forth_acl_allow_store()

```
static void forth_acl_allow_store (
            VM * vm )  [static]
```

Implements FORTH word `ACL-ALLOW!` — store cached allow/deny flag.

Stack effect: (flag xt –) Sets the cached ACL decision: any non-zero `flag` = allowed, 0 = denied. No-op if the word is pinned.

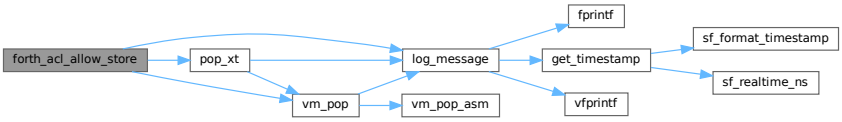
Parameters

vm	Pointer to the VM structure

References DictEntry::acl_allow, DictEntry::acl_pinned, VM::dsp, VM::error, LOG_ERROR, log_message(), pop_xt(), and vm_pop().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.4 forth_acl_heat_fetch()

```
static void forth_acl_heat_fetch (
            VM * vm )  [static]
```

Implements FORTH word `ACL-HEAT@` — fetch execution heat.

Stack effect: (xt – heat) Pushes the `execution_heat` counter for the word identified by `xt`. Used by `ACL-TTL-COMPUTE` in `ACL.4th` to calibrate the TTL value based on how frequently the word is executed.

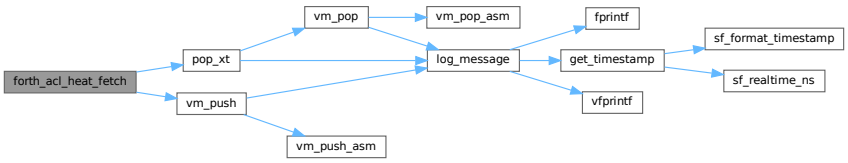
Parameters

vm	Pointer to the VM structure

References DictEntry::execution_heat, pop_xt(), and vm_push().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.5 forth_acl_inherit()

```
static void forth_acl_inherit (
            VM * vm )  [static]
```

Implements FORTH word ACL-INHERIT — inherit ACL policy from parent.

Stack effect: (src dst –) Copies acl_mode from src to dst, then clears dst->acl_pinned, resets acl_ttl to 0, and sets acl_allow to 1 (optimistic default). This is a C primitive because FORTH code cannot clear acl_pinned — pin is a one-way ratchet that only C can reset during inheritance.

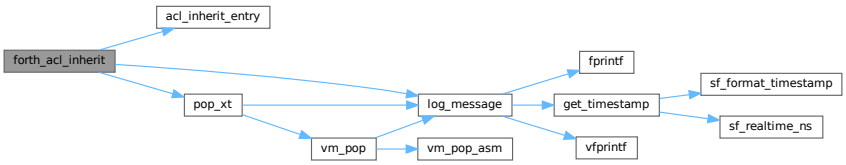
Parameters

vm	Pointer to the VM structure

References acl_inherit_entry(), VM::dsp, VM::error, LOG_ERROR, log_message(), and pop_xt().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.6 forth_acl_init_primitives()

```
static void forth_acl_init_primitives (
            VM * vm )  [static]
```

Implements FORTH word ACL-INIT-PRIMITIVES — reset ACL fields on all unpinned dictionary entries.

Stack effect: (–) Walks the entire dictionary and resets ACL fields on every entry that is not pinned: sets acl_ttl = 0 (force recheck on first execution), acl_allow = 1 (optimistic default), acl_mode = ACL_MODE_TTL. Called by ACL-BOOT in ACL.4th immediately after the ACL capsule finishes loading.

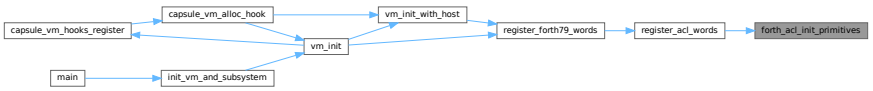
Parameters

vm	Pointer to the VM structure

References ACL_MODE_TTL, and VM::latest.

Referenced by register_acl_words().

Here is the caller graph for this function:



7.350.1.7 forth_acl_mode_fetch()

```
static void forth_acl_mode_fetch (
            VM * vm )  [static]
```

Implements FORTH word `ACL-MODE@` — fetch ACL enforcement mode.

Stack effect: (xt – mode) Pushes the enforcement mode of the word identified by `xt`: 0 = STRICT (allow/deny is permanent), 1 = TTL (expires after countdown).

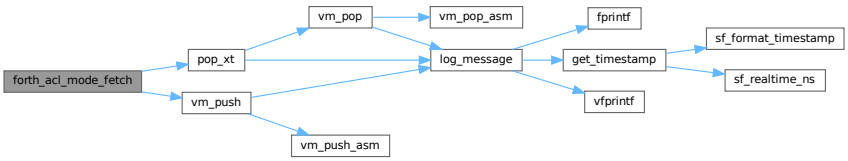
Parameters

vm	Pointer to the VM structure

References DictEntry::acl_mode, pop_xt(), and vm_push().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.8 forth_acl_mode_store()

```
static void forth_acl_mode_store (
            VM * vm )  [static]
```

Implements FORTH word `ACL-MODE!` — store ACL enforcement mode.

Stack effect: (mode xt –) Sets the enforcement mode of the word identified by `xt`. No-op if the word is pinned — pinned entries are immutable.

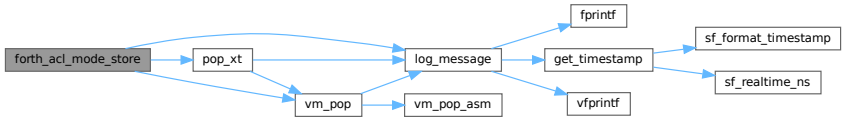
Parameters

vm	Pointer to the VM structure

References DictEntry::acl_mode, DictEntry::acl_pinned, VM::dsp, VM::error, LOG_ERROR, log_message(), pop_xt(), and vm_pop().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.9 forth_acl_pin()

```
static void forth_acl_pin (
            VM * vm )  [static]
```

Implements FORTH word `ACL-PIN` — pin a word's ACL entry.

Stack effect: (xt –) Sets `acl_pinned` = 1 on the word identified by `xt`. This is a one-way ratchet: once set it can never be cleared, even by ACL-INHERIT. Use with ' WORD ACL-PIN in FORTH, never from C policy code.

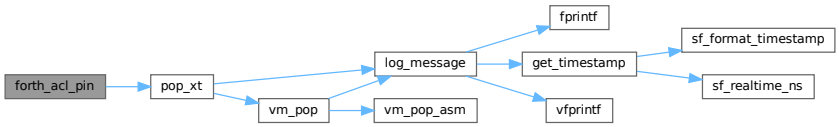
Parameters

vm	Pointer to the VM structure

References DictEntry::acl_pinned, and pop_xt().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.10 forth_acl_pinned_query()

```
static void forth_acl_pinned_query (
            VM * vm )  [static]
```

Implements FORTH word `ACL-PINNED?` — query pin status.

Stack effect: (xt – flag) Pushes -1 (true) if the word is pinned, 0 (false) otherwise. A pinned word's ACL fields cannot be modified by any FORTH operation.

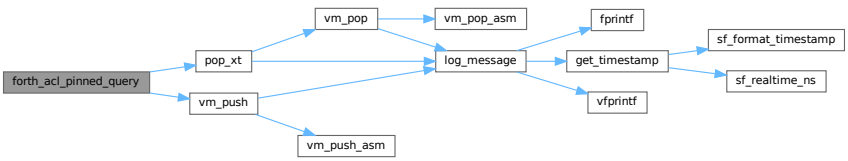
Parameters

vm	Pointer to the VM structure

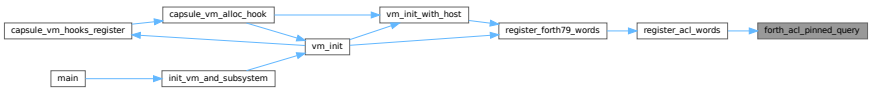
References DictEntry::acl_pinned, pop_xt(), and vm_push().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.350.1.11 forth_acl_ttl_fetch()

```
static void forth_acl_ttl_fetch (
            VM * vm )  [static]
```

Implements FORTH word `ACL-TTL@` — fetch TTL countdown value.

Stack effect: (xt – n) Pushes the current TTL (time-to-live) countdown of the word's ACL entry. When TTL reaches zero in TTL mode the interpreter calls `acl_recheck()`.

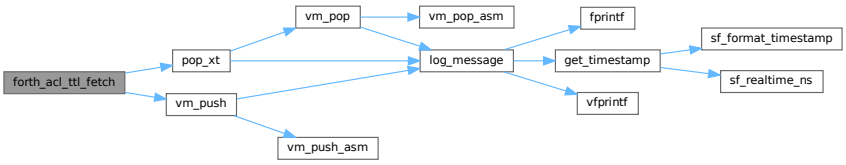
Parameters

vm	Pointer to the VM structure

References DictEntry::acl_ttl, pop_xt(), and vm_push().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.12 forth_acl_ttl_store()

```
static void forth_acl_ttl_store (
            VM * vm )  [static]
```

Implements FORTH word `ACL-TTL!` — store TTL countdown value.

Stack effect: (n xt –) Sets the TTL countdown on the word's ACL entry. Negative values are clamped to 0; values above UINT32_MAX are clamped to UINT32_MAX. No-op if the word is pinned.

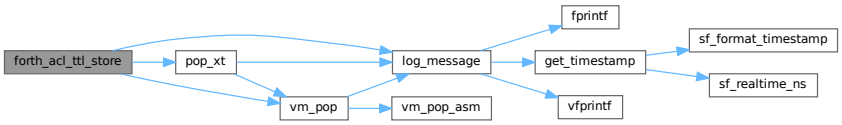
Parameters

vm	Pointer to the VM structure

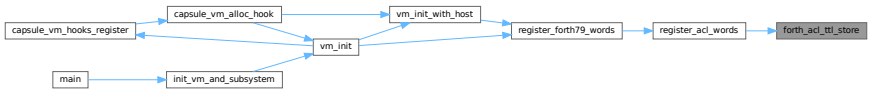
References DictEntry::acl_pinned, DictEntry::acl_ttl, VM::dsp, VM::error, LOG_ERROR, log_message(), pop_xt(), and vm_pop().

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.350.1.13 forth_acl_word_id()

```
static void forth_acl_word_id (
            VM * vm )  [static]
```

Implements FORTH word `ACL-WORD-ID` — fetch stable word identifier.

Stack effect: (xt – n) Pushes the stable `word_id` for the entry identified by `xt`. Unlike dictionary positions, `word_id` is assigned at registration and never changes, making it safe to use as a persistent table index in `ACL.4th`.

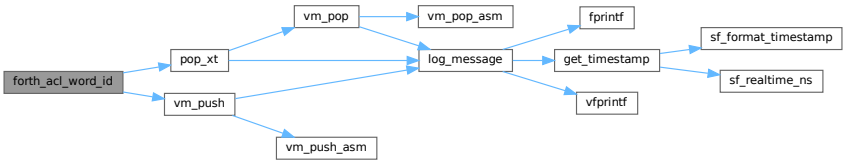
Parameters

vm	Pointer to the VM structure

References pop_xt(), vm_push(), and DictEntry::word_id.

Referenced by register_acl_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.350.1.14 pop_xt()

```
static DictEntry * pop_xt (
            VM * vm )  [static]
```

Pop an execution token (XT) from the data stack and return the corresponding DictEntry pointer.

Sets `vm->error` and returns NULL on stack underflow or if the XT is zero. All ACL word implementations call this before accessing DictEntry fields.

Parameters

vm	Pointer to the VM structure

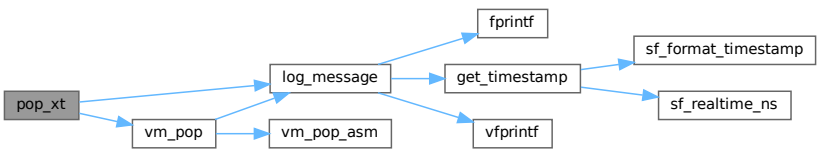
Returns

Pointer to the DictEntry, or NULL on error

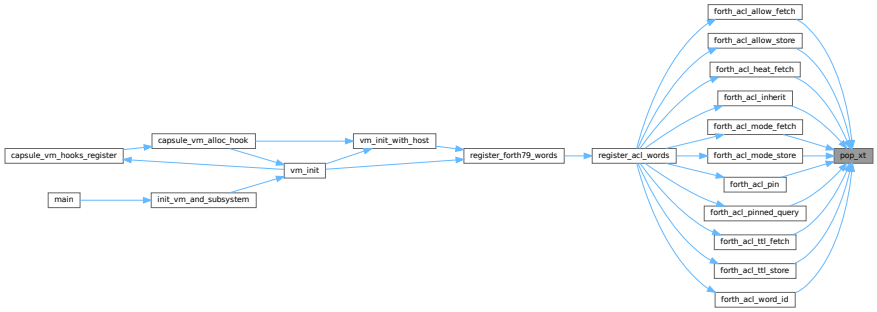
References VM::dsp, entry, VM::error, LOG_ERROR, log_message(), and vm_pop().

Referenced by forth_acl_allow_fetch(), forth_acl_allow_store(), forth_acl_heat_fetch(), forth_acl_inherit(), forth_acl_mode_fetch(), forth_acl_mode_store(), forth_acl_pin(), forth_acl_pinned_query(), forth_acl_ttl_fetch(), forth_acl_ttl_store(), and forth_acl_word_id().

Here is the call graph for this function:



Here is the caller graph for this function:



7.350.1.15 register_acl_words()

```
void register_acl_words (
            VM * vm )
```

Register all ACL C-primitive words with the VM dictionary.

Registers: field accessors (ACL-MODE@ / ACL-MODE! / ACL-PINNED? / ACL-TTL@ / ACL-TTL! / ACL-↩ ALLOW@ / ACL-ALLOW! / ACL-HEAT@ / ACL-WORD-ID), and mutation words (ACL-PIN / ACL-INHERIT / ACL-INIT-PRIMITIVES). Called from vm_bootstrap.c during VM initialisation.

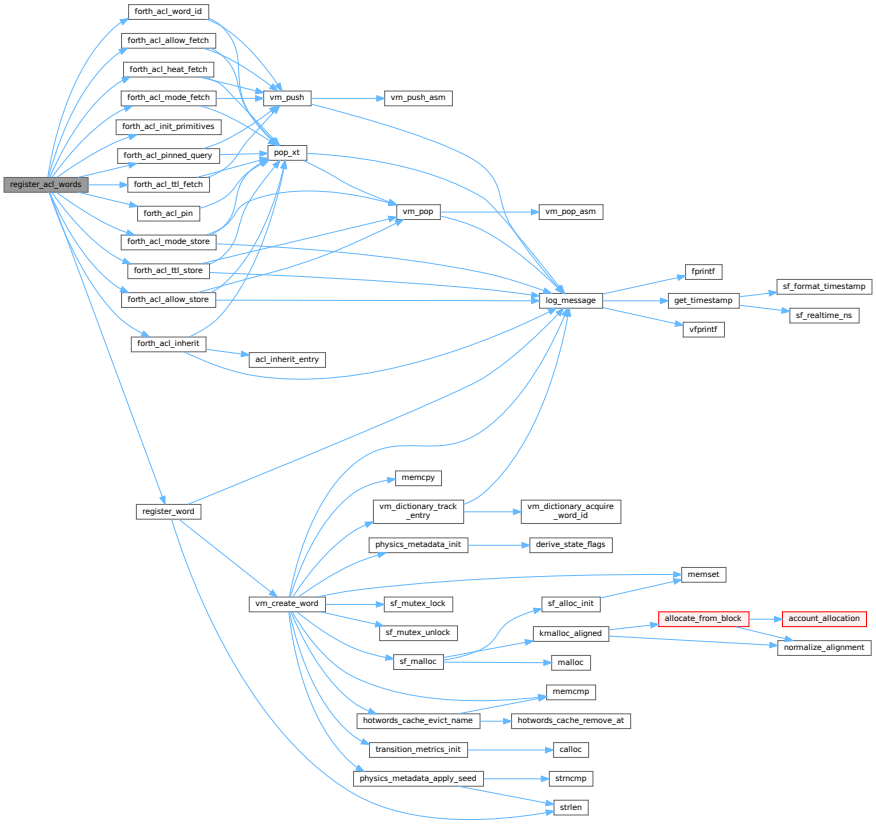
Parameters

vm	Pointer to the VM structure

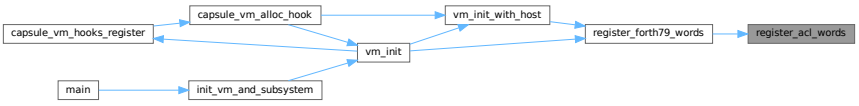
References forth_acl_allow_fetch(), forth_acl_allow_store(), forth_acl_heat_fetch(), forth_acl_inherit(), forth_acl_init_primitives(), forth_acl_mode_fetch(), forth_acl_mode_store(), forth_acl_pin(), forth_acl_pinned_query(), forth_acl_ttl_fetch(), forth_acl_ttl_store(), forth_acl_word_id(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



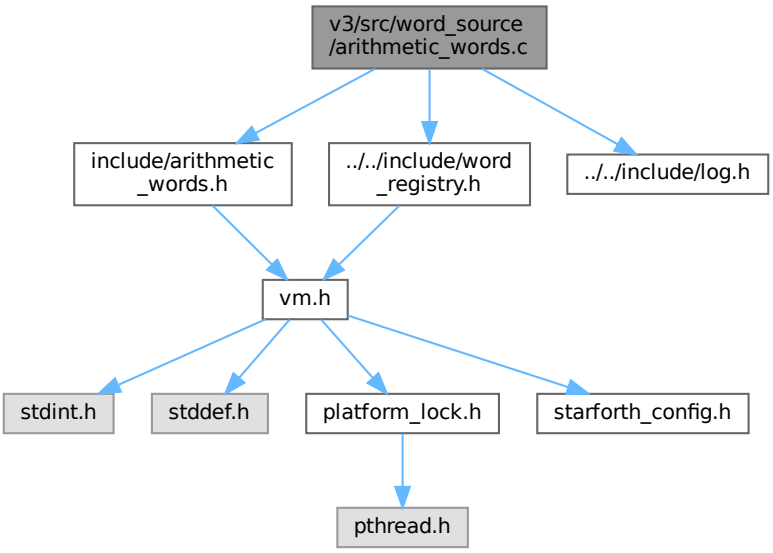
Here is the caller graph for this function:



7.351 v3/src/word_source/arithmetic_words.c File Reference

```
#include "include/arithmetic_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
```

Include dependency graph for arithmetic_words.c:



Functions

- static void arithmetic_word_star_slash (VM *vm)
- static void arithmetic_word_star_slash_mod (VM *vm)
- void arithmetic_word_plus (VM *vm)

  Implements FORTH word "+" that adds two numbers.
- static void arithmetic_word_minus (VM *vm)

  Implements FORTH word "-" that subtracts two numbers.
- static void arithmetic_word_multiply (VM *vm)

  Implements FORTH word "" that multiplies two numbers.*
- static void word_divide (VM *vm)
- static void mixed_math_word_mod (VM *vm)
- static void word_divmod (VM *vm)
- static void arithmetic_word_one_plus (VM *vm)
- static void arithmetic_word_one_minus (VM *vm)
- static void arithmetic_word_two_plus (VM *vm)
- static void arithmetic_word_two_minus (VM *vm)
- static void arithmetic_word_two_multiply (VM *vm)
- static void arithmetic_word_two_divide (VM *vm)
- static void arithmetic_word_abs (VM *vm)
- static void arithmetic_word_negate (VM *vm)
- static void arithmetic_word_min (VM *vm)
- static void arithmetic_word_max (VM *vm)
- void register_arithmetic_words (VM *vm)

  Registers all FORTH-79 arithmetic words with the virtual machine.

7.351.1 Function Documentation

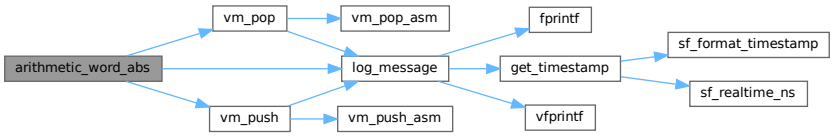
7.351.1.1 arithmetic_word_abs()

```
static void arithmetic_word_abs (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

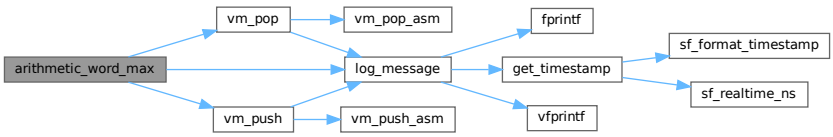
7.351.1.2 arithmetic_word_max()

```
static void arithmetic_word_max (
            VM * vm )  [static]
```

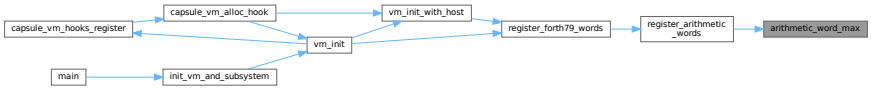
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:



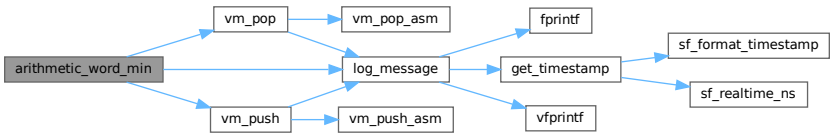
7.351.1.3 arithmetic_word_min()

```
static void arithmetic_word_min (
            VM * vm )  [static]
```

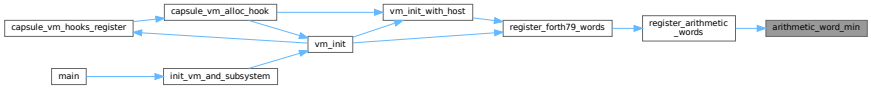
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.351.1.4 arithmetic_word_minus()

```
static void arithmetic_word_minus (
            VM * vm )  [static]
```

Implements FORTH word "-" that subtracts two numbers.

Stack effect: (n1 n2 – n3) Subtracts n2 from n1, pushing their difference n3 onto the stack.

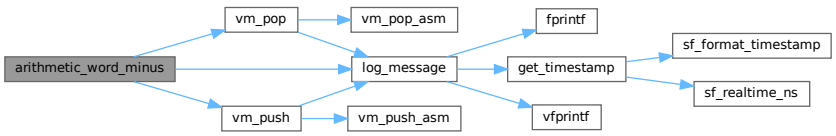
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.351.1.5 arithmetic_word_multiply()

```
static void arithmetic_word_multiply (
            VM * vm )  [static]
```

Implements FORTH word "∗" that multiplies two numbers.

Stack effect: (n1 n2 – n3) Multiplies n1 by n2, pushing their product n3 onto the stack.

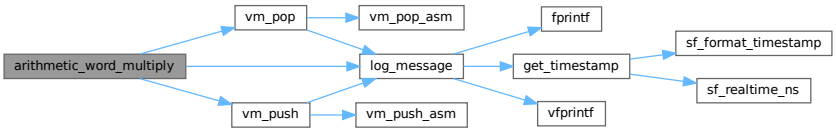
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

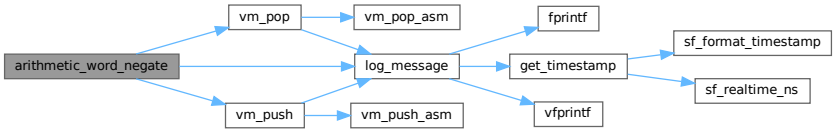
7.351.1.6 arithmetic_word_negate()

```
static void arithmetic_word_negate (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

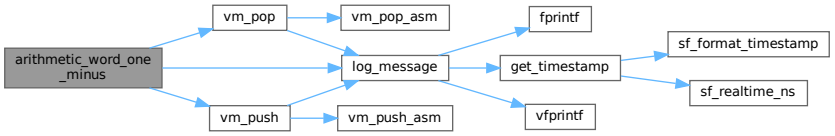
7.351.1.7 arithmetic_word_one_minus()

```
static void arithmetic_word_one_minus (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

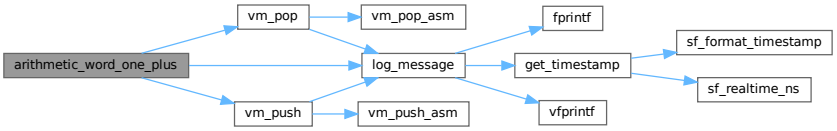
7.351.1.8 arithmetic_word_one_plus()

```
static void arithmetic_word_one_plus (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.351.1.9 arithmetic_word_plus()

```
void arithmetic_word_plus (
            VM * vm )
```

Implements FORTH word "+" that adds two numbers.

Stack effect: (n1 n2 – n3) Adds n1 and n2, pushing their sum n3 onto the stack.

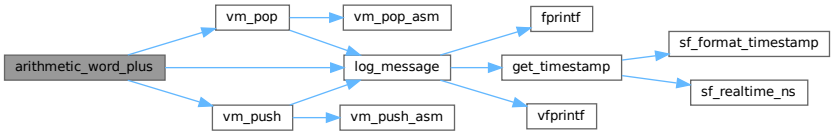
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

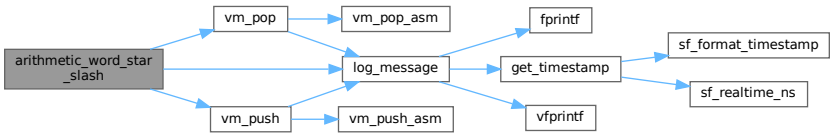
7.351.1.10 arithmetic_word_star_slash()

```
static void arithmetic_word_star_slash (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

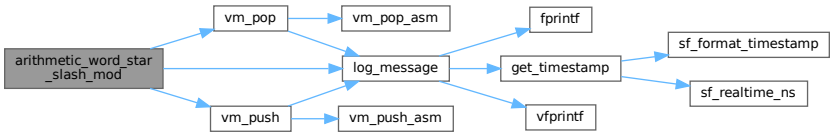
7.351.1.11 arithmetic_word_star_slash_mod()

```
static void arithmetic_word_star_slash_mod (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

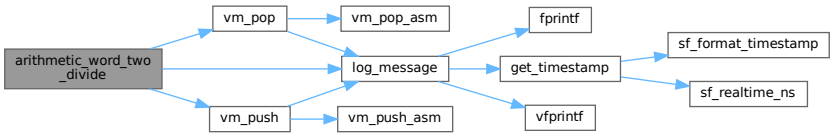
7.351.1.12 arithmetic_word_two_divide()

```
static void arithmetic_word_two_divide (
            VM * vm )  [static]
```

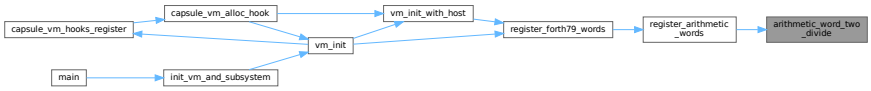
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:



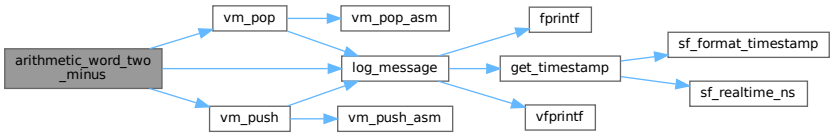
7.351.1.13 arithmetic_word_two_minus()

```
static void arithmetic_word_two_minus (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

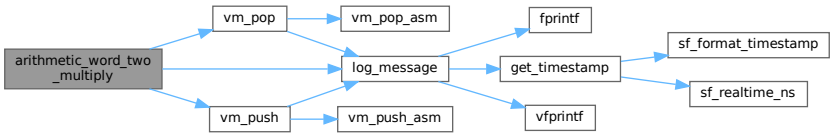
7.351.1.14 arithmetic_word_two_multiply()

```
static void arithmetic_word_two_multiply (
            VM * vm )  [static]
```

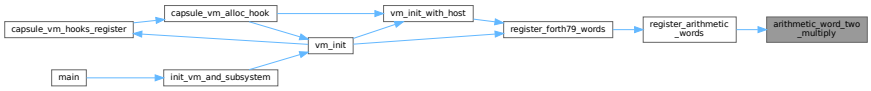
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:



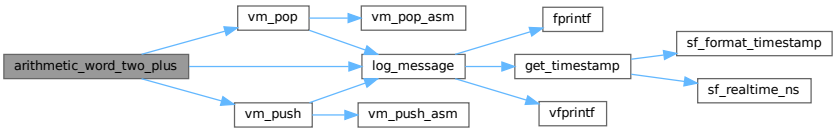
7.351.1.15 arithmetic_word_two_plus()

```
static void arithmetic_word_two_plus (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

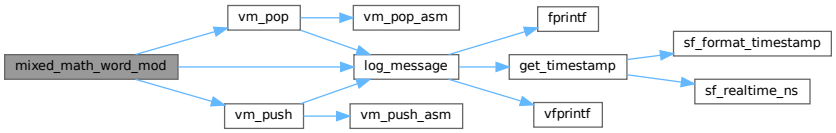
7.351.1.16 mixed_math_word_mod()

```
static void mixed_math_word_mod (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.351.1.17 register_arithmetic_words()

```
void register_arithmetic_words (
            VM * vm )
```

Registers all FORTH-79 arithmetic words with the virtual machine.

Registers the following words:

- + (add)
- - (subtract)
- * (multiply)
- / (divide)
- MOD (modulo)
- /MOD (divide with remainder)
- NEGATE
- ABS (absolute)
- MIN
- MAX
- 1+ (increment)
- 1- (decrement)
- 2* (multiply by 2)
- 2/ (divide by 2)
- STAR-SLASH (multiply then divide)

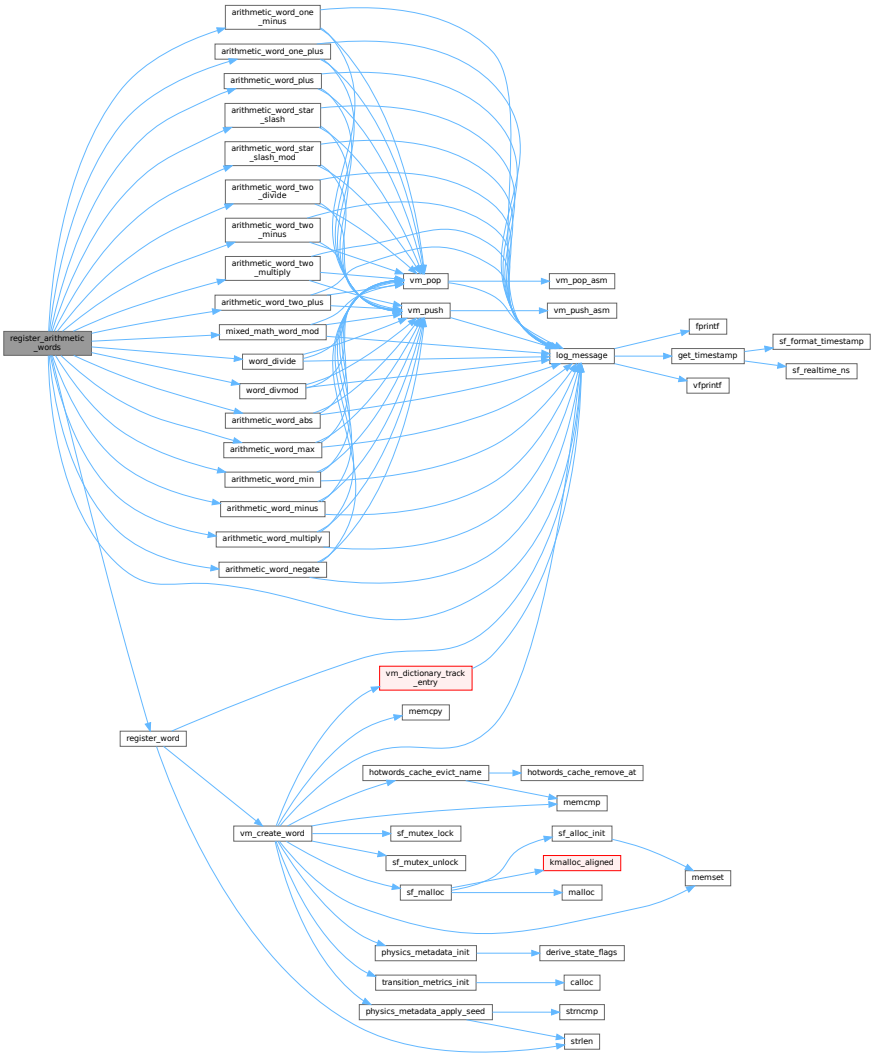
Parameters

vm	Pointer to the Forth virtual machine instance

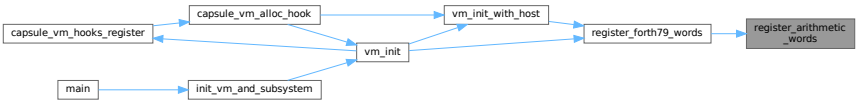
References arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), LOG_INFO, log_message(), mixed_math_word_mod(), register_word(), word_divide(), and word_divmod().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



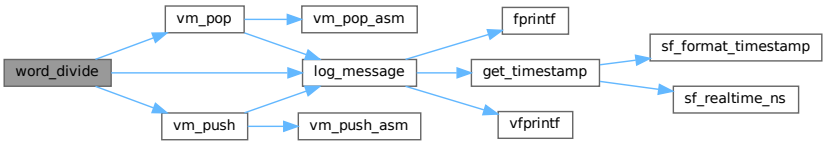
7.351.1.18 word_divide()

```
static void word_divide (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

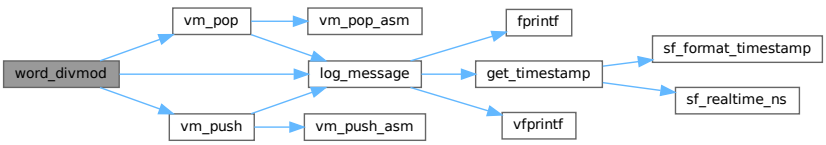
7.351.1.19 word_divmod()

```
static void word_divmod (
            VM * vm )  [static]
```

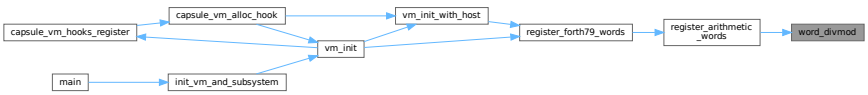
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_arithmetic_words().

Here is the call graph for this function:



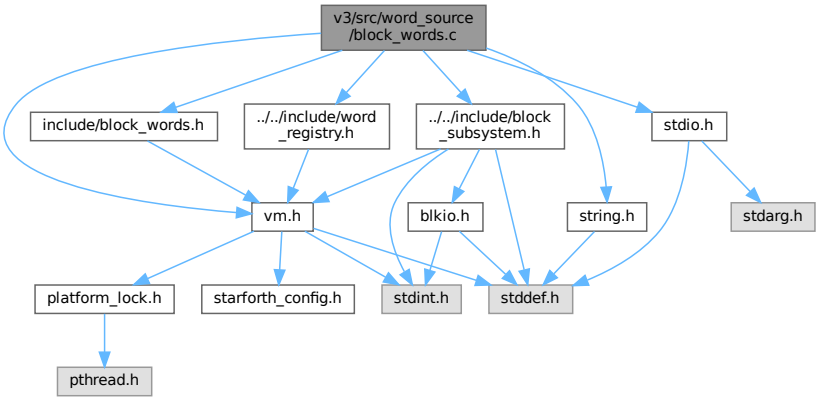
Here is the caller graph for this function:



7.352 v3/src/word_source/block_words.c File Reference

```
#include "include/block_words.h"
#include "../../include/word_registry.h"
#include "../../include/vm.h"
#include "../../include/block_subsystem.h"
#include <string.h>
#include <stdio.h>
```

Include dependency graph for block_words.c:



Functions

- static void set_scr (VM *vm, cell_t blk)
- void init_block_system (VM *vm)

  Initializes the block system.
- unsigned char * get_block_buffer (VM *vm, int block_num)

  Gets a buffer for the specified block.
- unsigned char * get_empty_buffer (VM *vm, int block_num)

  Gets an empty buffer for the specified block.
- void mark_buffer_dirty (VM *vm)

  Marks the current buffer as modified.
- void save_all_buffers (VM *vm)
- void empty_all_buffers (VM *vm)

  Empties all user block buffers.
- void blk_vm_check_epoch (VM *vm)

  Discards vm->blk_vm_lbn[]/cbuf[]/dirty[] if the device chain has changed since last validated (see this function's own doc comment in block_words.c for why).
- int blk_vm_slot_for_addr (vaddr_t addr)

  Does `addr` fall inside the block I/O window? Pure address arithmetic against the window's fixed compile-time base/size – no VM state needed.
- int blk_vm_is_cross_device_copy (VM *vm, vaddr_t addr1, vaddr_t addr2)

  True if addr1/addr2 are both block-window addresses currently loaded from two different physical devices (Phase 8 v3, 2026-09-23) – the check MOVE/CMOVE/CMOVE> call before copying to refuse a same-VM, cross-device raw block copy.
- static int blk_vm_find (VM *vm, uint32_t lbn)
- static int blk_vm_evict (VM *vm)

- static vaddr_t blk_vm_load (VM *vm, uint32_t lbn, int writable)
- static vaddr_t blk_vm_assign (VM *vm, uint32_t lbn)
- void blk_vm_flush_all (VM *vm)

  Sync every dirty VM block-window slot to the C-layer block cache, then flush the whole block subsystem to physical media – the complete SAVE-BUFFERS implementation (unlike save_all_buffers() above, which only calls blk_flush() and so misses any BUFFER-modified-but-not-yet-UPDATE'd content still sitting only in vm->memory's window).
- void block_word_block (VM *vm)
- void block_word_buffer (VM *vm)
- void block_word_update (VM *vm)
- void block_word_confirm_format (VM *vm)
- void block_word_relocate (VM *vm)
- void block_word_save_buffers (VM *vm)
- void block_word_empty_buffers (VM *vm)
- void block_word_flush (VM *vm)
- void block_word_load (VM *vm)
- void block_word_list (VM *vm)
- void block_word_thru (VM *vm)
- void block_word_scr (VM *vm)
- void block_word_next_block (VM *vm)
- static void block_word_acl_allow_fetch (VM *vm)
- static void block_word_acl_allow_store (VM *vm)
- static void block_word_acl_ttl_fetch (VM *vm)
- static void block_word_acl_ttl_store (VM *vm)
- static void block_word_owner_fetch (VM *vm)
- static void block_word_blk_attach (VM *vm)
- void register_block_words (VM *vm)

  Registers all block and buffer words with the virtual machine.

7.352.1 Function Documentation

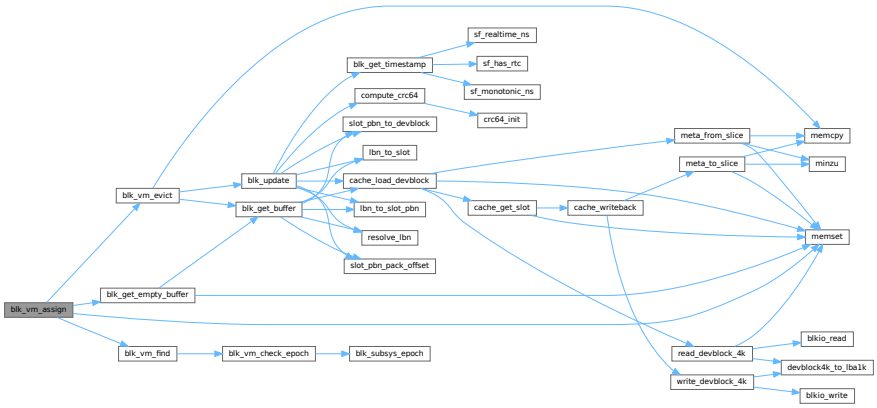
7.352.1.1 blk_vm_assign()

```
static vaddr_t blk_vm_assign (
            VM * vm,
            uint32_t lbn )  [static]
```

References base, blk_get_empty_buffer(), VM::blk_vm_cbuf, VM::blk_vm_dirty, blk_vm_evict(), blk_vm_find(), VM::blk_vm_lbn, BLK_VM_SLOTS, BLK_VM_WINDOW_BASE, BLOCK_SIZE, VM::memory, and memset().

Referenced by block_word_buffer().

Here is the call graph for this function:



Here is the caller graph for this function:

7.352.1.2 blk_vm_check_epoch()

```
void blk_vm_check_epoch (
            VM * vm )
```

Discards vm->blk_vm_lbn[]/cbuf[]/dirty[] if the device chain has changed since last validated (see this function's own doc comment in block_words.c for why).

Exposed (Phase 8 v3, 2026-09-23) so callers outside this file that read vm->blk_vm_lbn[] directly (MOVE/↩ CMOVE's cross-device block-copy check, memory_words.c/string_words.c) go through the same invalidation contract blk_vm_find() already uses internally, rather than a second, divergent copy of this logic.

Parameters

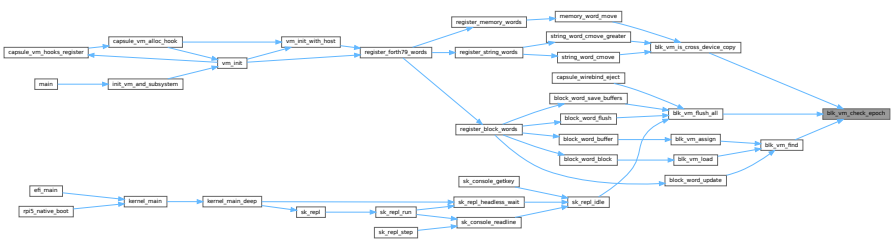
vm	Pointer to the Forth virtual machine instance

References blk_subsys_epoch(), VM::blk_vm_cbuf, VM::blk_vm_dirty, VM::blk_vm_epoch, VM::blk_vm_lbn, BLK_VM_SLOTS, and epoch.

Referenced by blk_vm_find(), blk_vm_flush_all(), and blk_vm_is_cross_device_copy().

Here is the call graph for this function:

Here is the caller graph for this function:



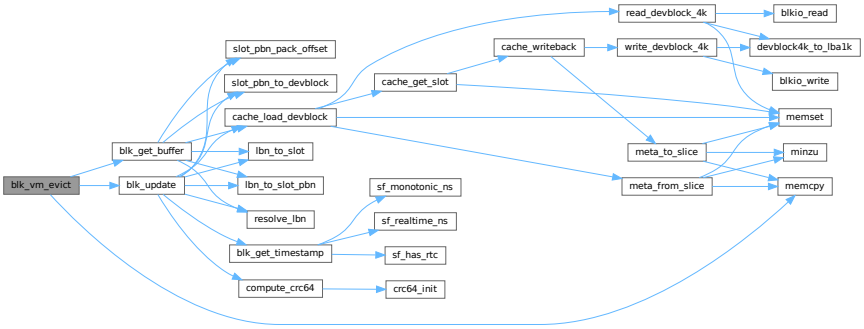
7.352.1.3 blk_vm_evict()

```
static int blk_vm_evict (
            VM * vm )  [static]
```

References base, blk_get_buffer(), blk_update(), VM::blk_vm_cbuf, VM::blk_vm_dirty, VM::blk_vm_lbn, VM::blk_vm_next, BLK_VM_SLOTS, BLK_VM_WINDOW_BASE, BLOCK_SIZE, memcpy(), and VM::memory.

Referenced by blk_vm_assign(), and blk_vm_load().

Here is the call graph for this function:



Here is the caller graph for this function:

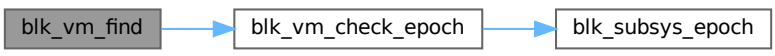
7.352.1.4 blk_vm_find()

```
static int blk_vm_find (
            VM * vm,
            uint32_t lbn )  [static]
```

References VM::blk_vm_cbuf, blk_vm_check_epoch(), VM::blk_vm_lbn, and BLK_VM_SLOTS.

Referenced by blk_vm_assign(), blk_vm_load(), and block_word_update().

Here is the call graph for this function:



Here is the caller graph for this function:

7.352.1.5 blk_vm_flush_all()

```
void blk_vm_flush_all (
            VM * vm )
```

Sync every dirty VM block-window slot to the C-layer block cache, then flush the whole block subsystem to physical media – the complete SAVE-BUFFERS implementation (unlike save_all_buffers() above, which only calls blk_flush() and so misses any BUFFER-modified-but-not-yet-UPDATE'd content still sitting only in vm->memory's window).

Cheap to call when nothing is dirty – every check inside is a small fixed-size scan, no I/O happens unless something actually needs writing. Safe to call from outside word dispatch (e.g. a periodic idle-tick auto-flush); vm must be non-NULL.

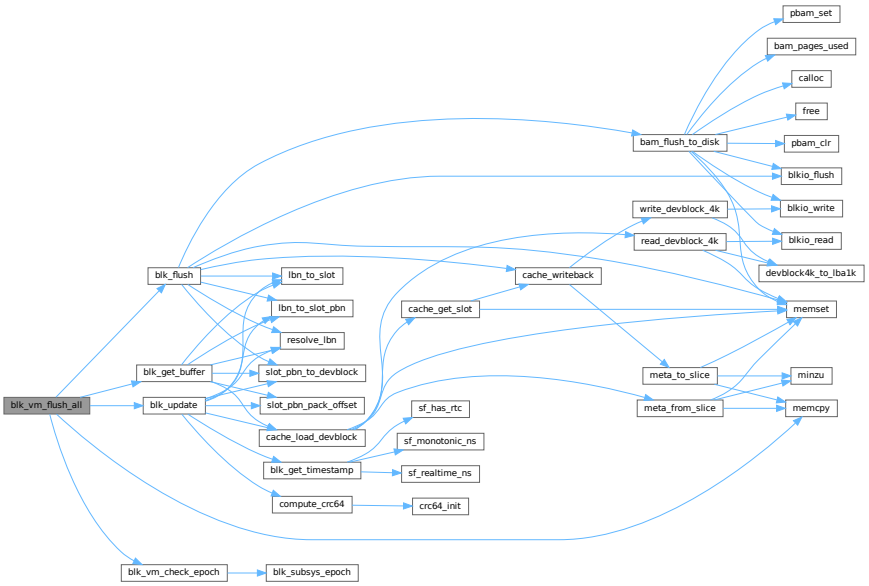
Parameters

vm	Pointer to the Forth virtual machine instance

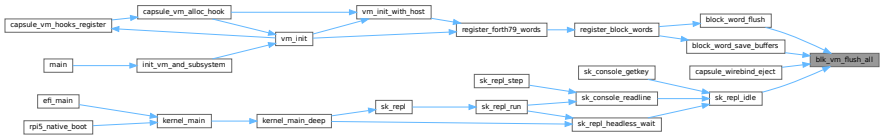
References base, blk_flush(), blk_get_buffer(), blk_update(), VM::blk_vm_cbuf, blk_vm_check_epoch(), VM::blk_vm_dirty, VM::blk_vm_lbn, BLK_VM_SLOTS, BLK_VM_WINDOW_BASE, BLOCK_SIZE, memcpy(), and VM::memory.

Referenced by block_word_flush(), block_word_save_buffers(), capsule_wirebind_eject(), and sk_repl_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.352.1.6 blk_vm_is_cross_device_copy()

```
int blk_vm_is_cross_device_copy (
            VM * vm,
            vaddr_t addr1,
            vaddr_t addr2 )
```

True if addr1/addr2 are both block-window addresses currently loaded from two different physical devices (Phase 8 v3, 2026-09-23) – the check MOVE/CMOVE/CMOVE> call before copying to refuse a same-VM, cross-device raw block copy.

See this function's own doc comment in block_words.c for exactly what is and isn't flagged.

Parameters

vm	VM performing the copy
addr1	first VM offset (source or destination, order doesn't matter)
addr2	second VM offset

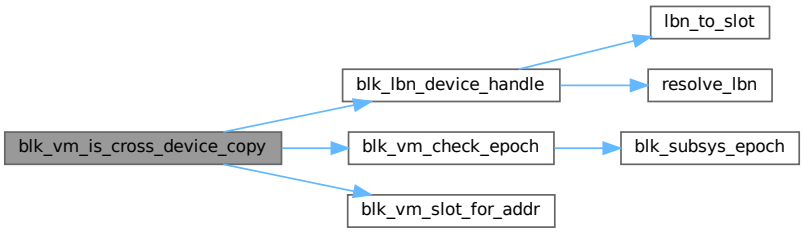
Returns

1 if this copy should be refused, 0 otherwise

References blk_lbn_device_handle(), blk_vm_check_epoch(), VM::blk_vm_lbn, and blk_vm_slot_for_addr().

Referenced by memory_word_move(), string_word_cmove(), and string_word_cmove_greater().

Here is the call graph for this function:



Here is the caller graph for this function:

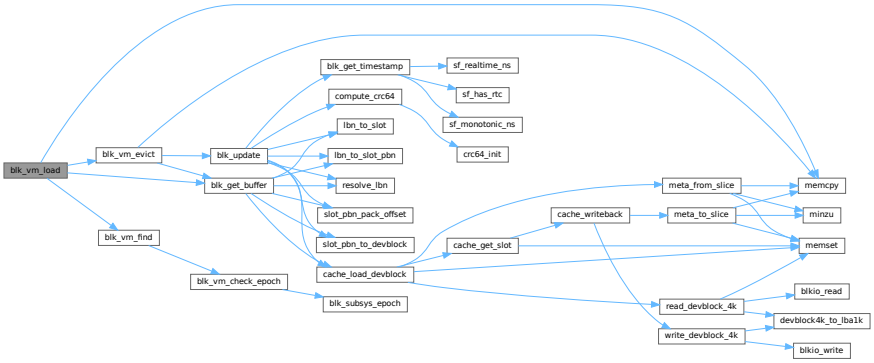
7.352.1.7 blk_vm_load()

```
static vaddr_t blk_vm_load (
            VM * vm,
            uint32_t lbn,
            int writable )  [static]
```

References base, blk_get_buffer(), VM::blk_vm_cbuf, VM::blk_vm_dirty, blk_vm_evict(), blk_vm_find(), VM::blk_vm_lbn, BLK_VM_SLOTS, BLK_VM_WINDOW_BASE, BLOCK_SIZE, memcpy(), and VM::memory.

Referenced by block_word_block().

Here is the call graph for this function:



Here is the caller graph for this function:

7.352.1.8 blk_vm_slot_for_addr()

```
int blk_vm_slot_for_addr (
            vaddr_t addr )
```

Does `addr` fall inside the block I/O window? Pure address arithmetic against the window's fixed compile-time base/size – no VM state needed.

Exposed (Phase 8 v3, 2026-09-23) for MOVE/CMOVE/CMOVE>'s cross-device block-copy check.

Parameters

addr	VM offset to test

Returns

slot index (0..BLK_VM_SLOTS-1) if inside the window, -1 otherwise

References BLK_VM_WINDOW_BASE, BLK_VM_WINDOW_SIZE, and BLOCK_SIZE.

Referenced by blk_vm_is_cross_device_copy().

Here is the caller graph for this function:

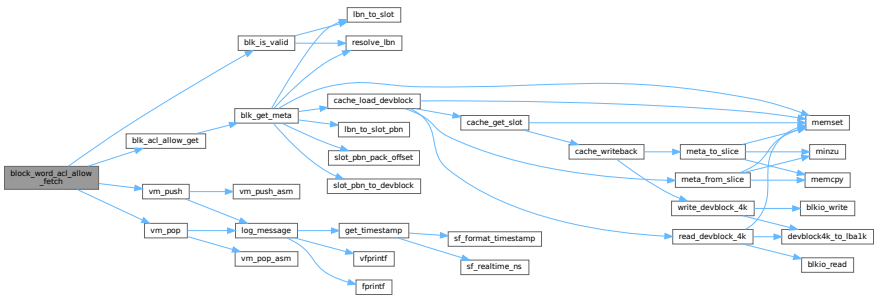
7.352.1.9 block_word_acl_allow_fetch()

```
static void block_word_acl_allow_fetch (
            VM * vm )  [static]
```

References blk_acl_allow_get(), blk_is_valid(), BLK_OK, VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

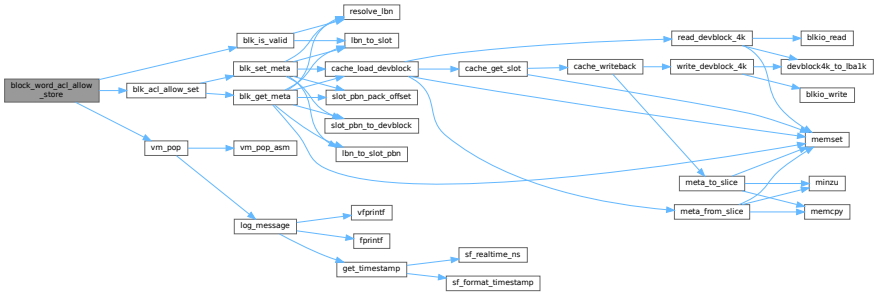
7.352.1.10 block_word_acl_allow_store()

```
static void block_word_acl_allow_store (
            VM * vm )  [static]
```

References blk_acl_allow_set(), blk_is_valid(), BLK_OK, VM::dsp, VM::error, and vm_pop().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

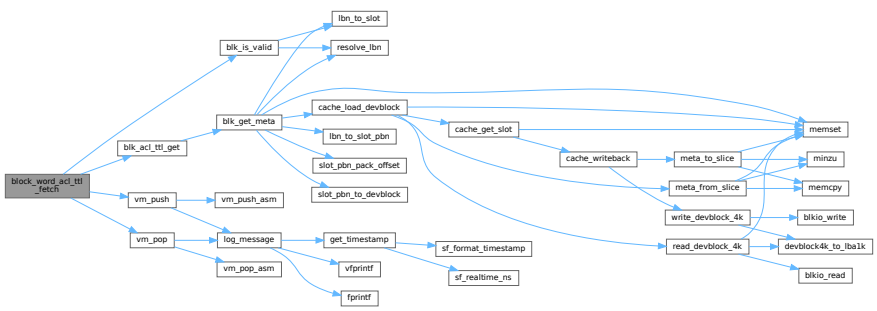
7.352.1.11 block_word_acl_ttl_fetch()

```
static void block_word_acl_ttl_fetch (
            VM * vm )  [static]
```

References blk_acl_ttl_get(), blk_is_valid(), BLK_OK, VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

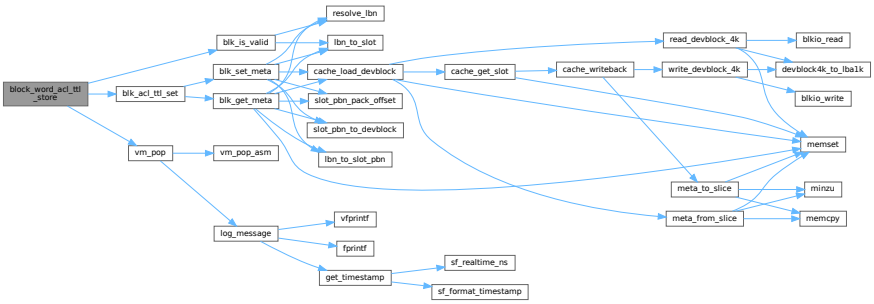
7.352.1.12 block_word_acl_ttl_store()

```
static void block_word_acl_ttl_store (
            VM * vm )  [static]
```

References blk_acl_ttl_set(), blk_is_valid(), BLK_OK, VM::dsp, VM::error, and vm_pop().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

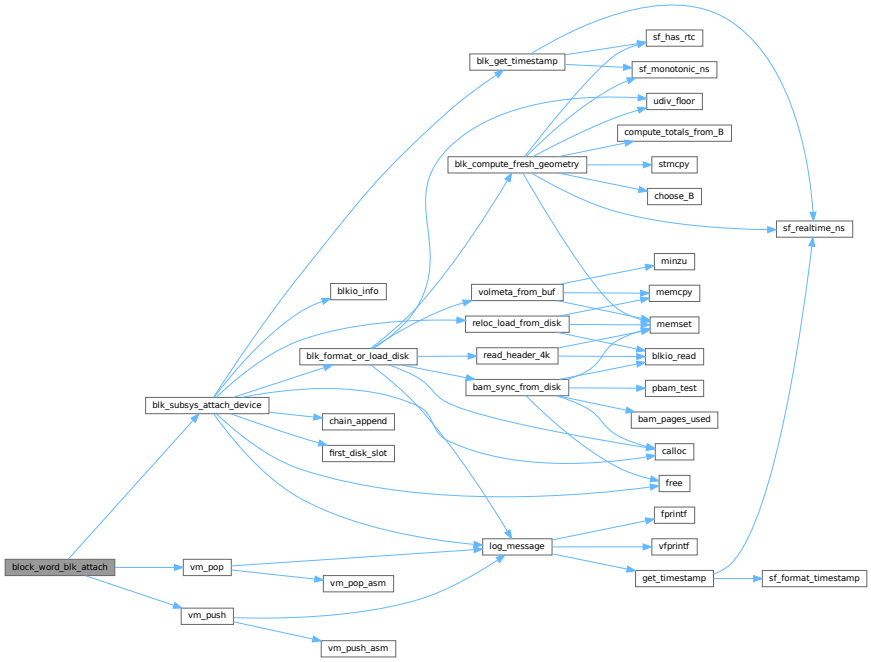
7.352.1.13 block_word_blk_attach()

```
static void block_word_blk_attach (
            VM * vm )  [static]
```

References BLK_OK, blk_subsys_attach_device(), VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

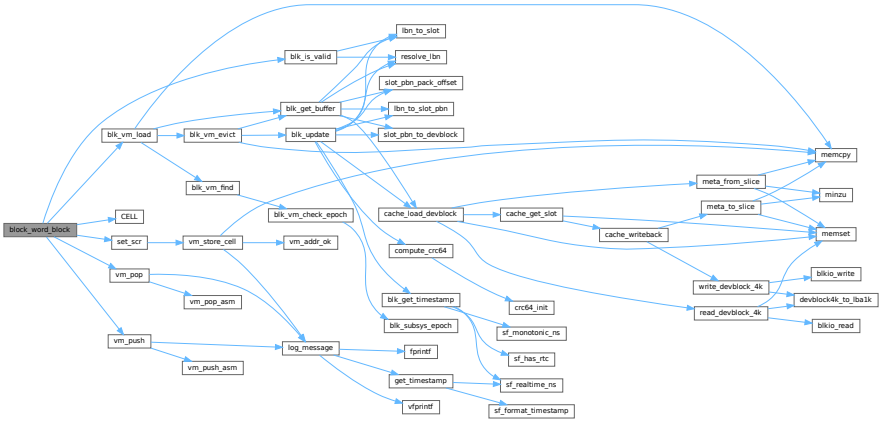
7.352.1.14 block_word_block()

```
void block_word_block (
            VM * vm )
```

References blk_is_valid(), blk_vm_load(), CELL(), VM::dsp, VM::error, VM::heartbeat, set_scr(), HeartbeatState::tick_count, vm_pop(), and vm_push().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

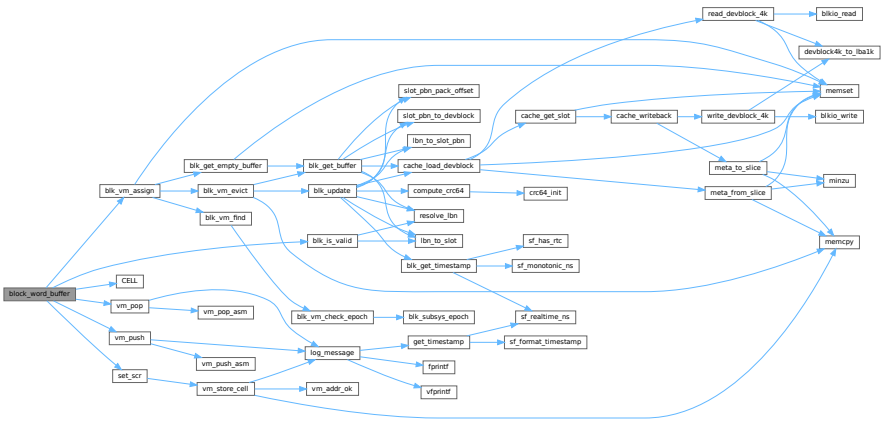
7.352.1.15 block_word_buffer()

```
void block_word_buffer (
            VM * vm )
```

References blk_is_valid(), blk_vm_assign(), CELL(), VM::dsp, VM::error, VM::heartbeat, set_scr(), HeartbeatState::tick_count, vm_pop(), and vm_push().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

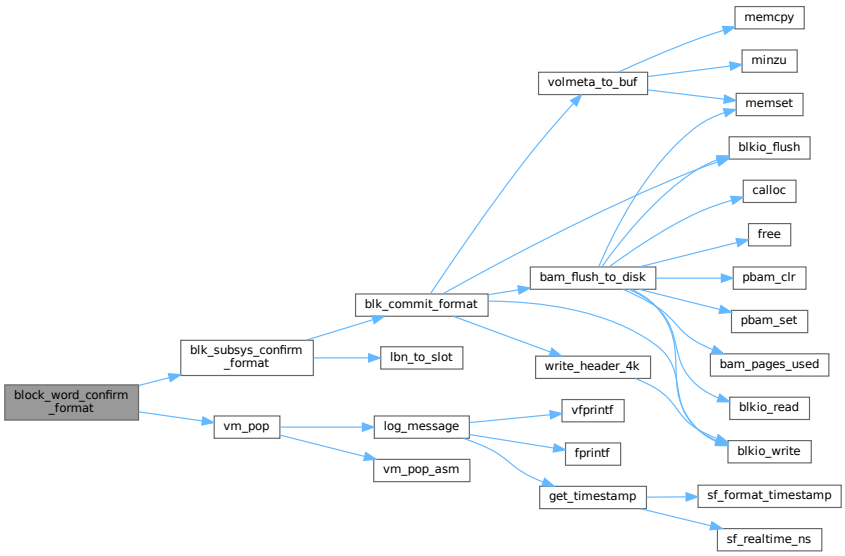
7.352.1.16 block_word_confirm_format()

```
void block_word_confirm_format (
            VM * vm )
```

References BLK_OK, blk_subsys_confirm_format(), VM::dsp, VM::error, and vm_pop().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

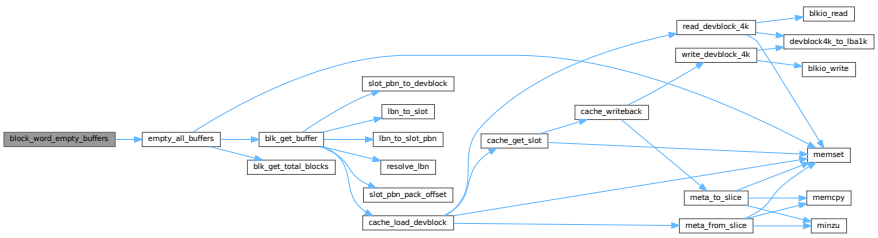
7.352.1.17 block_word_empty_buffers()

```
void block_word_empty_buffers (
            VM * vm )
```

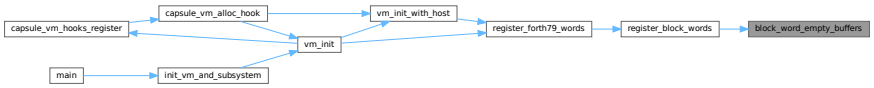
References VM::blk_vm_cbuf, VM::blk_vm_dirty, VM::blk_vm_lbn, BLK_VM_SLOTS, and empty_all_buffers().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:



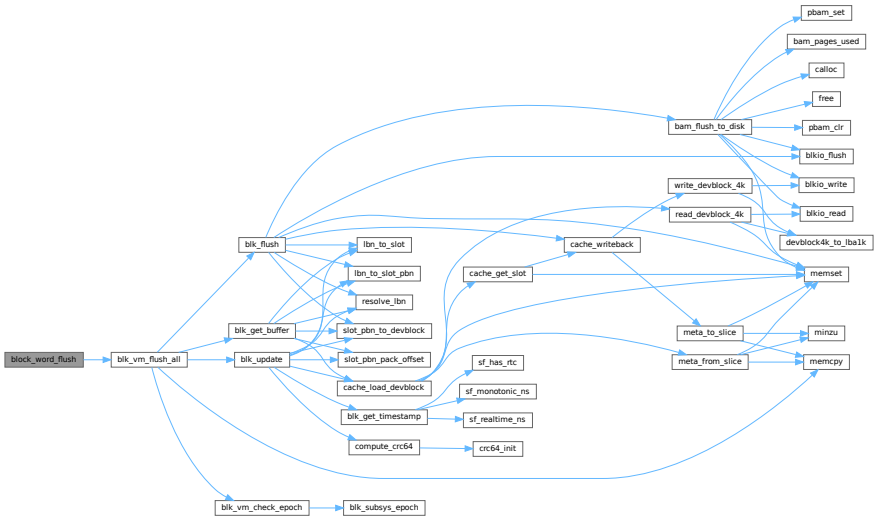
7.352.1.18 block_word_flush()

```
void block_word_flush (
            VM * vm )
```

References blk_vm_flush_all().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

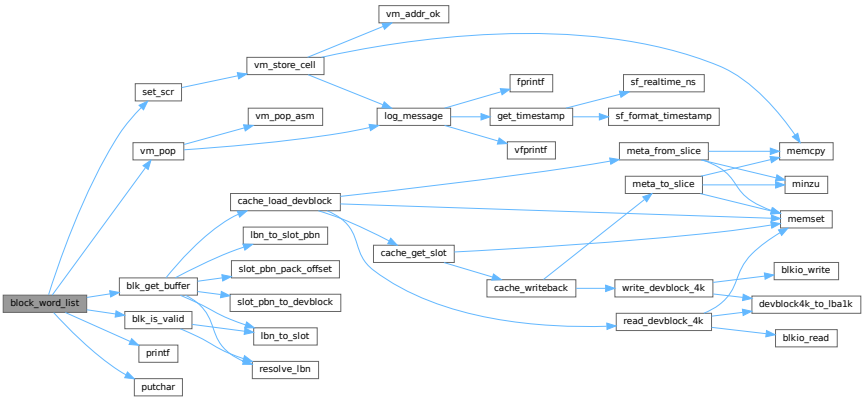
7.352.1.19 block_word_list()

```
void block_word_list (
            VM * vm )
```

References blk_get_buffer(), blk_is_valid(), VM::dsp, VM::error, printf(), putchar(), set_scr(), and vm_pop().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

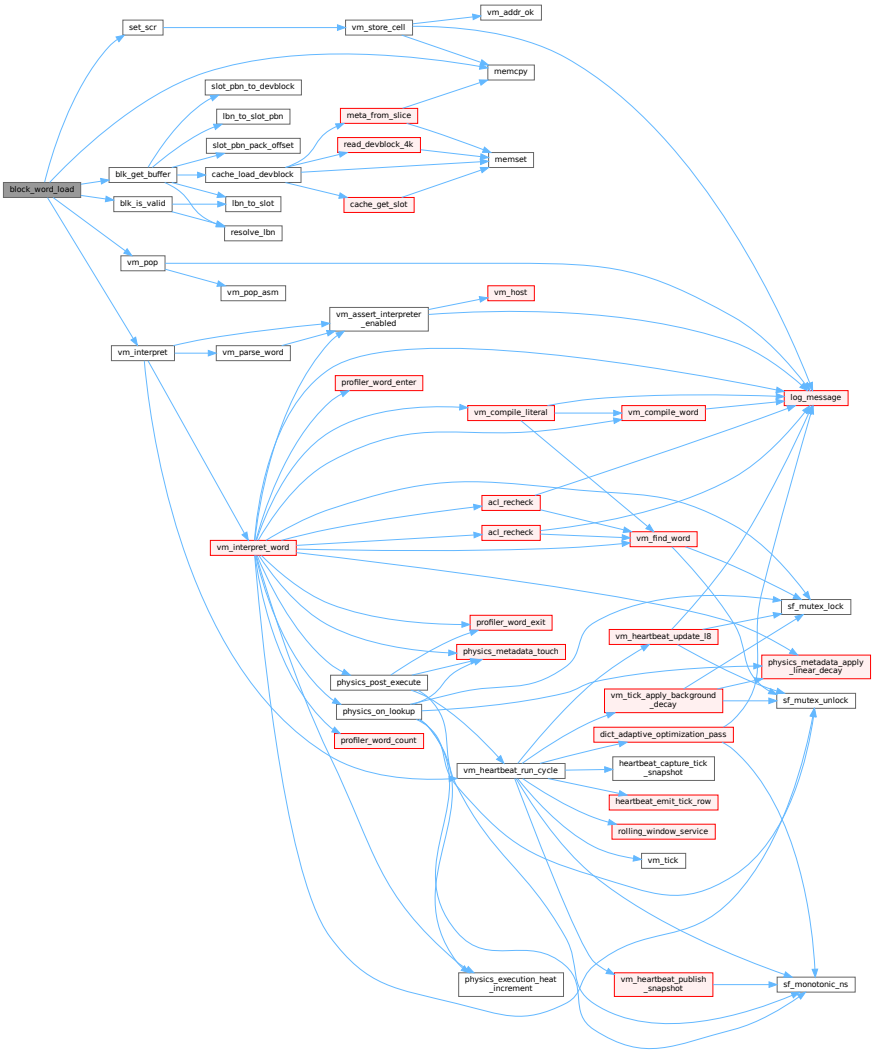
7.352.1.20 block_word_load()

```
void block_word_load (
            VM * vm )
```

References blk_get_buffer(), blk_is_valid(), VM::dsp, VM::error, memcpy(), set_scr(), vm_interpret(), and vm_pop().

Referenced by block_word_thru(), and register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

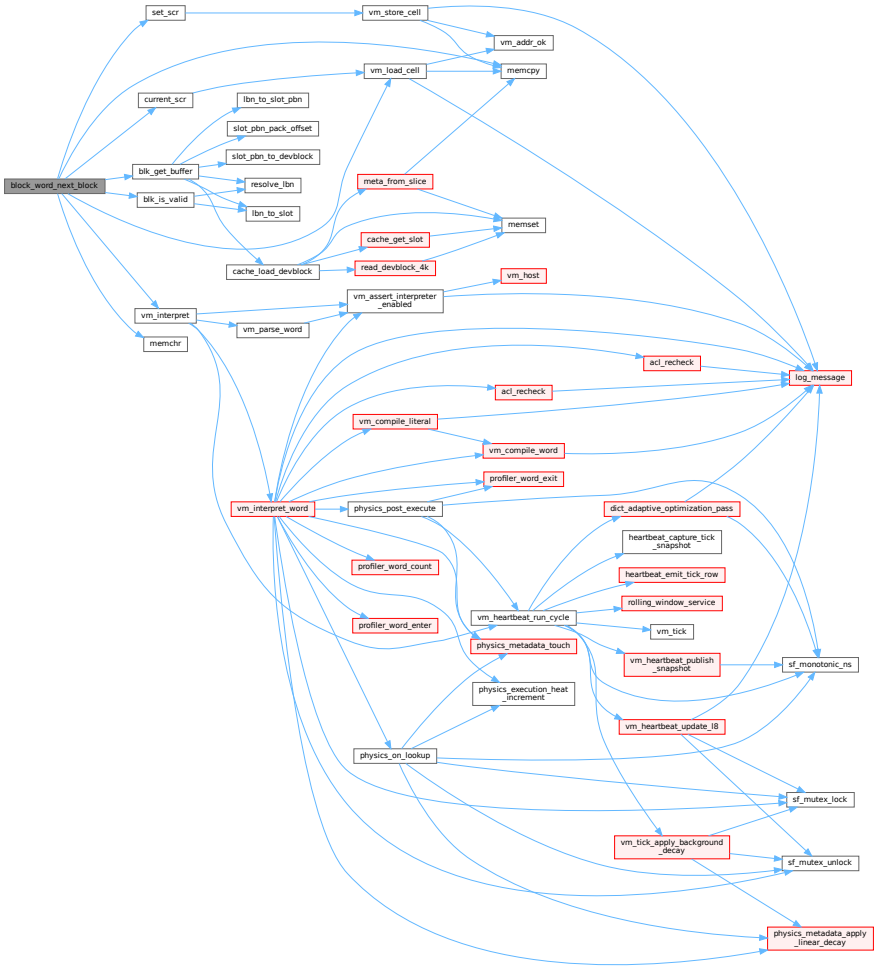
7.352.1.21 block_word_next_block()

```
void block_word_next_block (
            VM * vm )
```

References VM::abort_requested, blk_get_buffer(), blk_is_valid(), current_scr(), VM::error, memchr(), memcpy(), VM::scr_addr, set_scr(), vm_interpret(), and vm_load_cell().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

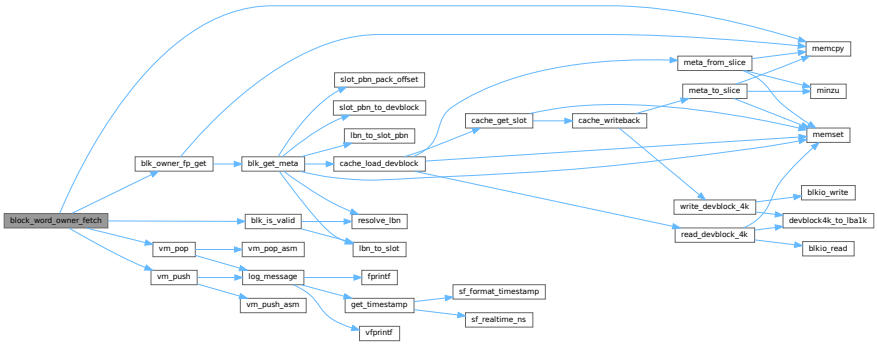
7.352.1.22 block_word_owner_fetch()

```
static void block_word_owner_fetch (
            VM * vm )  [static]
```

References blk_is_valid(), BLK_OK, blk_owner_fp_get(), VM::dsp, VM::error, memcpy(), vm_pop(), and vm_push().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

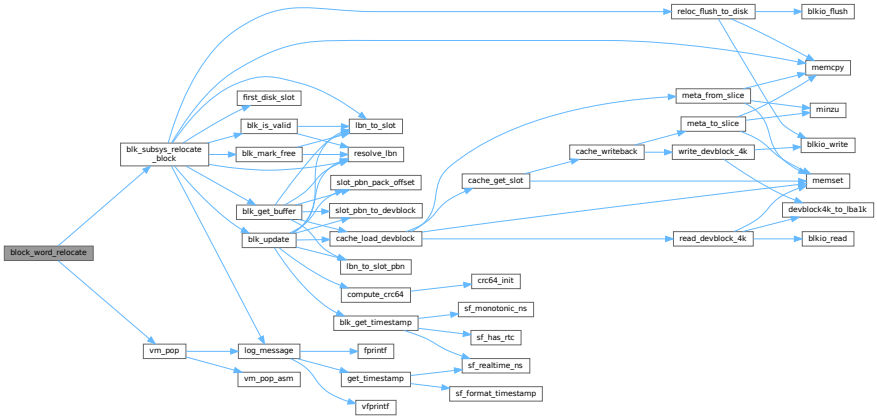
7.352.1.23 block_word_relocate()

```
void block_word_relocate (
            VM * vm )
```

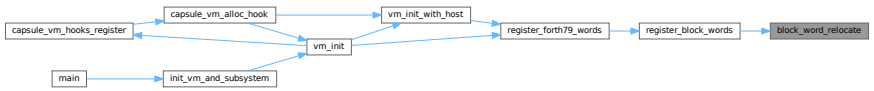
References BLK_OK, blk_subsys_relocate_block(), VM::dsp, VM::error, and vm_pop().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:



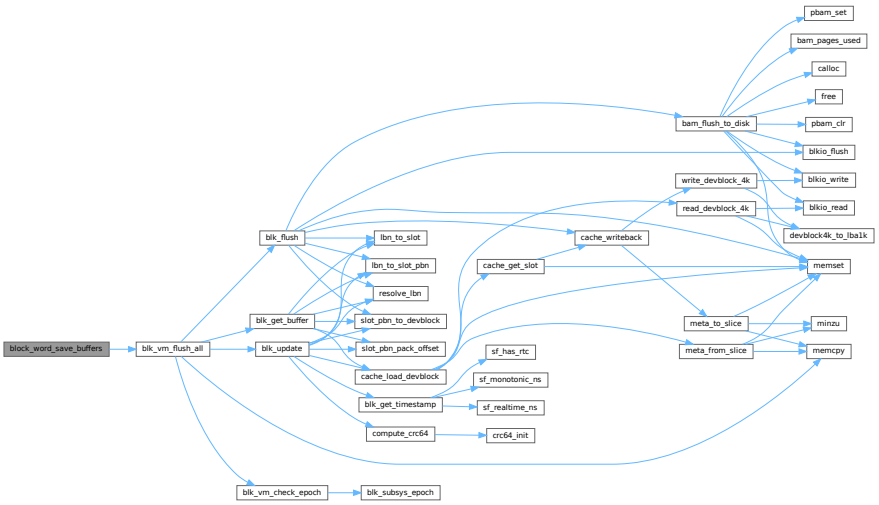
7.352.1.24 block_word_save_buffers()

```
void block_word_save_buffers (
            VM * vm )
```

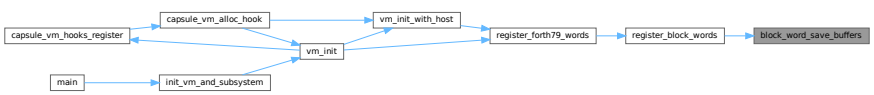
References blk_vm_flush_all().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:



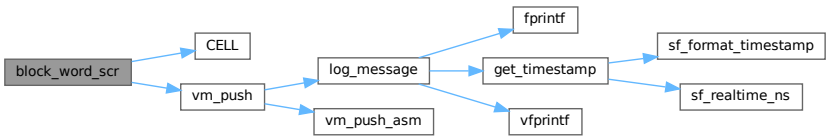
7.352.1.25 block_word_scr()

```
void block_word_scr (
            VM * vm )
```

References CELL(), VM::scr_addr, and vm_push().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

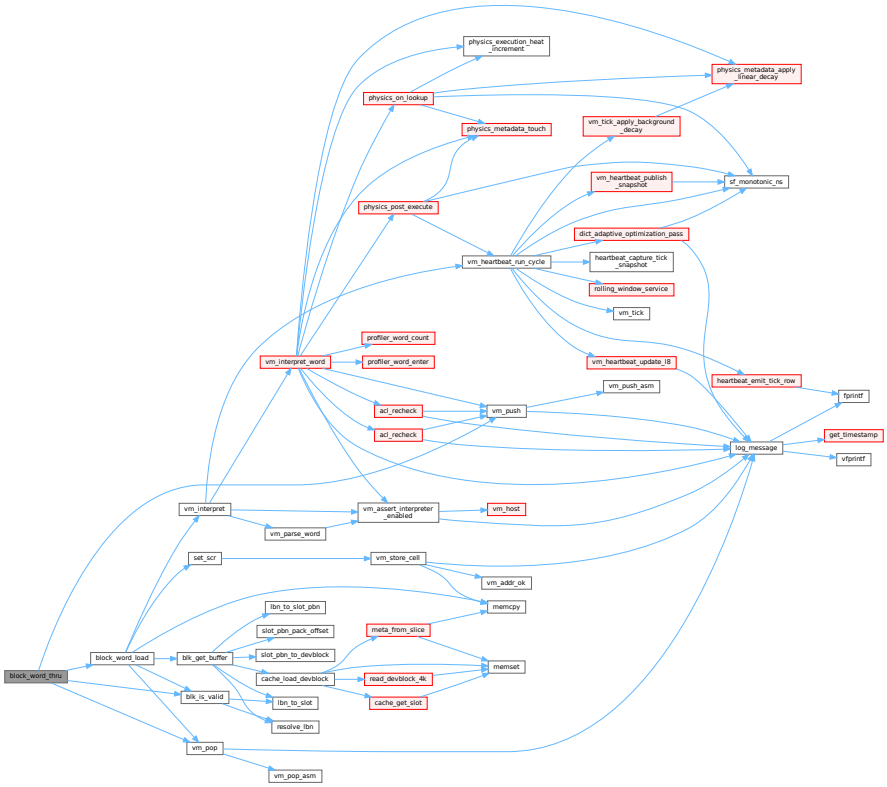
7.352.1.26 block_word_thru()

```
void block_word_thru (
            VM * vm )
```

References VM::abort_requested, blk_is_valid(), block_word_load(), VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

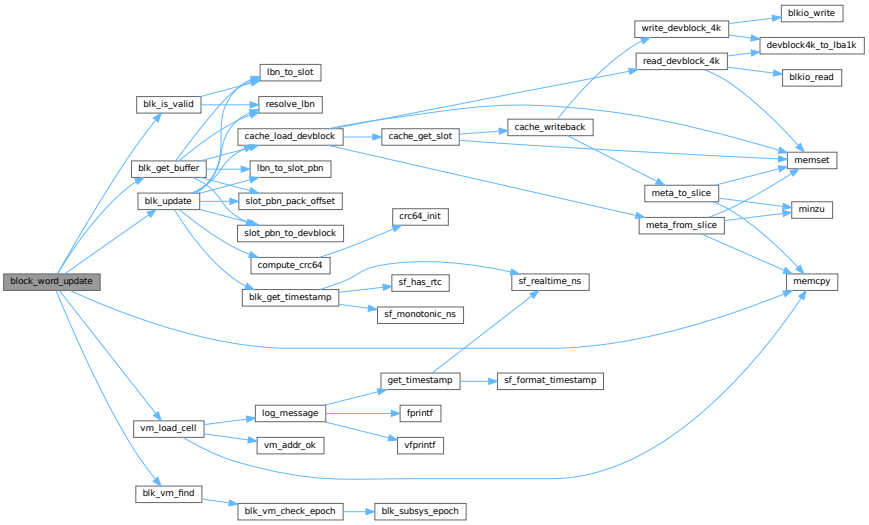
7.352.1.27 block_word_update()

```
void block_word_update (
            VM * vm )
```

References base, blk_get_buffer(), blk_is_valid(), blk_update(), VM::blk_vm_cbuf, VM::blk_vm_dirty, blk_vm_find(), BLK_VM_WINDOW_BASE, BLOCK_SIZE, VM::error, VM::heartbeat, memcpy(), VM::memory, VM::scr_addr, HeartbeatState::tick_count, and vm_load_cell().

Referenced by register_block_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.352.1.28 empty_all_buffers()

```
void empty_all_buffers (
            VM * vm )
```

Empties all user block buffers.

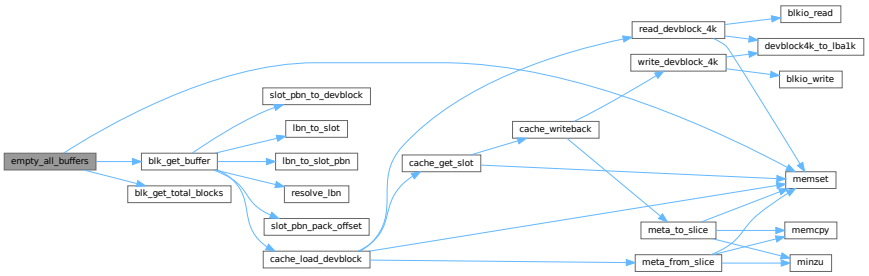
Parameters

vm	Pointer to the VM instance

References blk_get_buffer(), blk_get_total_blocks(), BLOCK_SIZE, memset(), and USER_BLOCKS_START.

Referenced by block_word_empty_buffers().

Here is the call graph for this function:



Here is the caller graph for this function:

7.352.1.29 get_block_buffer()

```
unsigned char * get_block_buffer (
            VM * vm,
            int block_num )
```

Gets a buffer for the specified block.

Parameters

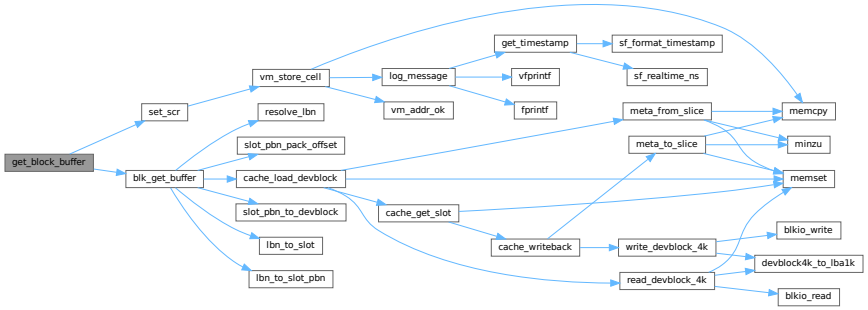
vm	Pointer to the VM instance
block_num	Block number to retrieve

Returns

Pointer to block buffer or NULL if invalid

References blk_get_buffer(), and set_scr().

Here is the call graph for this function:



7.352.1.30 get_empty_buffer()

```
unsigned char * get_empty_buffer (
            VM * vm,
            int block_num )
```

Gets an empty buffer for the specified block.

Parameters

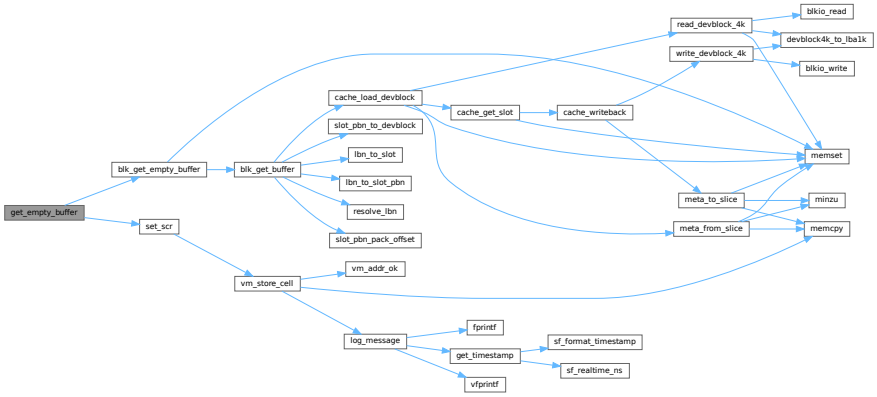
vm	Pointer to the VM instance
block_num	Block number to create

Returns

Pointer to empty block buffer or NULL if invalid

References blk_get_empty_buffer(), and set_scr().

Here is the call graph for this function:



7.352.1.31 init_block_system()

```
void init_block_system (
            VM * vm )
```

Initializes the block system.

Parameters

vm	Pointer to the Forth virtual machine instance

7.352.1.32 mark_buffer_dirty()

```
void mark_buffer_dirty (
            VM * vm )
```

Marks the current buffer as modified.

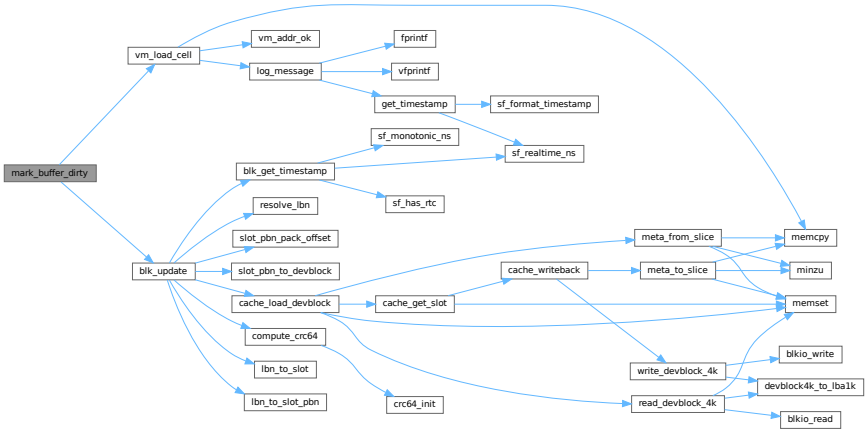
Parameters

vm	Pointer to the Forth virtual machine instance

References blk_update(), VM::scr_addr, and vm_load_cell().

Referenced by editor_word_edit(), and editor_word_s().

Here is the call graph for this function:



Here is the caller graph for this function:

7.352.1.33 register_block_words()

```
void register_block_words (
            VM * vm )
```

Registers all block and buffer words with the virtual machine.

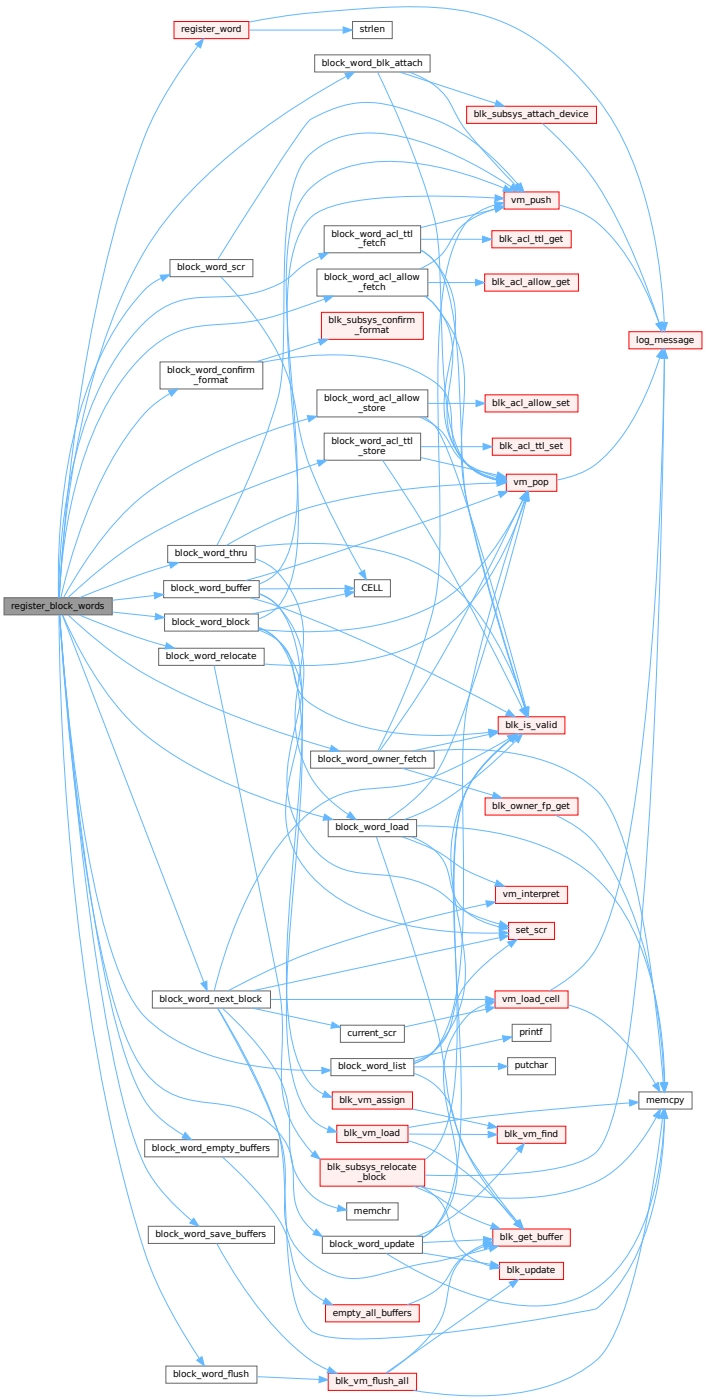
Parameters

vm	Pointer to the Forth virtual machine instance

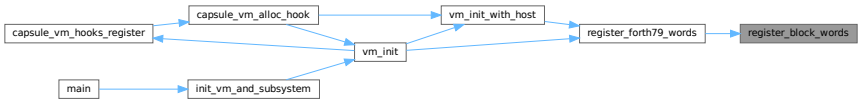
References block_word_acl_allow_fetch(), block_word_acl_allow_store(), block_word_acl_ttl_fetch(), block_word_acl_ttl_store(), block_word_blk_attach(), block_word_block(), block_word_buffer(), block_word_confirm_format(), block_word_empty_buffers(), block_word_flush(), block_word_list(), block_word_load(), block_word_next_block(), block_word_owner_fetch(), block_word_relocate(), block_word_save_buffers(), block_word_scr(), block_word_thru(), block_word_update(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



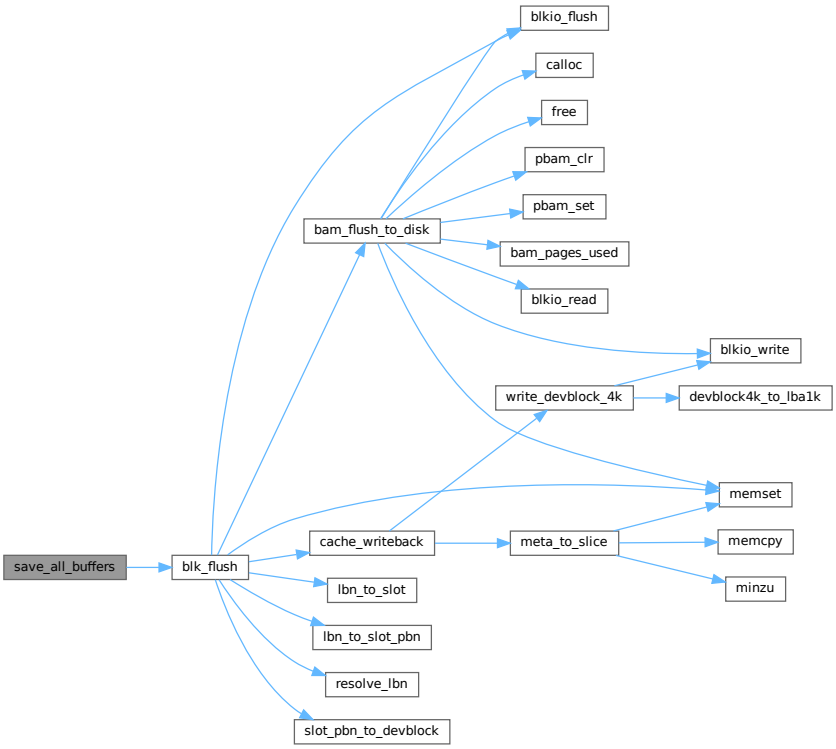
7.352.1.34 save_all_buffers()

```
void save_all_buffers (
            VM * vm )
```

References blk_flush().

Referenced by editor_word_edit().

Here is the call graph for this function:



Here is the caller graph for this function:

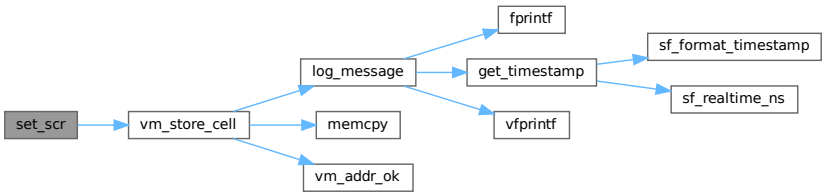
7.352.1.35 set_scr()

```
static void set_scr (
            VM * vm,
            cell_t blk )  [static]
```

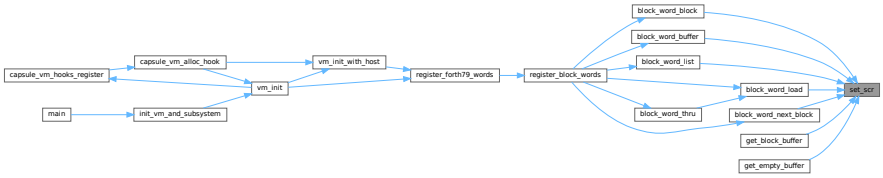
References VM::scr_addr, and vm_store_cell().

Referenced by block_word_block(), block_word_buffer(), block_word_list(), block_word_load(), block_word_next_block(), get_block_buffer(), and get_empty_buffer().

Here is the call graph for this function:



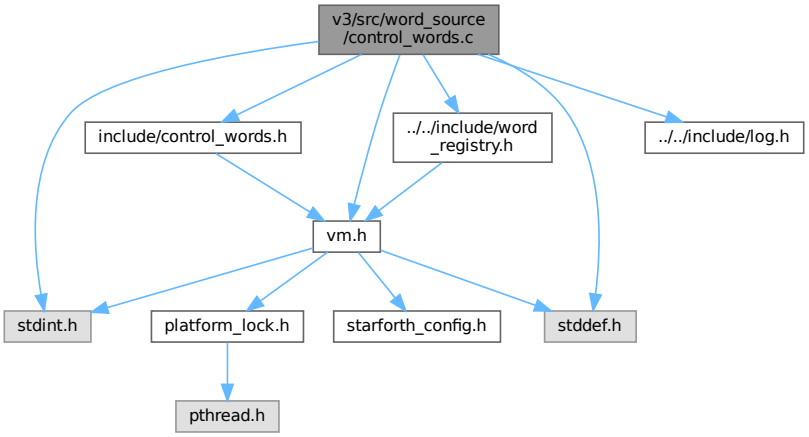
Here is the caller graph for this function:



7.353 v3/src/word_source/control_words.c File Reference

```
#include "include/control_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include "../../include/vm.h"
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for control_words.c:



Functions

- static void cf_epoch_sync (VM *vm)
- static int cf_push_item (VM *vm, cf_tag_t tag, size_t mark)
- static int cf_pop_item (VM *vm, cf_item_t *out)
- static int cf_peek_item (VM *vm, cf_item_t *out)
- static size_t emit_cell (VM *vm, cell_t value)
- static void control_forth_UNLOOP (VM *vm)
- static void control_forth_if (VM *vm)
- static void control_forth_else (VM *vm)
- static void control_forth_then (VM *vm)
- static void control_forth_begin (VM *vm)
- static void control_forth_until (VM *vm)
- static void control_forth_again (VM *vm)
- static void control_forth_while (VM *vm)
- static void control_forth_repeat (VM *vm)
- static void control_forth_qdo (VM *vm)
- static void control_forth_do (VM *vm)
- static void control_forth_leave (VM *vm)
- static void control_forth_loop (VM *vm)
- static void control_forth_plus_loop (VM *vm)
- static void control_forth_case (VM *vm)
- static void control_forth_of (VM *vm)
- static void control_forth_endof (VM *vm)
- static void control_forth_endcase (VM *vm)
- void register_control_words (VM *vm)

  Registers all control flow words with the Forth virtual machine.

Internal Runtime Functions

Forward declarations for internal runtime helpers and control flow primitives

- static void control_forth_branch (VM *vm)

  Execute unconditional branch.
- static void control_forth_0branch (VM *vm)

  Execute conditional branch based on top of stack.
- static void control_forth_runtime_qdo (VM *vm)

  Runtime handler for ?DO loops.
- static void control_forth_runtime_do (VM *vm)

  Runtime handler for DO loops.
- static void control_forth_runtime_loop (VM *vm)

  Runtime handler for LOOP.
- static void control_forth_runtime_plus_loop (VM *vm)

  Runtime handler for +LOOP.
- static void control_forth_runtime_leave (VM *vm)

  Runtime handler for LEAVE.
- static void control_forth_I (VM *vm)

  Get current loop index.
- static void control_forth_J (VM *vm)

  Get outer loop index.
- static void control_forth_EXIT (VM *vm)

  Exit from current definition.

7.353.1 Function Documentation

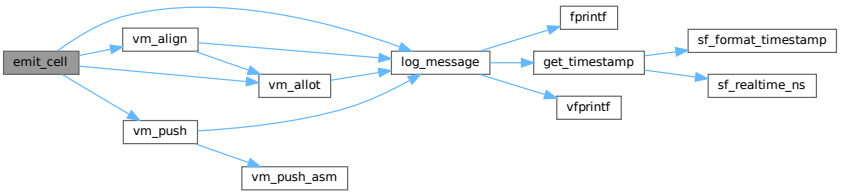
7.353.1.1 emit_cell()

```
static size_t emit_cell (
            VM * vm,
            cell_t value )  [inline], [static]
```

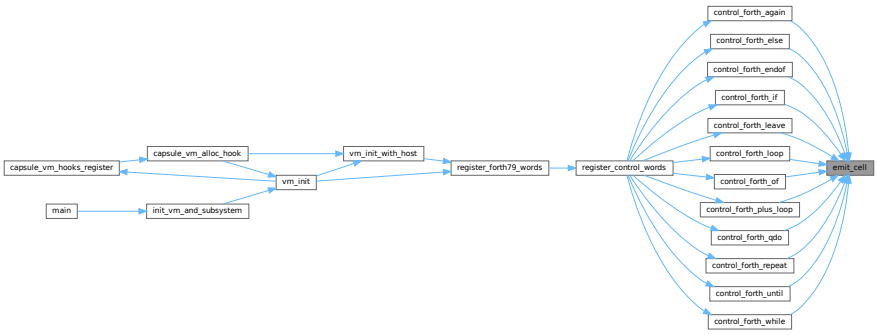
References VM::error, LOG_ERROR, log_message(), VM::memory, VM::mode, MODE_COMPILE, vm_align(), vm_allot(), and vm_push().

Referenced by control_forth_again(), control_forth_else(), control_forth_endof(), control_forth_if(), control_forth_leave(), control_forth_loop(), control_forth_of(), control_forth_plus_loop(), control_forth_qdo(), control_forth_repeat(), control_forth_until(), and control_forth_while().

Here is the call graph for this function:



Here is the caller graph for this function:

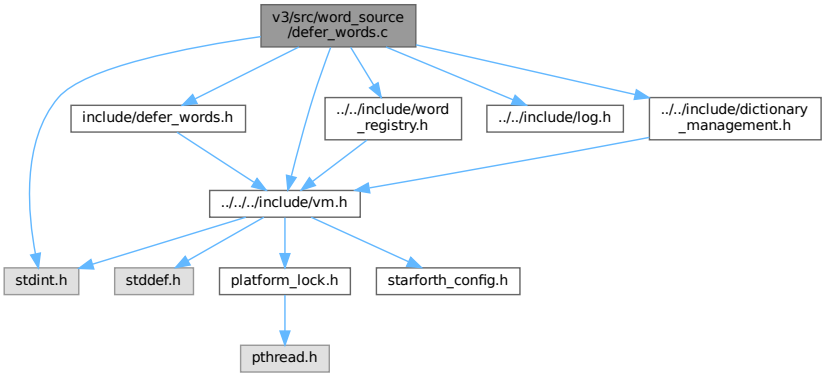


7.354 v3/src/word_source/defer_words.c File Reference

```
#include "include/defer_words.h"
#include "../../include/vm.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include "../../include/dictionary_management.h"
```

```
#include <stdint.h>
```

Include dependency graph for defer_words.c:



Functions

- static void defer_runtime (VM *vm)
- static void word_defer (VM *vm)
- static void word_is (VM *vm)
- static void word_defer_fetch (VM *vm)
- void register_defer_words (VM *vm)

7.354.1 Function Documentation

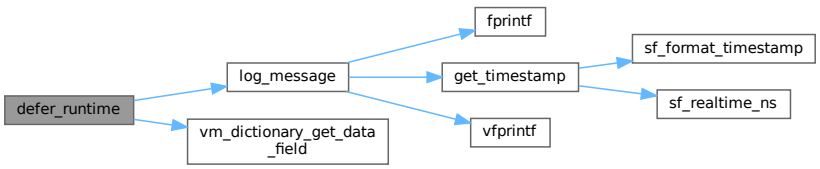
7.354.1.1 defer_runtime()

```
static void defer_runtime (
            VM * vm )  [static]
```

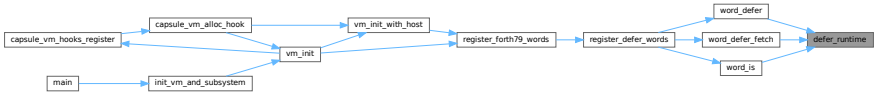
References VM::current_executing_entry, VM::error, DictEntry::func, LOG_ERROR, log_message(), DictEntry::name, DictEntry::name_len, and vm_dictionary_get_data_field().

Referenced by word_defer(), word_defer_fetch(), and word_is().

Here is the call graph for this function:



Here is the caller graph for this function:



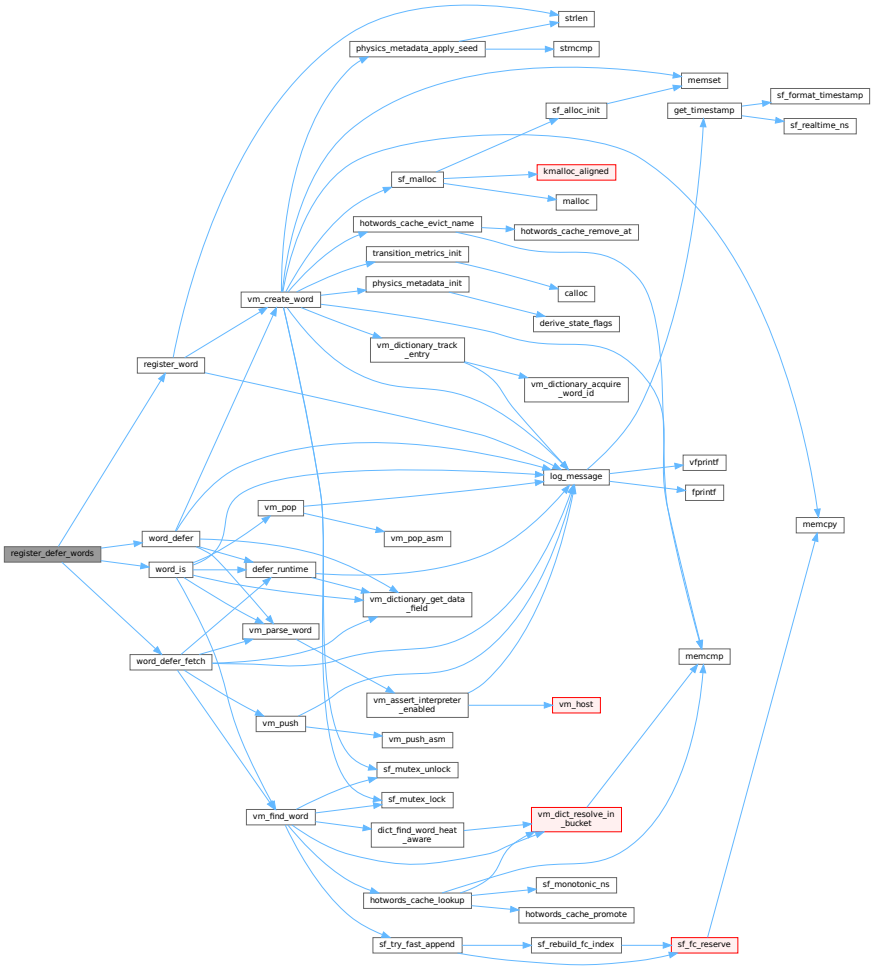
7.354.1.2 register_defer_words()

```
void register_defer_words (
            VM * vm )
```

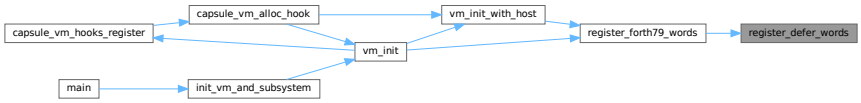
References register_word(), word_defer(), word_defer_fetch(), and word_is().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



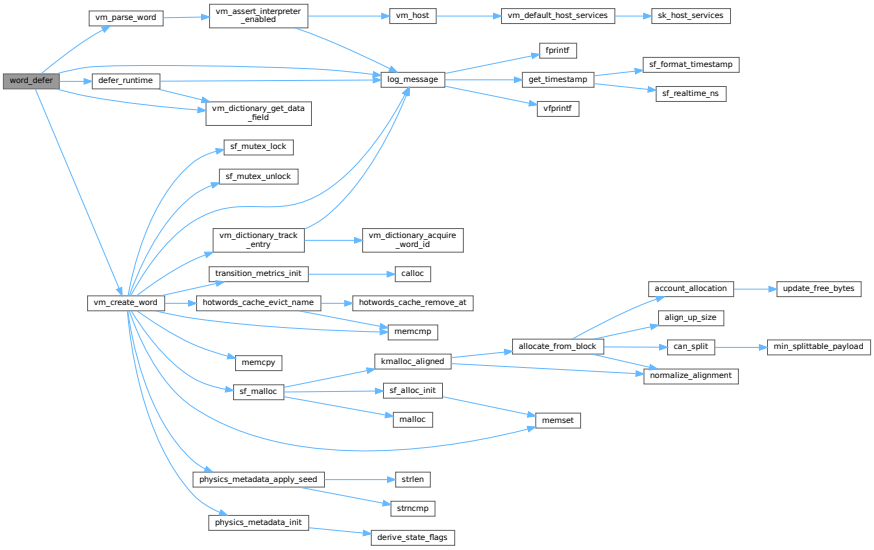
7.354.1.3 word_defer()

```
static void word_defer (
            VM * vm )  [static]
```

References defer_runtime(), VM::error, LOG_DEBUG, LOG_ERROR, log_message(), name, vm_create_word(), vm_dictionary_get_data_field(), vm_parse_word(), and WORD_NAME_MAX.

Referenced by register_defer_words().

Here is the call graph for this function:



Here is the caller graph for this function:

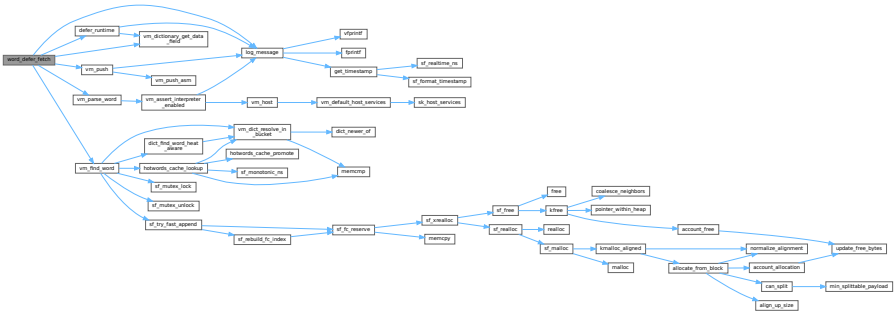
7.354.1.4 word_defer_fetch()

```
static void word_defer_fetch (
            VM * vm )  [static]
```

References defer_runtime(), VM::error, DictEntry::func, LOG_DEBUG, LOG_ERROR, log_message(), name, vm_dictionary_get_data_field(), vm_find_word(), vm_parse_word(), vm_push(), and WORD_NAME_MAX.

Referenced by register_defer_words().

Here is the call graph for this function:



Here is the caller graph for this function:

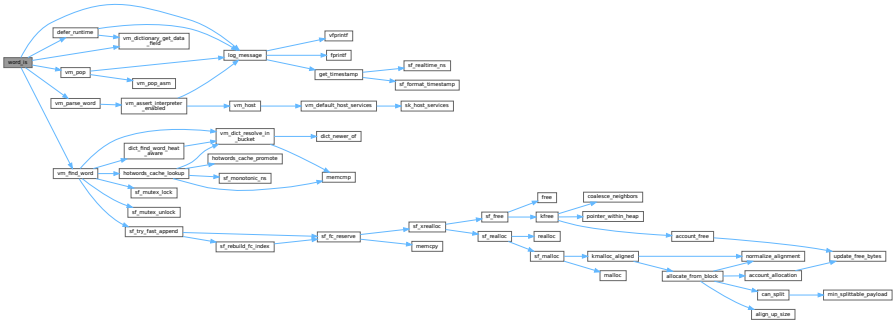
7.354.1.5 word_is()

```
static void word_is (
            VM * vm )  [static]
```

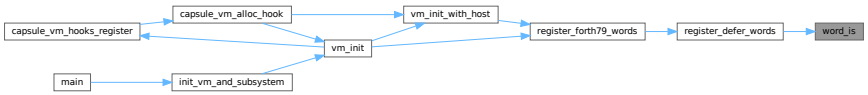
References defer_runtime(), VM::dsp, VM::error, DictEntry::func, LOG_DEBUG, LOG_ERROR, log_message(), name, vm_dictionary_get_data_field(), vm_find_word(), vm_parse_word(), vm_pop(), and WORD_NAME_MAX.

Referenced by register_defer_words().

Here is the call graph for this function:



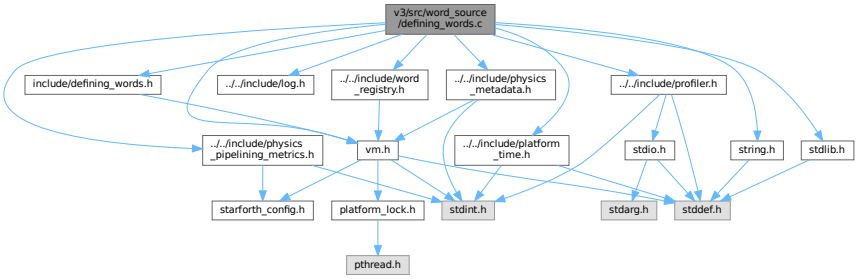
Here is the caller graph for this function:



7.355 v3/src/word_source/defining_words.c File Reference

```
#include "include/defining_words.h"
#include "../../include/vm.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include "../../include/profiler.h"
#include "../../include/physics_metadata.h"
#include "../../include/platform_time.h"
#include "../../include/physics_pipelining_metrics.h"
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for defining_words.c:



Functions

- static void defining_runtime_create (VM *vm)

  Runtime behavior for words defined by CREATE.
- static void defining_runtime_constant (VM *vm)

  Runtime behavior for CONSTANT words.
- static void defining_runtime_variable (VM *vm)

  Runtime behavior for VARIABLE words.
- static void defining_runtime_lit (VM *vm)

  Runtime behavior for literal values in threaded code.
- static void defining_runtime_dodoes (VM *vm)

  Runtime behavior for words defined with DOES>
- static void defining_runtime_does_rt (VM *vm)

  Helper routine executed at DOES> time.
- static void defining_word_colon (VM *vm)
- static void defining_word_semicolon (VM *vm)

- static void defining_word_create (VM *vm)
- static void defining_word_constant (VM *vm)
- static void defining_word_variable (VM *vm)
- static void defining_word_left_bracket (VM *vm)
- static void defining_word_right_bracket (VM *vm)
- static void defining_word_state (VM *vm)
- static void defining_word_compile (VM *vm)
- static void defining_word_bracket_compile (VM *vm)
- static void defining_word_literal (VM *vm)
- static void defining_word_does (VM *vm)
- static void defining_word_immediate (VM *vm)
- static void dictionary_word_forget (VM *vm)
- static void dictionary_word_fence (VM *vm)
- void register_defining_words (VM *vm)

  Register all defining words in the dictionary.

7.355.1 Function Documentation

7.355.1.1 defining_runtime_constant()

```
static void defining_runtime_constant (
            VM * vm )  [static]
```

Runtime behavior for CONSTANT words.

Pushes the constant value stored in the word's data field onto the stack. Stack effect: (– n)

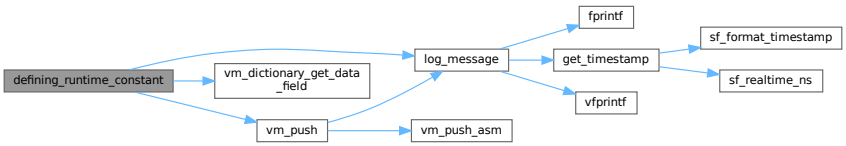
Parameters

vm	Pointer to the VM context

References VM::current_executing_entry, entry, VM::error, LOG_DEBUG, log_message(), vm_dictionary_get_data_field(), and vm_push().

Referenced by defining_word_constant().

Here is the call graph for this function:



Here is the caller graph for this function:

7.355.1.2 defining_runtime_create()

```
static void defining_runtime_create (
            VM * vm )  [static]
```

Runtime behavior for words defined by CREATE.

Pushes the word's data field address (DFA) captured at CREATE time. Stack effect: (– addr)

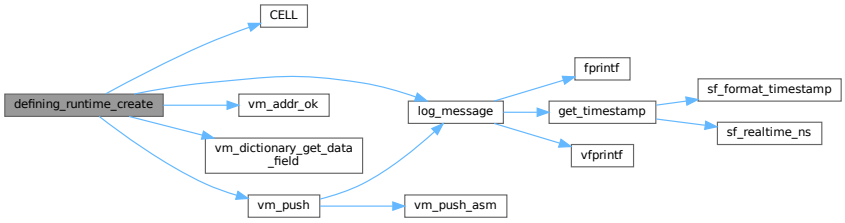
Parameters

vm	Pointer to the VM context

References CELL(), VM::current_executing_entry, entry, VM::error, LOG_DEBUG, log_message(), vm_addr_ok(), vm_dictionary_get_data_field(), and vm_push().

Referenced by defining_word_create().

Here is the call graph for this function:



Here is the caller graph for this function:

7.355.1.3 defining_runtime_dodoes()

```
static void defining_runtime_dodoes (
            VM * vm )  [static]
```

Runtime behavior for words defined with DOES>

Pushes the address of user data area then executes the DOES> body code. PFA layout: [body_vaddr][user_data...]

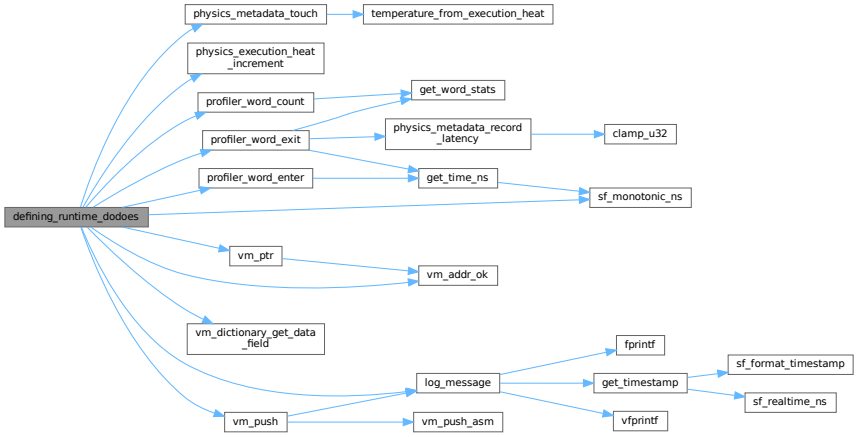
Parameters

vm	Pointer to the VM context

References VM::abort_requested, VM::current_executing_entry, entry, VM::error, VM::exit_colon, LOG_DEBUG, LOG_ERROR, log_message(), physics_execution_heat_increment(), physics_metadata_touch(), profiler_word_count(), profiler_word_enter(), profiler_word_exit(), VM::return_stack, VM::rsp, sf_monotonic_ns(), STACK_SIZE, vm_addr_ok(), vm_dictionary_get_data_field(), vm_ptr(), and vm_push().

Referenced by defining_runtime_does_rt().

Here is the call graph for this function:



Here is the caller graph for this function:

7.355.1.4 defining_runtime_does_rt()

```
static void defining_runtime_does_rt (
            VM * vm )  [static]
```

Helper routine executed at DOES> time.

Converts the just-created child word to use DODOES runtime behavior and records the DOES> body address in the child's PFA.

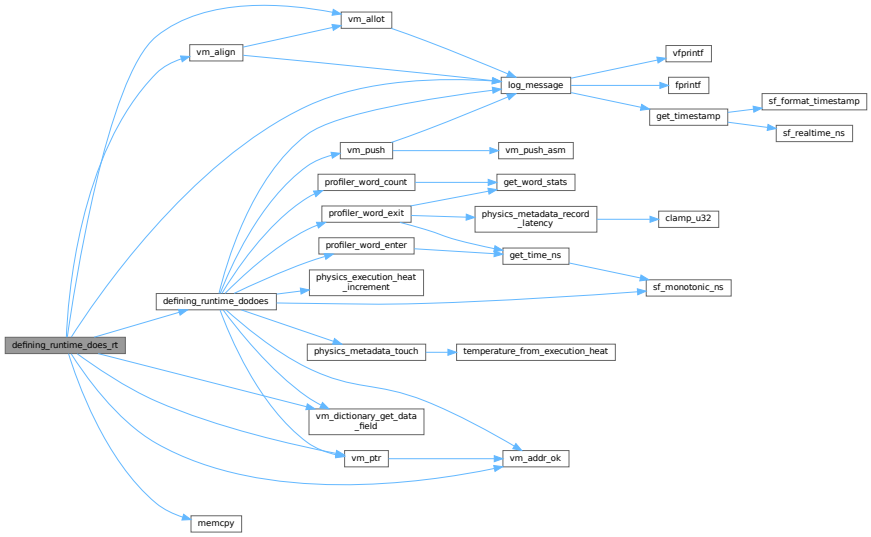
Parameters

vm	Pointer to the VM context

References defining_runtime_dodoes(), VM::error, DictEntry::func, VM::here, VM::latest, LOG_ERROR, log_message(), memcpy(), VM::memory, VM::return_stack, VM::rsp, vm_addr_ok(), vm_align(), vm_allot(), vm_dictionary_get_data_field(), and vm_ptr().

Referenced by defining_word_does(), and register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.355.1.5 defining_runtime_lit()

```
static void defining_runtime_lit (
            VM * vm )  [static]
```

Runtime behavior for literal values in threaded code.

Fetches the next cell from threaded code via IP and pushes it. Stack effect: (– n)

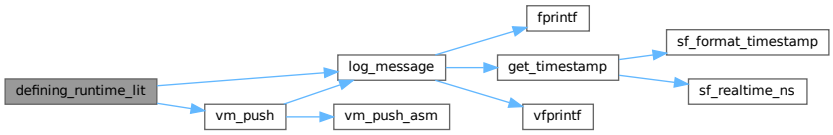
Parameters

vm	Pointer to the VM context

References VM::error, LOG_DEBUG, log_message(), VM::return_stack, VM::rsp, and vm_push().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.355.1.6 defining_runtime_variable()

```
static void defining_runtime_variable (
            VM * vm )  [static]
```

Runtime behavior for VARIABLE words.

Pushes the variable's address (VM offset) stored in the word's data field. Stack effect: (– addr)

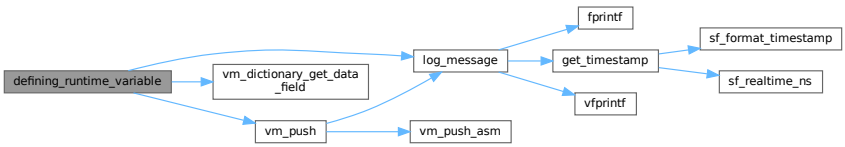
Parameters

vm	Pointer to the VM context

References VM::current_executing_entry, entry, VM::error, LOG_DEBUG, log_message(), vm_dictionary_get_data_field(), and vm_push().

Referenced by defining_word_variable().

Here is the call graph for this function:



Here is the caller graph for this function:

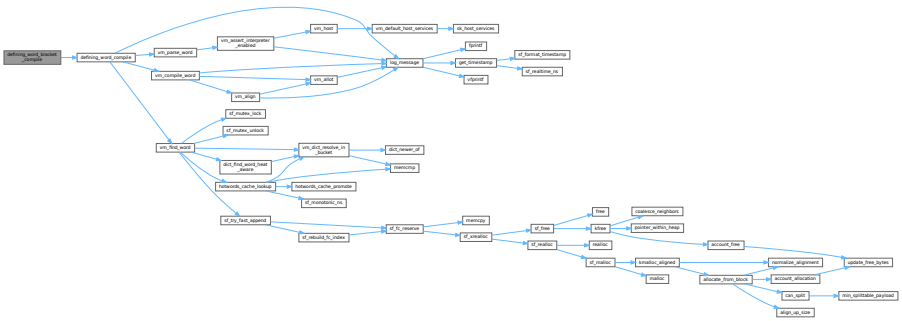
7.355.1.7 defining_word_bracket_compile()

```
static void defining_word_bracket_compile (
            VM * vm )  [static]
```

References defining_word_compile().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

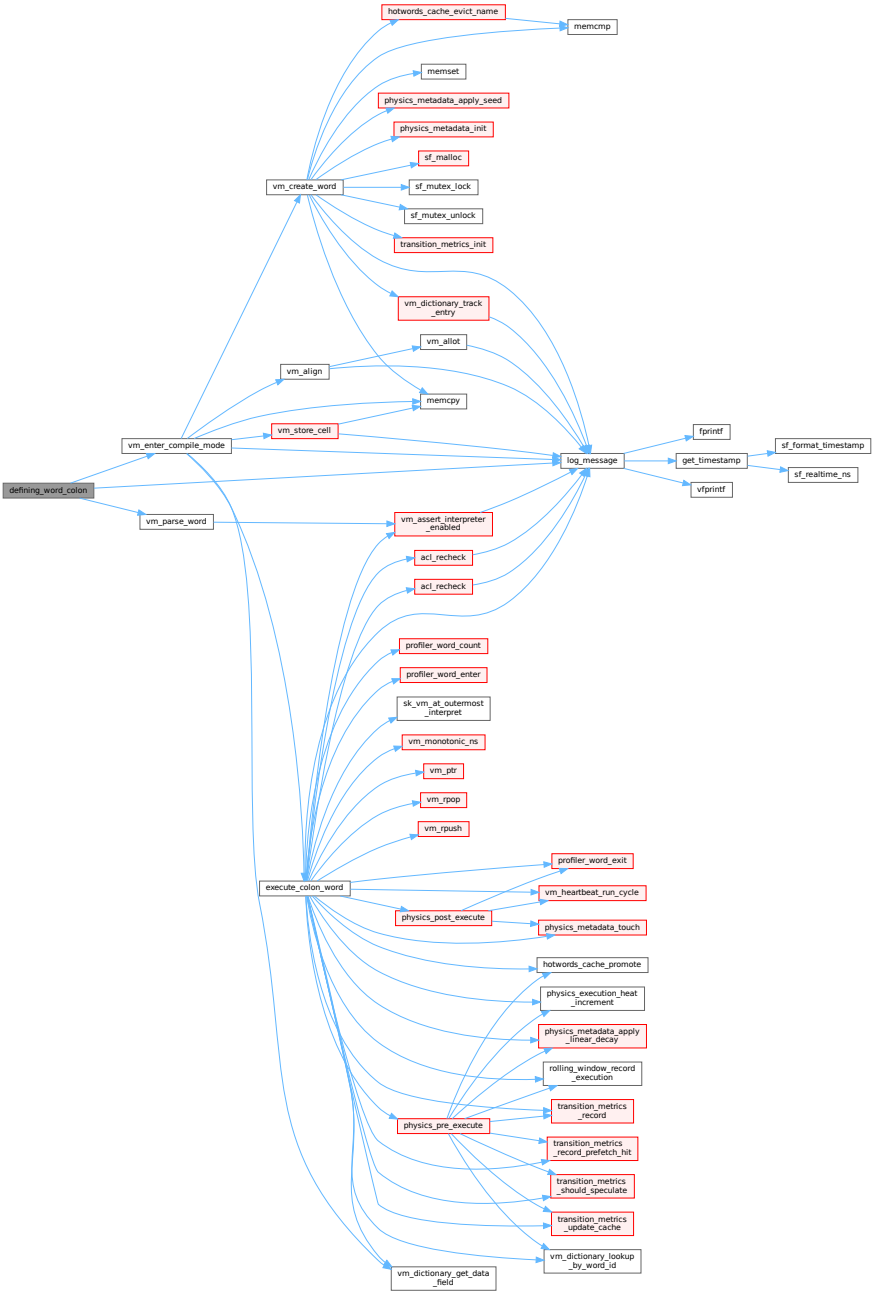
7.355.1.8 defining_word_colon()

```
static void defining_word_colon (
            VM * vm )  [static]
```

References VM::error, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_enter_compile_mode(), vm_parse_word(), and WORD_NAME_MAX.

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

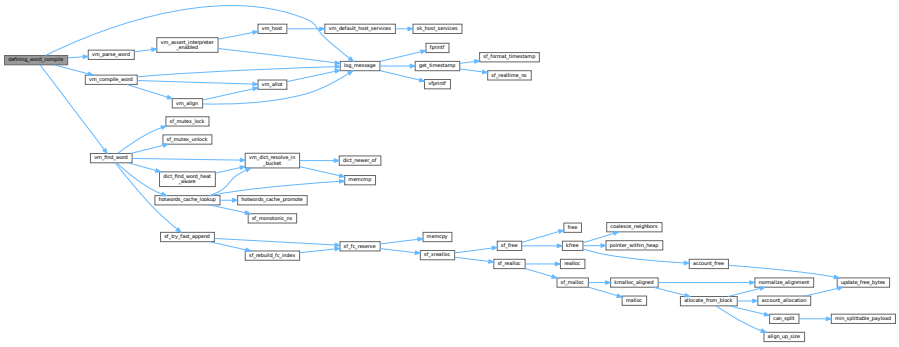
7.355.1.9 defining_word_compile()

```
static void defining_word_compile (
            VM * vm )  [static]
```

References entry, VM::error, LOG_DEBUG, log_message(), vm_compile_word(), vm_find_word(), vm_parse_word(), and WORD_NAME_MAX.

Referenced by defining_word_bracket_compile(), and register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

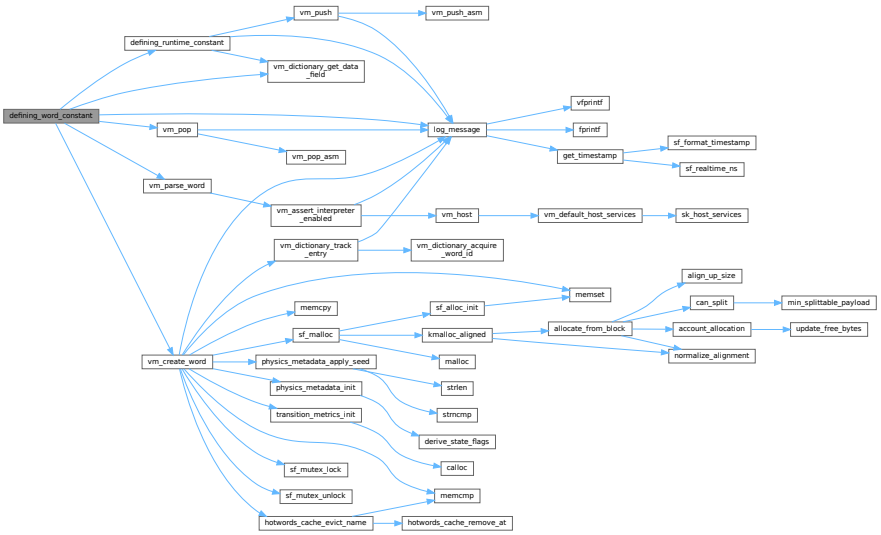
7.355.1.10 defining_word_constant()

```
static void defining_word_constant (
            VM * vm )  [static]
```

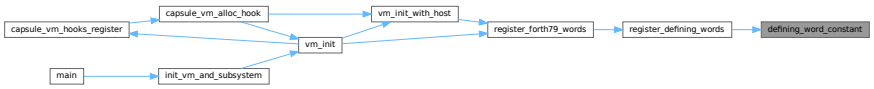
References defining_runtime_constant(), VM::dsp, entry, VM::error, LOG_DEBUG, log_message(), vm_create_word(), vm_dictionary_get_data_field(), vm_parse_word(), vm_pop(), and WORD_NAME_MAX.

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:



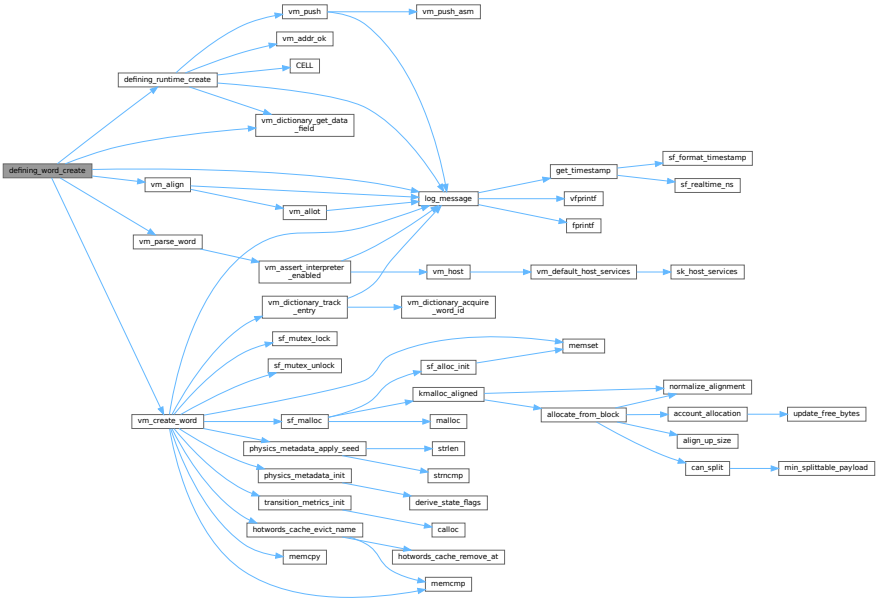
7.355.1.11 defining_word_create()

```
static void defining_word_create (
            VM * vm )  [static]
```

References defining_runtime_create(), entry, VM::error, VM::here, LOG_DEBUG, log_message(), vm_align(), vm_create_word(), vm_dictionary_get_data_field(), vm_parse_word(), and WORD_NAME_MAX.

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

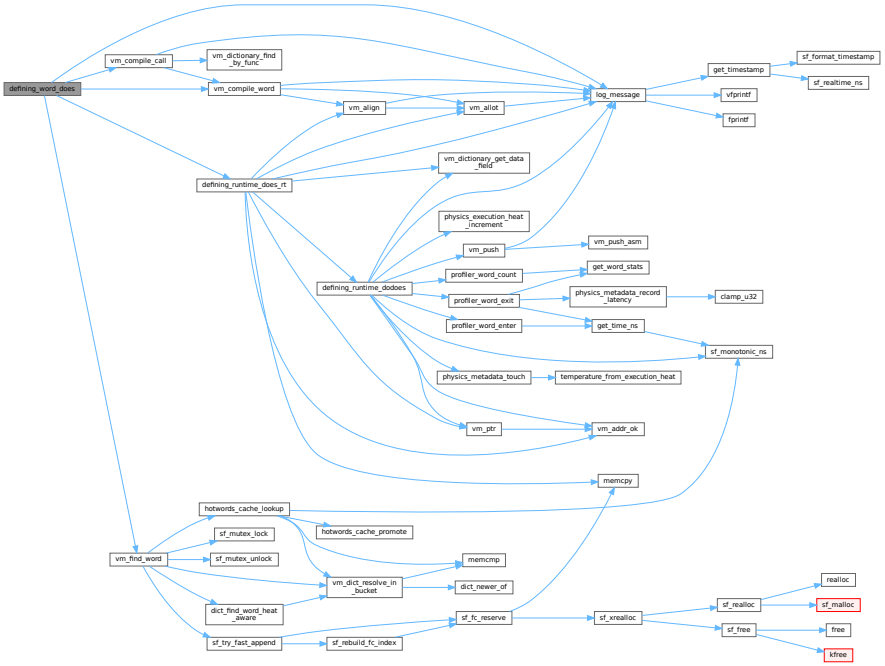
7.355.1.12 defining_word_does()

```
static void defining_word_does (
            VM * vm )  [static]
```

References defining_runtime_does_rt(), VM::error, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_compile_call(), vm_compile_word(), and vm_find_word().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

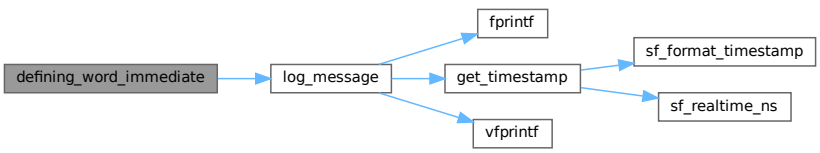
7.355.1.13 defining_word_immediate()

```
static void defining_word_immediate (
            VM * vm )  [static]
```

References VM::error, DictEntry::flags, VM::latest, LOG_DEBUG, log_message(), DictEntry::name, DictEntry::name_len, and WORD_IMMEDIATE.

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

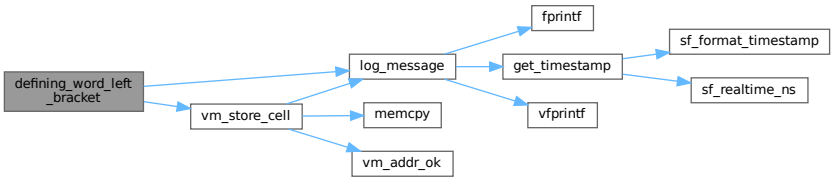
7.355.1.14 defining_word_left_bracket()

```
static void defining_word_left_bracket (
            VM * vm )  [static]
```

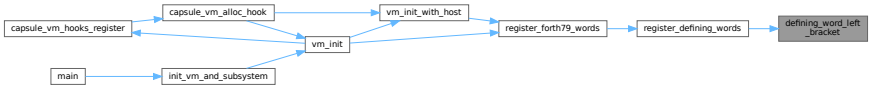
References LOG_DEBUG, log_message(), VM::mode, MODE_INTERPRET, VM::state_addr, and vm_store_cell().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:



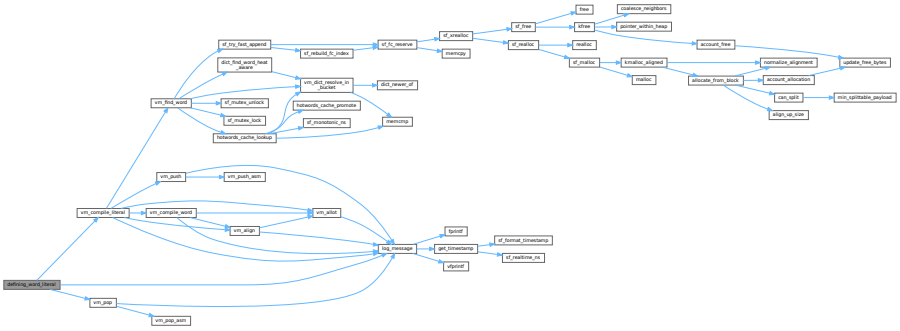
7.355.1.15 defining_word_literal()

```
static void defining_word_literal (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, log_message(), vm_compile_literal(), and vm_pop().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

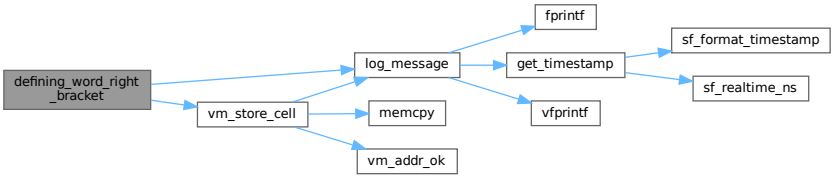
7.355.1.16 defining_word_right_bracket()

```
static void defining_word_right_bracket (
            VM * vm )  [static]
```

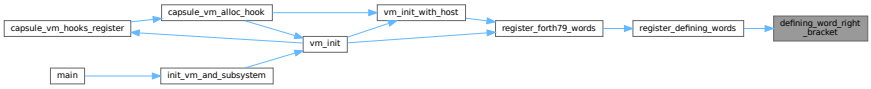
References LOG_DEBUG, log_message(), VM::mode, MODE_COMPILE, VM::state_addr, and vm_store_cell().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:



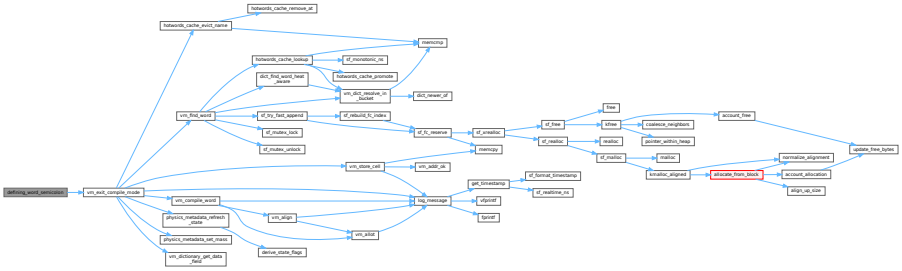
7.355.1.17 defining_word_semicolon()

```
static void defining_word_semicolon (
            VM * vm )  [static]
```

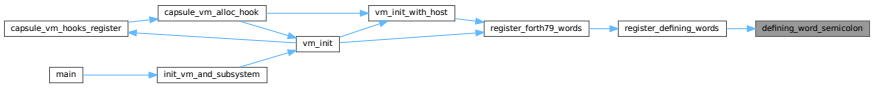
References VM::error, VM::mode, MODE_COMPILE, and vm_exit_compile_mode().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:



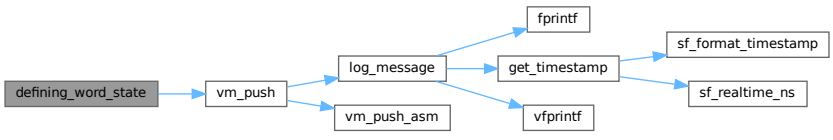
7.355.1.18 defining_word_state()

```
static void defining_word_state (
            VM * vm )  [static]
```

References VM::state_addr, and vm_push().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

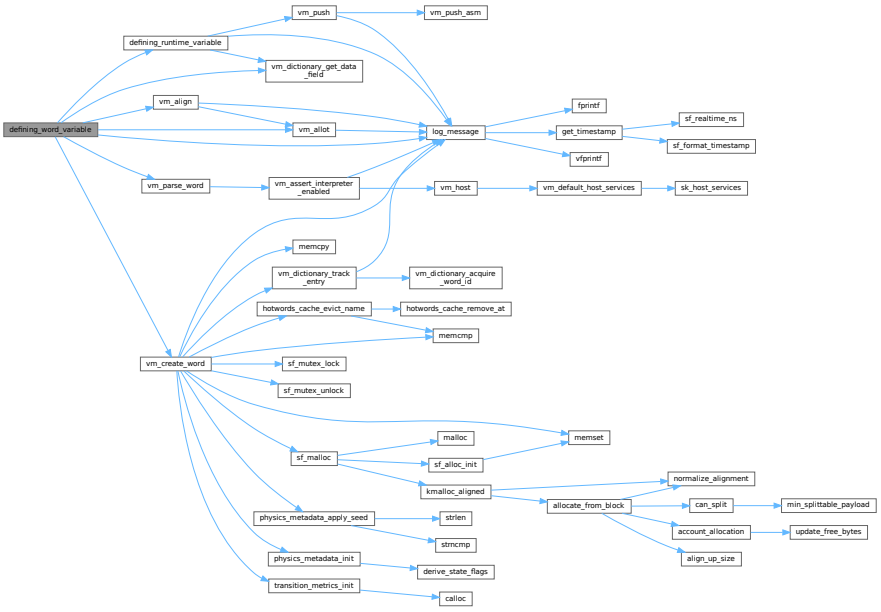
7.355.1.19 defining_word_variable()

```
static void defining_word_variable (
            VM * vm )  [static]
```

References defining_runtime_variable(), entry, VM::error, VM::here, LOG_DEBUG, log_message(), vm_align(), vm_allot(), vm_create_word(), vm_dictionary_get_data_field(), vm_parse_word(), and WORD_NAME_MAX.

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

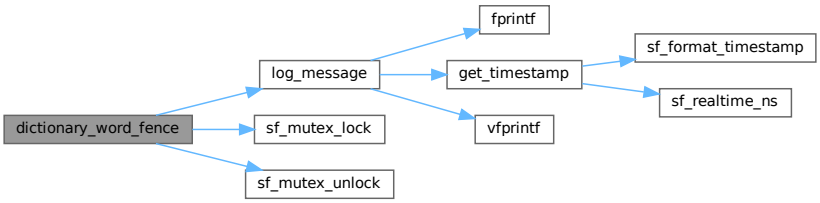
7.355.1.20 dictionary_word_fence()

```
static void dictionary_word_fence (
            VM * vm )  [static]
```

References VM::dict_fence_here, VM::dict_fence_latest, VM::dict_lock, VM::here, VM::latest, LOG_DEBUG, log_message(), sf_mutex_lock(), and sf_mutex_unlock().

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:

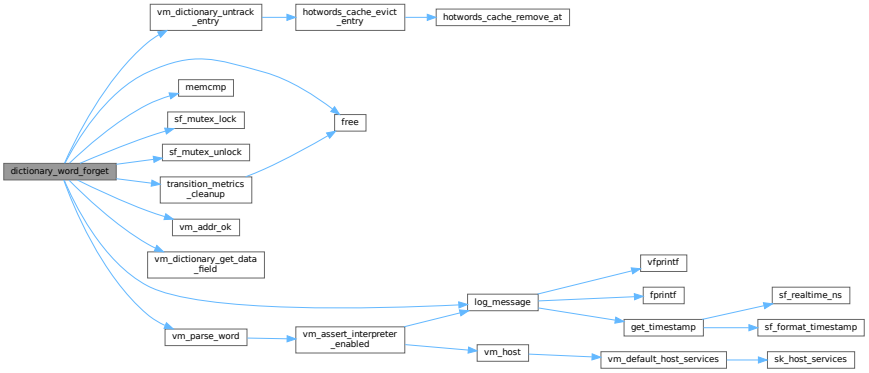
7.355.1.21 dictionary_word_forget()

```
static void dictionary_word_forget (
            VM * vm )  [static]
```

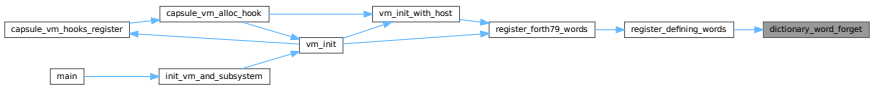
References VM::dict_fence_here, VM::dict_fence_latest, VM::dict_lock, VM::error, DictEntry::flags, free(), VM::here, VM::latest, DictEntry::link, LOG_DEBUG, log_message(), memcmp(), DictEntry::name, DictEntry::name_len, sf_mutex_lock(), sf_mutex_unlock(), DictEntry::transition_metrics, transition_metrics_cleanup(), vm_addr_ok(), vm_dictionary_get_data_field(), vm_dictionary_untrack_entry(), vm_parse_word(), WORD_HIDDEN, and WORD_NAME_MAX.

Referenced by register_defining_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.355.1.22 register_defining_words()

```
void register_defining_words (
            VM * vm )
```

Register all defining words in the dictionary.

Registers core defining words like :, ;, CREATE, CONSTANT, VARIABLE, and their support words like IMMEDIATE, STATE, etc.

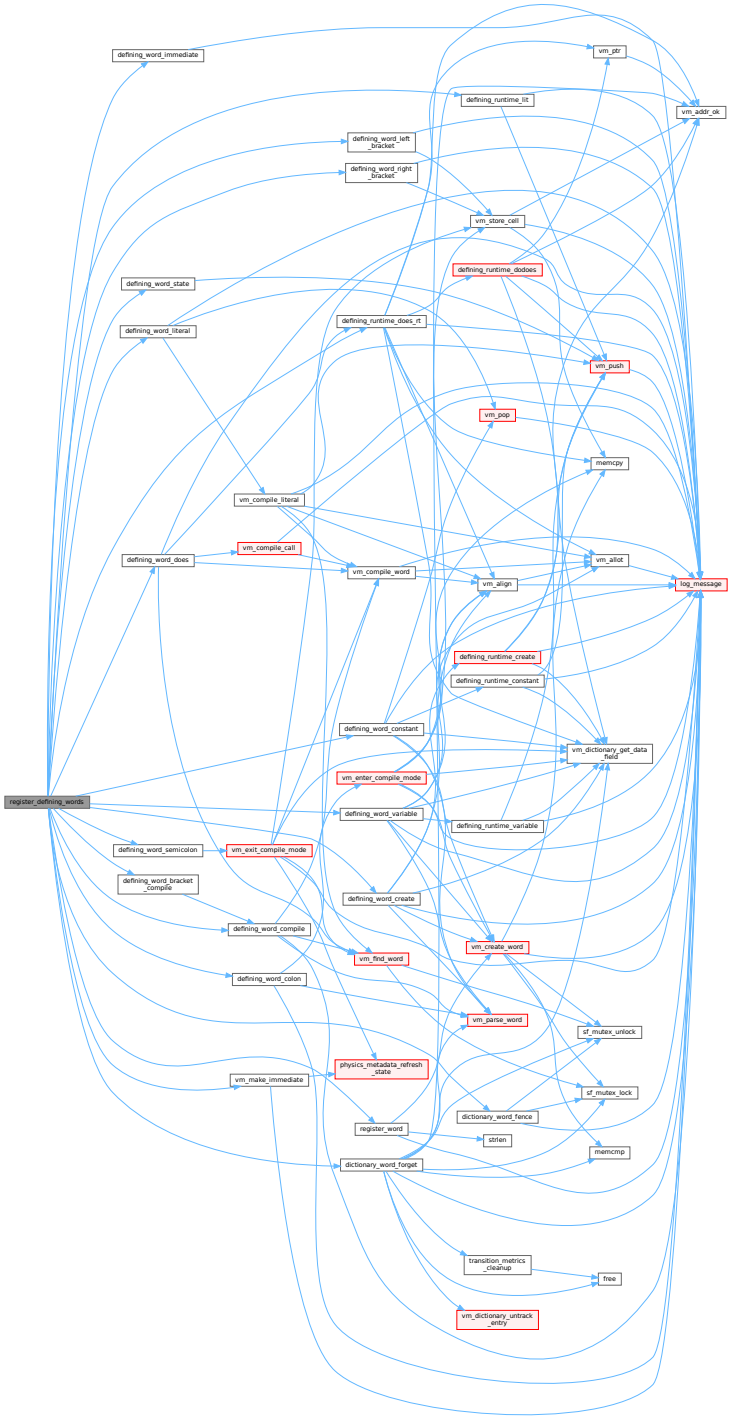
Parameters

vm	Pointer to the VM context

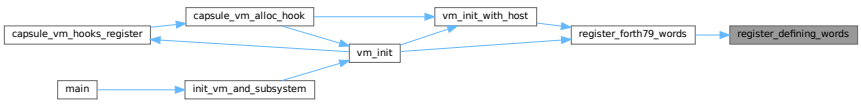
References defining_runtime_does_rt(), defining_runtime_lit(), defining_word_bracket_compile(), defining_word_colon(), defining_word_compile(), defining_word_constant(), defining_word_create(), defining_word_does(), defining_word_immediate(), defining_word_left_bracket(), defining_word_literal(), defining_word_right_bracket(), defining_word_semicolon(), defining_word_state(), defining_word_variable(), dictionary_word_fence(), dictionary_word_forget(), register_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



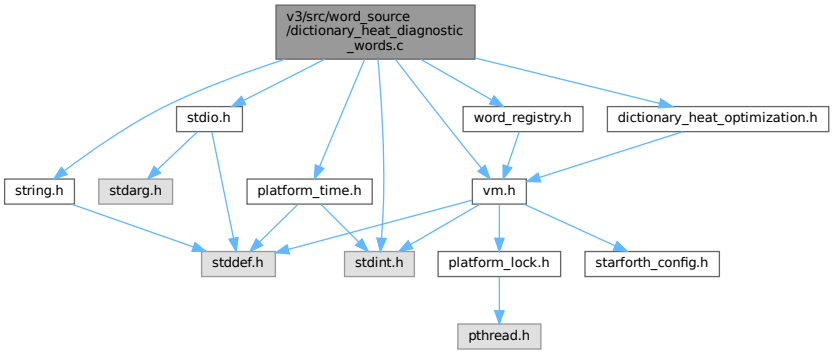
Here is the caller graph for this function:



7.356 v3/src/word_source/dictionary_heat_diagnostic_words.c File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "vm.h"
#include "word_registry.h"
#include "platform_time.h"
#include "dictionary_heat_optimization.h"
```

Include dependency graph for dictionary_heat_diagnostic_words.c:



Functions

- void forth_HEAT_PERCENTILES (VM *vm)

  HEAT-PERCENTILES (– 25th 50th 75th)
- void forth_LOOKUP_STRATEGY_FETCH (VM *vm)

  LOOKUP-STRATEGY@ (– strategy)
- void forth_LOOKUP_STRATEGY_STORE (VM *vm)

  LOOKUP-STRATEGY! (strategy –)
- void forth_REORG_BUCKETS (VM *vm)

  REORG-BUCKETS (–)
- void forth_SHOW_HEAT_OPTIMIZATION (VM *vm)

  SHOW-HEAT-OPTIMIZATION (–)
- void forth_COMPARE_LOOKUPS (VM *vm)

  COMPARE-LOOKUPS (iterations –)
- void register_dictionary_heat_diagnostic_words (VM *vm)

  Register all dictionary heat diagnostic words with the VM dictionary.

7.356.1 Function Documentation

7.356.1.1 forth_COMPARE_LOOKUPS()

```
void forth_COMPARE_LOOKUPS (
            VM * vm )
```

COMPARE-LOOKUPS (iterations –)

Micro-benchmark that measures naive vs heat-aware dictionary lookup performance over `iterations` passes through six common test words (DUP, DROP, SWAP, @, !, EMIT). Prints elapsed time in milliseconds and the percentage speedup (or slowdown) for heat-aware vs naive. Restores the original `vm->lookup_strategy` on completion. Intended for interactive diagnostics; not used in DoE mode.

Stack effect: (iterations –)

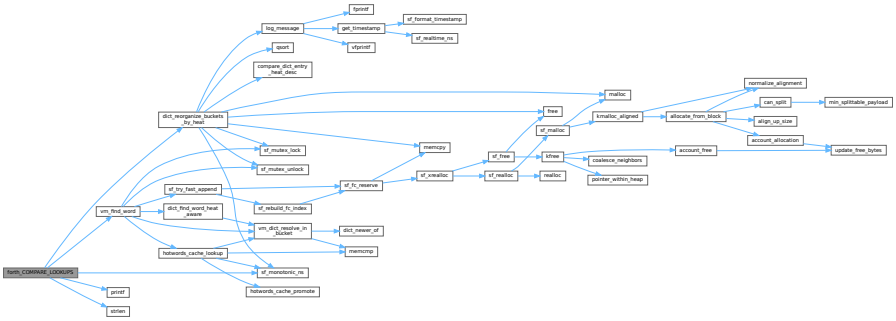
Parameters

vm	Active VM; sets `vm->error` = 1 on underflow or non-positive `iterations`

References VM::data_stack, dict_reorganize_buckets_by_heat(), VM::dsp, VM::error, VM::lookup_strategy, printf(), sf_monotonic_ns(), strlen(), and vm_find_word().

Referenced by register_dictionary_heat_diagnostic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.356.1.2 forth_HEAT_PERCENTILES()

```
void forth_HEAT_PERCENTILES (
            VM * vm )
```

HEAT-PERCENTILES (– 25th 50th 75th)

Pushes the three heat percentile thresholds currently used by the heat-aware dictionary lookup strategy: `heat_threshold_25th`, `heat_threshold_50th`, and `heat_threshold_75th`. These are updated by `dict_update_heat_percentiles()` during bucket reorganisation.

Stack effect: (– 25th 50th 75th) TOS = 75th

Parameters

vm	Active VM; sets `vm->error` = 1 on stack overflow

References VM::data_stack, VM::dsp, VM::error, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, and STACK_SIZE.

Referenced by register_dictionary_heat_diagnostic_words().

Here is the caller graph for this function:

7.356.1.3 forth_LOOKUP_STRATEGY_FETCH()

```
void forth_LOOKUP_STRATEGY_FETCH (
            VM * vm )
```

LOOKUP-STRATEGY@ (– strategy)

Pushes the current dictionary lookup strategy: 0 = naive (newest-first linear scan), 1 = heat-aware (hot-bucket-first).

Stack effect: (– strategy)

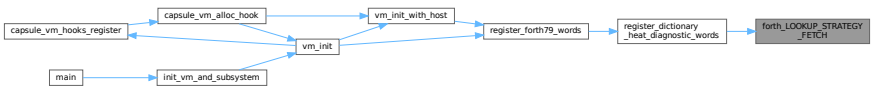
Parameters

vm	Active VM; sets `vm->error` = 1 on stack overflow

References VM::data_stack, VM::dsp, VM::error, VM::lookup_strategy, and STACK_SIZE.

Referenced by register_dictionary_heat_diagnostic_words().

Here is the caller graph for this function:



7.356.1.4 forth_LOOKUP_STRATEGY_STORE()

```
void forth_LOOKUP_STRATEGY_STORE (
            VM * vm )
```

LOOKUP-STRATEGY! (strategy –)

Sets `vm->lookup_strategy` to `strategy`. Only values 0 (naive) and 1 (heat-aware) are accepted; out-of-range values are silently discarded.

Stack effect: (strategy –)

Parameters

vm	Active VM; sets `vm->error` = 1 on stack underflow

References VM::data_stack, VM::dsp, VM::error, and VM::lookup_strategy.

Referenced by register_dictionary_heat_diagnostic_words().

Here is the caller graph for this function:

7.356.1.5 forth_REORG_BUCKETS()

```
void forth_REORG_BUCKETS (
            VM * vm )
```

REORG-BUCKETS (–)

Immediately triggers `dict_reorganize_buckets_by_heat()` followed by `dict_update_heat_percentiles()` to re-sort the dictionary's lookup buckets by current execution heat and refresh the percentile thresholds. Normally this happens automatically on a heartbeat cycle; this word forces it on-demand for testing or after a large batch of `HEAT!` assignments.

Stack effect: (–)

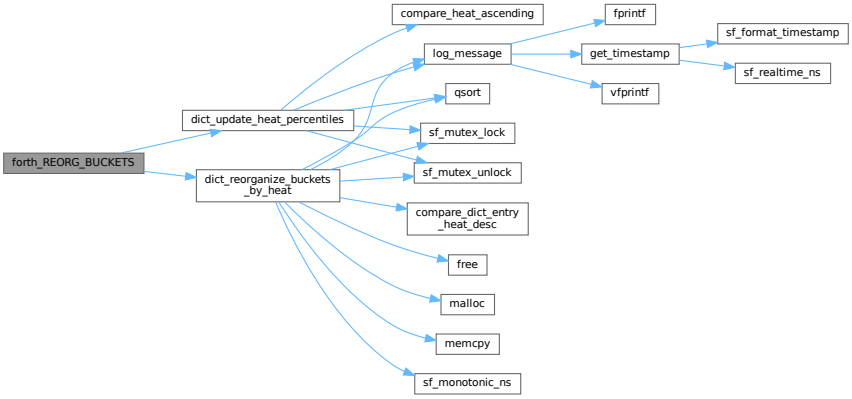
Parameters

vm	Active VM; no-op if `vm->error` is set

References dict_reorganize_buckets_by_heat(), dict_update_heat_percentiles(), and VM::error.

Referenced by register_dictionary_heat_diagnostic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.356.1.6 forth_SHOW_HEAT_OPTIMIZATION()

```
void forth_SHOW_HEAT_OPTIMIZATION (
            VM * vm )
```

SHOW-HEAT-OPTIMIZATION (–)

Prints a summary of the current dictionary heat optimization state to `stdout`: the active lookup strategy (naive or heat-aware), the three percentile threshold values, and the resulting hot/warm/cool heat zones. Intended for interactive diagnostics.

Stack effect: (–)

Parameters

vm	Active VM; no-op if `vm->error` is set

References VM::error, VM::heat_threshold_25th, VM::heat_threshold_50th, VM::heat_threshold_75th, VM::lookup_strategy, and printf().

Referenced by register_dictionary_heat_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:

7.356.1.7 register_dictionary_heat_diagnostic_words()

```
void register_dictionary_heat_diagnostic_words (
            VM * vm )
```

Register all dictionary heat diagnostic words with the VM dictionary.

Register all dictionary heat diagnostic words with the VM.

Registers: `HEAT-PERCENTILES`, `LOOKUP-STRATEGY@`, `LOOKUP-STRATEGY!`, `REORG-BUCKETS`, `SHOW-HEAT-OPTIMIZATION`, `COMPARE-LOOKUPS`. Called during VM bootstrap.

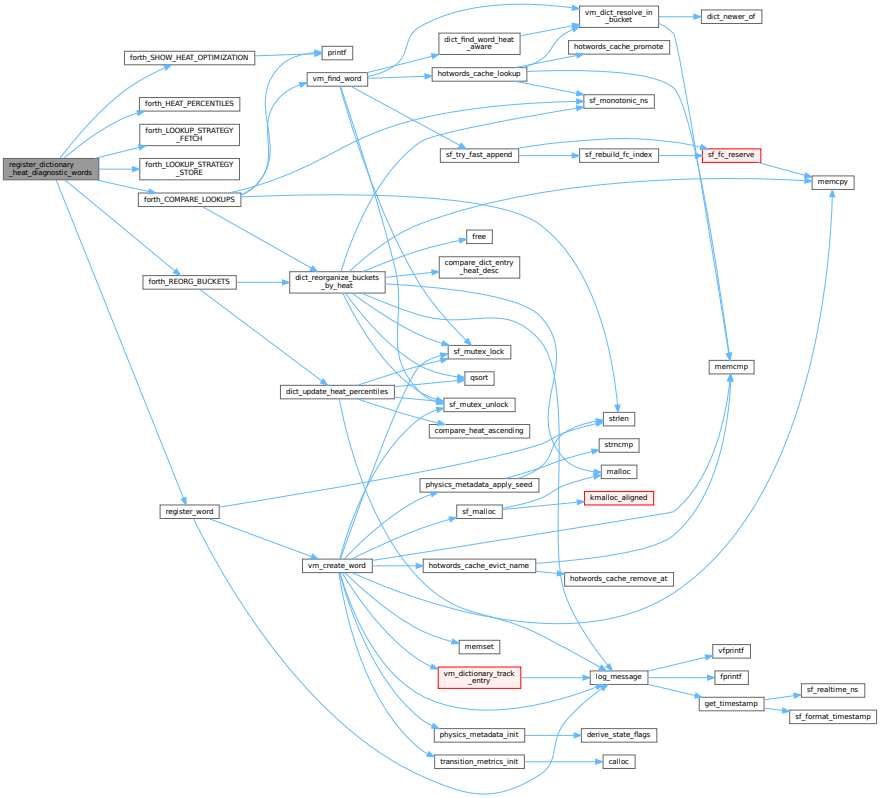
Parameters

vm	Active VM to register words into

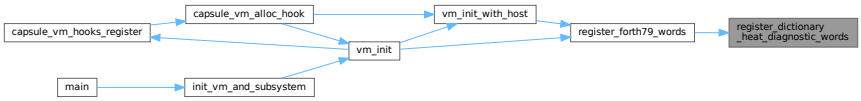
References forth_COMPARE_LOOKUPS(), forth_HEAT_PERCENTILES(), forth_LOOKUP_STRATEGY_FETCH(), forth_LOOKUP_STRATEGY_STORE(), forth_REORG_BUCKETS(), forth_SHOW_HEAT_OPTIMIZATION(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



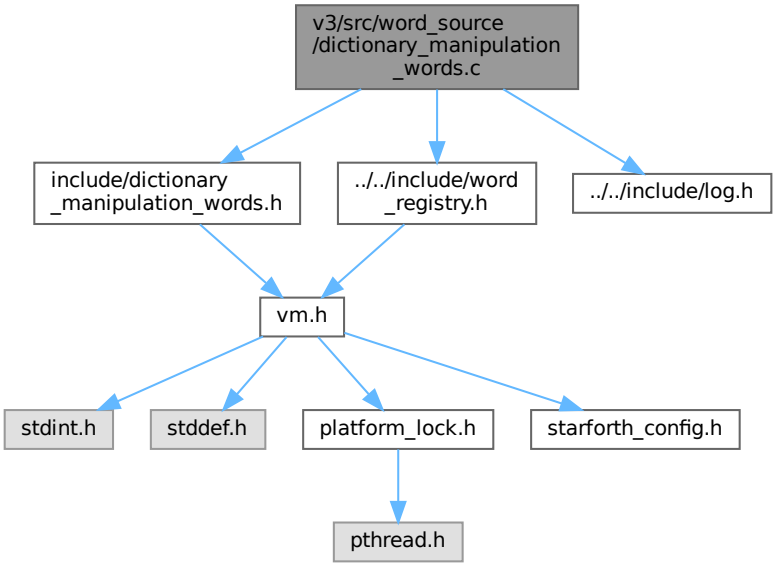
Here is the caller graph for this function:



7.357 v3/src/word_source/dictionary_manipulation_words.c File Reference

```
#include "include/dictionary_manipulation_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
```

Include dependency graph for dictionary_manipulation_words.c:



Functions

- static uintptr_t align_address (uintptr_t addr)
- static char * get_name_field (DictEntry *entry)
- static void * get_body_address (DictEntry *entry)
- static DictEntry * find_entry_from_name (VM *vm, char *name_field)
- static char * traverse_name_field (char *name_addr, int direction)
- static void dictionary_m_word_smudge (VM *vm)

  FORTH word SMUDGE - Toggle smudge bit of latest word.
- void dictionary_m_word_to_body (VM *vm)
- void dictionary_m_word_to_name (VM *vm)
- void dictionary_m_word_name_to (VM *vm)
- void dictionary_m_word_to_link (VM *vm)
- void dictionary_m_word_link_from (VM *vm)
- void dictionary_m_word_cfa (VM *vm)
- void dictionary_m_word_lfa (VM *vm)
- void dictionary_m_word_nfa (VM *vm)
- void dictionary_m_word_pfa (VM *vm)
- void dictionary_m_word_traverse (VM *vm)
- void dictionary_m_word_interpret (VM *vm)
- static void dictionary_m_word_find (VM *vm)
- static void dictionary_m_word_tick (VM *vm)
- static void dictionary_m_word_hidden (VM *vm)
- void register_dictionary_manipulation_words (VM *vm)

  Register all dictionary manipulation words with the virtual machine.

Variables

- static cell_t state_variable = 0

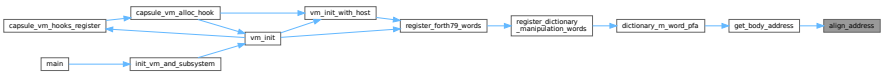
7.357.1 Function Documentation

7.357.1.1 align_address()

```
static uintptr_t align_address (
            uintptr_t addr )  [static]
```

Referenced by get_body_address().

Here is the caller graph for this function:



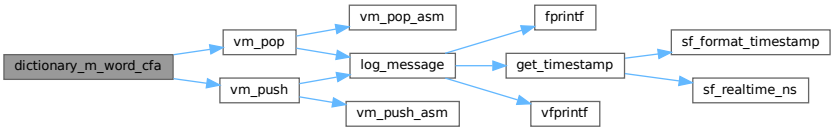
7.357.1.2 dictionary_m_word_cfa()

```
void dictionary_m_word_cfa (
            VM * vm )
```

References VM::dsp, entry, VM::error, vm_pop(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

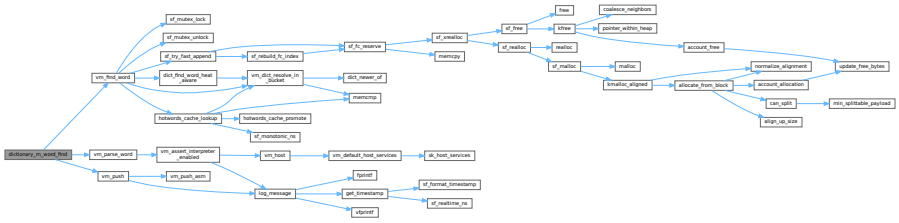
7.357.1.3 dictionary_m_word_find()

```
static void dictionary_m_word_find (
            VM * vm )  [static]
```

References VM::error, vm_find_word(), vm_parse_word(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

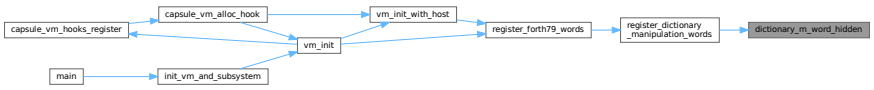
7.357.1.4 dictionary_m_word_hidden()

```
static void dictionary_m_word_hidden (
            VM * vm )  [static]
```

References VM::error, DictEntry::flags, VM::latest, VM::mode, MODE_COMPILE, WORD_HIDDEN, and WORD_SMUDGED.

Referenced by register_dictionary_manipulation_words().

Here is the caller graph for this function:



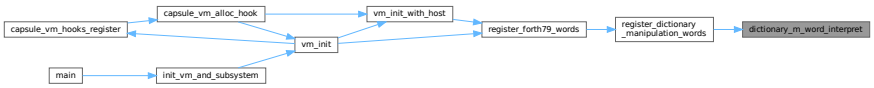
7.357.1.5 dictionary_m_word_interpret()

```
void dictionary_m_word_interpret (
            VM * vm )
```

References VM::mode, MODE_INTERPRET, and state_variable.

Referenced by register_dictionary_manipulation_words().

Here is the caller graph for this function:



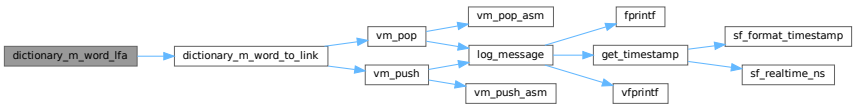
7.357.1.6 dictionary_m_word_lfa()

```
void dictionary_m_word_lfa (
            VM * vm )
```

References dictionary_m_word_to_link().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

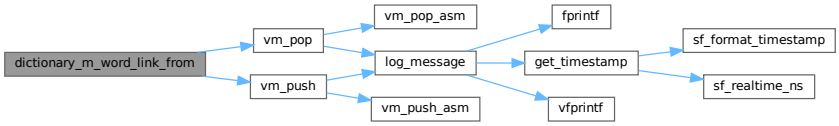
7.357.1.7 dictionary_m_word_link_from()

```
void dictionary_m_word_link_from (
            VM * vm )
```

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

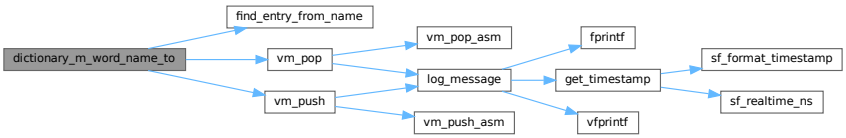
7.357.1.8 dictionary_m_word_name_to()

```
void dictionary_m_word_name_to (
            VM * vm )
```

References VM::dsp, entry, VM::error, find_entry_from_name(), vm_pop(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

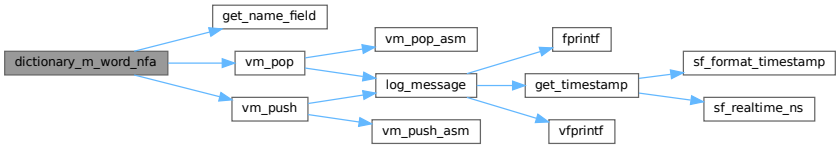
7.357.1.9 dictionary_m_word_nfa()

```
void dictionary_m_word_nfa (
            VM * vm )
```

References VM::dsp, entry, VM::error, get_name_field(), vm_pop(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

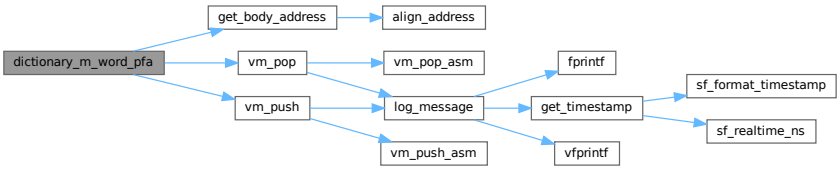
7.357.1.10 dictionary_m_word_pfa()

```
void dictionary_m_word_pfa (
            VM * vm )
```

References VM::dsp, entry, VM::error, get_body_address(), vm_pop(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.357.1.11 dictionary_m_word_smudge()

```
static void dictionary_m_word_smudge (
            VM * vm )  [static]
```

FORTH word SMUDGE - Toggle smudge bit of latest word.

Parameters

vm	Pointer to VM instance @stack (–)

Note

Compile-only word

References VM::error, VM::mode, and MODE_COMPILE.

Referenced by register_dictionary_manipulation_words().

Here is the caller graph for this function:

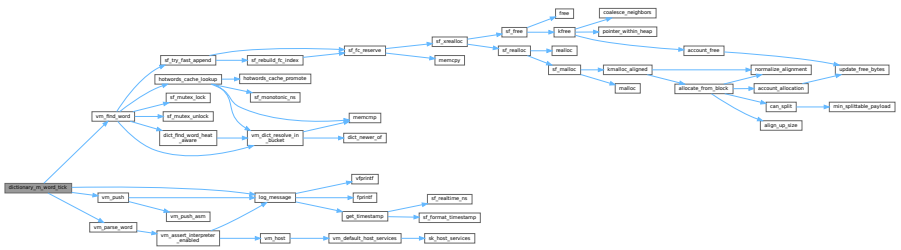
7.357.1.12 dictionary_m_word_tick()

```
static void dictionary_m_word_tick (
            VM * vm )  [static]
```

References VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_find_word(), vm_parse_word(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

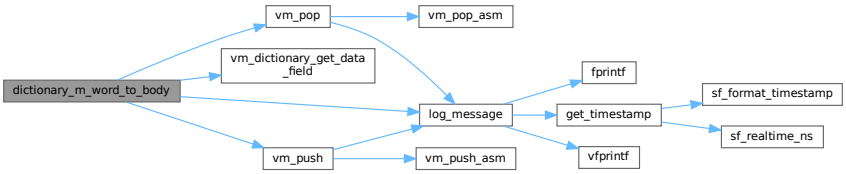
7.357.1.13 dictionary_m_word_to_body()

```
void dictionary_m_word_to_body (
            VM * vm )
```

References VM::dsp, entry, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_dictionary_get_data_field(), vm_pop(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

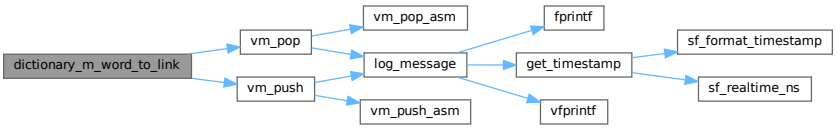
7.357.1.14 dictionary_m_word_to_link()

```
void dictionary_m_word_to_link (
            VM * vm )
```

References VM::dsp, entry, VM::error, vm_pop(), and vm_push().

Referenced by dictionary_m_word_lfa(), and register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

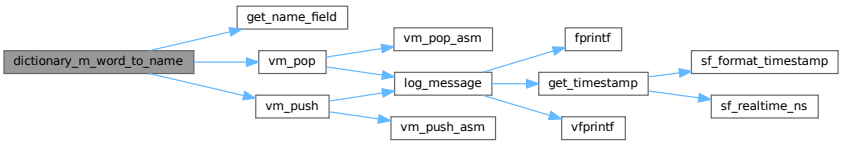
7.357.1.15 dictionary_m_word_to_name()

```
void dictionary_m_word_to_name (
            VM * vm )
```

References VM::dsp, entry, VM::error, get_name_field(), vm_pop(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

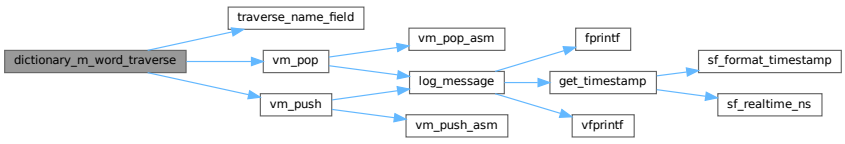
7.357.1.16 dictionary_m_word_traverse()

```
void dictionary_m_word_traverse (
            VM * vm )
```

References VM::dsp, VM::error, traverse_name_field(), vm_pop(), and vm_push().

Referenced by register_dictionary_manipulation_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.357.1.17 find_entry_from_name()

```
static DictEntry * find_entry_from_name (
            VM * vm,
            char * name_field )  [static]
```

References entry, VM::latest, and DictEntry::link.

Referenced by dictionary_m_word_name_to().

Here is the caller graph for this function:

7.357.1.18 get_body_address()

```
static void * get_body_address (
            DictEntry * entry )  [static]
```

References align_address(), and entry.

Referenced by dictionary_m_word_pfa().

Here is the call graph for this function:

Here is the caller graph for this function:

7.357.1.19 get_name_field()

```
static char * get_name_field (
            DictEntry * entry )  [static]
```

References entry.

Referenced by dictionary_m_word_nfa(), and dictionary_m_word_to_name().

Here is the caller graph for this function:

7.357.1.20 register_dictionary_manipulation_words()

```
void register_dictionary_manipulation_words (
            VM * vm )
```

Register all dictionary manipulation words with the virtual machine.

Parameters

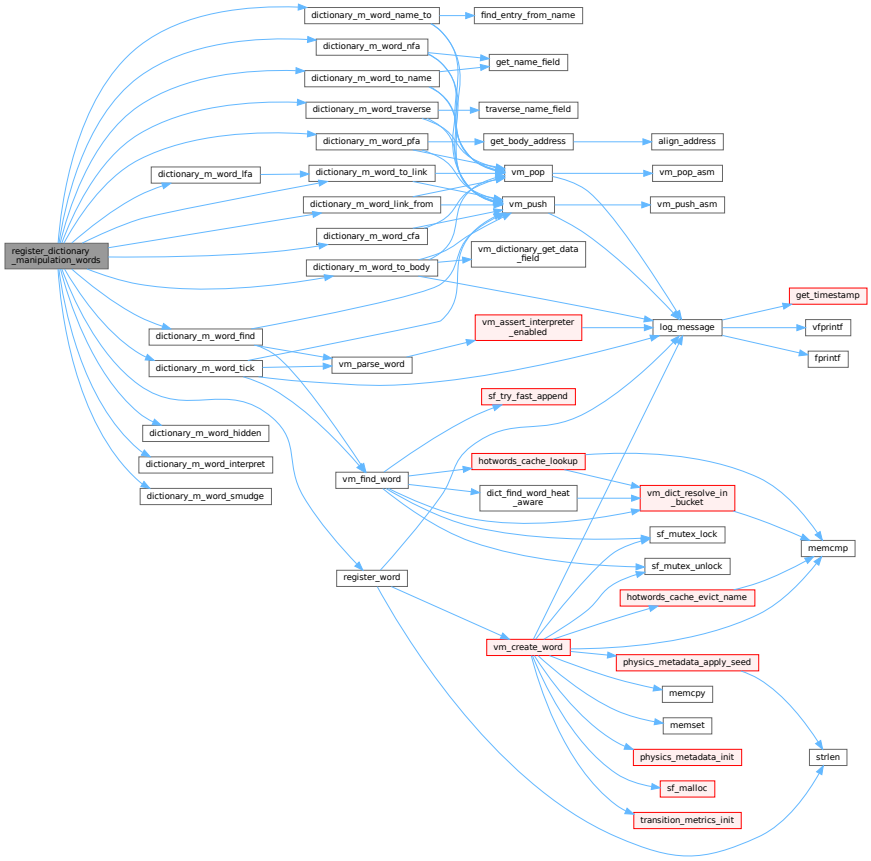
vm	Pointer to the virtual machine instance

This function registers all FORTH-79 standard dictionary manipulation words with the provided virtual machine instance. These words are essential for dictionary management and word definition functionality.

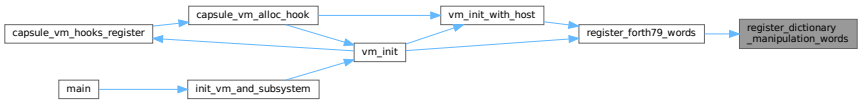
References dictionary_m_word_cfa(), dictionary_m_word_find(), dictionary_m_word_hidden(), dictionary_m_word_interpret(), dictionary_m_word_lfa(), dictionary_m_word_link_from(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_smudge(), dictionary_m_word_tick(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), dictionary_m_word_traverse(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.357.1.21 traverse_name_field()

```
static char * traverse_name_field (
            char * name_addr,
            int direction )  [static]
```

References name_len.

Referenced by dictionary_m_word_traverse().

Here is the caller graph for this function:

7.357.2 Variable Documentation

7.357.2.1 state_variable

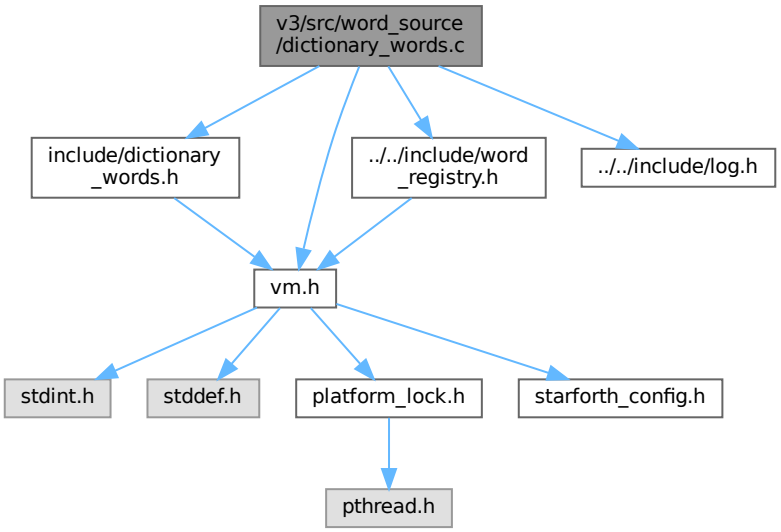
```
cell_t state_variable = 0  [static]
```

Referenced by dictionary_m_word_interpret().

7.358 v3/src/word_source/dictionary_words.c File Reference

```
#include "include/dictionary_words.h"
#include "../../include/word_registry.h"
#include "vm.h"
#include "../../include/log.h"
```

Include dependency graph for dictionary_words.c:



Functions

- void dictionary_word_here (VM *vm)

  HERE (– addr) Returns the dictionary pointer.
- static void dictionary_word_align (VM *vm)
- void dictionary_word_allot (VM *vm)

  ALLOT (n –) Allocate n bytes in dictionary.
- void dictionary_word_comma (VM *vm)

  , (n –) Compile cell into dictionary
- void dictionary_word_c_comma (VM *vm)

  C, (c –) Compile byte into dictionary.
- void dictionary_word_2comma (VM *vm)

2, (d –) Compile double-cell into dictionary

- void dictionary_word_pad (VM *vm)

  PAD (– addr) Get scratch pad buffer address.

- void dictionary_word_sp_fetch (VM *vm)

  SP@ (– sp) Get current stack pointer.

- void dictionary_word_sp_store (VM *vm)

  SP! (sp –) Set stack pointer.

- void dictionary_word_latest (VM *vm)

  LATEST (– addr) Get latest definition address.

- void register_dictionary_words (VM *vm)

  Registers all dictionary and compilation words with the Forth VM.

7.358.1 Function Documentation

7.358.1.1 dictionary_word_2comma()

```
void dictionary_word_2comma (
            VM * vm )
```

2, (d –) Compile double-cell into dictionary

Parameters

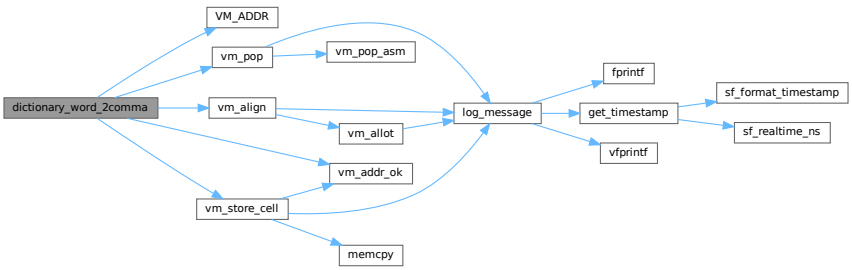
vm	Pointer to the VM structure

Compiles a double cell value into the dictionary at HERE, low cell first

References DICTIONARY_MEMORY_SIZE, VM::dsp, VM::error, VM::here, VM_ADDR(), vm_addr_ok(), vm_align(), vm_pop(), and vm_store_cell().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

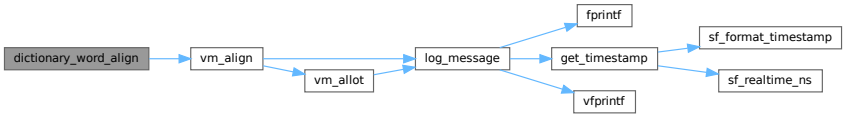
7.358.1.2 dictionary_word_align()

```
static void dictionary_word_align (
            VM * vm )  [static]
```

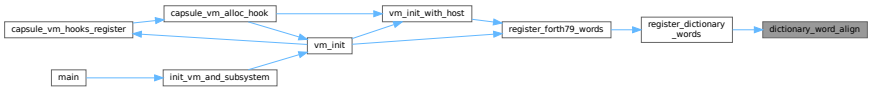
References vm_align().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.358.1.3 dictionary_word_allot()

```
void dictionary_word_allot (
            VM * vm )
```

ALLOT (n –) Allocate n bytes in dictionary.

Parameters

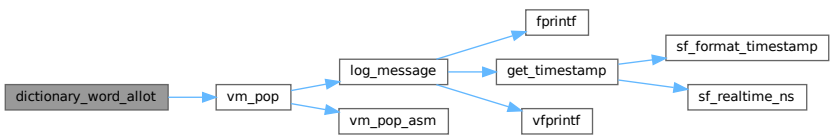
vm	Pointer to the VM structure

Allocates n bytes in the dictionary. n can be negative to deallocate space

References DICTIONARY_MEMORY_SIZE, VM::dsp, VM::error, VM::here, and vm_pop().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.358.1.4 dictionary_word_c_comma()

```
void dictionary_word_c_comma (
            VM * vm )
```

C, (c –) Compile byte into dictionary.

Parameters

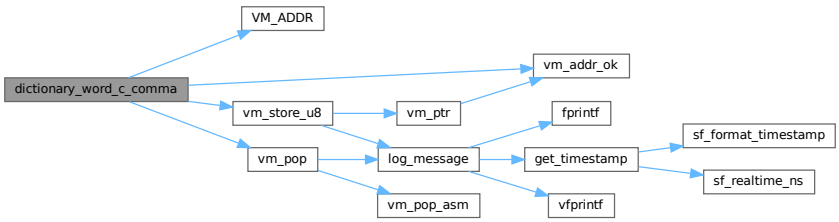
vm	Pointer to the VM structure

Compiles a single byte value into the dictionary at HERE

References DICTIONARY_MEMORY_SIZE, VM::dsp, VM::error, VM::here, VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_store_u8().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.358.1.5 dictionary_word_comma()

```
void dictionary_word_comma (
            VM * vm )
```

, (n –) Compile cell into dictionary

Parameters

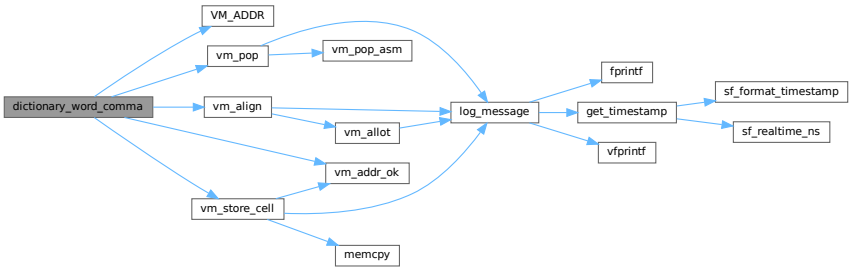
vm	Pointer to the VM structure

Compiles a single cell value into the dictionary at HERE

References DICTIONARY_MEMORY_SIZE, VM::dsp, VM::error, VM::here, VM_ADDR(), vm_addr_ok(), vm_align(), vm_pop(), and vm_store_cell().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.358.1.6 dictionary_word_here()

```
void dictionary_word_here (
            VM * vm )
```

HERE (– addr) Returns the dictionary pointer.

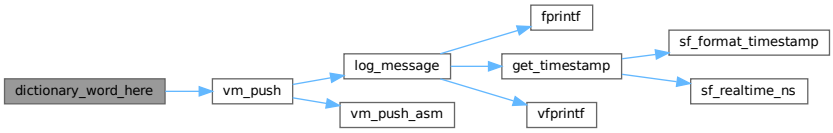
Parameters

vm	Pointer to the VM structure

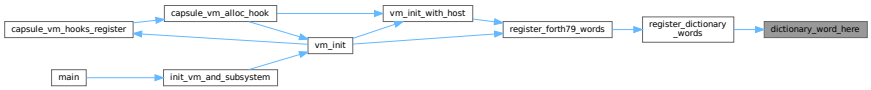
References VM::here, and vm_push().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.358.1.7 dictionary_word_latest()

```
void dictionary_word_latest (
            VM * vm )
```

LATEST (– addr) Get latest definition address.

Parameters

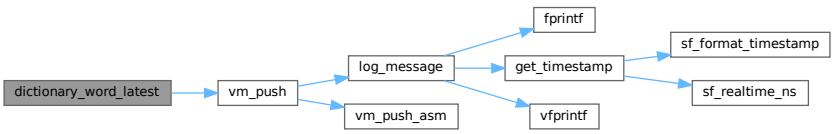
vm	Pointer to the VM structure

Returns VM address near most recent compiled definition (end of dictionary)

References VM::here, and vm_push().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.358.1.8 dictionary_word_pad()

```
void dictionary_word_pad (
            VM * vm )
```

PAD (– addr) Get scratch pad buffer address.

Parameters

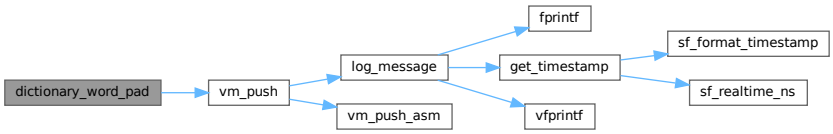
vm	Pointer to the VM structure

Returns VM address of 512-byte temporary text buffer at top of memory

References VM_MEMORY_SIZE, and vm_push().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.358.1.9 dictionary_word_sp_fetch()

```
void dictionary_word_sp_fetch (
            VM * vm )
```

SP@ (– sp) Get current stack pointer.

Parameters

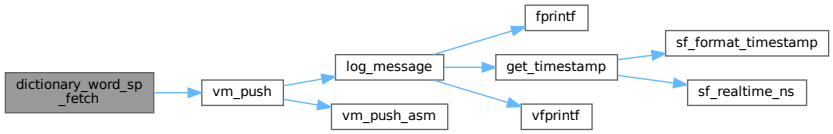
vm	Pointer to the VM structure

Returns the current stack-pointer index (top is 0)

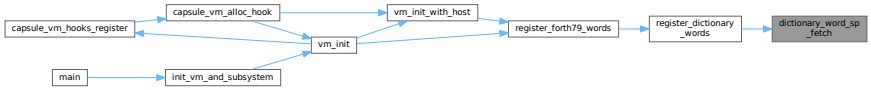
References VM::dsp, and vm_push().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.358.1.10 dictionary_word_sp_store()

```
void dictionary_word_sp_store (
            VM * vm )
```

SP! (sp –) Set stack pointer.

Parameters

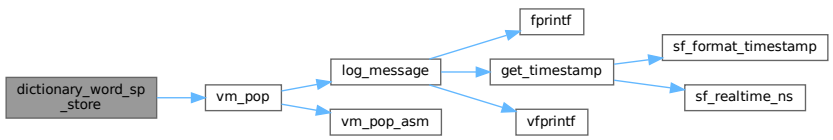
vm	Pointer to the VM structure

Restores the stack-pointer, but never allows growing the stack

References VM::dsp, VM::error, and vm_pop().

Referenced by register_dictionary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.358.1.11 register_dictionary_words()

```
void register_dictionary_words (
            VM * vm )
```

Registers all dictionary and compilation words with the Forth VM.

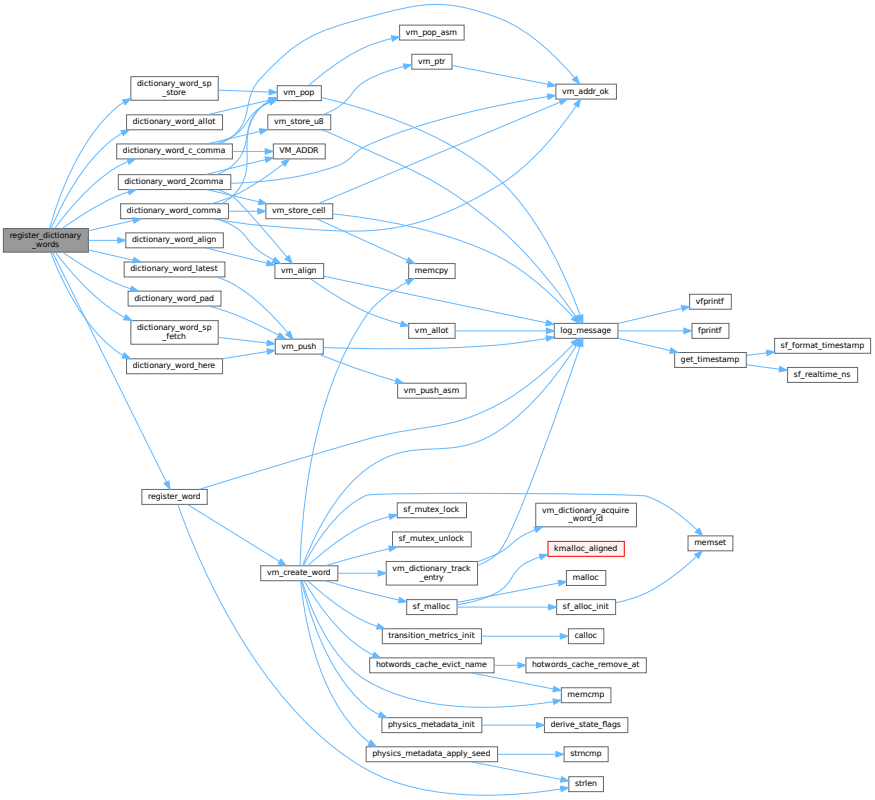
Parameters

vm	Pointer to the Forth virtual machine instance

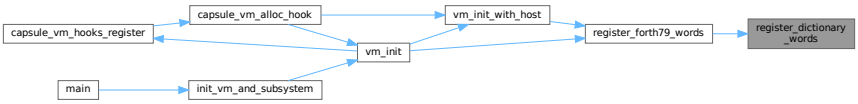
References dictionary_word_2comma(), dictionary_word_align(), dictionary_word_allot(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_here(), dictionary_word_latest(), dictionary_word_pad(), dictionary_word_sp_fetch(), dictionary_word_sp_store(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



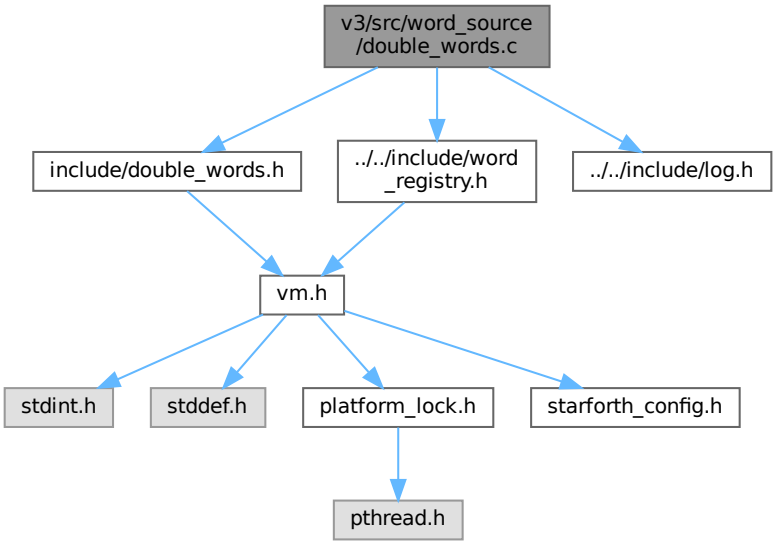
Here is the caller graph for this function:



7.359 v3/src/word_source/double_words.c File Reference

```
#include "include/double_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
```

Include dependency graph for double_words.c:



Functions

- void double_word_s_to_d (VM *vm)

  Convert single number to double (n – d)
- void double_word_d_plus (VM *vm)
- void double_word_d_minus (VM *vm)
- void double_word_dnegate (VM *vm)
- void double_word_dabs (VM *vm)
- static int d_compare (cell_t d1h, cell_t d1l, cell_t d2h, cell_t d2l)

  Compare two double numbers.
- void double_word_dmax (VM *vm)
- void double_word_dmin (VM *vm)
- void double_word_d_less (VM *vm)

- void double_word_d_equals (VM *vm)
- void double_word_2drop (VM *vm)
- void double_word_2dup (VM *vm)
- void double_word_2swap (VM *vm)
- void double_word_2over (VM *vm)
- void double_word_2rot (VM *vm)
- void double_word_2to_r (VM *vm)
- void double_word_2r_from (VM *vm)
- void double_word_2r_fetch (VM *vm)
- void double_word_d_zero_equals (VM *vm)
- void double_word_d_zero_less (VM *vm)
- void double_word_d_two_star (VM *vm)
- void double_word_d_two_slash (VM *vm)
- void register_double_words (VM *vm)

  Register all double number words.

7.359.1 Function Documentation

7.359.1.1 d_compare()

```
static int d_compare (
            cell_t d1h,
            cell_t d1l,
            cell_t d2h,
            cell_t d2l )  [static]
```

Compare two double numbers.

Parameters

d1h	High cell of first double number
d1l	Low cell of first double number
d2h	High cell of second double number
d2l	Low cell of second double number

Returns

int -1 if d1<d2, 0 if d1=d2, 1 if d1>d2

Referenced by double_word_d_equals(), double_word_d_less(), double_word_dmax(), and double_word_dmin().

Here is the caller graph for this function:

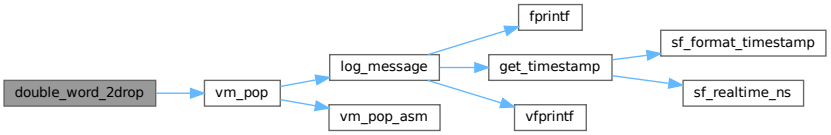
7.359.1.2 double_word_2drop()

```
void double_word_2drop (
            VM * vm )
```

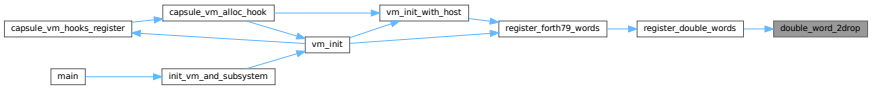
References VM::dsp, VM::error, and vm_pop().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



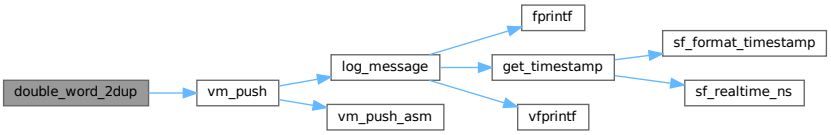
7.359.1.3 double_word_2dup()

```
void double_word_2dup (
            VM * vm )
```

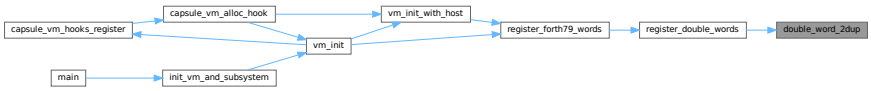
References VM::data_stack, VM::dsp, VM::error, and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



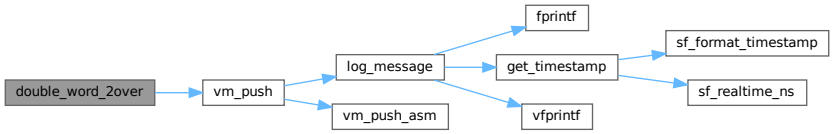
7.359.1.4 double_word_2over()

```
void double_word_2over (
            VM * vm )
```

References VM::data_stack, VM::dsp, VM::error, and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

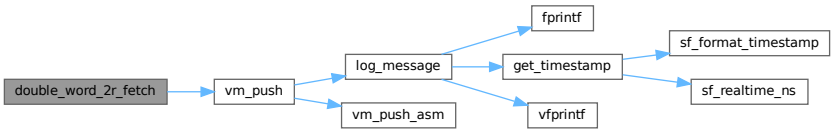
7.359.1.5 double_word_2r_fetch()

```
void double_word_2r_fetch (
            VM * vm )
```

References VM::ecw_nesting, VM::error, VM::return_stack, VM::rsp, and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

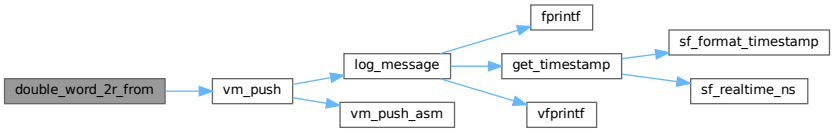
7.359.1.6 double_word_2r_from()

```
void double_word_2r_from (
            VM * vm )
```

References VM::ecw_nesting, VM::error, VM::return_stack, VM::rsp, and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

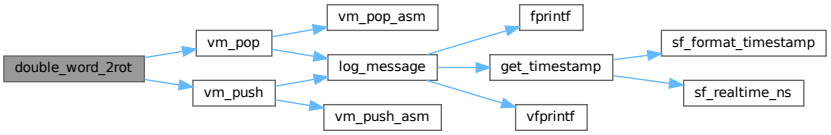
7.359.1.7 double_word_2rot()

```
void double_word_2rot (
            VM * vm )
```

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

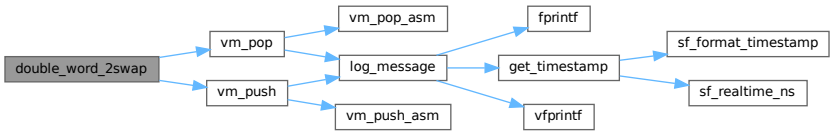
7.359.1.8 double_word_2swap()

```
void double_word_2swap (
            VM * vm )
```

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

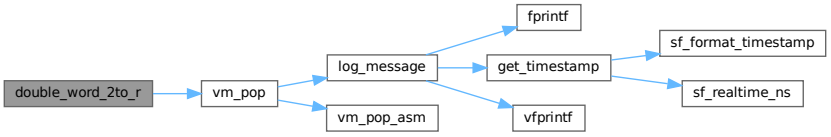
7.359.1.9 double_word_2to_r()

```
void double_word_2to_r (
            VM * vm )
```

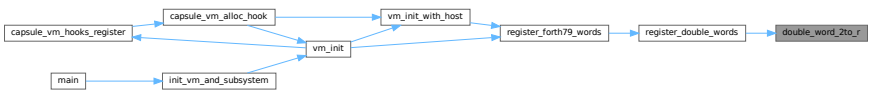
References VM::dsp, VM::ecw_nesting, VM::error, VM::return_stack, VM::rsp, STACK_SIZE, and vm_pop().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



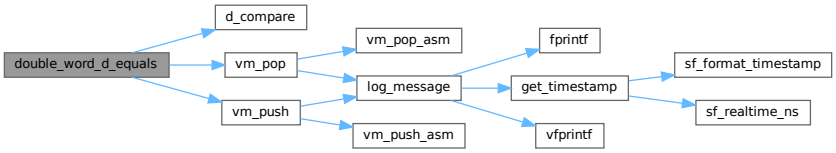
7.359.1.10 double_word_d_equals()

```
void double_word_d_equals (
            VM * vm )
```

References d_compare(), VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

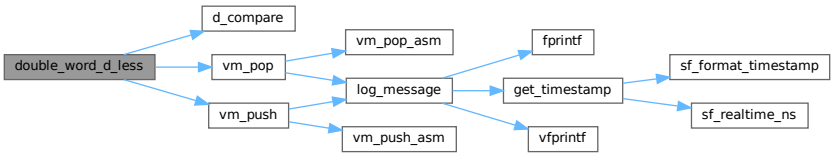
7.359.1.11 double_word_d_less()

```
void double_word_d_less (
            VM * vm )
```

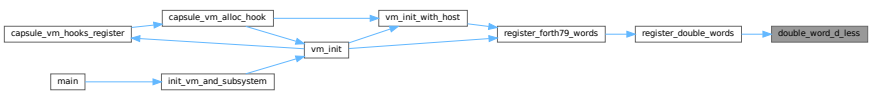
References d_compare(), VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



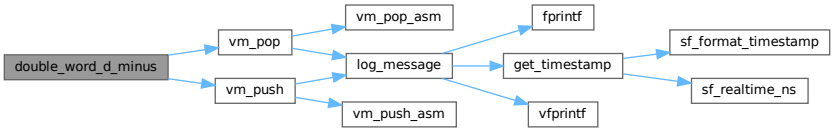
7.359.1.12 double_word_d_minus()

```
void double_word_d_minus (
            VM * vm )
```

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

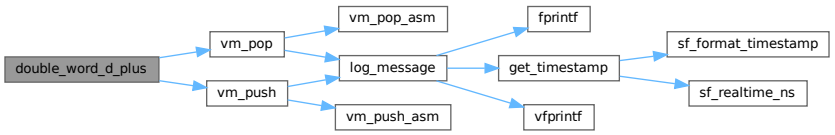
7.359.1.13 double_word_d_plus()

```
void double_word_d_plus (
            VM * vm )
```

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

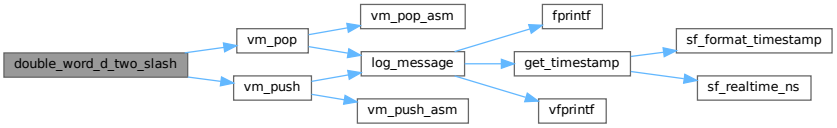
7.359.1.14 double_word_d_two_slash()

```
void double_word_d_two_slash (
            VM * vm )
```

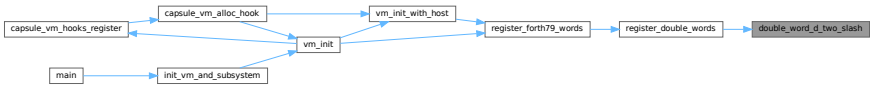
References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



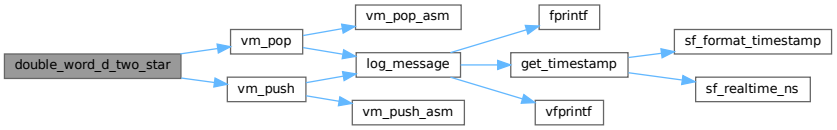
7.359.1.15 double_word_d_two_star()

```
void double_word_d_two_star (
            VM * vm )
```

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

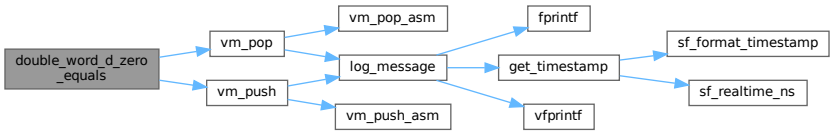
7.359.1.16 double_word_d_zero_equals()

```
void double_word_d_zero_equals (
            VM * vm )
```

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

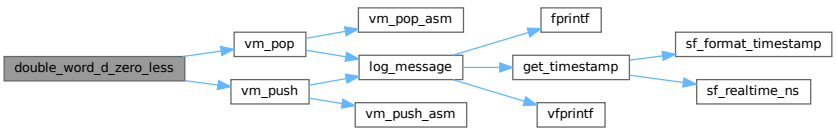
7.359.1.17 double_word_d_zero_less()

```
void double_word_d_zero_less (
            VM * vm )
```

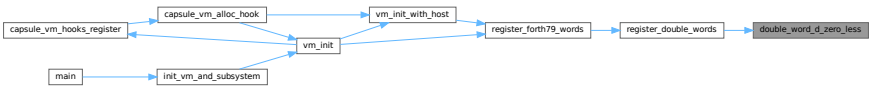
References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



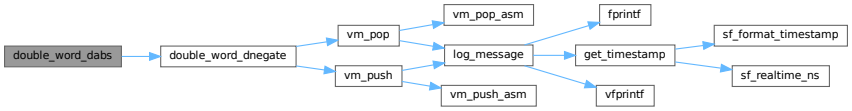
7.359.1.18 double_word_dabs()

```
void double_word_dabs (
            VM * vm )
```

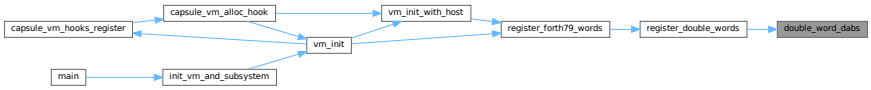
References VM::data_stack, double_word_dnegate(), VM::dsp, and VM::error.

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



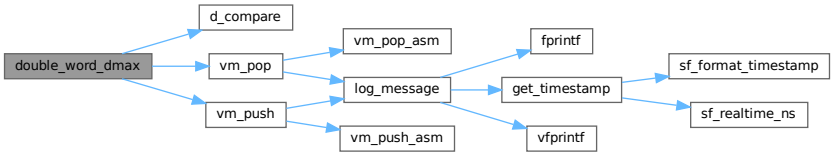
7.359.1.19 double_word_dmax()

```
void double_word_dmax (
            VM * vm )
```

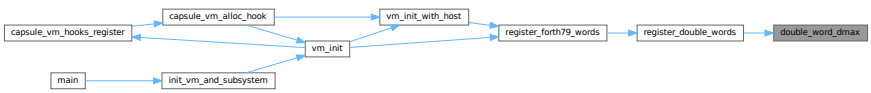
References d_compare(), VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



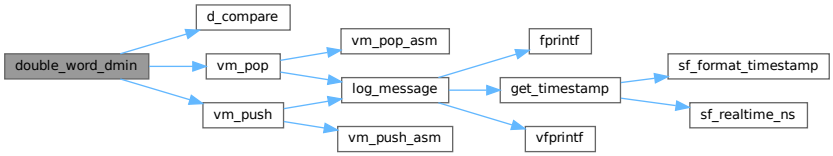
7.359.1.20 double_word_dmin()

```
void double_word_dmin (
            VM * vm )
```

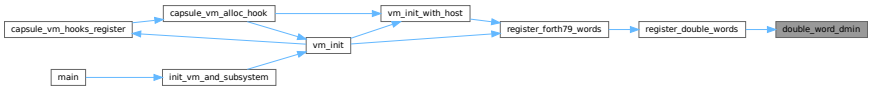
References d_compare(), VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:



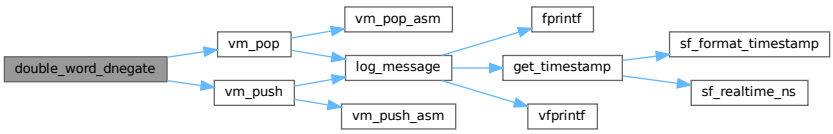
7.359.1.21 double_word_dnegate()

```
void double_word_dnegate (
            VM * vm )
```

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by double_word_dabs(), and register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.359.1.22 double_word_s_to_d()

```
void double_word_s_to_d (
            VM * vm )
```

Convert single number to double (n – d)

Converts a single-precision number to double-precision format. Result is pushed as (dlow dhigh) with high cell on top.

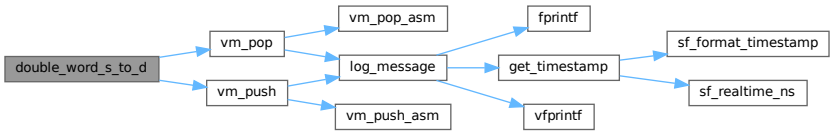
Parameters

vm	Pointer to the virtual machine state

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_double_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.359.1.23 register_double_words()

```
void register_double_words (
            VM * vm )
```

Register all double number words.

Registers all double-precision number operations with the virtual machine.

Parameters

vm	Pointer to the virtual machine state

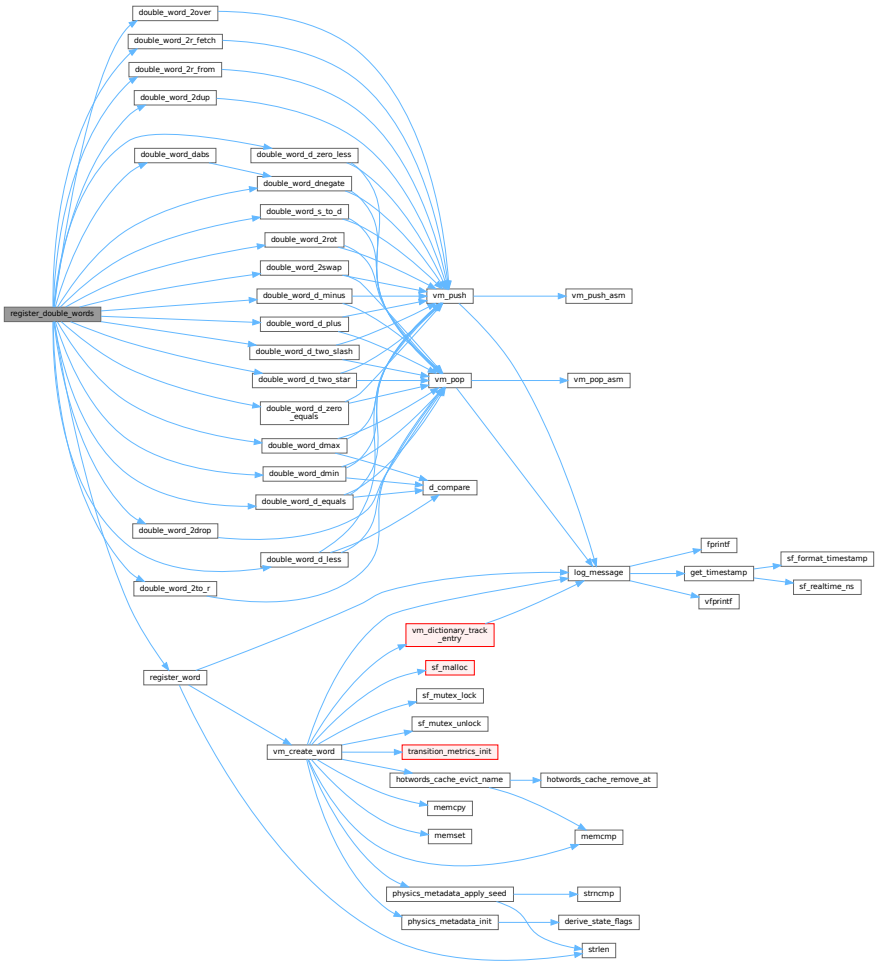
Note

This function must be called during VM initialization.

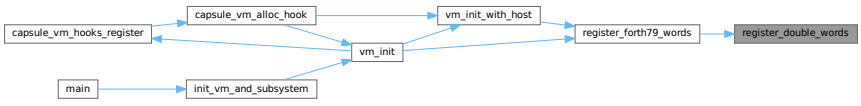
References double_word_2drop(), double_word_2dup(), double_word_2over(), double_word_2r_fetch(), double_word_2r_from(), double_word_2rot(), double_word_2swap(), double_word_2to_r(), double_word_d_equals(), double_word_d_less(), double_word_d_minus(), double_word_d_plus(), double_word_d_two_slash(), double_word_d_two_star(), double_word_d_zero_equals(), double_word_d_zero_less(), double_word_dabs(), double_word_dmax(), double_word_dmin(), double_word_dnegate(), double_word_s_to_d(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



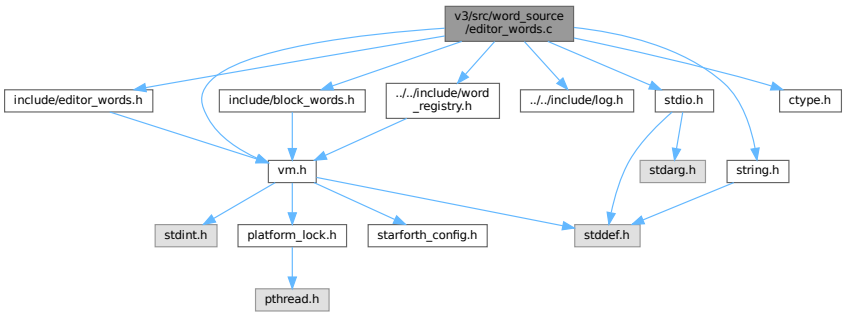
Here is the caller graph for this function:



7.360 v3/src/word_source/editor_words.c File Reference

```
#include "include/editor_words.h"
#include "include/block_words.h"
#include "../../include/vm.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>
```

Include dependency graph for editor_words.c:



Functions

- static cell_t current_scr (VM *vm)

  Get current screen number from VM.
- static int line_ptr (VM *vm, cell_t scr, cell_t line, unsigned char **out)

  Get pointer to specific line in a screen.
- static void print_line_64 (const unsigned char *p)

  Print 64-character line with character filtering.
- static void editor_word_l (VM *vm)
- static void editor_word_s (VM *vm)
- static void editor_word_show (VM *vm)
- static void editor_word_edit (VM *vm)
- void register_editor_words (VM *vm)

  Register all editor words with the Forth virtual machine.

7.360.1 Function Documentation

7.360.1.1 current_scr()

```
static cell_t current_scr (
            VM * vm )  [inline], [static]
```

Get current screen number from VM.

Parameters

vm	Pointer to VM instance

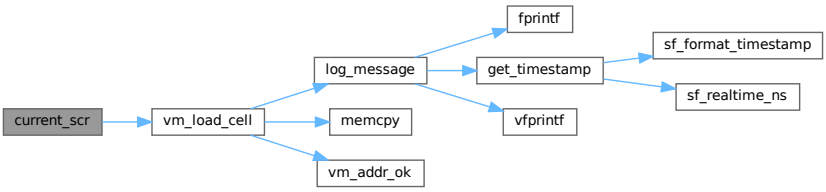
Returns

Current screen number

References VM::scr_addr, and vm_load_cell().

Referenced by block_word_next_block(), editor_word_l(), editor_word_s(), and editor_word_show().

Here is the call graph for this function:



Here is the caller graph for this function:

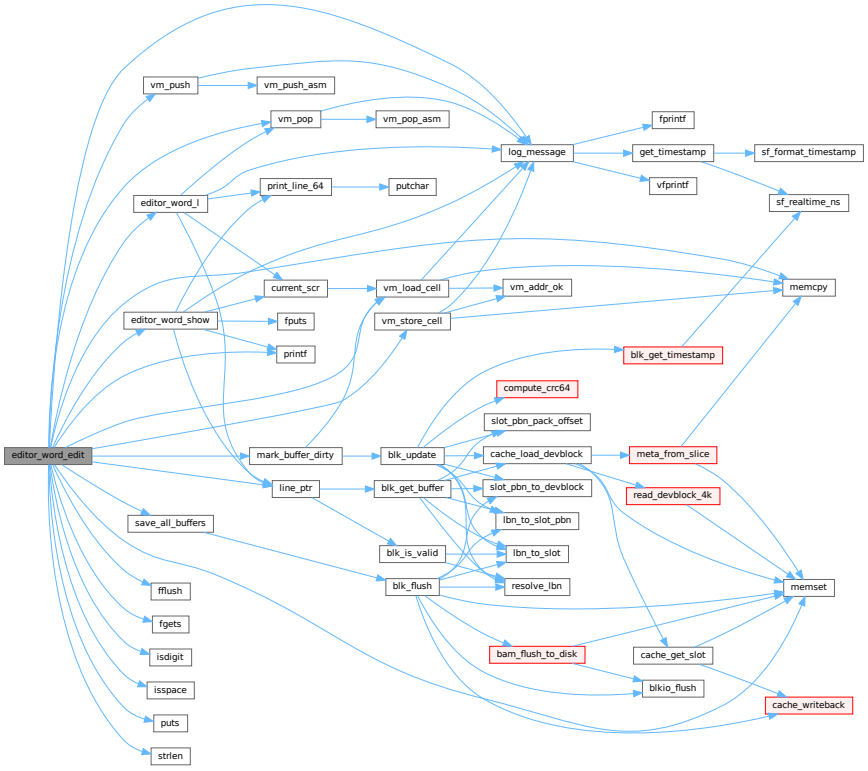
7.360.1.2 editor_word_edit()

```
static void editor_word_edit (
            VM * vm )  [static]
```

References VM::dsp, editor_word_l(), editor_word_show(), VM::error, fflush(), fgets(), isdigit(), isspace(), line_ptr(), LOG_ERROR, log_message(), mark_buffer_dirty(), MAX_BLOCKS, memcpy(), memset(), printf(), puts(), save_all_buffers(), VM::scr_addr, stdin, stdout, strlen(), vm_load_cell(), vm_pop(), vm_push(), and vm_store_cell().

Referenced by register_editor_words().

Here is the call graph for this function:



Here is the caller graph for this function:

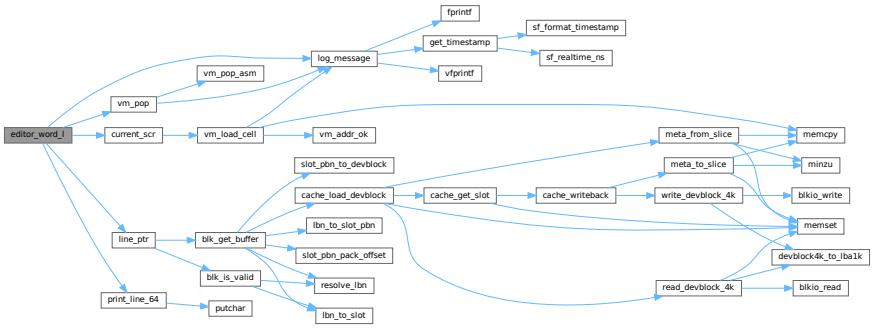
7.360.1.3 editor_word_l()

```
static void editor_word_l (
            VM * vm )  [static]
```

References current_scr(), VM::dsp, VM::error, line_ptr(), LOG_ERROR, log_message(), MAX_BLOCKS, print_line_64(), and vm_pop().

Referenced by editor_word_edit(), and register_editor_words().

Here is the call graph for this function:



Here is the caller graph for this function:

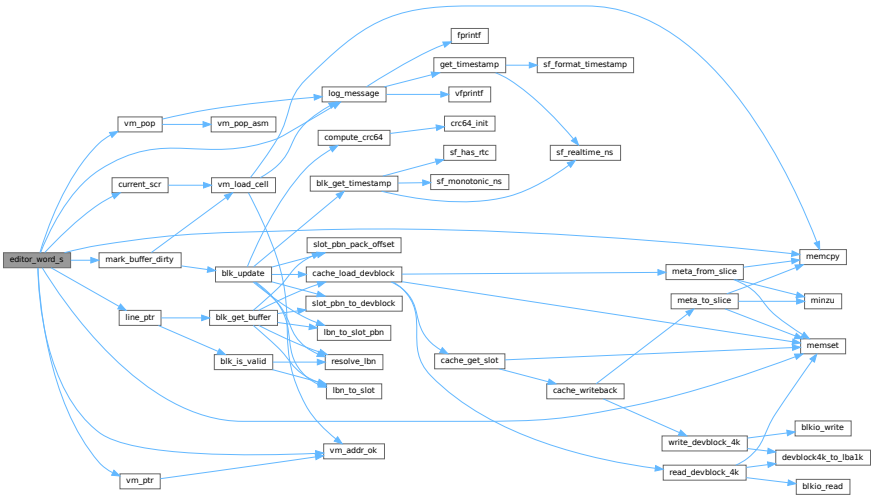
7.360.1.4 editor_word_s()

```
static void editor_word_s (
            VM * vm )  [static]
```

References current_scr(), VM::dsp, VM::error, line_ptr(), LOG_ERROR, log_message(), mark_buffer_dirty(), MAX_BLOCKS, memcpy(), memset(), vm_addr_ok(), vm_pop(), and vm_ptr().

Referenced by register_editor_words().

Here is the call graph for this function:



Here is the caller graph for this function:

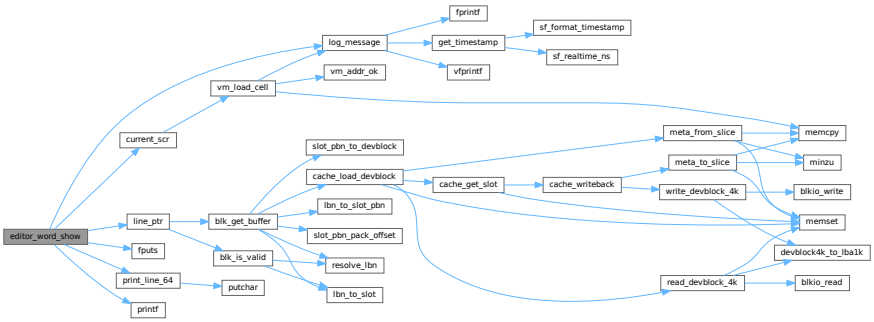
7.360.1.5 editor_word_show()

```
static void editor_word_show (
            VM * vm )  [static]
```

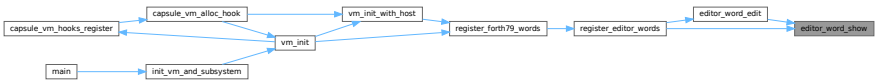
References current_scr(), VM::error, fputs(), line_ptr(), LOG_ERROR, log_message(), MAX_BLOCKS, print_line_64(), printf(), and stdout.

Referenced by editor_word_edit(), and register_editor_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.360.1.6 line_ptr()

```
static int line_ptr (
            VM * vm,
            cell_t scr,
            cell_t line,
            unsigned char ** out )  [static]
```

Get pointer to specific line in a screen.

Parameters

vm	Pointer to VM instance
scr	Screen number (1-based)
line	Line number (0-15)
out	Output parameter for line pointer

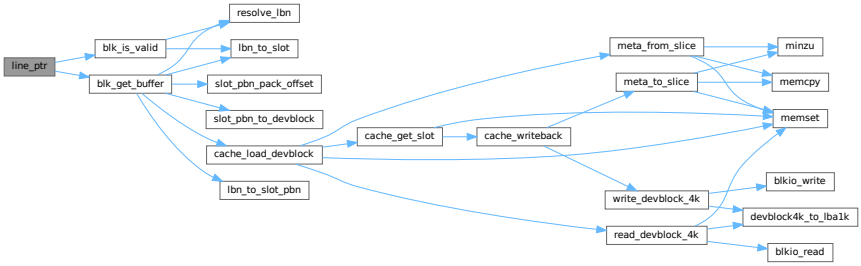
Returns

1 if successful, 0 if parameters invalid

References blk_get_buffer(), blk_is_valid(), and MAX_BLOCKS.

Referenced by editor_word_edit(), editor_word_l(), editor_word_s(), and editor_word_show().

Here is the call graph for this function:



Here is the caller graph for this function:

7.360.1.7 print_line_64()

```
static void print_line_64 (
            const unsigned char * p )  [static]
```

Print 64-character line with character filtering.

Parameters

p	Pointer to 64-byte line buffer

References putchar().

Referenced by editor_word_l(), and editor_word_show().

Here is the call graph for this function:

Here is the caller graph for this function:

7.360.1.8 register_editor_words()

```
void register_editor_words (
            VM * vm )
```

Register all editor words with the Forth virtual machine.

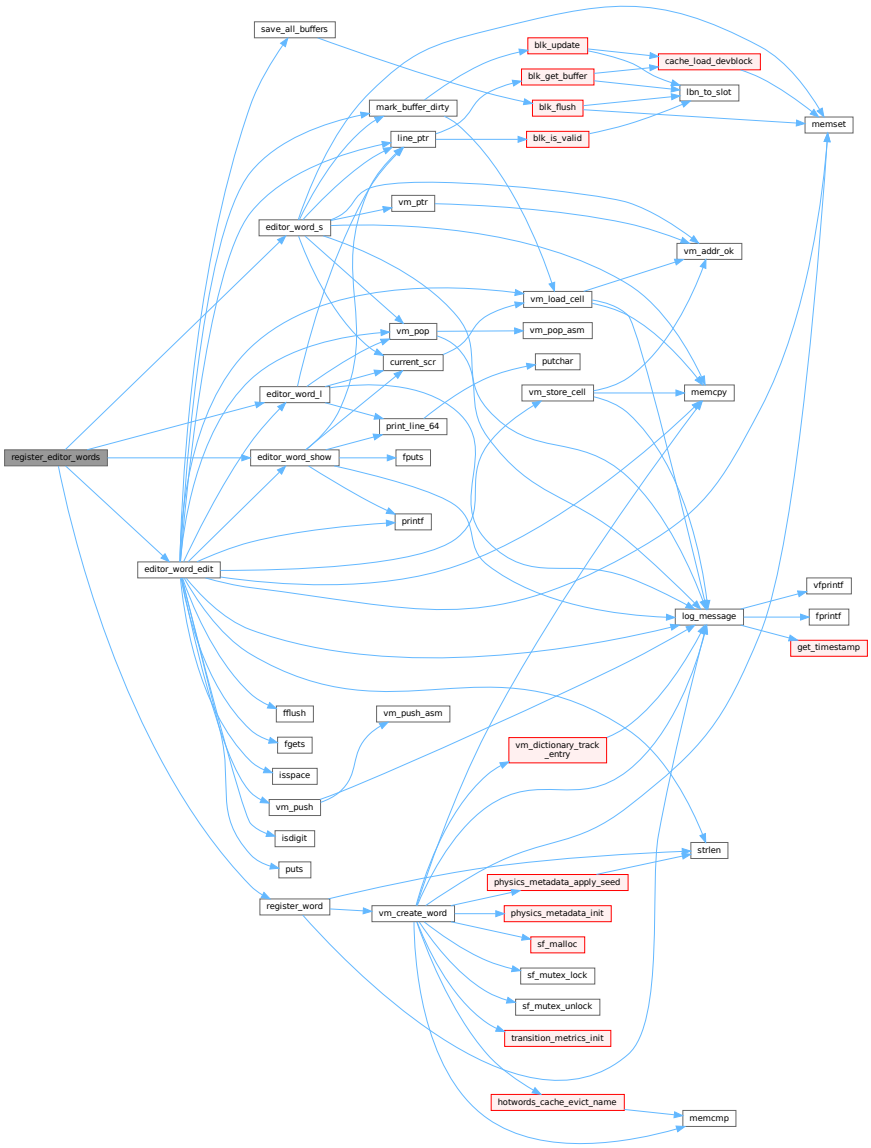
Parameters

vm	Pointer to the Forth virtual machine instance

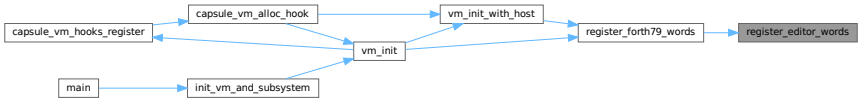
References editor_word_edit(), editor_word_l(), editor_word_s(), editor_word_show(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

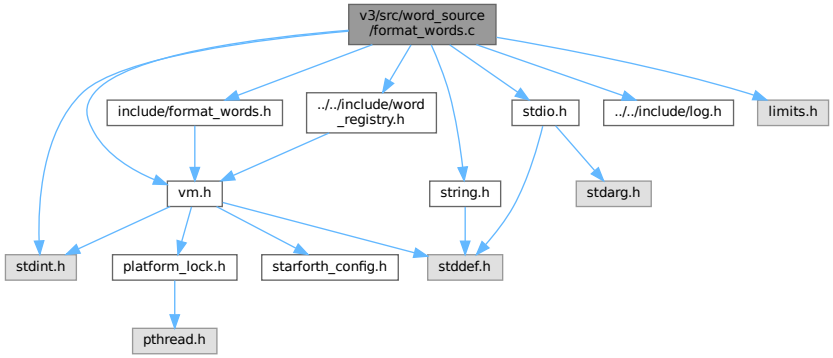


7.361 v3/src/word_source/format_words.c File Reference

```
#include "include/format_words.h"
#include "../../include/word_registry.h"
```

```
#include "../../include/log.h"
#include "../../include/vm.h"
#include <stdint.h>
#include <limits.h>
#include <string.h>
#include <stdio.h>
```

Include dependency graph for format_words.c:



Macros

- #define PN_CAP 64
- #define PN_BUF(vm) ((unsigned char ∗)((vm)->memory + (vm)->hold_addr))
- #define CELL_BITS ((int)(sizeof(cell_t) ∗ CHAR_BIT))
- #define CHUNK_BITS 16
- #define CHUNK_MASK ((uint64_t)(UINT64_C(1) << CHUNK_BITS) - 1)

Functions

- static unsigned current_base (VM ∗vm)

  Returns current numeric conversion base, defaulting to 10 if invalid.
- static char digit_for (unsigned v)

  Converts value 0-35 to character '0'-'9' or 'A'-'Z'.
- static void div_ud_by_base (cell_t dhigh_in, cell_t dlow_in, unsigned base, cell_t ∗qhigh_out, cell_t ∗qlow↩ _out, unsigned ∗rem_out)

  Divides unsigned double-cell number by base using portable C99 code.
- static void print_number_formatted (VM ∗vm, cell_t n, int width, int is_unsigned)

  Prints number with optional width field and sign handling.
- void format_word_base (VM ∗vm)
- void format_word_decimal (VM ∗vm)
- void format_word_hex (VM ∗vm)
- void format_word_octal (VM ∗vm)
- void format_word_begin_conversion (VM ∗vm)
- void format_word_hold (VM ∗vm)
- void format_word_sign (VM ∗vm)
- void format_word_hash (VM ∗vm)
- void format_word_hash_s (VM ∗vm)
- void format_word_end_conversion (VM ∗vm)

- void format_word_dot (VM *vm)
- void format_word_dot_r (VM *vm)
- void format_word_u_dot (VM *vm)
- void format_word_u_dot_r (VM *vm)
- void format_word_d_dot (VM *vm)
- void format_word_d_dot_r (VM *vm)
- void format_word_dot_s (VM *vm)
- void format_word_question (VM *vm)
- void format_word_dump (VM *vm)
- void register_format_words (VM *vm)

  Registers all formatting and numeric conversion words with the VM.

7.361.1 Macro Definition Documentation

7.361.1.1 CELL_BITS

```
#define CELL_BITS ((int)(sizeof(cell_t) * CHAR_BIT))
```

7.361.1.2 CHUNK_BITS

```
#define CHUNK_BITS 16
```

7.361.1.3 CHUNK_MASK

```
#define CHUNK_MASK ((uint64_t)(UINT64_C(1) << CHUNK_BITS) - 1)
```

7.361.1.4 PN_BUF

```
#define PN_BUF(
              vm ) ((unsigned char *)((vm)->memory + (vm)->hold_addr))
```

7.361.1.5 PN_CAP

```
#define PN_CAP 64
```

7.361.2 Function Documentation

7.361.2.1 current_base()

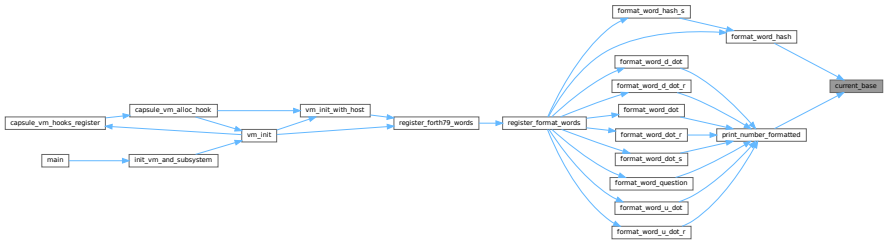
```
static unsigned current_base (
            VM * vm )  [static]
```

Returns current numeric conversion base, defaulting to 10 if invalid.

References VM::base.

Referenced by format_word_hash(), and print_number_formatted().

Here is the caller graph for this function:



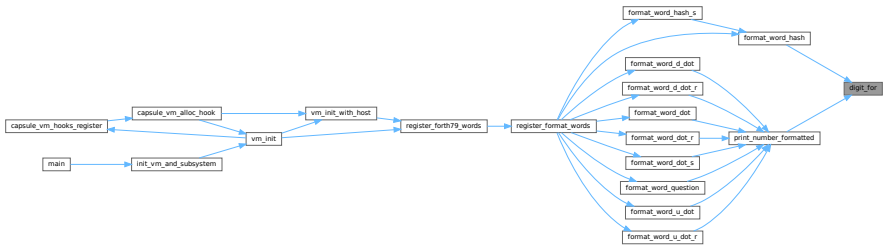
7.361.2.2 digit_for()

```
static char digit_for (
            unsigned v )  [static]
```

Converts value 0-35 to character '0'-'9' or 'A'-'Z'.

Referenced by format_word_hash(), and print_number_formatted().

Here is the caller graph for this function:



7.361.2.3 div_ud_by_base()

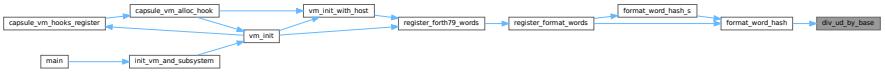
```
static void div_ud_by_base (
            cell_t dhigh_in,
            cell_t dlow_in,
            unsigned base,
            cell_t * qhigh_out,
            cell_t * qlow_out,
            unsigned * rem_out )  [static]
```

Divides unsigned double-cell number by base using portable C99 code.

References base, CELL_BITS, CHUNK_BITS, and CHUNK_MASK.

Referenced by format_word_hash().

Here is the caller graph for this function:



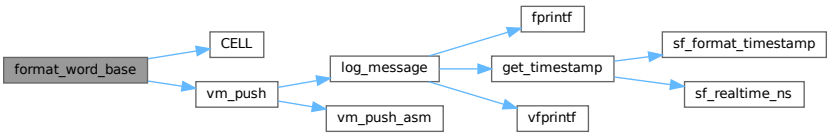
7.361.2.4 format_word_base()

```
void format_word_base (
            VM * vm )
```

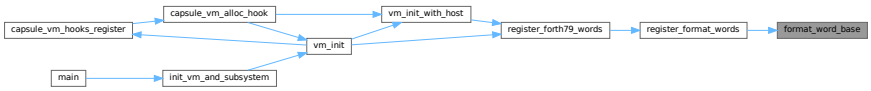
References VM::base_addr, CELL(), and vm_push().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.361.2.5 format_word_begin_conversion()

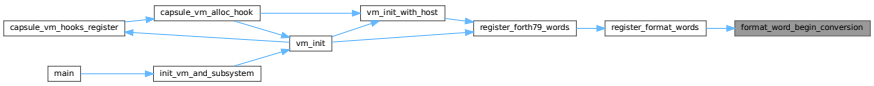
```
void format_word_begin_conversion (
            VM * vm )
```

References VM::hold_pos, memset(), PN_BUF, and PN_CAP.

Referenced by register_format_words().

Here is the call graph for this function:

Here is the caller graph for this function:



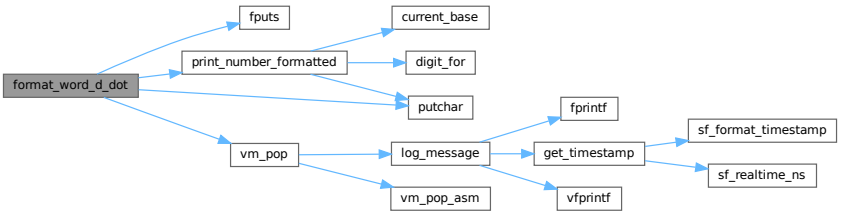
7.361.2.6 format_word_d_dot()

```
void format_word_d_dot (
            VM * vm )
```

References VM::dsp, VM::error, fputs(), print_number_formatted(), putchar(), stdout, and vm_pop().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

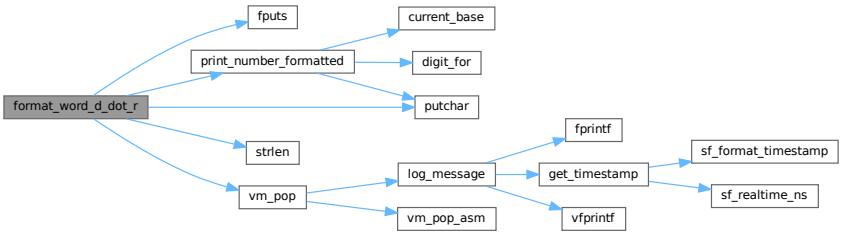
7.361.2.7 format_word_d_dot_r()

```
void format_word_d_dot_r (
            VM * vm )
```

References VM::dsp, VM::error, fputs(), print_number_formatted(), putchar(), stdout, strlen(), and vm_pop().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

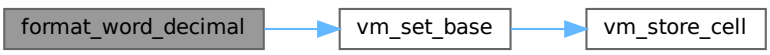
7.361.2.8 format_word_decimal()

```
void format_word_decimal (
            VM * vm )
```

References vm_set_base().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

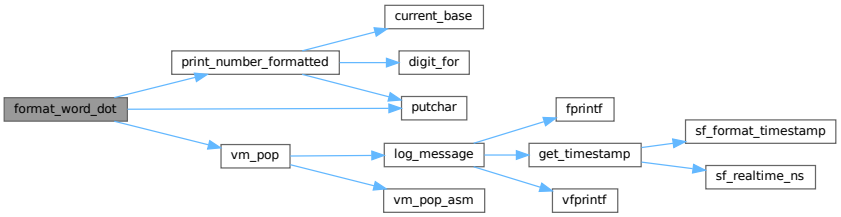
7.361.2.9 format_word_dot()

```
void format_word_dot (
            VM * vm )
```

References VM::dsp, VM::error, print_number_formatted(), putchar(), and vm_pop().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

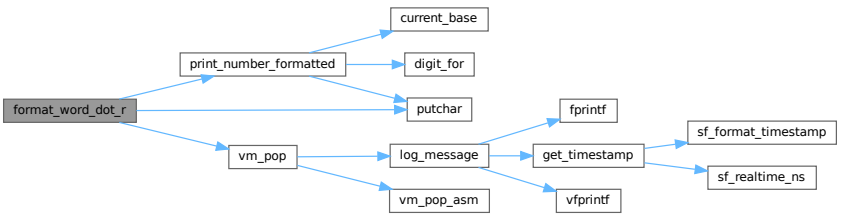
7.361.2.10 format_word_dot_r()

```
void format_word_dot_r (
            VM * vm )
```

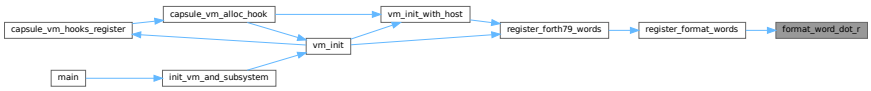
References VM::dsp, VM::error, print_number_formatted(), putchar(), and vm_pop().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:



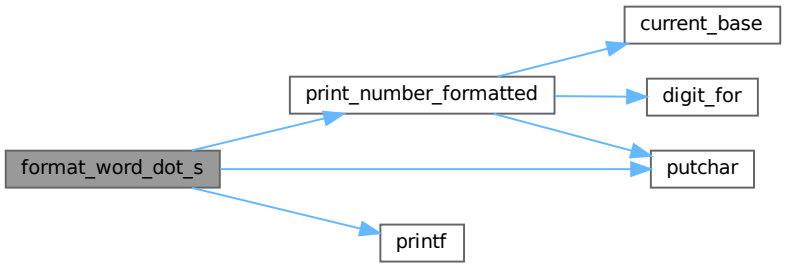
7.361.2.11 format_word_dot_s()

```
void format_word_dot_s (
            VM * vm )
```

References VM::data_stack, VM::dsp, print_number_formatted(), printf(), and putchar().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

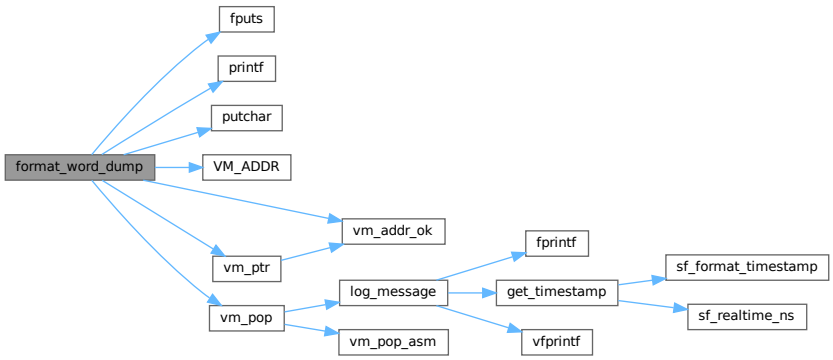
7.361.2.12 format_word_dump()

```
void format_word_dump (
            VM * vm )
```

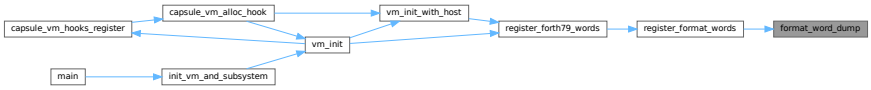
References VM::dsp, VM::error, fputs(), printf(), putchar(), stdout, VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_ptr().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:



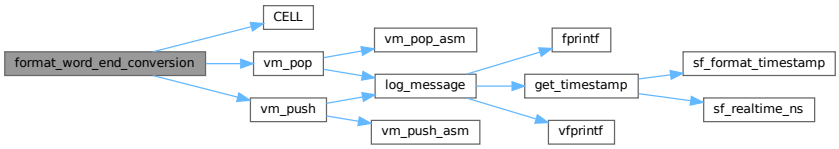
7.361.2.13 format_word_end_conversion()

```
void format_word_end_conversion (
            VM * vm )
```

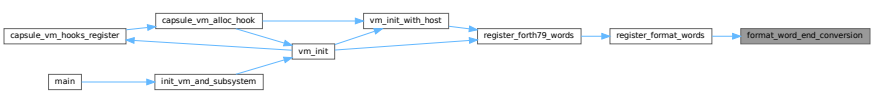
References CELL(), VM::dsp, VM::hold_addr, VM::hold_pos, vm_pop(), and vm_push().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:



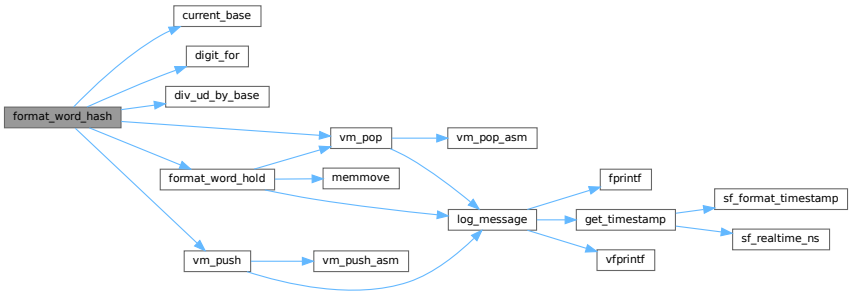
7.361.2.14 format_word_hash()

```
void format_word_hash (
            VM * vm )
```

References base, current_base(), digit_for(), div_ud_by_base(), VM::dsp, VM::error, format_word_hold(), vm_pop(), and vm_push().

Referenced by format_word_hash_s(), and register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

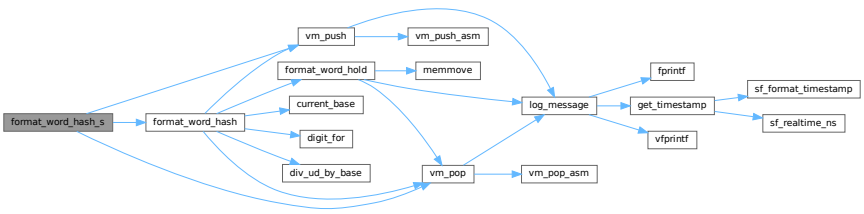
7.361.2.15 format_word_hash_s()

```
void format_word_hash_s (
            VM * vm )
```

References VM::data_stack, VM::dsp, VM::error, format_word_hash(), vm_pop(), and vm_push().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

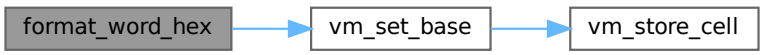
7.361.2.16 format_word_hex()

```
void format_word_hex (
            VM * vm )
```

References vm_set_base().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

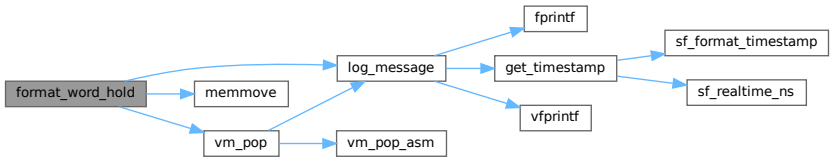
7.361.2.17 format_word_hold()

```
void format_word_hold (
            VM * vm )
```

References VM::dsp, VM::error, VM::hold_pos, LOG_ERROR, log_message(), memmove(), PN_BUF, PN_CAP, and vm_pop().

Referenced by format_word_hash(), format_word_sign(), and register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

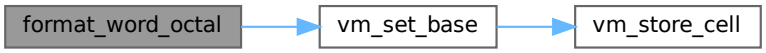
7.361.2.18 format_word_octal()

```
void format_word_octal (
            VM * vm )
```

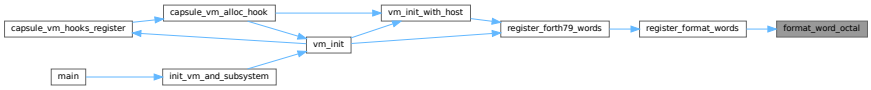
References vm_set_base().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:



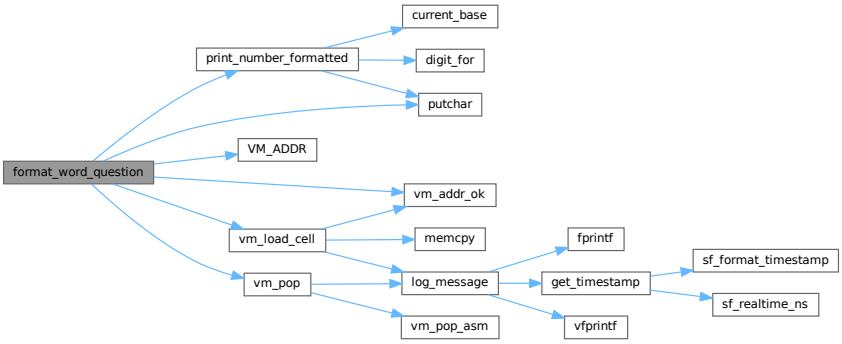
7.361.2.19 format_word_question()

```
void format_word_question (
            VM * vm )
```

References VM::dsp, VM::error, print_number_formatted(), putchar(), VM_ADDR(), vm_addr_ok(), vm_load_cell(), and vm_pop().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

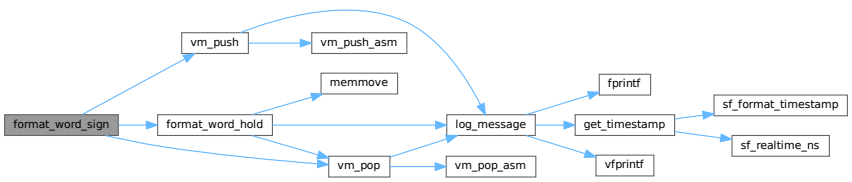
7.361.2.20 format_word_sign()

```
void format_word_sign (
            VM * vm )
```

References VM::dsp, VM::error, format_word_hold(), vm_pop(), and vm_push().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

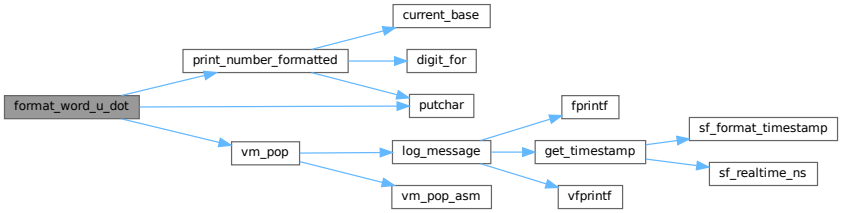
7.361.2.21 format_word_u_dot()

```
void format_word_u_dot (
            VM * vm )
```

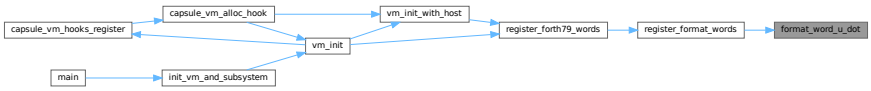
References VM::dsp, VM::error, print_number_formatted(), putchar(), and vm_pop().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:



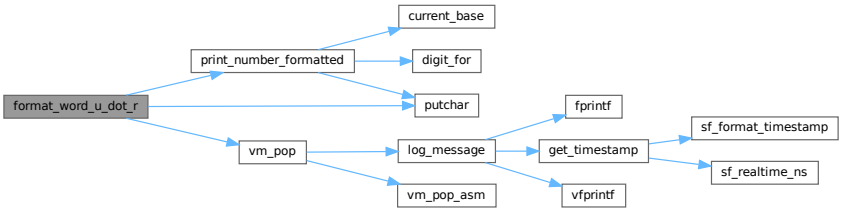
7.361.2.22 format_word_u_dot_r()

```
void format_word_u_dot_r (
            VM * vm )
```

References VM::dsp, VM::error, print_number_formatted(), putchar(), and vm_pop().

Referenced by register_format_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.361.2.23 print_number_formatted()

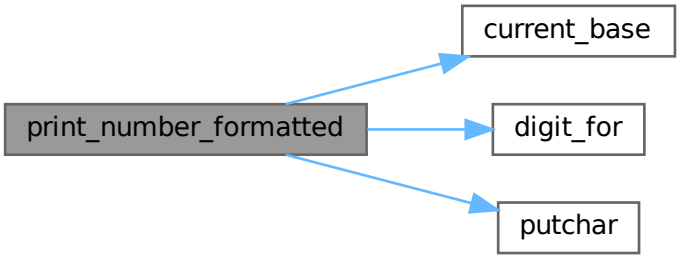
```
static void print_number_formatted (
            VM * vm,
            cell_t n,
            int width,
            int is_unsigned )  [static]
```

Prints number with optional width field and sign handling.

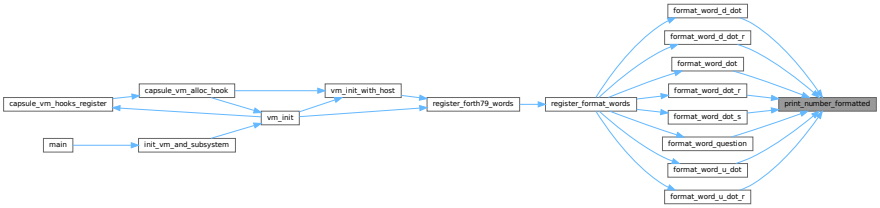
References base, current_base(), digit_for(), and putchar().

Referenced by format_word_d_dot(), format_word_d_dot_r(), format_word_dot(), format_word_dot_r(), format_word_dot_s(), format_word_question(), format_word_u_dot(), and format_word_u_dot_r().

Here is the call graph for this function:



Here is the caller graph for this function:



7.361.2.24 register_format_words()

```
void register_format_words (
            VM * vm )
```

Registers all formatting and numeric conversion words with the VM.

This function registers the standard FORTH formatting words including:

- Number output words (., .R, U., U.R, D., D.R, .S, ?)
- Picture number formatting (<#, #, #S, #>, HOLD, SIGN)
- Base conversion (BASE, DECIMAL, HEX, OCTAL)
- Memory dump (DUMP)

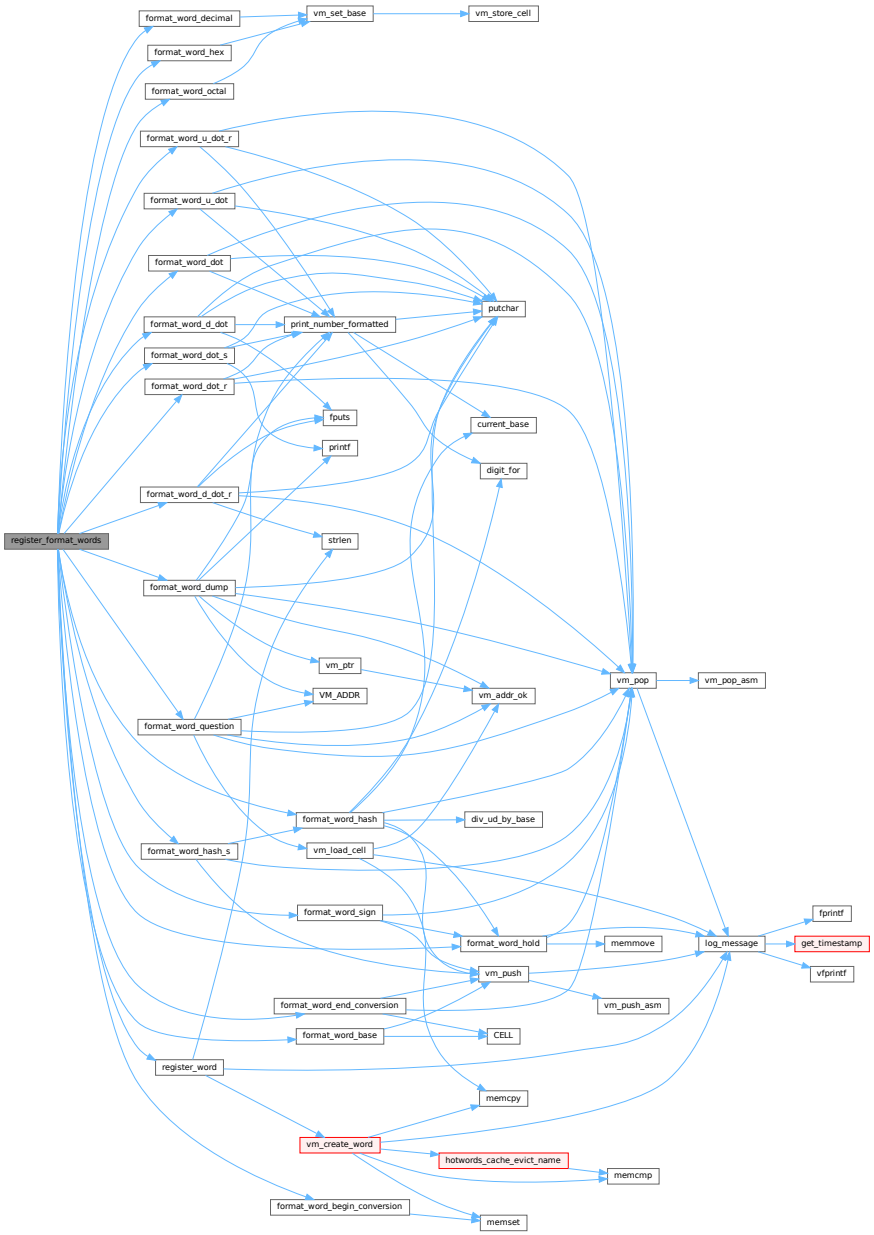
Parameters

vm	Pointer to the virtual machine instance

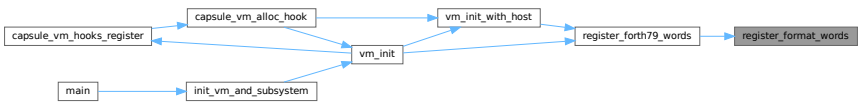
References format_word_base(), format_word_begin_conversion(), format_word_d_dot(), format_word_d_dot_r(), format_word_decimal(), format_word_dot(), format_word_dot_r(), format_word_dot_s(), format_word_dump(), format_word_end_conversion(), format_word_hash(), format_word_hash_s(), format_word_hex(), format_word_hold(), format_word_octal(), format_word_question(), format_word_sign(), format_word_u_dot(), format_word_u_dot_r(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

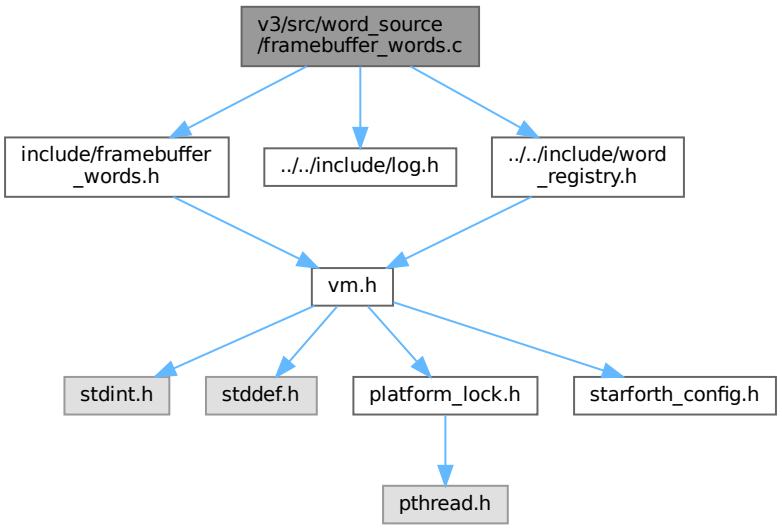


7.362 v3/src/word_source/framebuffer_words.c File Reference

```
#include "include/framebuffer_words.h"
```

```
#include "../../include/log.h"
#include "../../include/word_registry.h"
```

Include dependency graph for framebuffer_words.c:



Functions

- static void fbw_plot (VM *vm)
- static void fbw_width (VM *vm)
- static void fbw_height (VM *vm)
- void register_framebuffer_words (VM *vm)

7.362.1 Function Documentation

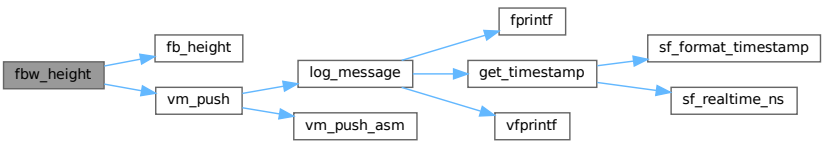
7.362.1.1 fbw_height()

```
static void fbw_height (
            VM * vm )  [static]
```

References fb_height(), and vm_push().

Referenced by register_framebuffer_words().

Here is the call graph for this function:



Here is the caller graph for this function:

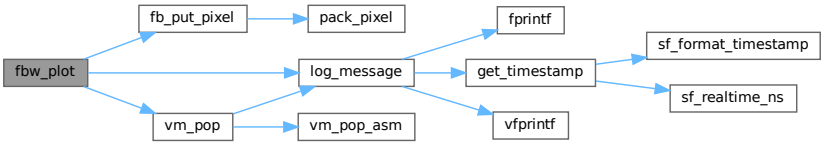
7.362.1.2 fbw_plot()

```
static void fbw_plot (
            VM * vm )  [static]
```

References VM::dsp, VM::error, fb_put_pixel(), LOG_ERROR, log_message(), and vm_pop().

Referenced by register_framebuffer_words().

Here is the call graph for this function:



Here is the caller graph for this function:

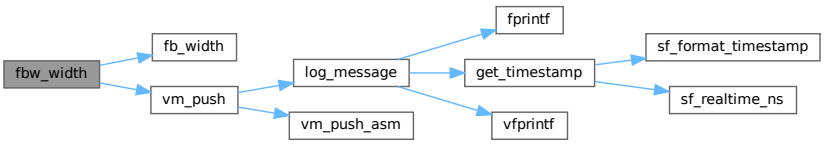
7.362.1.3 fbw_width()

```
static void fbw_width (
            VM * vm )  [static]
```

References fb_width(), and vm_push().

Referenced by register_framebuffer_words().

Here is the call graph for this function:



Here is the caller graph for this function:

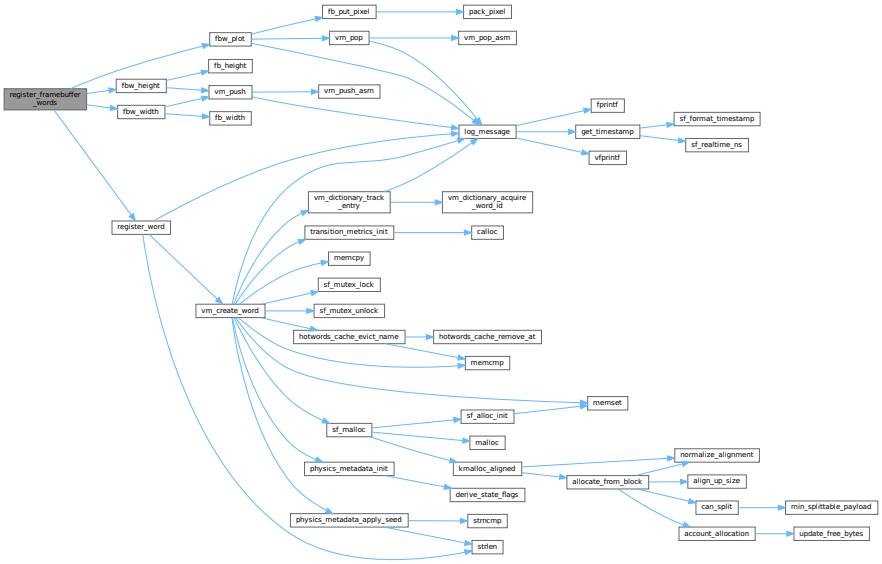
7.362.1.4 register_framebuffer_words()

```
void register_framebuffer_words (
            VM * vm )
```

References fbw_height(), fbw_plot(), fbw_width(), and register_word().

Referenced by capsule_birth_baby().

Here is the call graph for this function:

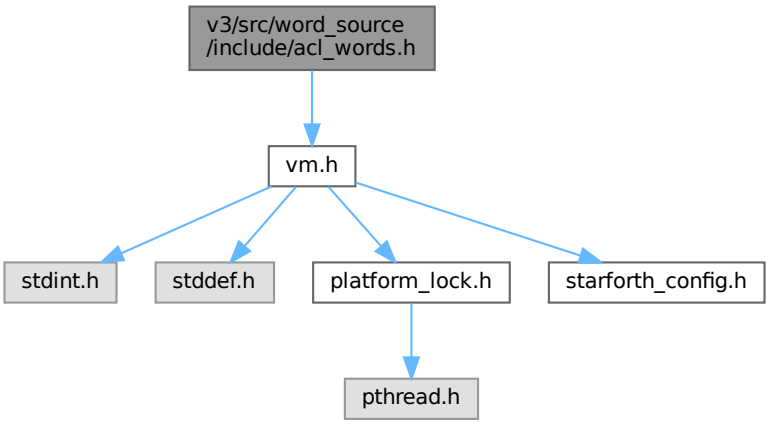


Here is the caller graph for this function:

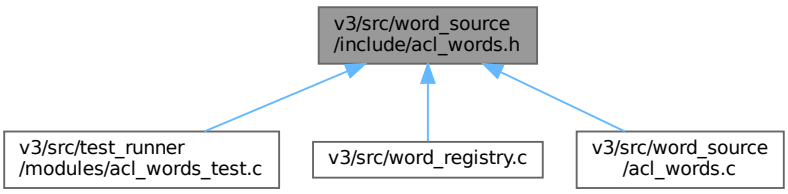
7.363 v3/src/word_source/include/acl_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for acl_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_acl_words (VM *vm)

  Register all ACL C-primitive words with the VM dictionary.
- void acl_inherit_entry (DictEntry *src, DictEntry *dst)

  Direct C entry point for ACL policy inheritance on the capsule birth path.

7.363.1 Function Documentation

7.363.1.1 acl_inherit_entry()

```
void acl_inherit_entry (
            DictEntry * src,
            DictEntry * dst )
```

Direct C entry point for ACL policy inheritance on the capsule birth path.

Copies `acl_mode` from `src` to `dst`, clears `acl_pinned`, resets `acl_ttl` to 0, and sets `acl_allow` to 1. Called from capsule birth code (not from FORTH) — no stack manipulation is performed.

Parameters

src	Source DictEntry whose `acl_mode` is inherited
dst	Destination DictEntry whose ACL fields are updated

References DictEntry::acl_allow, DictEntry::acl_mode, DictEntry::acl_pinned, and DictEntry::acl_ttl.

Referenced by forth_acl_inherit(), and test_acl_inherit().

Here is the caller graph for this function:

7.363.1.2 register_acl_words()

```
void register_acl_words (
              VM * vm )
```

Register all ACL C-primitive words with the VM dictionary.

Registers: field accessors (`ACL-MODE@` / `ACL-MODE!` / `ACL-PINNED?` / `ACL-TTL@` / `ACL-TTL!` / `ACL-ALLOW@` / `ACL-ALLOW!` / `ACL-HEAT@` / `ACL-WORD-ID`), and mutation words (`ACL-PIN` / `ACL-INHERIT` / `ACL-INIT-PRIMITIVES`). Called from `vm_bootstrap.c` during VM initialisation.

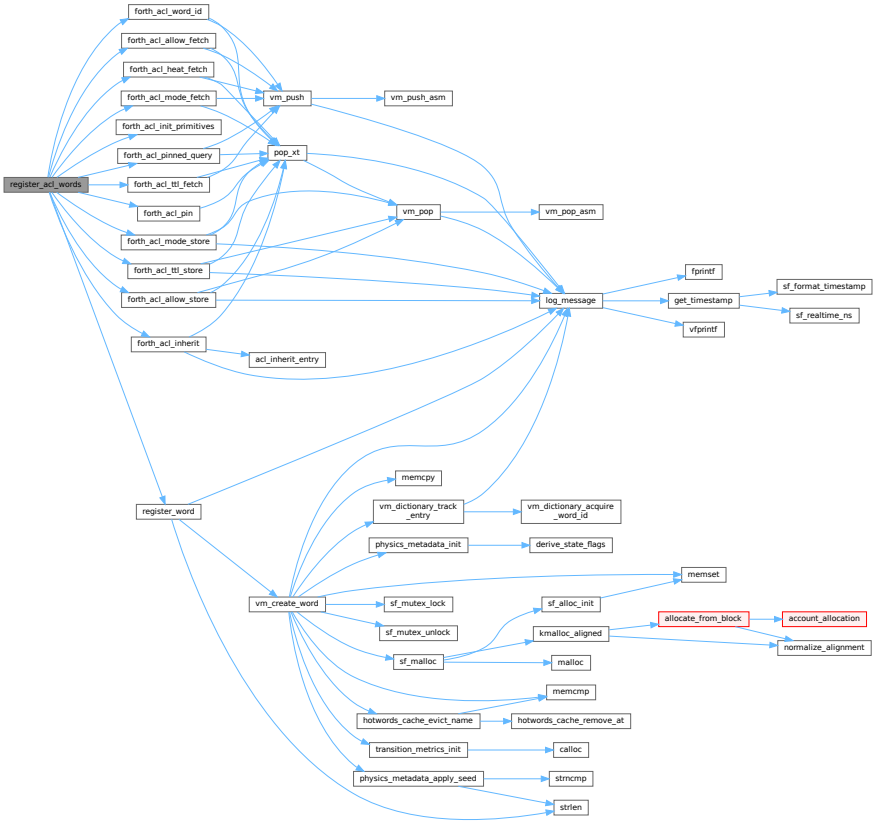
Parameters

vm	Pointer to the VM structure

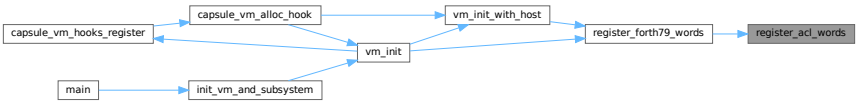
References forth_acl_allow_fetch(), forth_acl_allow_store(), forth_acl_heat_fetch(), forth_acl_inherit(), forth_acl_init_primitives(), forth_acl_mode_fetch(), forth_acl_mode_store(), forth_acl_pin(), forth_acl_pinned_query(), forth_acl_ttl_fetch(), forth_acl_ttl_store(), forth_acl_word_id(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



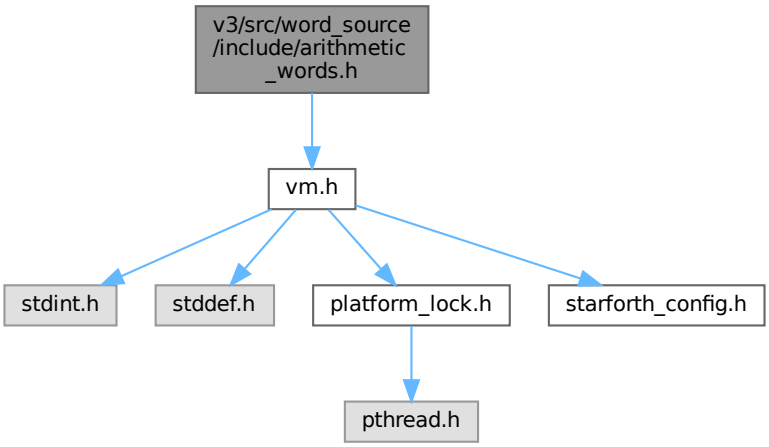
Here is the caller graph for this function:



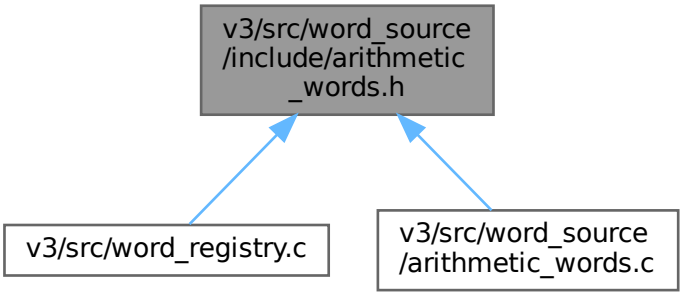
7.364 v3/src/word_source/include/arithmetic_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for arithmetic_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_arithmetic_words (VM *vm)

  Registers all FORTH-79 arithmetic words with the virtual machine.

7.364.1 Function Documentation

7.364.1.1 register_arithmetic_words()

```
void register_arithmetic_words (
            VM * vm )
```

Registers all FORTH-79 arithmetic words with the virtual machine.

Registers the following words:

- + (add)
- - (subtract)
- ∗ (multiply)
- / (divide)
- MOD (modulo)
- /MOD (divide with remainder)
- NEGATE
- ABS (absolute)
- MIN
- MAX
- 1+ (increment)
- 1- (decrement)
- 2∗ (multiply by 2)
- 2/ (divide by 2)
- STAR-SLASH (multiply then divide)

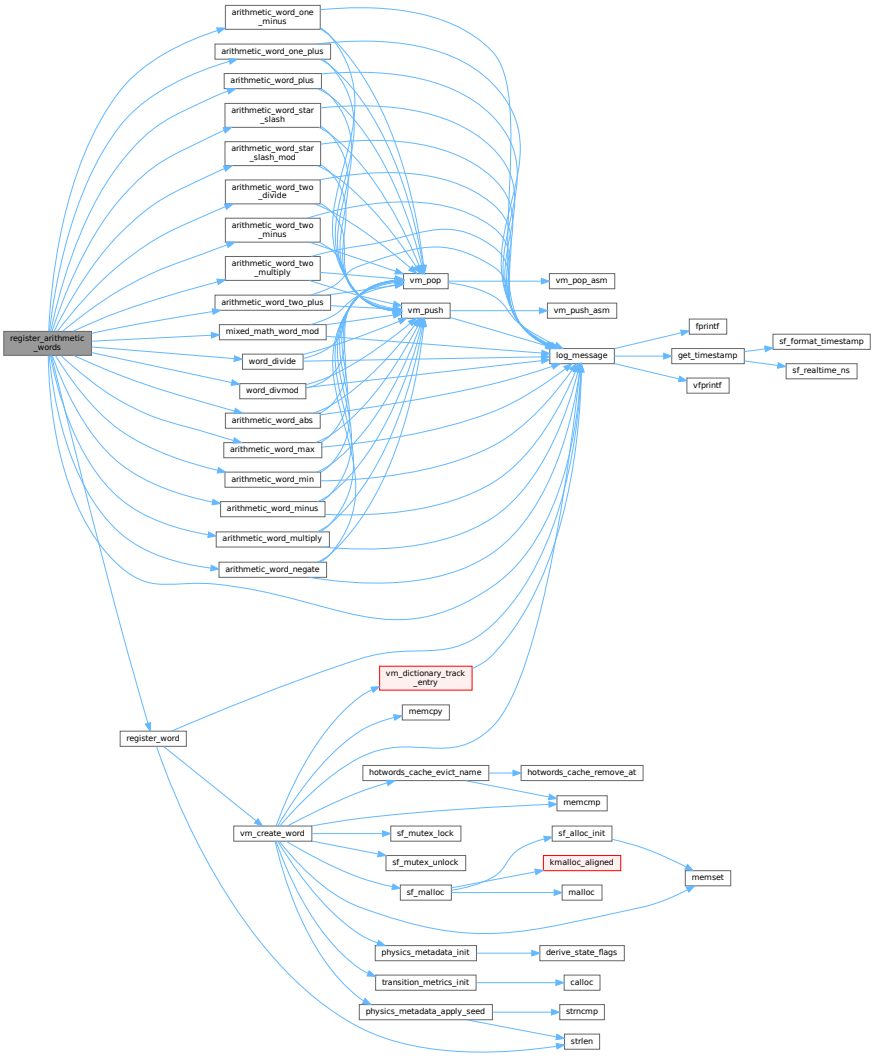
Parameters

vm	Pointer to the Forth virtual machine instance

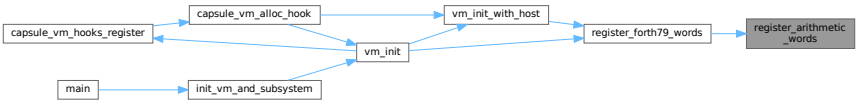
References arithmetic_word_abs(), arithmetic_word_max(), arithmetic_word_min(), arithmetic_word_minus(), arithmetic_word_multiply(), arithmetic_word_negate(), arithmetic_word_one_minus(), arithmetic_word_one_plus(), arithmetic_word_plus(), arithmetic_word_star_slash(), arithmetic_word_star_slash_mod(), arithmetic_word_two_divide(), arithmetic_word_two_minus(), arithmetic_word_two_multiply(), arithmetic_word_two_plus(), LOG_INFO, log_message(), mixed_math_word_mod(), register_word(), word_divide(), and word_divmod().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

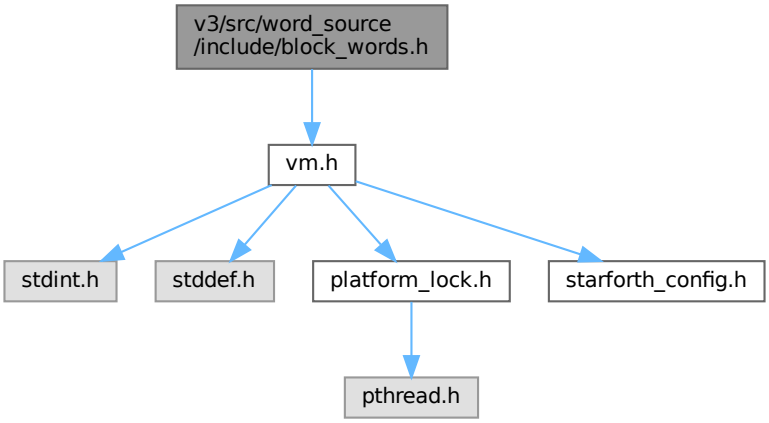


7.365 v3/src/word_source/include/block_words.h File Reference

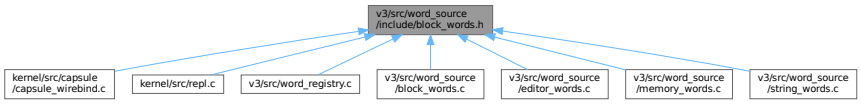
FORTH-79 Block and Buffer System Implementation.

```
#include "vm.h"
```

Include dependency graph for block_words.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct BlockBuffer

  Block buffer structure maintaining buffer state.

Macros

Block System Configuration

- #define BLOCK_SIZE 1024

  Size of each block in bytes (1KB)
- #define MAX_BLOCKS (VM_MEMORY_SIZE / BLOCK_SIZE)

  Maximum number of blocks (1024 blocks from 1MB memory)
- #define BUFFER_COUNT 4

  Number of block buffers in memory.

Enumerations

- enum buffer_type_t { BUFFER_TYPE_MEMORY , BUFFER_TYPE_HARDWARE , BUFFER_TYPE_STORAGE , BUFFER_TYPE_NETWORK }

  Block buffer types for different storage mechanisms.

Functions

- void register_block_words (VM *vm)

  Registers all block and buffer words with the virtual machine.
- void init_block_system (VM *vm)

  Initializes the block system.
- unsigned char * get_block_buffer (VM *vm, int block_num)

  Gets a buffer for the specified block.
- unsigned char * get_empty_buffer (VM *vm, int block_num)

  Gets an empty buffer for the specified block.
- void mark_buffer_dirty (VM *vm)

  Marks the current buffer as modified.
- void save_all_buffers (VM *vm)
- void blk_vm_flush_all (VM *vm)

  Sync every dirty VM block-window slot to the C-layer block cache, then flush the whole block subsystem to physical media – the complete SAVE-BUFFERS implementation (unlike save_all_buffers() above, which only calls blk_flush() and so misses any BUFFER-modified-but-not-yet-UPDATE'd content still sitting only in vm->memory's window).
- void empty_all_buffers (VM *vm)

  Empties all user block buffers.
- void blk_vm_check_epoch (VM *vm)

  Discards vm->blk_vm_lbn[]/cbuf[]/dirty[] if the device chain has changed since last validated (see this function's own doc comment in block_words.c for why).
- int blk_vm_slot_for_addr (vaddr_t addr)

  Does `addr` fall inside the block I/O window? Pure address arithmetic against the window's fixed compile-time base/size – no VM state needed.
- int blk_vm_is_cross_device_copy (VM *vm, vaddr_t addr1, vaddr_t addr2)

  True if addr1/addr2 are both block-window addresses currently loaded from two different physical devices (Phase 8 v3, 2026-09-23) – the check MOVE/CMOVE/CMOVE> call before copying to refuse a same-VM, cross-device raw block copy.

7.365.1 Detailed Description

FORTH-79 Block and Buffer System Implementation.

This file implements the standard FORTH-79 block and buffer words:

- BLOCK (u – addr) Get block buffer address
- BUFFER (u – addr) Get empty buffer for block
- UPDATE (–) Mark current block as modified
- SAVE-BUFFERS (–) Save all modified blocks
- EMPTY-BUFFERS (–) Mark all buffers as empty
- FLUSH (–) Save buffers and empty them
- LIST (u –) List block contents
- LOAD (u –) Load and interpret block
- SCR (– addr) Variable: screen/block being listed
- THRU (u1 u2 –) Load blocks from u1 to u2
- --> or → (–) Continue to next block

7.365.2 Macro Definition Documentation

7.365.2.1 BLOCK_SIZE

```
#define BLOCK_SIZE 1024
```

Size of each block in bytes (1KB)

7.365.2.2 BUFFER_COUNT

```
#define BUFFER_COUNT 4
```

Number of block buffers in memory.

7.365.2.3 MAX_BLOCKS

```
#define MAX_BLOCKS (VM_MEMORY_SIZE / BLOCK_SIZE)
```

Maximum number of blocks (1024 blocks from 1MB memory)

7.365.3 Enumeration Type Documentation

7.365.3.1 buffer_type_t

```
enum buffer_type_t
```

Block buffer types for different storage mechanisms.

Enumerator

BUFFER_TYPE_MEMORY	Direct memory access (current implementation)
BUFFER_TYPE_HARDWARE	Future: Hardware register blocks.
BUFFER_TYPE_STORAGE	Future: Persistent storage blocks.
BUFFER_TYPE_NETWORK	Future: Network packet buffers.

7.365.4 Function Documentation

7.365.4.1 blk_vm_check_epoch()

```
void blk_vm_check_epoch (
            VM * vm )
```

Discards vm->blk_vm_lbn[]/cbuf[]/dirty[] if the device chain has changed since last validated (see this function's own doc comment in block_words.c for why).

Exposed (Phase 8 v3, 2026-09-23) so callers outside this file that read vm->blk_vm_lbn[] directly (MOVE/↩ CMOVE's cross-device block-copy check, memory_words.c/string_words.c) go through the same invalidation contract blk_vm_find() already uses internally, rather than a second, divergent copy of this logic.

Parameters

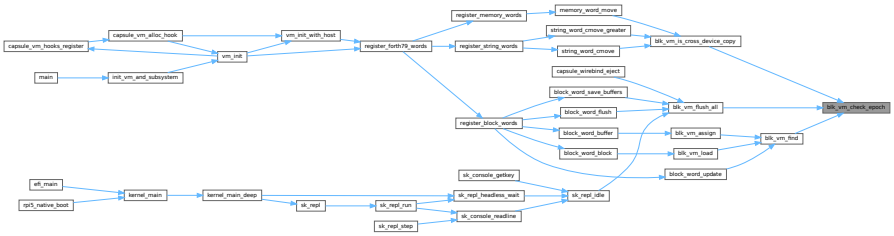
vm	Pointer to the Forth virtual machine instance

References blk_subsys_epoch(), VM::blk_vm_cbuf, VM::blk_vm_dirty, VM::blk_vm_epoch, VM::blk_vm_lbn, BLK_VM_SLOTS, and epoch.

Referenced by blk_vm_find(), blk_vm_flush_all(), and blk_vm_is_cross_device_copy().

Here is the call graph for this function:

Here is the caller graph for this function:



7.365.4.2 blk_vm_flush_all()

```
void blk_vm_flush_all (
            VM * vm )
```

Sync every dirty VM block-window slot to the C-layer block cache, then flush the whole block subsystem to physical media – the complete SAVE-BUFFERS implementation (unlike save_all_buffers() above, which only calls blk_flush() and so misses any BUFFER-modified-but-not-yet-UPDATE'd content still sitting only in vm->memory's window).

Cheap to call when nothing is dirty – every check inside is a small fixed-size scan, no I/O happens unless something actually needs writing. Safe to call from outside word dispatch (e.g. a periodic idle-tick auto-flush); vm must be non-NULL.

Parameters

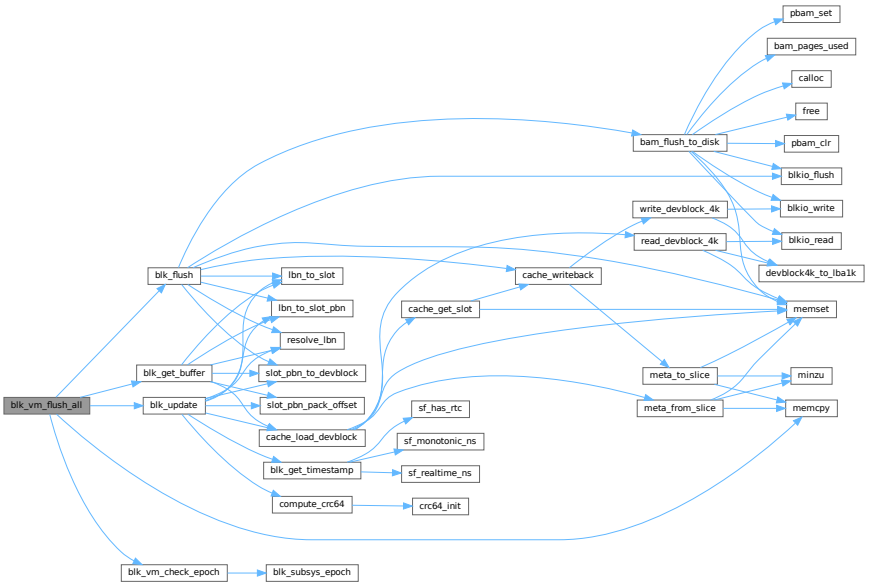
vm	Pointer to the Forth virtual machine instance

References base, blk_flush(), blk_get_buffer(), blk_update(), VM::blk_vm_cbuf, blk_vm_check_epoch(),

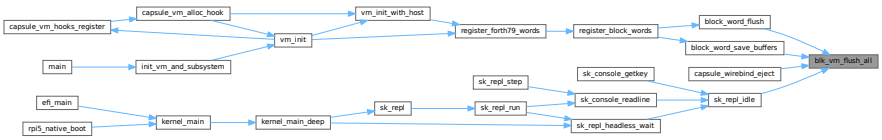
VM::blk_vm_dirty, VM::blk_vm_lbn, BLK_VM_SLOTS, BLK_VM_WINDOW_BASE, BLOCK_SIZE, memcpy(), and VM::memory.

Referenced by block_word_flush(), block_word_save_buffers(), capsule_wirebind_eject(), and sk_repl_idle().

Here is the call graph for this function:



Here is the caller graph for this function:



7.365.4.3 blk_vm_is_cross_device_copy()

```
int blk_vm_is_cross_device_copy (
            VM * vm,
            vaddr_t addr1,
            vaddr_t addr2 )
```

True if addr1/addr2 are both block-window addresses currently loaded from two different physical devices (Phase 8 v3, 2026-09-23) – the check MOVE/CMOVE/CMOVE> call before copying to refuse a same-VM, cross-device raw block copy.

See this function's own doc comment in block_words.c for exactly what is and isn't flagged.

Parameters

vm	VM performing the copy
addr1	first VM offset (source or destination, order doesn't matter)
addr2	second VM offset

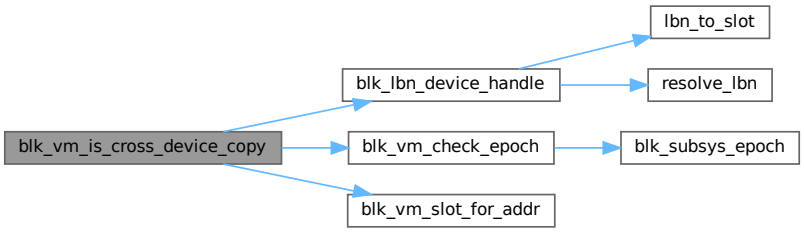
Returns

1 if this copy should be refused, 0 otherwise

References blk_lbn_device_handle(), blk_vm_check_epoch(), VM::blk_vm_lbn, and blk_vm_slot_for_addr().

Referenced by memory_word_move(), string_word_cmove(), and string_word_cmove_greater().

Here is the call graph for this function:



Here is the caller graph for this function:

7.365.4.4 blk_vm_slot_for_addr()

```
int blk_vm_slot_for_addr (
            vaddr_t addr )
```

Does `addr` fall inside the block I/O window? Pure address arithmetic against the window's fixed compile-time base/size – no VM state needed.

Exposed (Phase 8 v3, 2026-09-23) for MOVE/CMOVE/CMOVE>'s cross-device block-copy check.

Parameters

addr	VM offset to test

Returns

slot index (0..BLK_VM_SLOTS-1) if inside the window, -1 otherwise

References BLK_VM_WINDOW_BASE, BLK_VM_WINDOW_SIZE, and BLOCK_SIZE.

Referenced by blk_vm_is_cross_device_copy().

Here is the caller graph for this function:

7.365.4.5 empty_all_buffers()

```
void empty_all_buffers (
            VM * vm )
```

Empties all user block buffers.

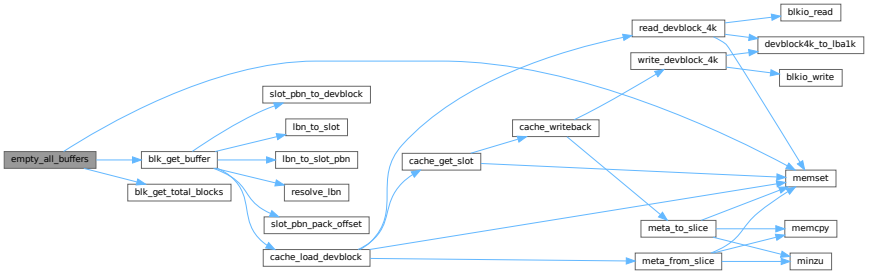
Parameters

vm	Pointer to the VM instance

References blk_get_buffer(), blk_get_total_blocks(), BLOCK_SIZE, memset(), and USER_BLOCKS_START.

Referenced by block_word_empty_buffers().

Here is the call graph for this function:



Here is the caller graph for this function:

7.365.4.6 get_block_buffer()

```
unsigned char * get_block_buffer (
            VM * vm,
            int block_num )
```

Gets a buffer for the specified block.

Parameters

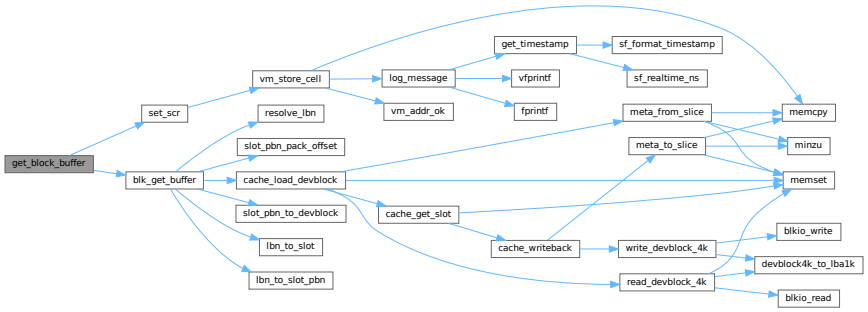
vm	Pointer to the VM instance
block_num	Block number to retrieve

Returns

Pointer to block buffer or NULL if invalid

References blk_get_buffer(), and set_scr().

Here is the call graph for this function:



7.365.4.7 get_empty_buffer()

```
unsigned char * get_empty_buffer (
            VM * vm,
            int block_num )
```

Gets an empty buffer for the specified block.

Parameters

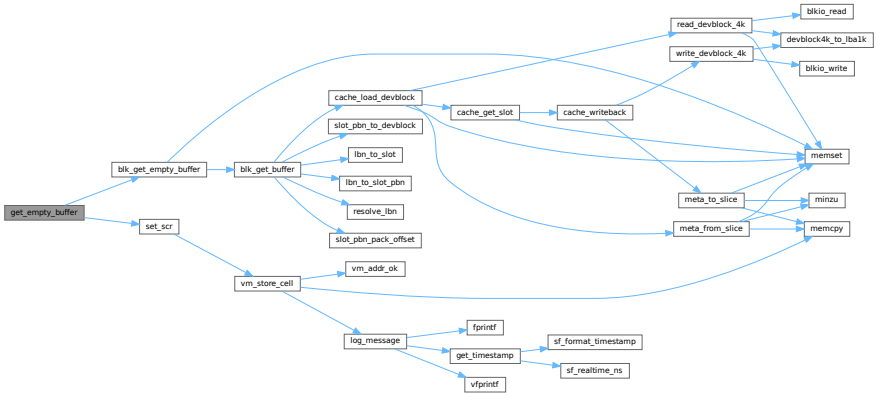
vm	Pointer to the VM instance
block_num	Block number to create

Returns

Pointer to empty block buffer or NULL if invalid

References blk_get_empty_buffer(), and set_scr().

Here is the call graph for this function:



7.365.4.8 init_block_system()

```
void init_block_system (
            VM * vm )
```

Initializes the block system.

Parameters

vm	Pointer to the Forth virtual machine instance

7.365.4.9 mark_buffer_dirty()

```
void mark_buffer_dirty (
            VM * vm )
```

Marks the current buffer as modified.

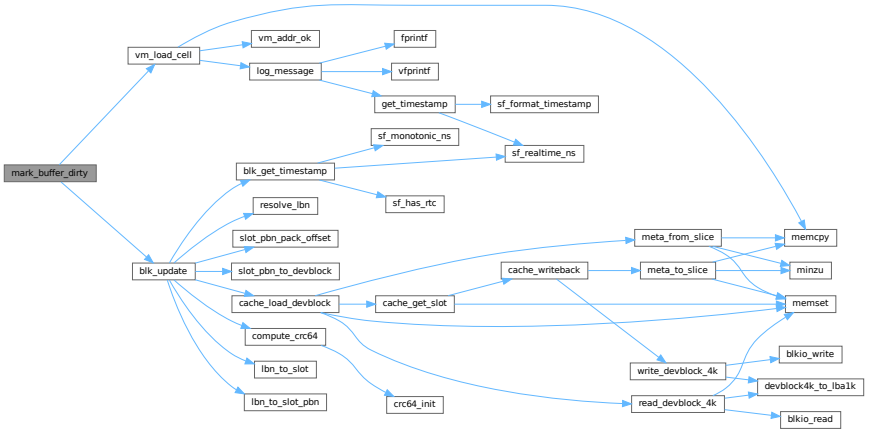
Parameters

vm	Pointer to the Forth virtual machine instance

References blk_update(), VM::scr_addr, and vm_load_cell().

Referenced by editor_word_edit(), and editor_word_s().

Here is the call graph for this function:



Here is the caller graph for this function:

7.365.4.10 register_block_words()

```
void register_block_words (
            VM * vm )
```

Registers all block and buffer words with the virtual machine.

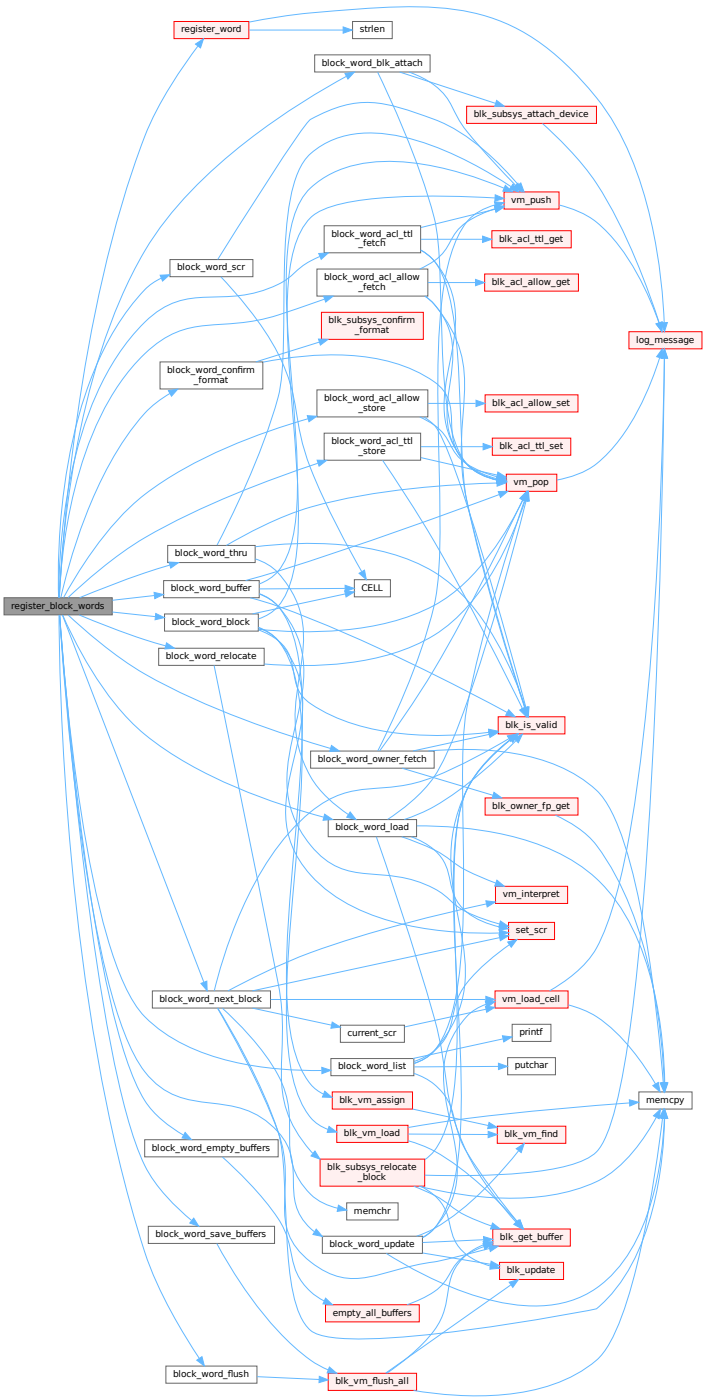
Parameters

vm	Pointer to the Forth virtual machine instance

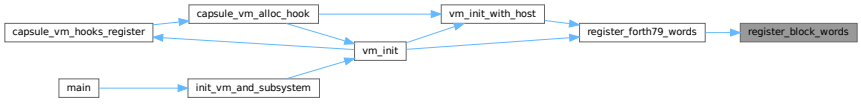
References block_word_acl_allow_fetch(), block_word_acl_allow_store(), block_word_acl_ttl_fetch(), block_word_acl_ttl_store(), block_word_blk_attach(), block_word_block(), block_word_buffer(), block_word_confirm_format(), block_word_empty_buffers(), block_word_flush(), block_word_list(), block_word_load(), block_word_next_block(), block_word_owner_fetch(), block_word_relocate(), block_word_save_buffers(), block_word_scr(), block_word_thru(), block_word_update(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



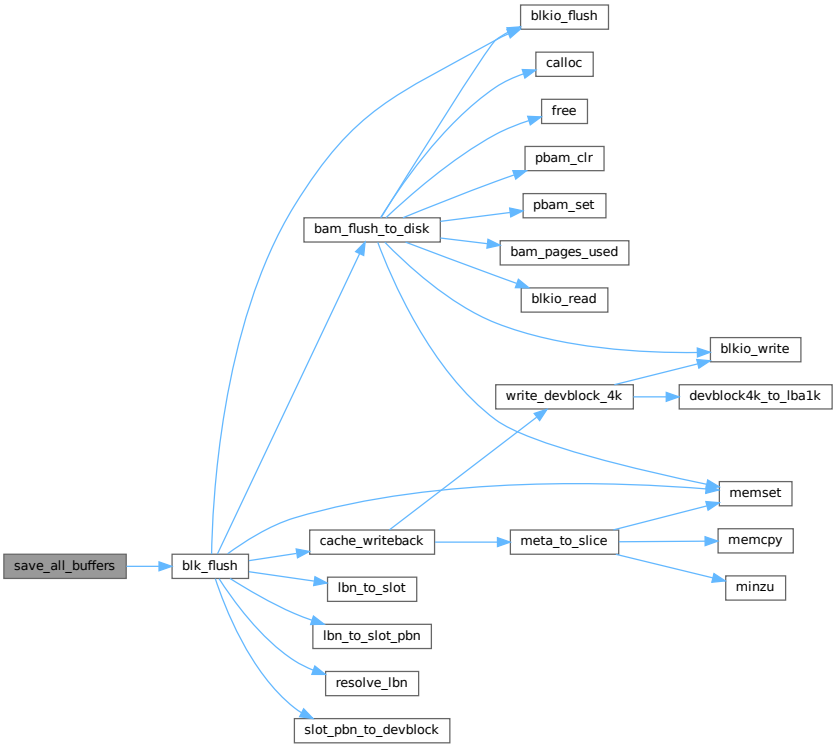
7.365.4.11 save_all_buffers()

```
void save_all_buffers (
            VM * vm )
```

References blk_flush().

Referenced by editor_word_edit().

Here is the call graph for this function:

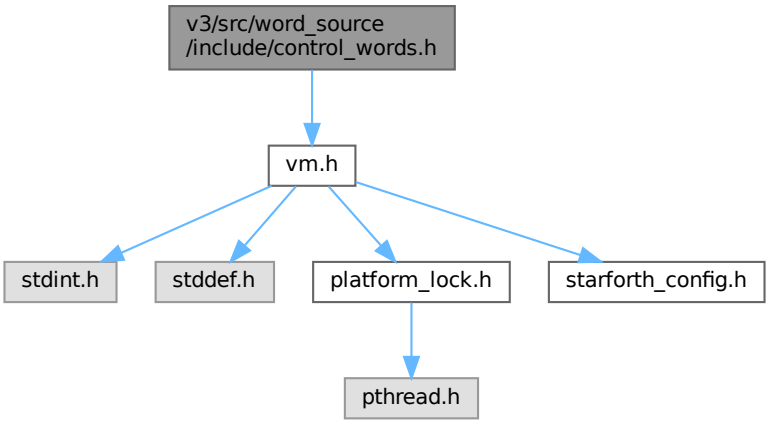


Here is the caller graph for this function:

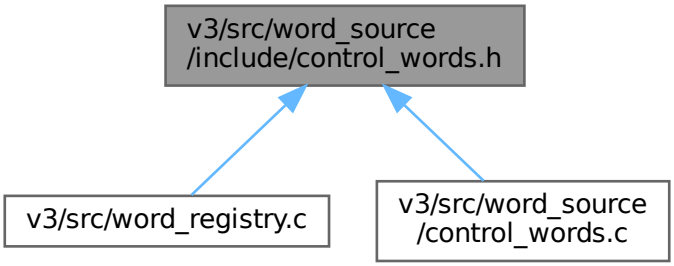
7.366 v3/src/word_source/include/control_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for control_words.h:



This graph shows which files directly or indirectly include this file:



Functions

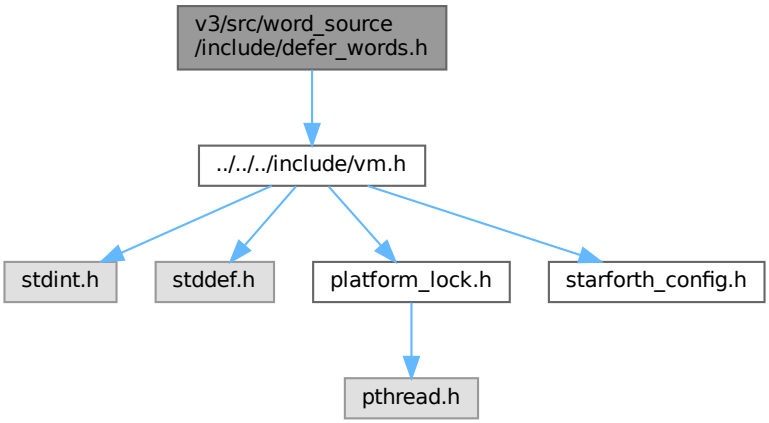
- void register_control_words (VM *vm)

  Registers all control flow words with the Forth virtual machine.

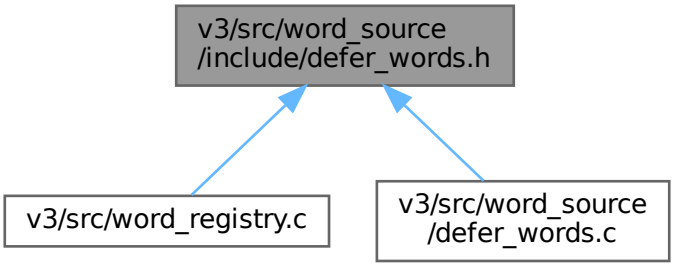
7.367 v3/src/word_source/include/defer_words.h File Reference

```
#include "../../../include/vm.h"
```

Include dependency graph for defer_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_defer_words (VM *vm)

7.367.1 Function Documentation

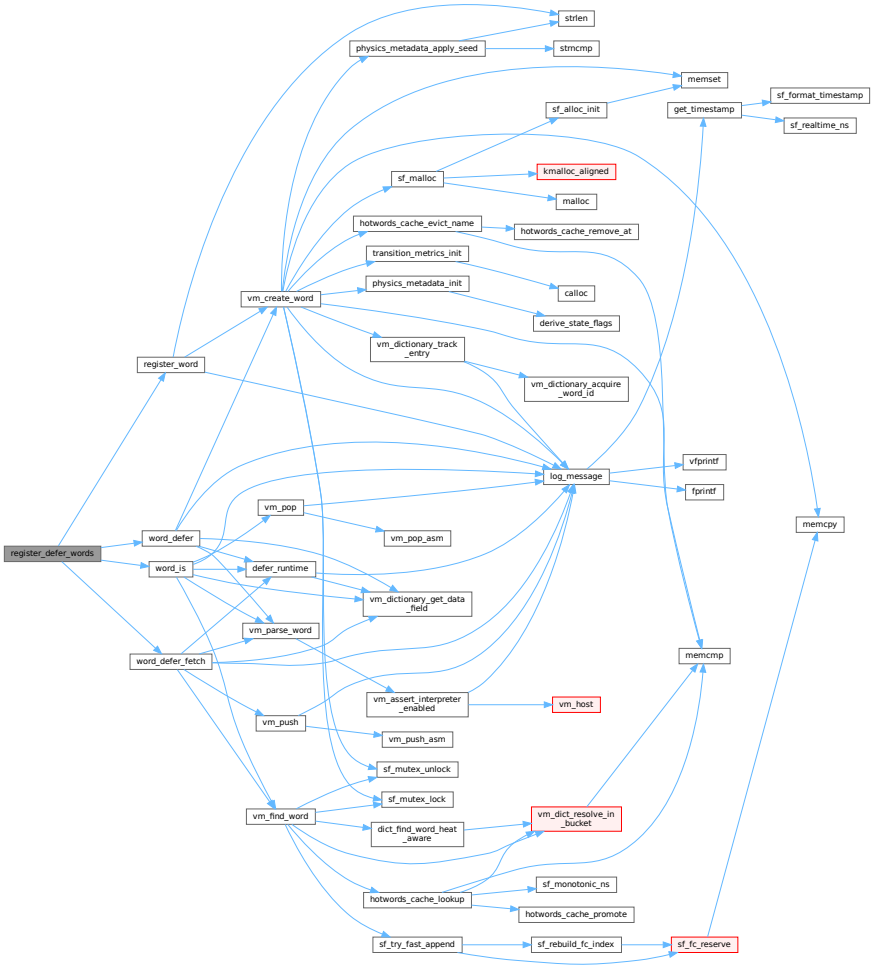
7.367.1.1 register_defer_words()

```
void register_defer_words (
            VM * vm )
```

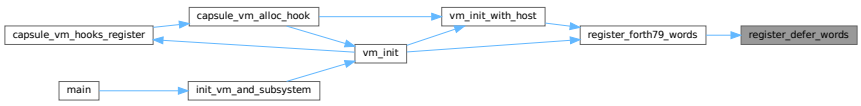
References register_word(), word_defer(), word_defer_fetch(), and word_is().

Referenced by register_forth79_words().

Here is the call graph for this function:



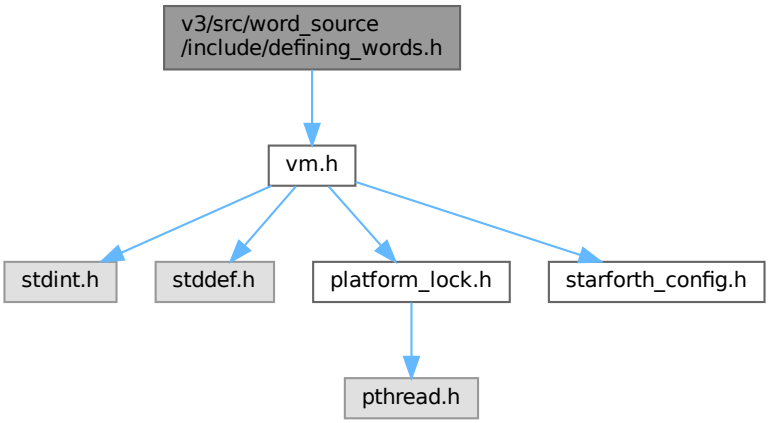
Here is the caller graph for this function:



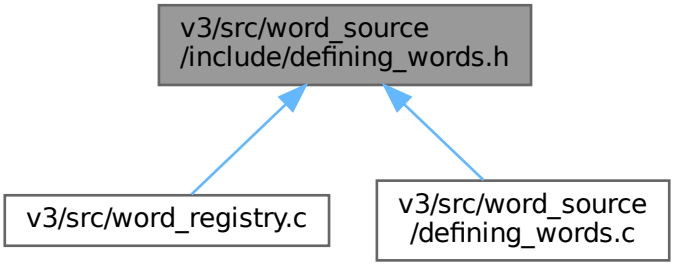
7.368 v3/src/word_source/include/defining_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for defining_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_defining_words (VM *vm)
  Register all defining words in the dictionary.

7.368.1 Function Documentation

7.368.1.1 register_defining_words()

```
void register_defining_words (
            VM * vm )
```

Register all defining words in the dictionary.

Registers core defining words like :, ;, CREATE, CONSTANT, VARIABLE, and their support words like IMMEDIATE, STATE, etc.

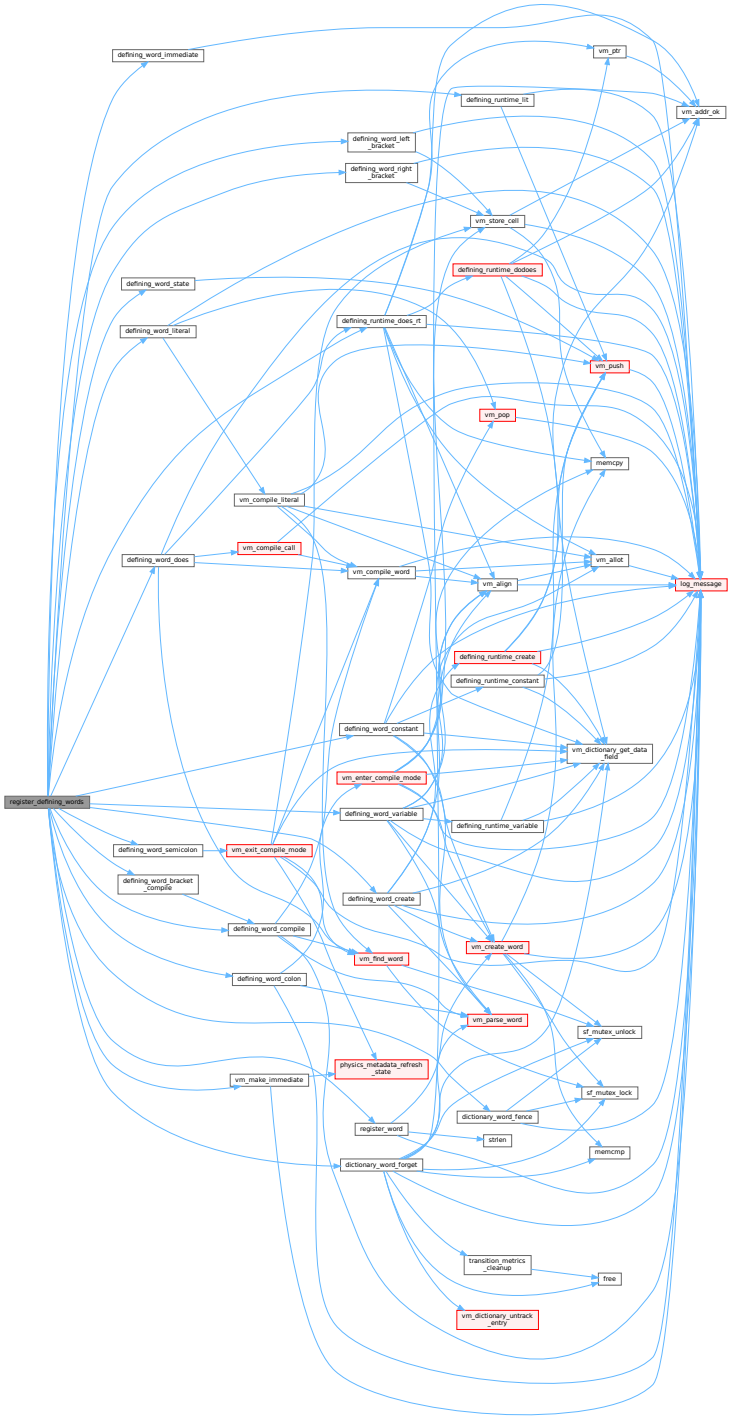
Parameters

vm	Pointer to the VM context

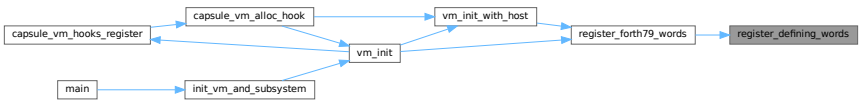
References defining_runtime_does_rt(), defining_runtime_lit(), defining_word_bracket_compile(), defining_word_colon(), defining_word_compile(), defining_word_constant(), defining_word_create(), defining_word_does(), defining_word_immediate(), defining_word_left_bracket(), defining_word_literal(), defining_word_right_bracket(), defining_word_semicolon(), defining_word_state(), defining_word_variable(), dictionary_word_fence(), dictionary_word_forget(), register_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



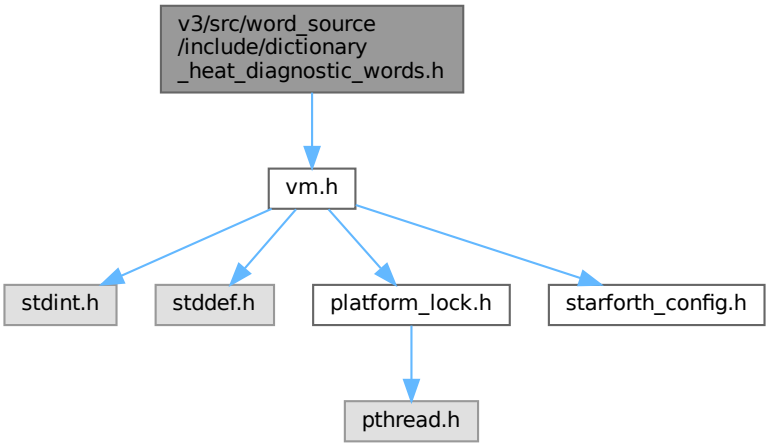
Here is the caller graph for this function:



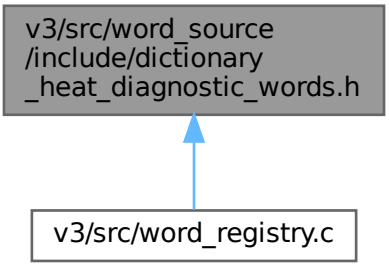
7.369 v3/src/word_source/include/dictionary_heat_diagnostic_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for dictionary_heat_diagnostic_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_dictionary_heat_diagnostic_words (VM *vm)

  Register all dictionary heat diagnostic words with the VM.

7.369.1 Function Documentation

7.369.1.1 register_dictionary_heat_diagnostic_words()

```
void register_dictionary_heat_diagnostic_words (
            VM * vm )
```

Register all dictionary heat diagnostic words with the VM.

Register all dictionary heat diagnostic words with the VM.

Registers: `HEAT-PERCENTILES`, `LOOKUP-STRATEGY@`, `LOOKUP-STRATEGY!`, `REORG-BUCKETS`, `SHOW-HEAT-OPTIMIZATION`, `COMPARE-LOOKUPS`. Called during VM bootstrap.

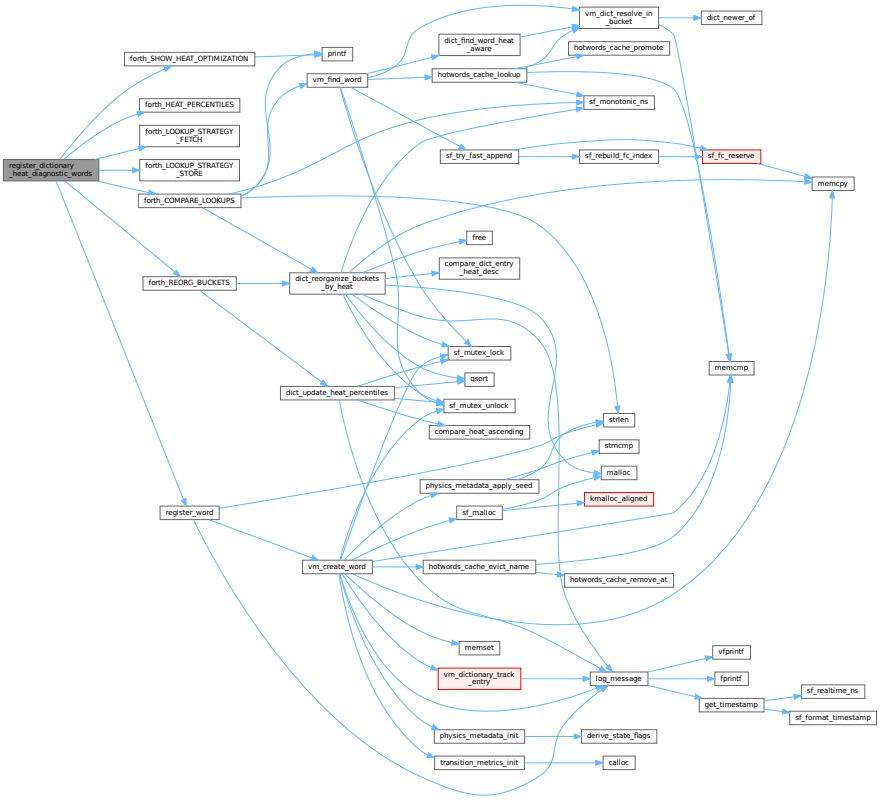
Parameters

vm	Active VM to register words into

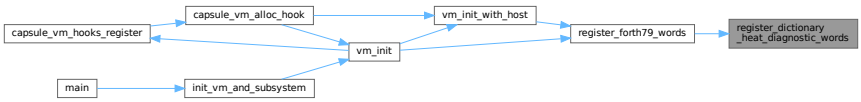
References forth_COMPARE_LOOKUPS(), forth_HEAT_PERCENTILES(), forth_LOOKUP_STRATEGY_FETCH(), forth_LOOKUP_STRATEGY_STORE(), forth_REORG_BUCKETS(), forth_SHOW_HEAT_OPTIMIZATION(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:

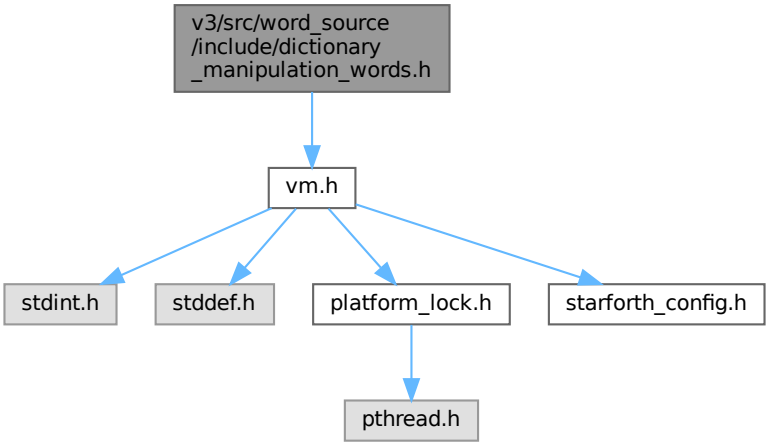


Here is the caller graph for this function:

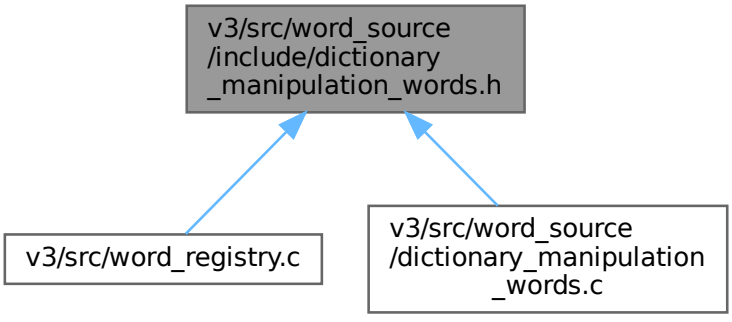


7.370 v3/src/word_source/include/dictionary_manipulation_words.h File Reference

`#include "vm.h"`
Include dependency graph for dictionary_manipulation_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_dictionary_manipulation_words (VM *vm)

  Register all dictionary manipulation words with the virtual machine.

7.370.1 Function Documentation

7.370.1.1 register_dictionary_manipulation_words()

```
void register_dictionary_manipulation_words (
            VM * vm )
```

Register all dictionary manipulation words with the virtual machine.

Parameters

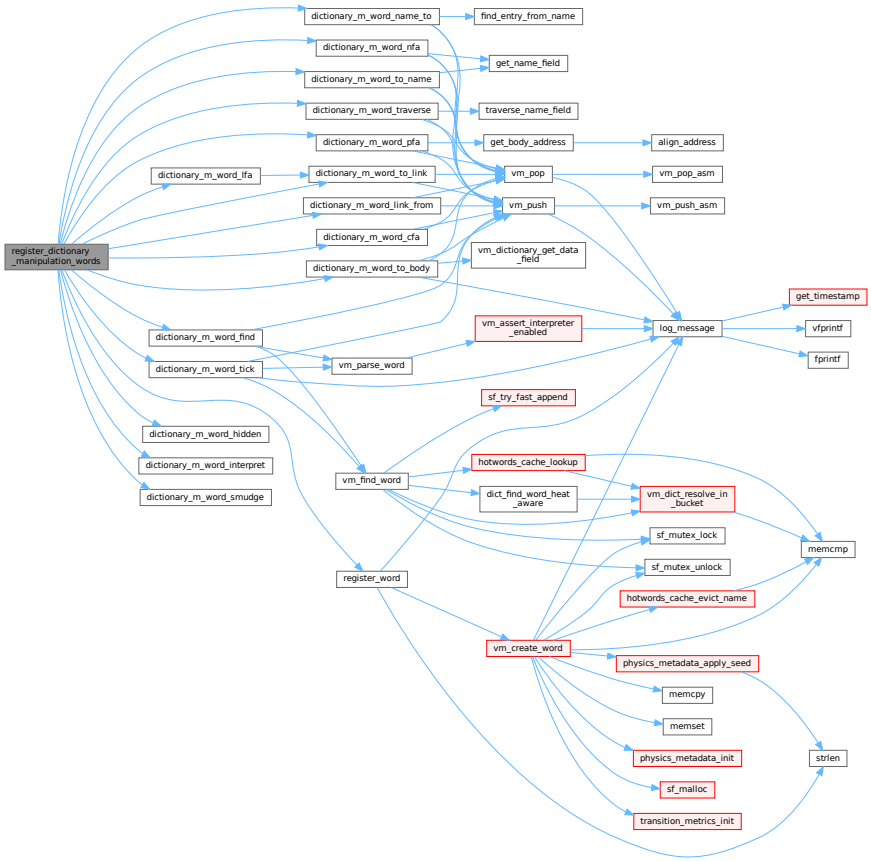
vm	Pointer to the virtual machine instance

This function registers all FORTH-79 standard dictionary manipulation words with the provided virtual machine instance. These words are essential for dictionary management and word definition functionality.

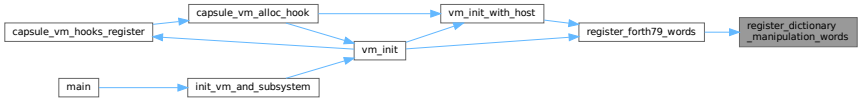
References dictionary_m_word_cfa(), dictionary_m_word_find(), dictionary_m_word_hidden(), dictionary_m_word_interpret(), dictionary_m_word_lfa(), dictionary_m_word_link_from(), dictionary_m_word_name_to(), dictionary_m_word_nfa(), dictionary_m_word_pfa(), dictionary_m_word_smudge(), dictionary_m_word_tick(), dictionary_m_word_to_body(), dictionary_m_word_to_link(), dictionary_m_word_to_name(), dictionary_m_word_traverse(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



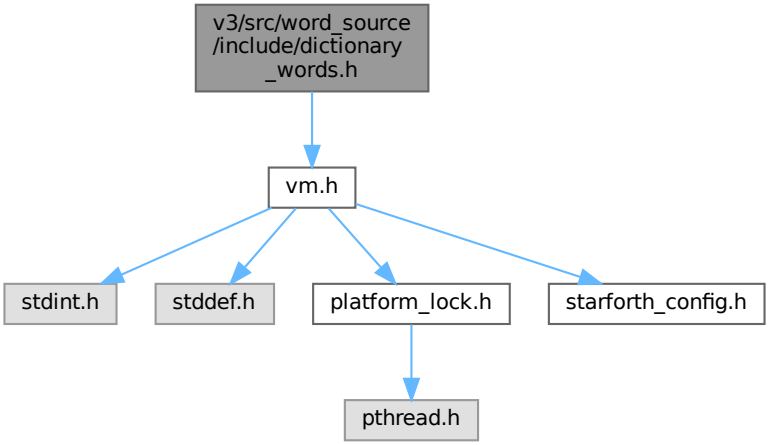
Here is the caller graph for this function:



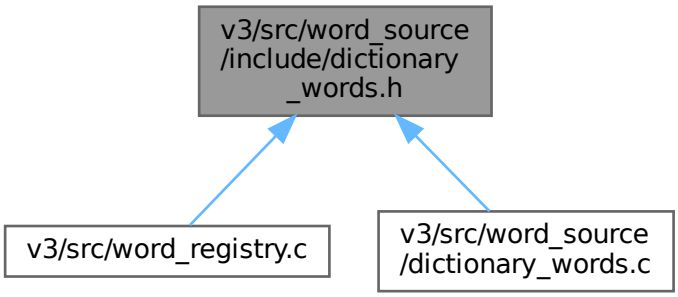
7.371 v3/src/word_source/include/dictionary_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for dictionary_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_dictionary_words (VM *vm)

  Registers all dictionary and compilation words with the Forth VM.

7.371.1 Function Documentation

7.371.1.1 register_dictionary_words()

```
void register_dictionary_words (
            VM * vm )
```

Registers all dictionary and compilation words with the Forth VM.

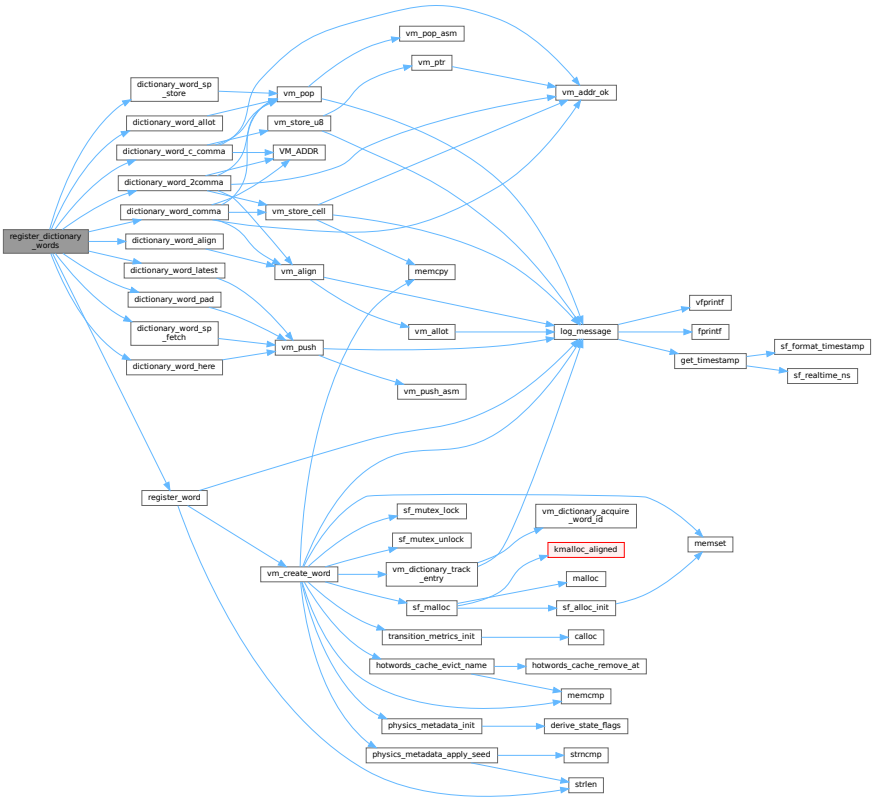
Parameters

vm	Pointer to the Forth virtual machine instance

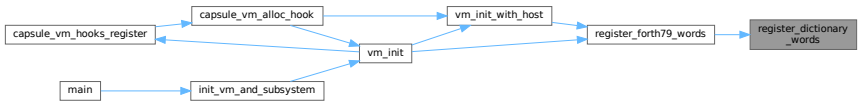
References dictionary_word_2comma(), dictionary_word_align(), dictionary_word_allot(), dictionary_word_c_comma(), dictionary_word_comma(), dictionary_word_here(), dictionary_word_latest(), dictionary_word_pad(), dictionary_word_sp_fetch(), dictionary_word_sp_store(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



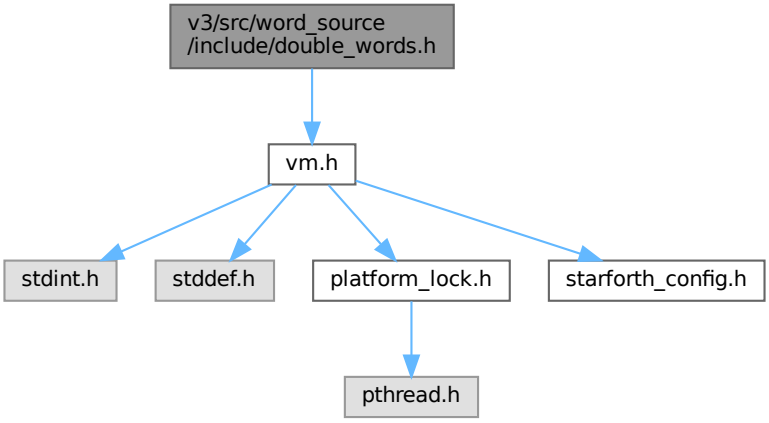
Here is the caller graph for this function:



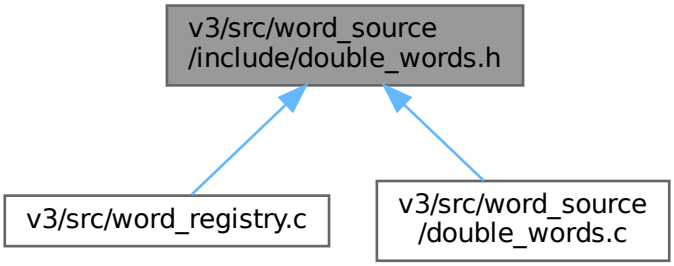
7.372 v3/src/word_source/include/double_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for double_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_double_words (VM *vm)

  Register all double number words.

7.372.1 Function Documentation

7.372.1.1 register_double_words()

```
void register_double_words (
            VM * vm )
```

Register all double number words.

Registers all double-precision number operations with the virtual machine.

Parameters

vm	Pointer to the virtual machine state

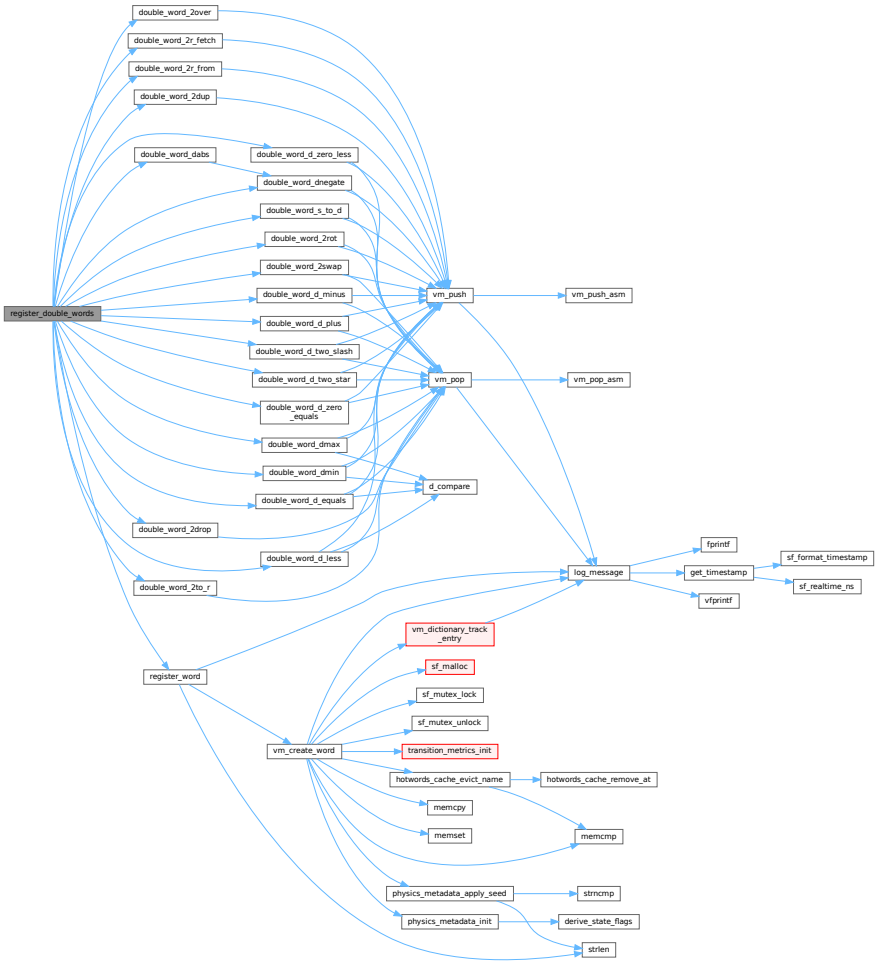
Note

This function must be called during VM initialization.

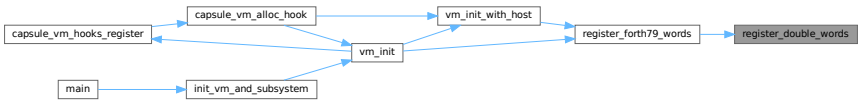
References double_word_2drop(), double_word_2dup(), double_word_2over(), double_word_2r_fetch(), double_word_2r_from(), double_word_2rot(), double_word_2swap(), double_word_2to_r(), double_word_d_equals(), double_word_d_less(), double_word_d_minus(), double_word_d_plus(), double_word_d_two_slash(), double_word_d_two_star(), double_word_d_zero_equals(), double_word_d_zero_less(), double_word_dabs(), double_word_dmax(), double_word_dmin(), double_word_dnegate(), double_word_s_to_d(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



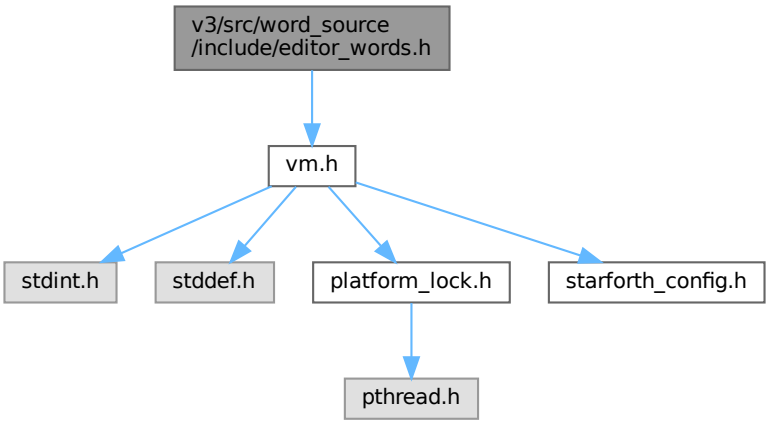
Here is the caller graph for this function:



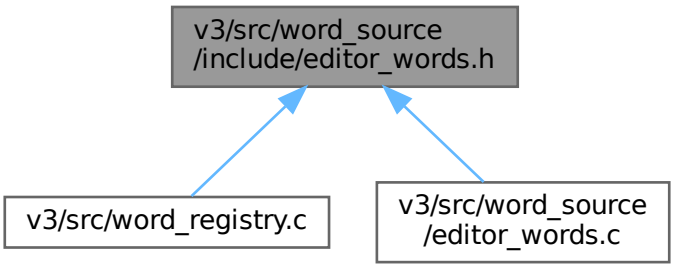
7.373 v3/src/word_source/include/editor_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for editor_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_editor_words (VM *vm)

  Register all editor words with the Forth virtual machine.

7.373.1 Function Documentation

7.373.1.1 register_editor_words()

```
void register_editor_words (
            VM * vm )
```

Register all editor words with the Forth virtual machine.

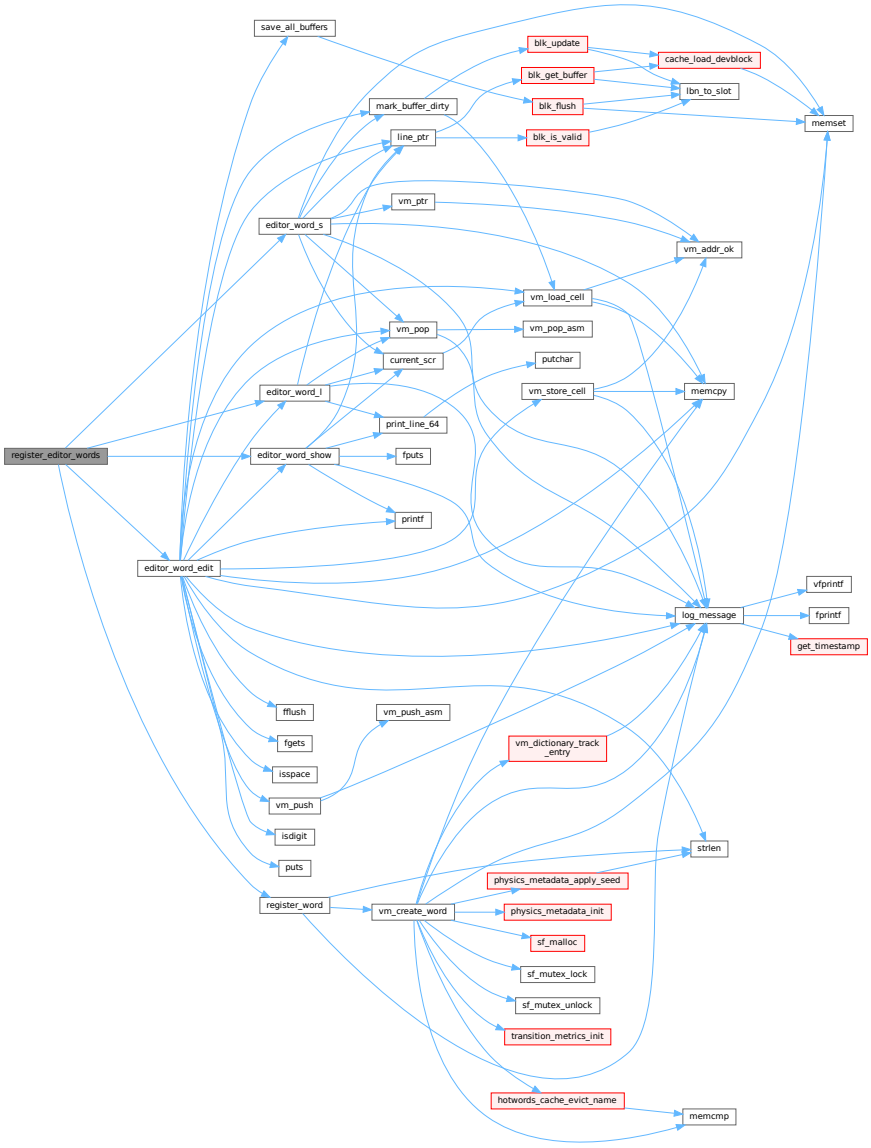
Parameters

vm	Pointer to the Forth virtual machine instance

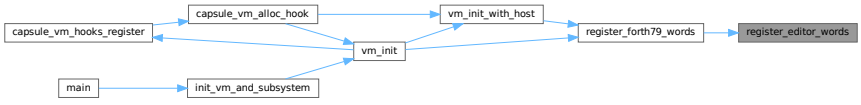
References editor_word_edit(), editor_word_l(), editor_word_s(), editor_word_show(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



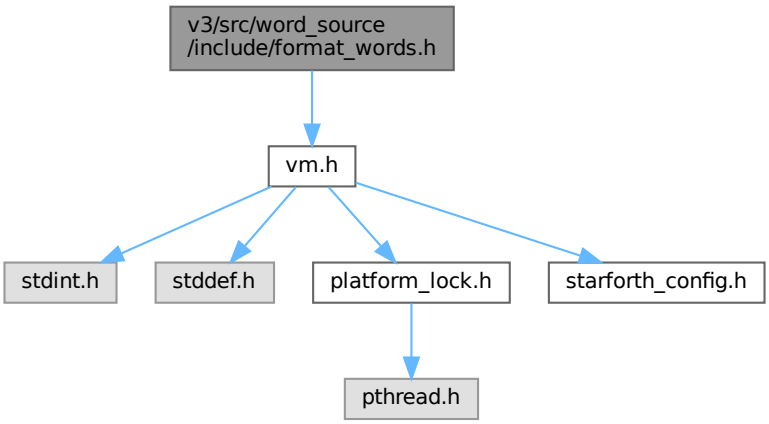
Here is the caller graph for this function:



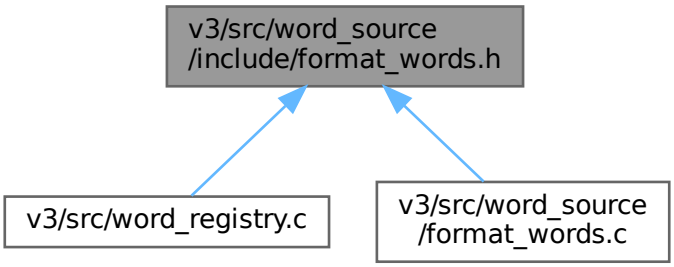
7.374 v3/src/word_source/include/format_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for format_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_format_words (VM *vm)

  Registers all formatting and numeric conversion words with the VM.

7.374.1 Function Documentation

7.374.1.1 register_format_words()

```
void register_format_words (
            VM * vm )
```

Registers all formatting and numeric conversion words with the VM.

This function registers the standard FORTH formatting words including:

- Number output words (., .R, U., U.R, D., D.R, .S, ?)
- Picture number formatting (<#, #, #S, #>, HOLD, SIGN)
- Base conversion (BASE, DECIMAL, HEX, OCTAL)
- Memory dump (DUMP)

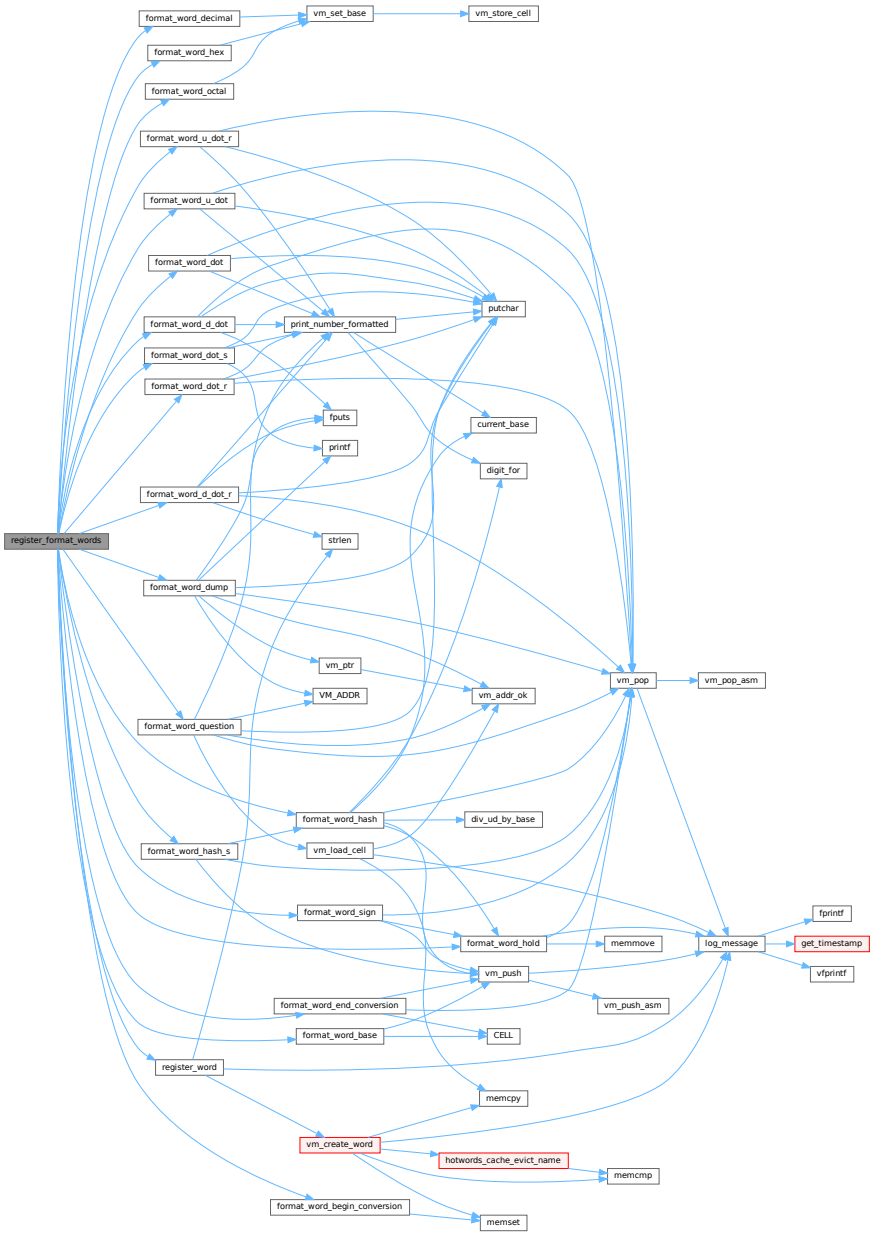
Parameters

vm	Pointer to the virtual machine instance

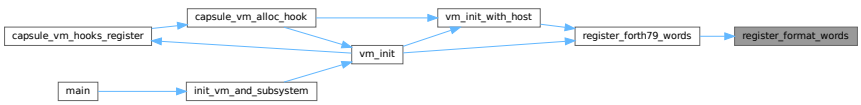
References format_word_base(), format_word_begin_conversion(), format_word_d_dot(), format_word_d_dot_r(), format_word_decimal(), format_word_dot(), format_word_dot_r(), format_word_dot_s(), format_word_dump(), format_word_end_conversion(), format_word_hash(), format_word_hash_s(), format_word_hex(), format_word_hold(), format_word_octal(), format_word_question(), format_word_sign(), format_word_u_dot(), format_word_u_dot_r(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



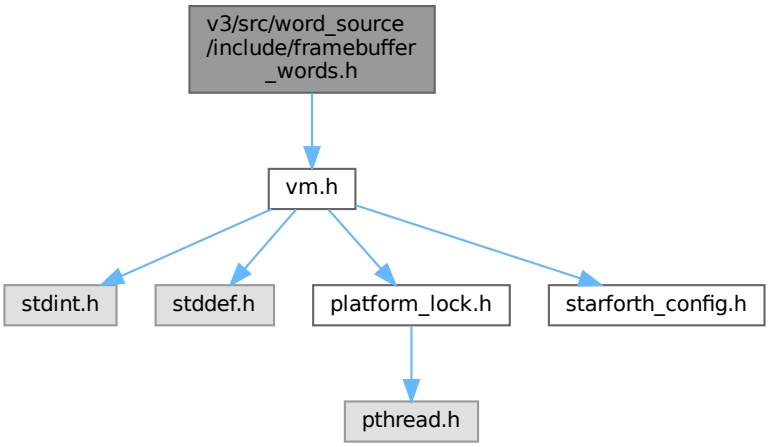
Here is the caller graph for this function:



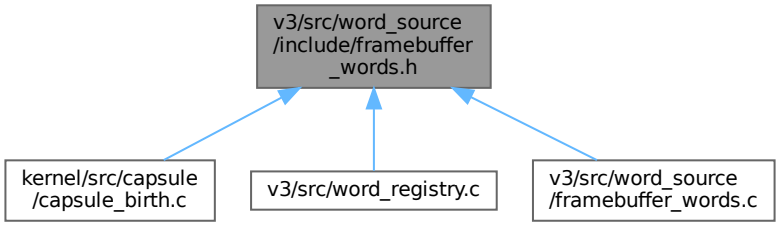
7.375 v3/src/word_source/include/framebuffer_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for framebuffer_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_framebuffer_words (VM *vm)

7.375.1 Function Documentation

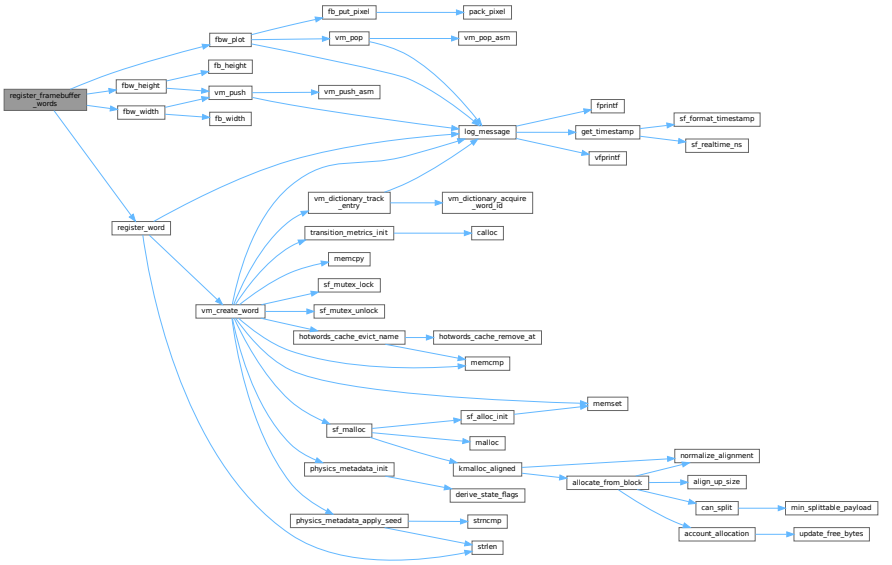
7.375.1.1 register_framebuffer_words()

```
void register_framebuffer_words (
            VM * vm )
```

References fbw_height(), fbw_plot(), fbw_width(), and register_word().

Referenced by capsule_birth_baby().

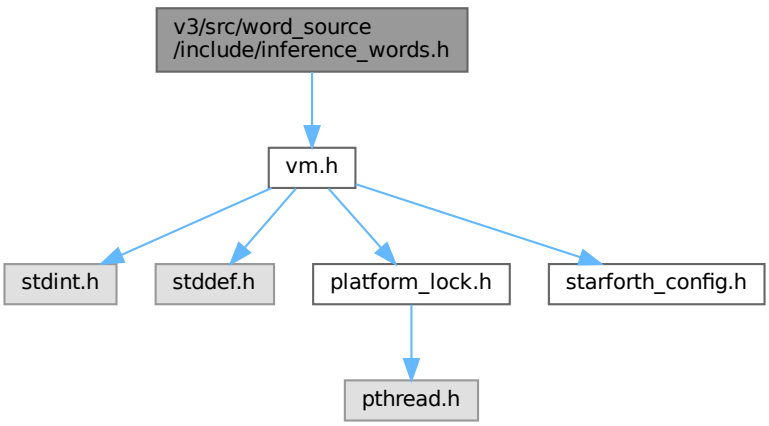
Here is the call graph for this function:



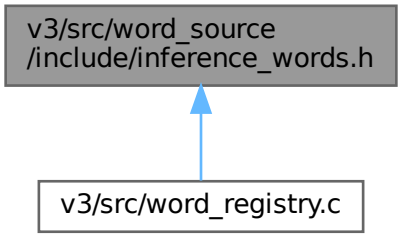
Here is the caller graph for this function:

7.376 v3/src/word_source/include/inference_words.h File Reference

`#include "vm.h"`
Include dependency graph for inference_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_inference_words (VM ∗vm)

  inference_words.h — Module 26: SSM inference engine + Jacquard FORTH words

7.376.1 Function Documentation

7.376.1.1 register_inference_words()

```
void register_inference_words (
            VM * vm )
```

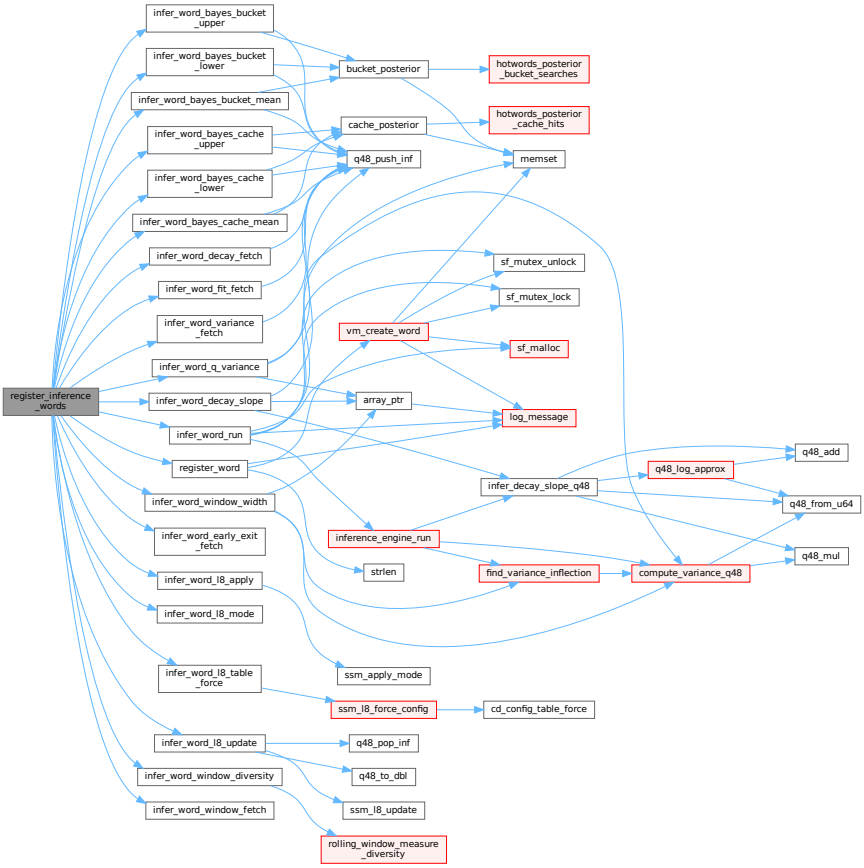
inference_words.h — Module 26: SSM inference engine + Jacquard FORTH words

Register all inference engine / Jacquard FORTH words with the VM. Module 26 in register_forth79_words().

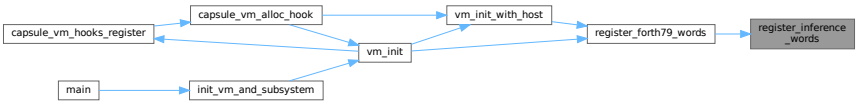
References infer_word_bayes_bucket_lower(), infer_word_bayes_bucket_mean(), infer_word_bayes_bucket_upper(), infer_word_bayes_cache_lower(), infer_word_bayes_cache_mean(), infer_word_bayes_cache_upper(), infer_word_decay_fetch(), infer_word_decay_slope(), infer_word_early_exit_fetch(), infer_word_fit_fetch(), infer_word_l8_apply(), infer_word_l8_mode(), infer_word_l8_table_force(), infer_word_l8_update(), infer_word_q_variance(), infer_word_run(), infer_word_variance_fetch(), infer_word_window_diversity(), infer_word_window_fetch(), infer_word_window_width(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



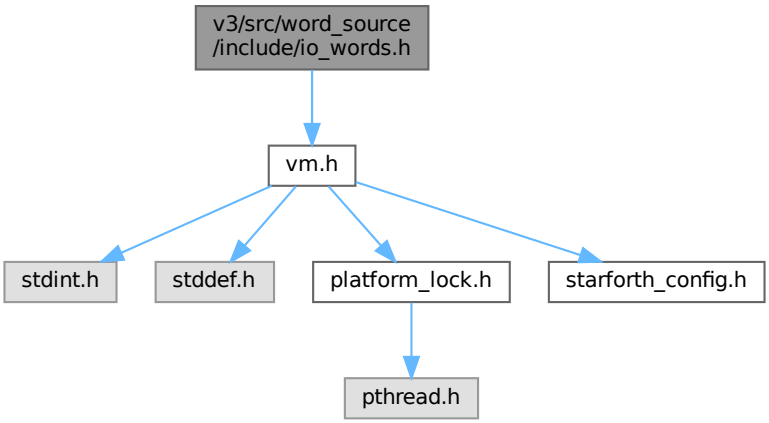
Here is the caller graph for this function:



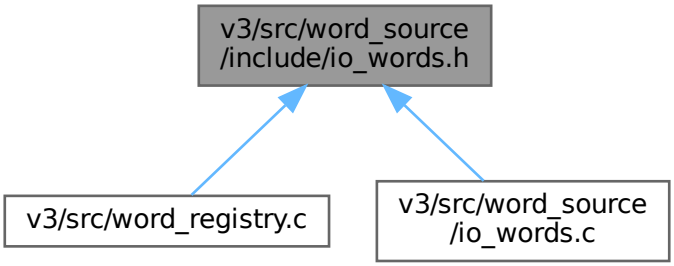
7.377 v3/src/word_source/include/io_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for io_words.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define IO_WORDS_H
  < *Include guard*

Functions

- void register_io_words (VM *vm)
  Registers all I/O related FORTH words with the virtual machine.

7.377.1 Macro Definition Documentation

7.377.1.1 IO_WORDS_H

```
#define IO_WORDS_H
```

< Include guard

Include guard

7.377.2 Function Documentation

7.377.2.1 register_io_words()

```
void register_io_words (
            VM * vm )
```

Registers all I/O related FORTH words with the virtual machine.

Parameters

vm	Pointer to the virtual machine instance

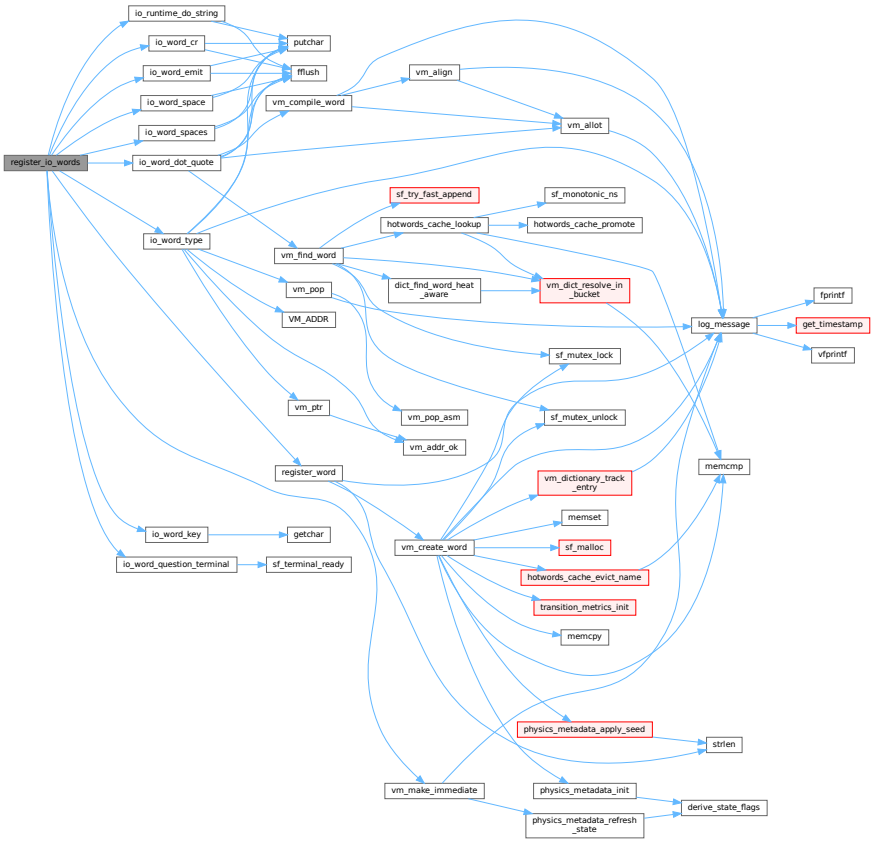
Registers the following FORTH-79 I/O & Terminal Words:

- EMIT (c –) Output character c
- CR (–) Output carriage return
- KEY (– c) Input character from keyboard
- ?TERMINAL (– flag) True if key pressed
- TYPE (addr u –) Output u characters from addr
- SPACE (–) Output one space
- SPACES (n –) Output n spaces

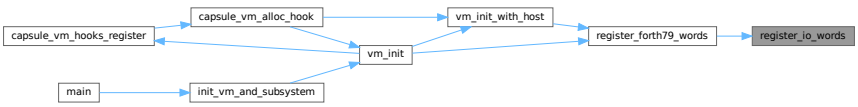
References io_runtime_do_string(), io_word_cr(), io_word_dot_quote(), io_word_emit(), io_word_key(), io_word_question_terminal(), io_word_space(), io_word_spaces(), io_word_type(), register_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



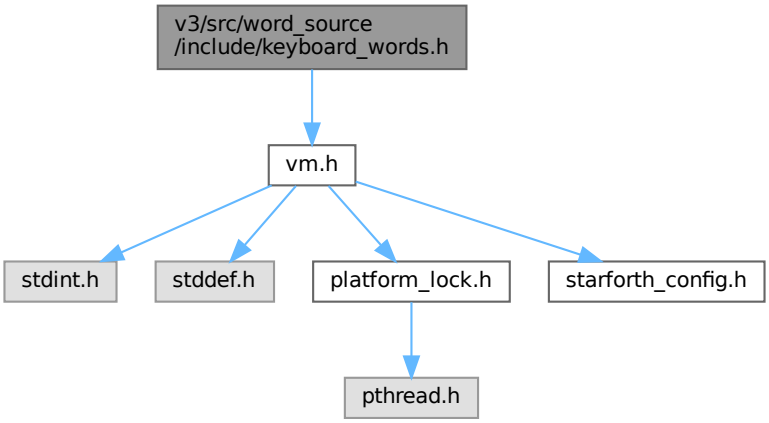
Here is the caller graph for this function:



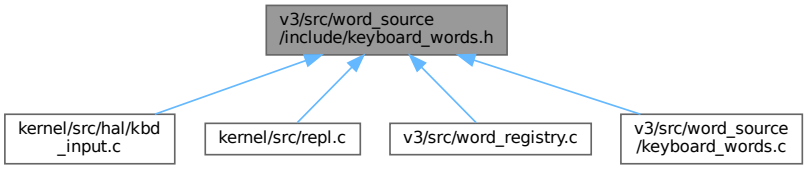
7.378 v3/src/word_source/include/keyboard_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for keyboard_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_keyboard_words (VM *vm)
- int sk_key_event_poll (uint16_t *keycode, int *pressed)

  FABRIC-0.md item 4.4v: shared C-level poll, one converged event stream (Linux input keycode namespace, all three architectures) – see KEY-EVENT's doc comment above for the full per-arch translation rationale.

7.378.1 Function Documentation

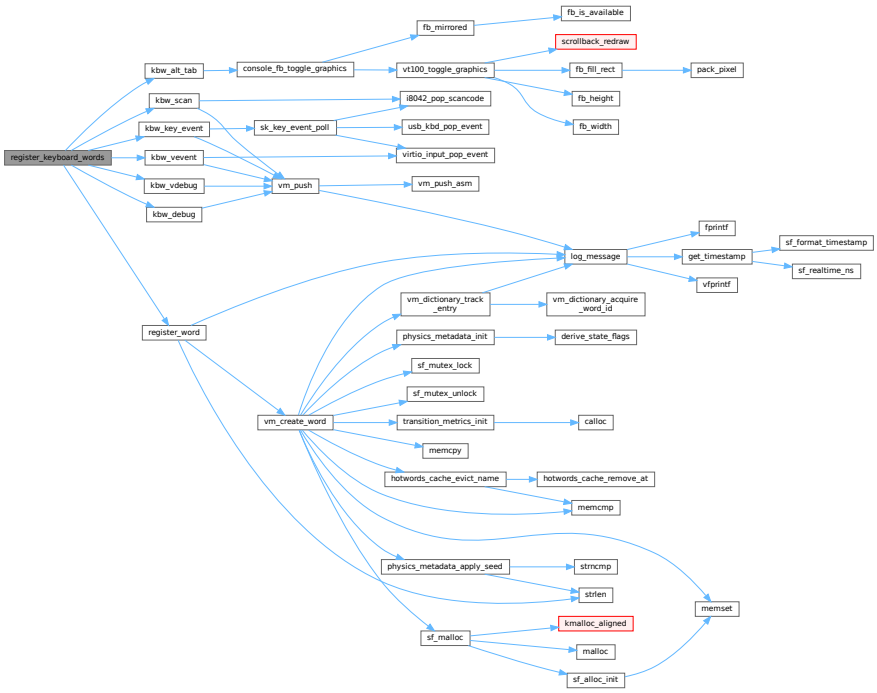
7.378.1.1 register_keyboard_words()

```
void register_keyboard_words (
            VM * vm )
```

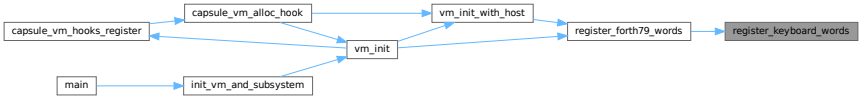
References kbw_alt_tab(), kbw_debug(), kbw_key_event(), kbw_scan(), kbw_vdebug(), kbw_vevent(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.378.1.2 sk_key_event_poll()

```
int sk_key_event_poll (
            uint16_t * keycode,
            int * pressed )
```

FABRIC-0.md item 4.4v: shared C-level poll, one converged event stream (Linux input keycode namespace, all three architectures) – see KEY-EVENT's doc comment above for the full per-arch translation rationale.

Used by both kbw_key_event() (the FORTH word) and the REPL keyboard bridge (repl.c), so there is exactly one implementation.

Parameters

keycode	Written with the Linux input keycode on success.
pressed	Written with 1 (press/repeat) or 0 (release) on success.

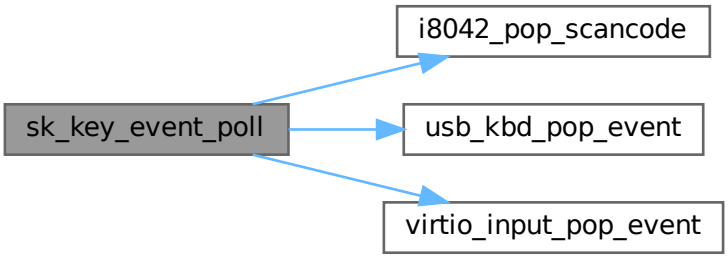
Returns

1 if an event was available, 0 otherwise.

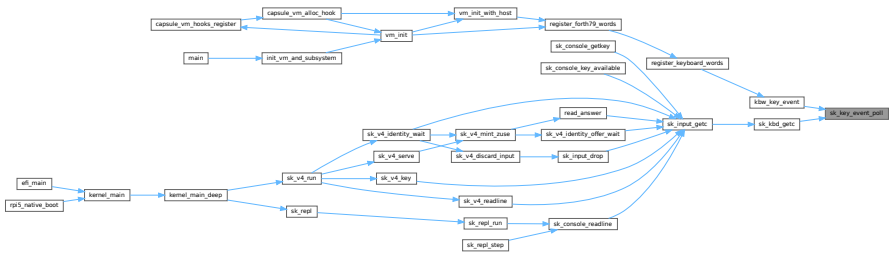
References i8042_pop_scancode(), keycode, pressed, usb_kbd_pop_event(), and virtio_input_pop_event().

Referenced by kbw_key_event(), and sk_kbd_getc().

Here is the call graph for this function:



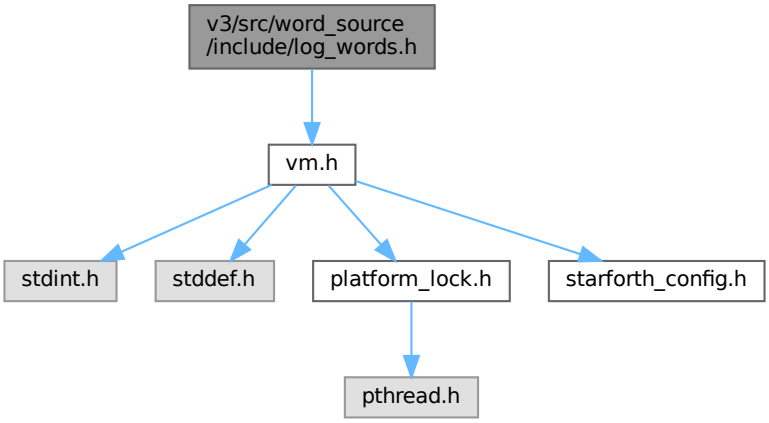
Here is the caller graph for this function:



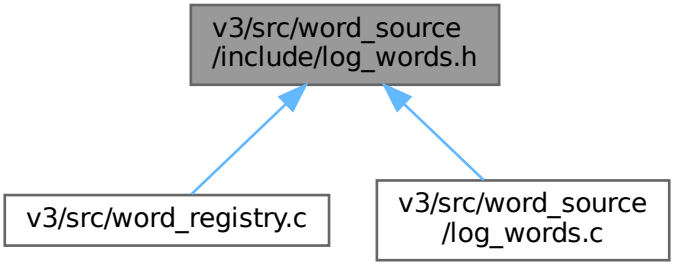
7.379 v3/src/word_source/include/log_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for log_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_log_words (VM *vm)

  Register all log words with the VM dictionary.

7.379.1 Function Documentation

7.379.1.1 register_log_words()

```
void register_log_words (
            VM * vm )
```

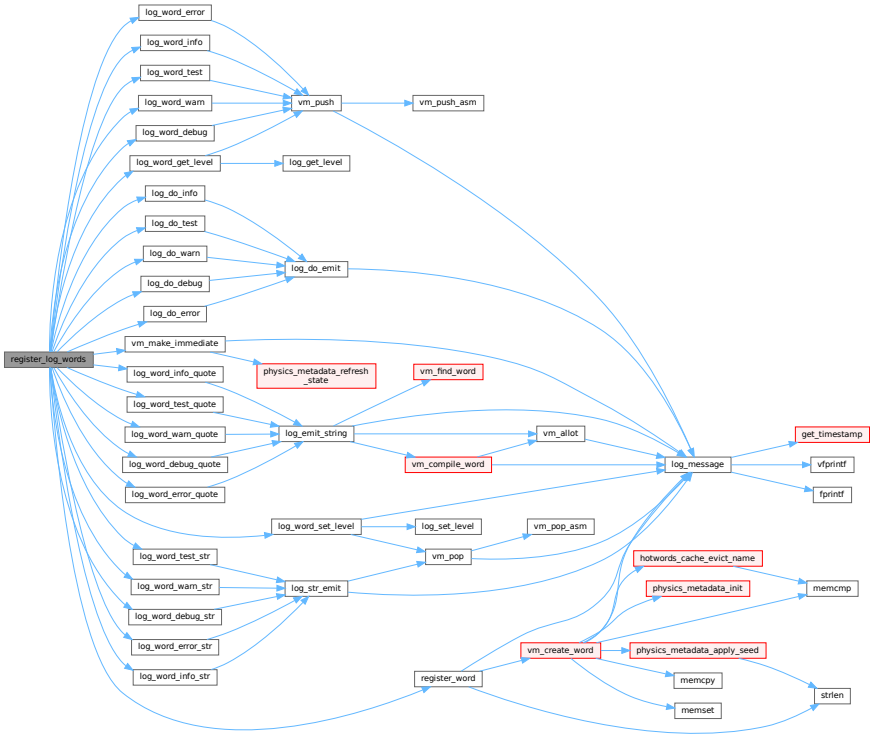
Register all log words with the VM dictionary.

Level constants: LOG-ERROR LOG-WARN LOG-INFO LOG-TEST LOG-DEBUG Level control: LOG-LEVEL! LOG-LEVEL@ Runtime (hidden): (do-log-error) (do-log-warn) (do-log-info) (do-log-test) (do-log-debug) String literals: LOG-ERROR" LOG-WARN" LOG-INFO" LOG-TEST" LOG-DEBUG" Stack string: LOG-ERROR-STR LOG-WARN-STR LOG-INFO-STR LOG-TEST-STR LOG-DEBUG-STR Kernel only: (LOG-APPEND-RAW) LOG-FLUSH

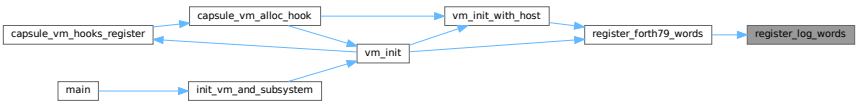
References log_do_debug(), log_do_error(), log_do_info(), log_do_test(), log_do_warn(), log_word_debug(), log_word_debug_quote(), log_word_debug_str(), log_word_error(), log_word_error_quote(), log_word_error_str(), log_word_get_level(), log_word_info(), log_word_info_quote(), log_word_info_str(), log_word_set_level(), log_word_test(), log_word_test_quote(), log_word_test_str(), log_word_warn(), log_word_warn_quote(), log_word_warn_str(), register_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



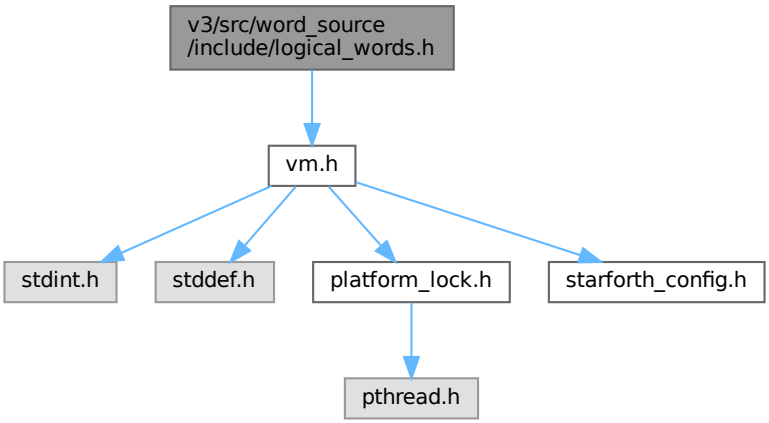
Here is the caller graph for this function:



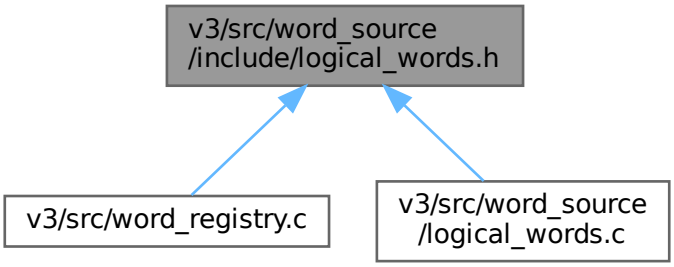
7.380 v3/src/word_source/include/logical_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for logical_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_logical_words (VM *vm)

  Registers all FORTH-79 logical and comparison words with the VM.

7.380.1 Function Documentation

7.380.1.1 register_logical_words()

```
void register_logical_words (
            VM * vm )
```

Registers all FORTH-79 logical and comparison words with the VM.

Parameters

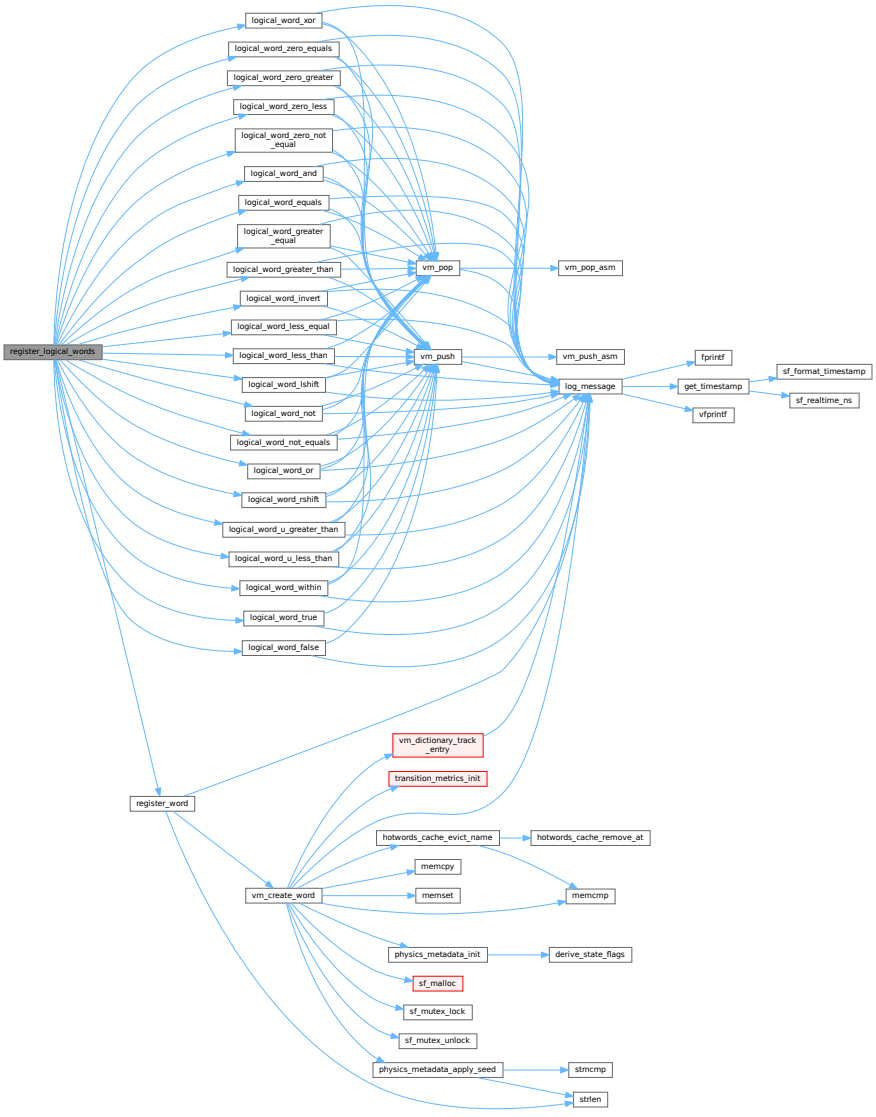
vm	Pointer to the virtual machine instance

Registers logical operations (AND, OR, XOR, NOT), comparison operations (=, <>, <, >, etc.), and constant words (TRUE, FALSE) with the virtual machine.

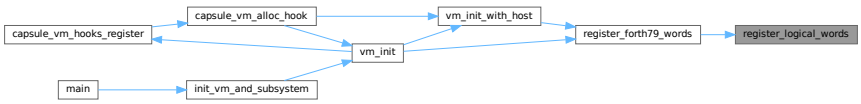
References logical_word_and(), logical_word_equals(), logical_word_false(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_true(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

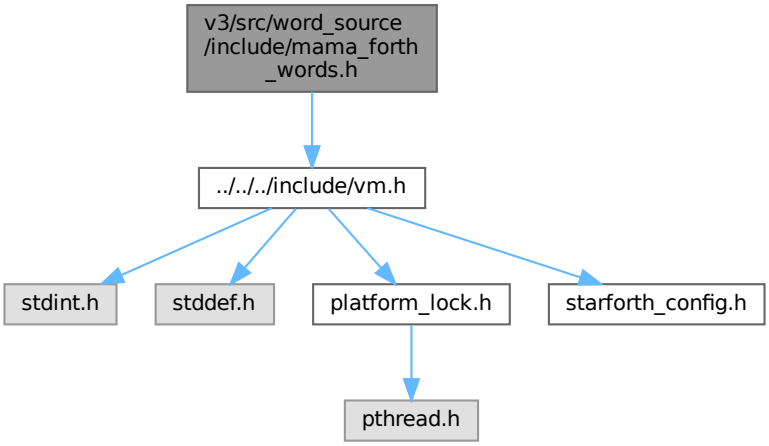


7.381 v3/src/word_source/include/mama_forth_words.h File Reference

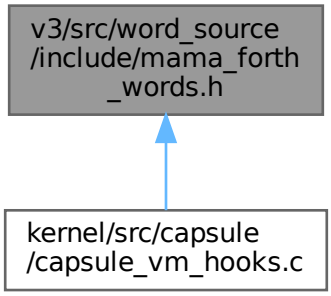
Mama FORTH vocabulary words for kernel capsule system (M7.1)

```
#include "../../../include/vm.h"
```

Include dependency graph for mama_forth_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- static void register_mama_forth_words (VM *vm)

  Stub for hosted builds - does nothing.

7.381.1 Detailed Description

Mama FORTH vocabulary words for kernel capsule system (M7.1)

This header defines the Mama FORTH vocabulary - kernel-only words for the LithosAnanke capsule birth protocol.

These words are ONLY available in **STARKERNEL** builds. Hosted StarForth builds do not include capsule functionality.

Mama FORTH Vocabulary: CAPSULE-COUNT (– n) Number of capsules in directory CAPSULE@ (idx – desc) Get descriptor address by index CAPSULE-HASH@ (desc – hash) Get capsule content hash CAPSULE-FLAGS@ (desc – flags) Get capsule flags CAPSULE-LEN@ (desc – len) Get payload length CAPSULE-BIRTH (hash – vm-id) Birth baby VM from (p) capsule CAPSULE-RUN (hash –) Run (e) capsule on Mama MAMA-VM-ID (– 0) Mama's VM ID (always 0) VM-COUNT (– n) Number of live VMs

7.381.2 Function Documentation

7.381.2.1 register_mama_forth_words()

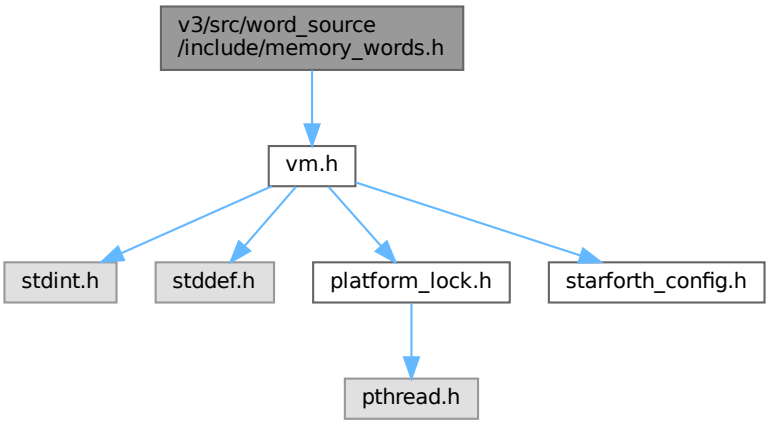
```
static void register_mama_forth_words (
            VM * vm )  [inline], [static]
```

Stub for hosted builds - does nothing.

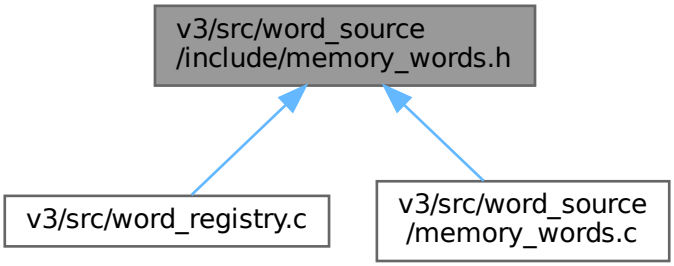
7.382 v3/src/word_source/include/memory_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for memory_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_memory_words (VM *vm)

  Register all memory manipulation words with the VM.

7.382.1 Function Documentation

7.382.1.1 register_memory_words()

```
void register_memory_words (
            VM * vm )
```

Register all memory manipulation words with the VM.

Registers all standard memory words (@, !, C@, C!, etc.) with the VM's word dictionary.

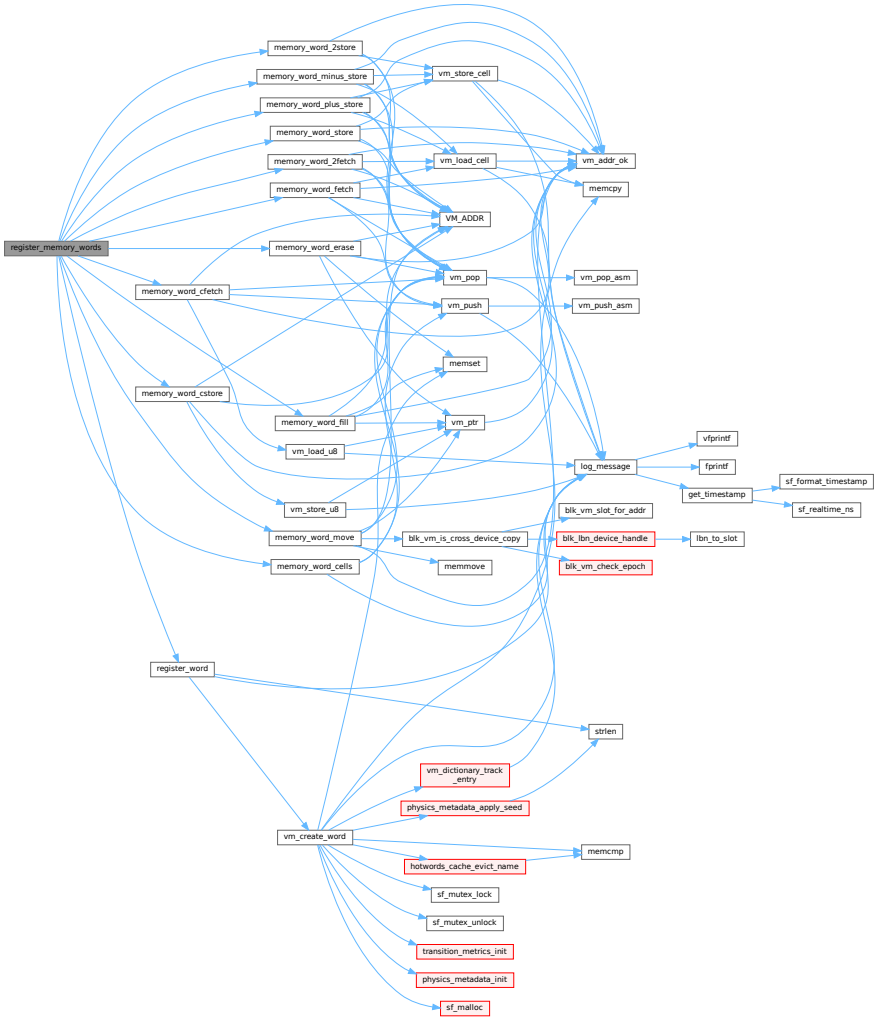
Parameters

vm	Pointer to the VM structure

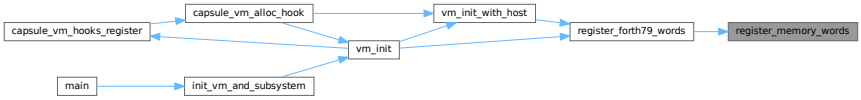
References memory_word_2fetch(), memory_word_2store(), memory_word_cells(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



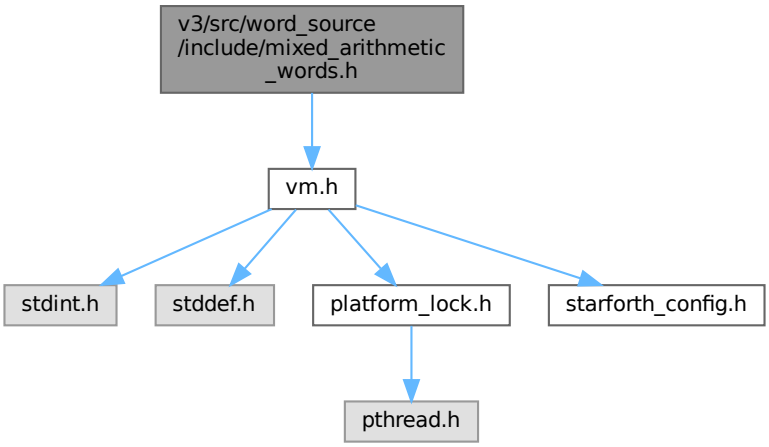
Here is the caller graph for this function:



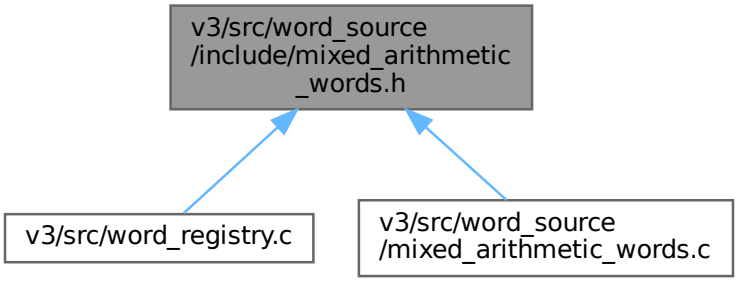
7.383 v3/src/word_source/include/mixed_arithmetic_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for mixed_arithmetic_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_mixed_arithmetic_words (VM *vm)

  Register all FORTH-79 mixed arithmetic words with the VM dictionary.

7.383.1 Function Documentation

7.383.1.1 register_mixed_arithmetic_words()

```
void register_mixed_arithmetic_words (
            VM * vm )
```

Register all FORTH-79 mixed arithmetic words with the VM dictionary.

Registers: `M+`, `M-`, `M*`, `M/MOD`, `MOD`, `/MOD`, `*\/`, `*\/MOD`. Called during VM bootstrap.

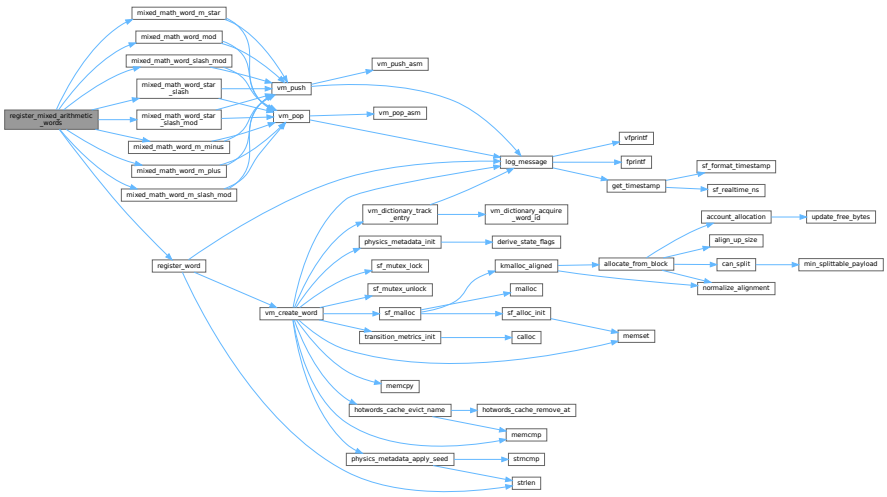
Parameters

vm	Active VM to register words into

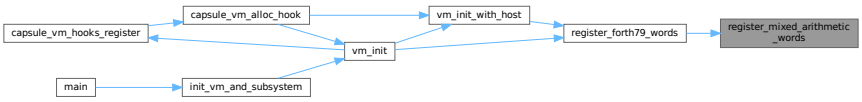
References mixed_math_word_m_minus(), mixed_math_word_m_plus(), mixed_math_word_m_slash_mod(), mixed_math_word_m_star(), mixed_math_word_mod(), mixed_math_word_slash_mod(), mixed_math_word_star_slash(), mixed_math_word_star_slash_mod(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



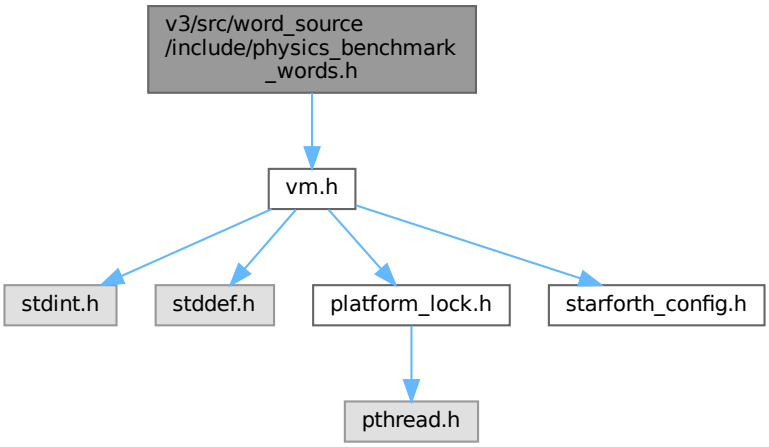
Here is the caller graph for this function:



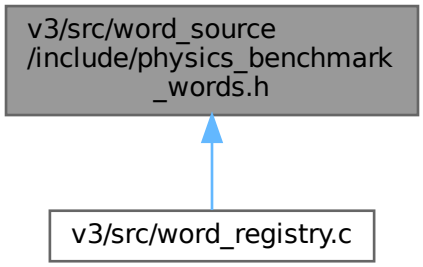
7.384 v3/src/word_source/include/physics_benchmark_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for physics_benchmark_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_physics_benchmark_words (VM *vm)

  Register all physics benchmark words with the VM.

7.384.1 Function Documentation

7.384.1.1 register_physics_benchmark_words()

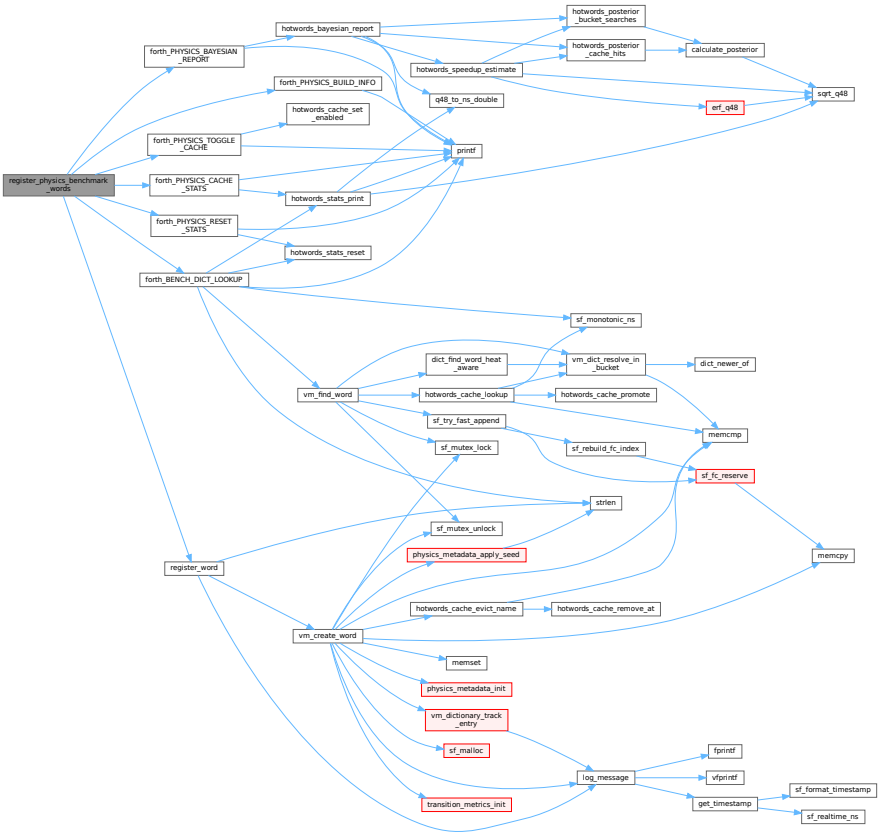
```
void register_physics_benchmark_words (
            VM * vm )
```

Register all physics benchmark words with the VM.

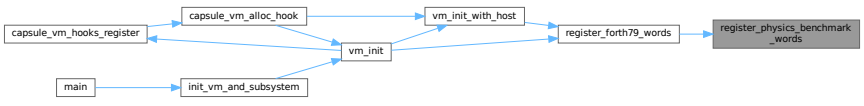
References forth_BENCH_DICT_LOOKUP(), forth_PHYSICS_BAYESIAN_REPORT(), forth_PHYSICS_BUILD_INFO(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_RESET_STATS(), forth_PHYSICS_TOGGLE_CACHE(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



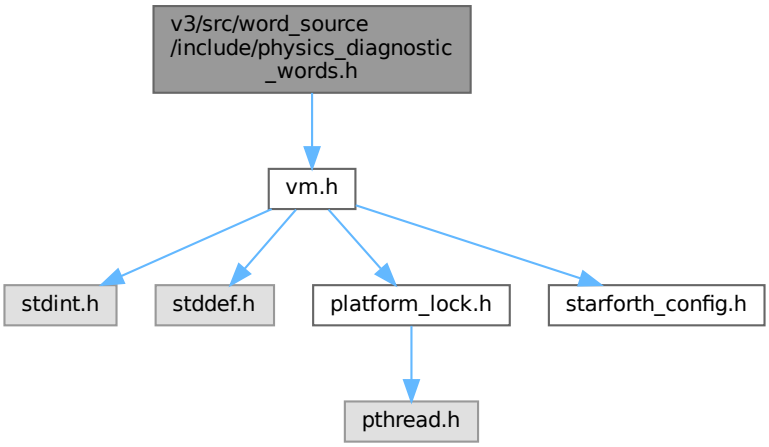
Here is the caller graph for this function:



7.385 v3/src/word_source/include/physics_diagnostic_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for physics_diagnostic_words.h:



Functions

- void forth_PHYSICS_WORD_METRICS (VM *vm)

  Display detailed physics metrics for the most recently executed word.
- void forth_PHYSICS_CALC_KNOBS (VM *vm)

  Calculate and show knob adjustments based on current thermal metrics.
- void forth_PHYSICS_BURN (VM *vm)

  Execute a word repeatedly (burn) and show thermal feedback.
- void forth_PHYSICS_SHOW_FEEDBACK (VM *vm)

  Display the complete feedback loop: inputs → math → knobs → effect.
- void register_physics_diagnostic_words (VM *vm)

  Register all physics diagnostic words with the VM.

7.385.1 Function Documentation

7.385.1.1 forth_PHYSICS_BURN()

```
void forth_PHYSICS_BURN (
            VM * vm )
```

Execute a word repeatedly (burn) and show thermal feedback.

Execute a word repeatedly (burn) and show thermal feedback.

Execute the top-of-stack word n times and show thermal feedback.

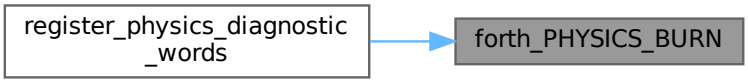
Usage: : HOTTEST 100 0 DO 1 2 + 3 * LOOP ; HOTTEST 10 PHYSICS-BURN (execute 10 times) PHYSICS-↩ SHOW-FEEDBACK

References VM::data_stack, VM::dsp, VM::error, DictEntry::execution_heat, DictEntry::func, DictPhysics::last_active_ns, VM::latest, DictEntry::name, DictEntry::physics, printf(), and DictPhysics::temperature_q8.

Referenced by register_physics_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



7.385.1.2 forth_PHYSICS_CALC_KNOBS()

```
void forth_PHYSICS_CALC_KNOBS (
            VM * vm )
```

Calculate and show knob adjustments based on current thermal metrics.

Calculate and show knob adjustments based on current thermal metrics.

Calculate what knobs should be adjusted based on current word metrics. Shows:

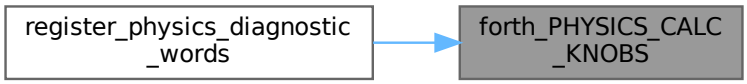
- Thermal pressure (0.0 to 1.0)
- Recommended priority adjustment
- Recommended sampling rate
- Stack depth limit
- Core affinity preference

References VM::error, DictPhysics::last_active_ns, VM::latest, DictEntry::name, DictEntry::physics, printf(), and DictPhysics::temperature_q8.

Referenced by register_physics_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



7.385.1.3 forth_PHYSICS_SHOW_FEEDBACK()

```
void forth_PHYSICS_SHOW_FEEDBACK (
            VM * vm )
```

Display the complete feedback loop: inputs → math → knobs → effect.

Display the complete feedback loop: inputs → math → knobs → effect.

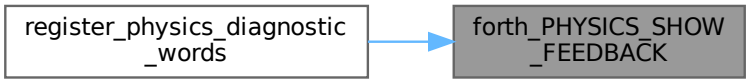
Display the complete feedback loop for the most recent word. Shows: Metrics → Calculations → Knobs → Effect

References DictPhysics::avg_latency_ns, VM::error, DictEntry::execution_heat, DictPhysics::last_active_ns, VM::latest, DictPhysics::mass_bytes, DictEntry::name, DictEntry::physics, printf(), and DictPhysics::temperature_q8.

Referenced by register_physics_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



7.385.1.4 forth_PHYSICS_WORD_METRICS()

```
void forth_PHYSICS_WORD_METRICS (
            VM * vm )
```

Display detailed physics metrics for the most recently executed word.

Display detailed physics metrics for the most recently executed word.

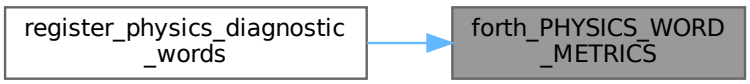
Display detailed physics metrics for a word. Looks up the most recently executed word and shows its state.

References DictPhysics::avg_latency_ns, VM::error, DictEntry::execution_heat, DictPhysics::last_active_ns, VM::latest, DictPhysics::mass_bytes, DictEntry::name, DictEntry::physics, printf(), and DictPhysics::temperature_q8.

Referenced by register_physics_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



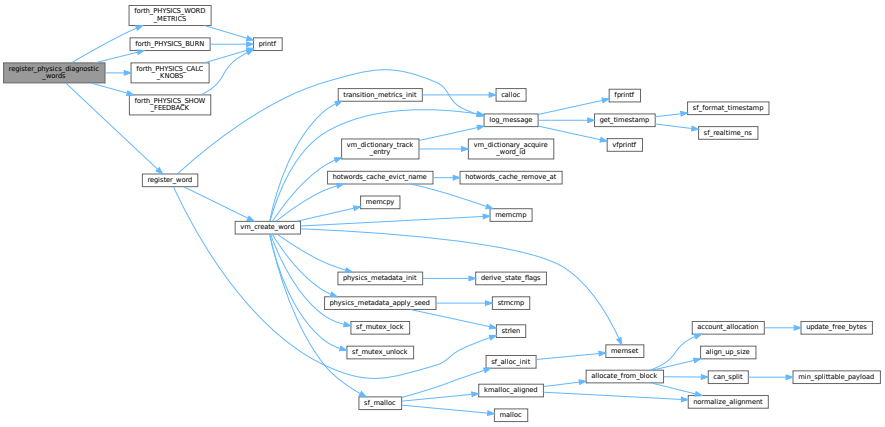
7.385.1.5 register_physics_diagnostic_words()

```
void register_physics_diagnostic_words (
            VM * vm )
```

Register all physics diagnostic words with the VM.

References forth_PHYSICS_BURN(), forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_SHOW_FEEDBACK(), forth_PHYSICS_WORD_METRICS(), and register_word().

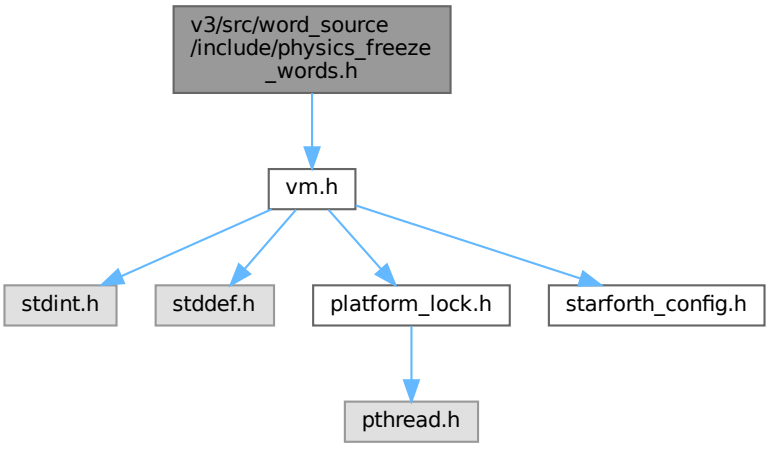
Here is the call graph for this function:



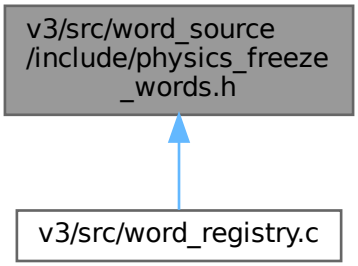
7.386 v3/src/word_source/include/physics_freeze_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for physics_freeze_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_physics_freeze_words (VM *vm)

  Register all Phase 2 freeze/decay control words with the VM.

7.386.1 Function Documentation

7.386.1.1 register_physics_freeze_words()

```
void register_physics_freeze_words (
            VM * vm )
```

Register all Phase 2 freeze/decay control words with the VM.

Register all Phase 2 freeze/decay control words with the VM.

Registers: FREEZE-WORD, UNFREEZE-WORD, FROZEN?, HEAT!, HEAT@, SHOW-HEAT, ALL-HEATS, DECAY-RATE@, FREEZE-CRITICAL. Called during VM bootstrap by the word registration subsystem.

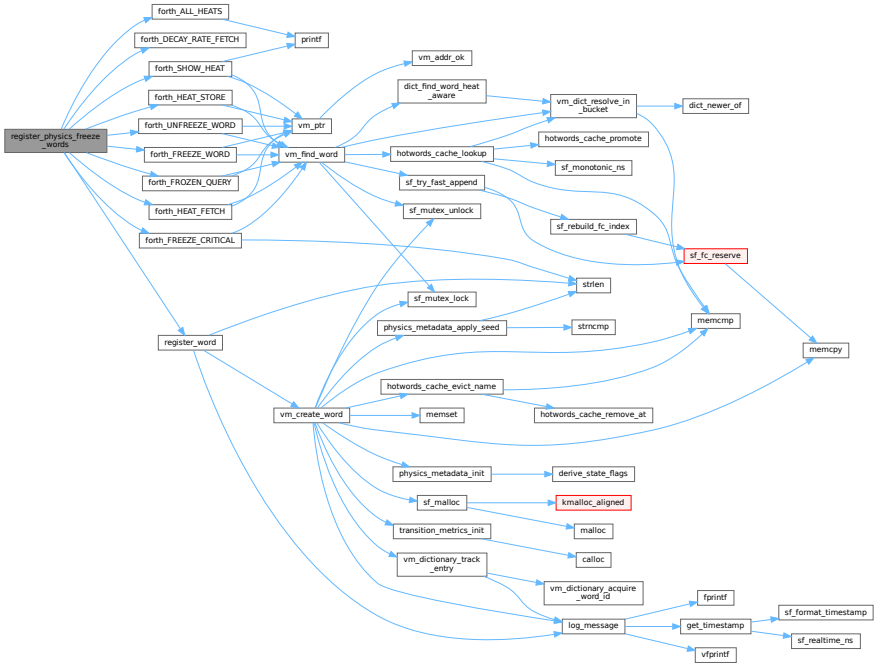
Parameters

vm	Active VM to register words into

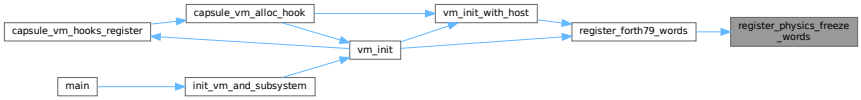
References forth_ALL_HEATS(), forth_DECAY_RATE_FETCH(), forth_FREEZE_CRITICAL(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_STORE(), forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



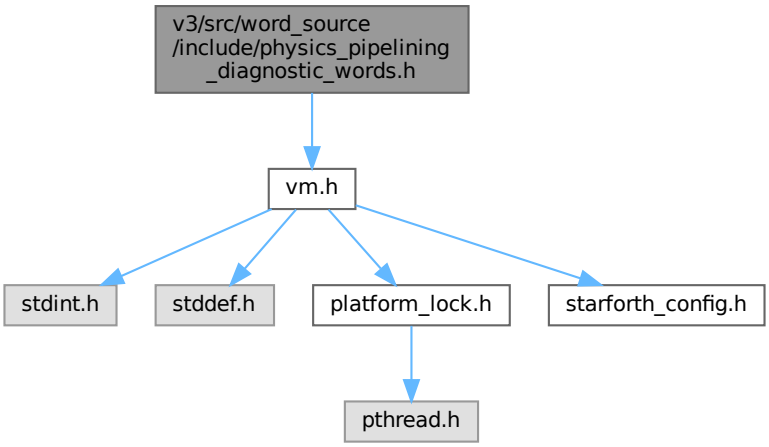
Here is the caller graph for this function:



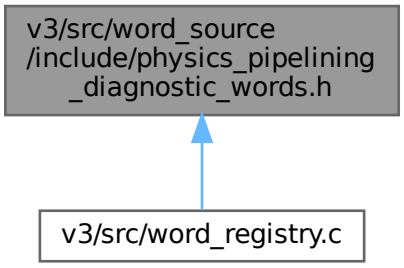
7.387 v3/src/word_source/include/physics_pipelining_diagnostic_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for physics_pipelining_diagnostic_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_physics_pipelining_diagnostic_words (VM *vm)

  Register all pipelining diagnostic words Called during VM initialization.

7.387.1 Function Documentation

7.387.1.1 register_physics_pipelining_diagnostic_words()

```
void register_physics_pipelining_diagnostic_words (
            VM * vm )
```

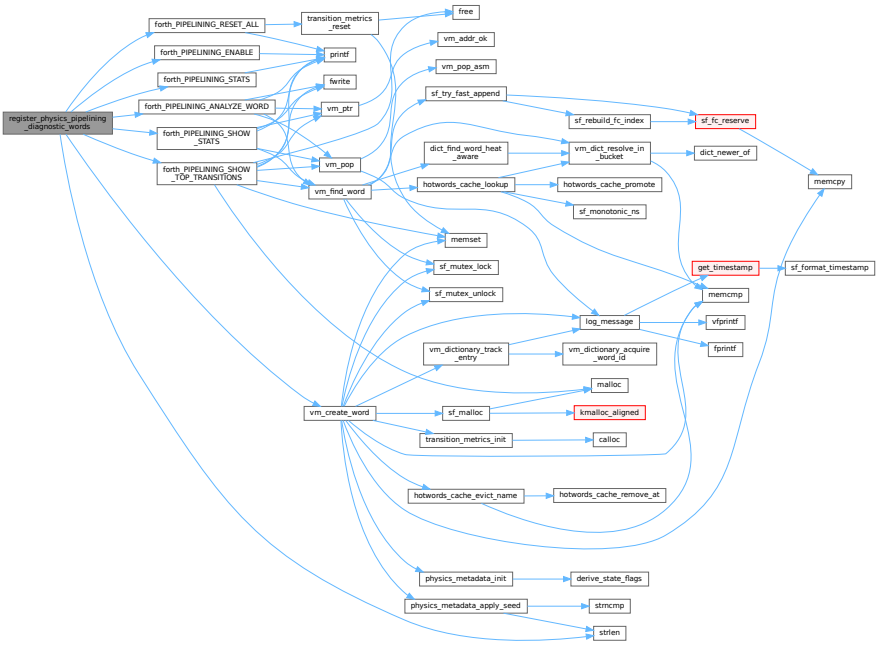
Register all pipelining diagnostic words Called during VM initialization.

Register all pipelining diagnostic words Called during VM initialization.

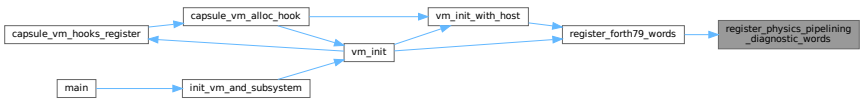
References forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_ENABLE(), forth_PIPELINING_RESET_ALL(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), forth_PIPELINING_STATS(), strlen(), and vm_create_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



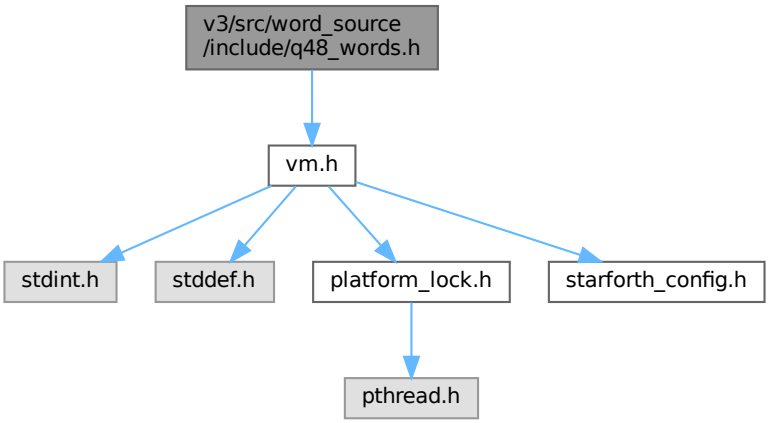
Here is the caller graph for this function:



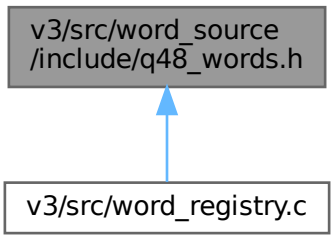
7.388 v3/src/word_source/include/q48_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for q48_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_q48_words (VM *vm)

  q48_words.h — Module 25: Q48.16 fixed-point FORTH words

7.388.1 Function Documentation

7.388.1.1 register_q48_words()

```
void register_q48_words (
            VM * vm )
```

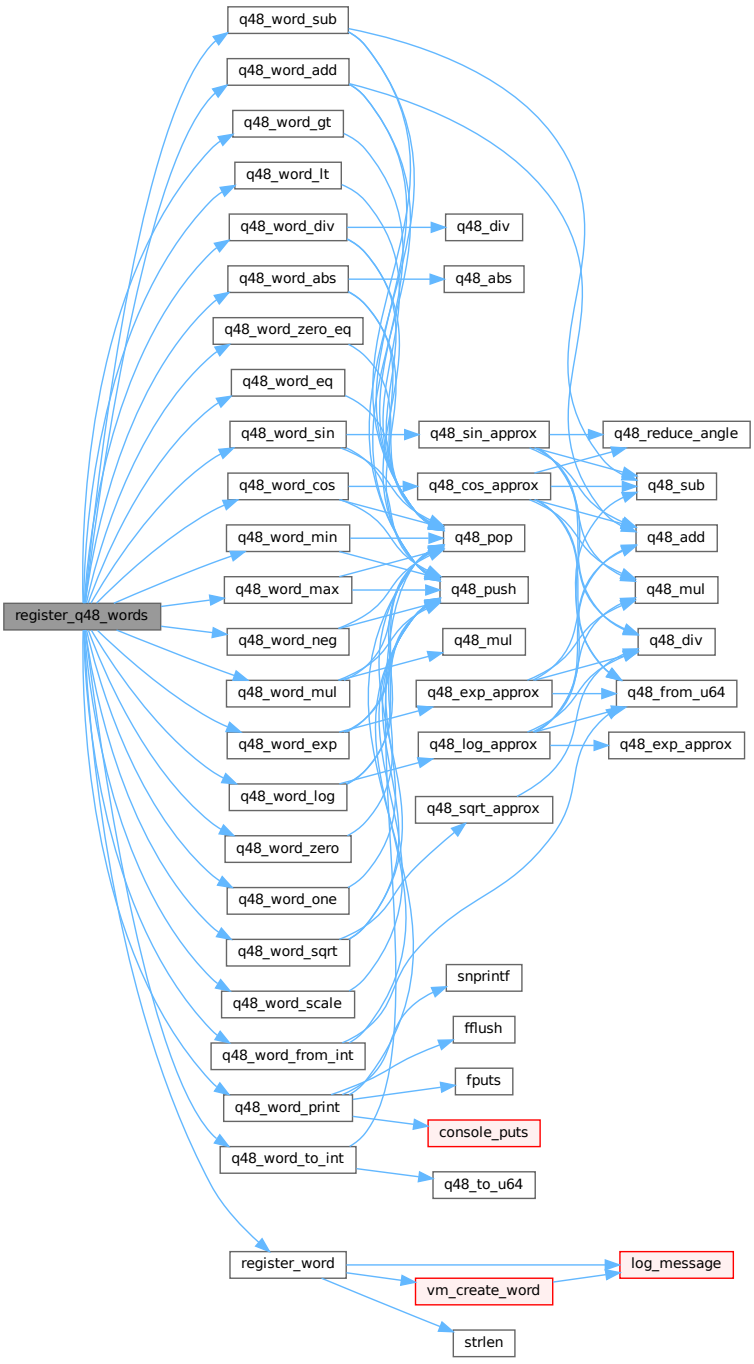
q48_words.h — Module 25: Q48.16 fixed-point FORTH words

Register all Q48.16 FORTH words with the VM. Module 25 in register_forth79_words().

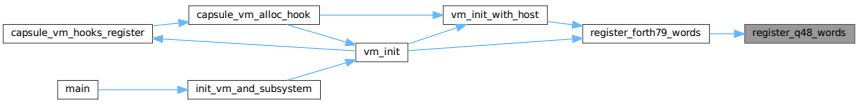
References q48_word_abs(), q48_word_add(), q48_word_cos(), q48_word_div(), q48_word_eq(), q48_word_exp(), q48_word_from_int(), q48_word_gt(), q48_word_log(), q48_word_lt(), q48_word_max(), q48_word_min(), q48_word_mul(), q48_word_neg(), q48_word_one(), q48_word_print(), q48_word_scale(), q48_word_sin(), q48_word_sqrt(), q48_word_sub(), q48_word_to_int(), q48_word_zero(), q48_word_zero_eq(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:

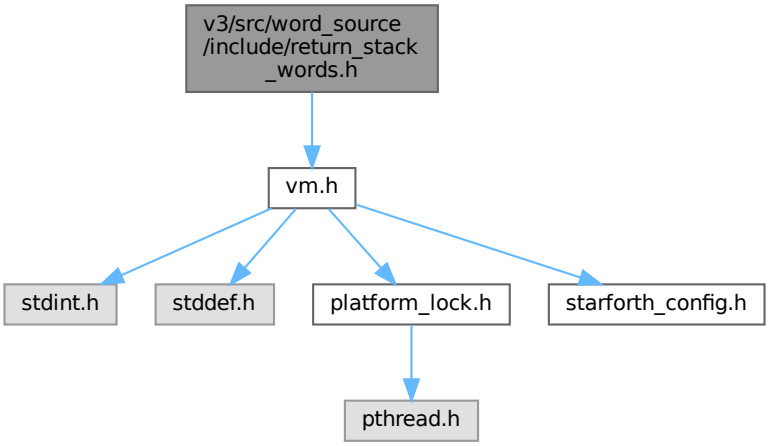


Here is the caller graph for this function:

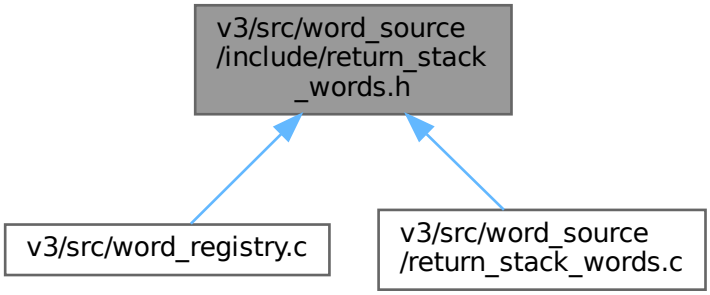


7.389 v3/src/word_source/include/return_stack_words.h File Reference

`#include "vm.h"`
Include dependency graph for return_stack_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_return_stack_words (VM *vm)

  Register all FORTH-79 return stack manipulation words.

7.389.1 Function Documentation

7.389.1.1 register_return_stack_words()

```
void register_return_stack_words (
            VM * vm )
```

Register all FORTH-79 return stack manipulation words.

Parameters

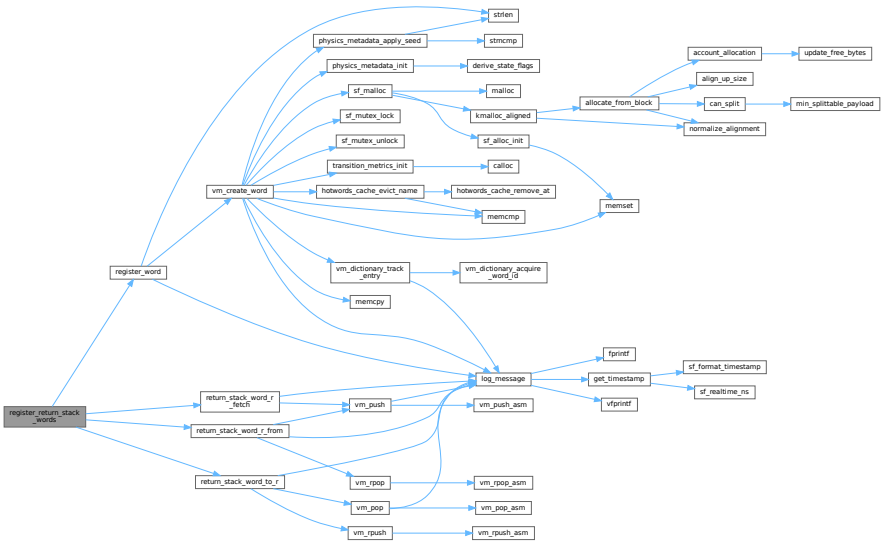
vm	Pointer to VM instance

Registers >R, R>, and R@ words. Direct stack addressing words (RP!, RP@) are deliberately omitted for FORTH-79 compliance.

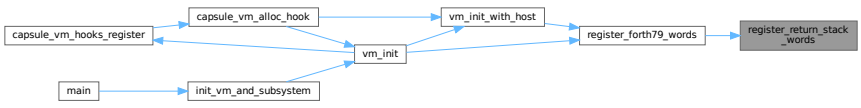
References register_word(), return_stack_word_r_fetch(), return_stack_word_r_from(), and return_stack_word_to_r().

Referenced by register_forth79_words().

Here is the call graph for this function:



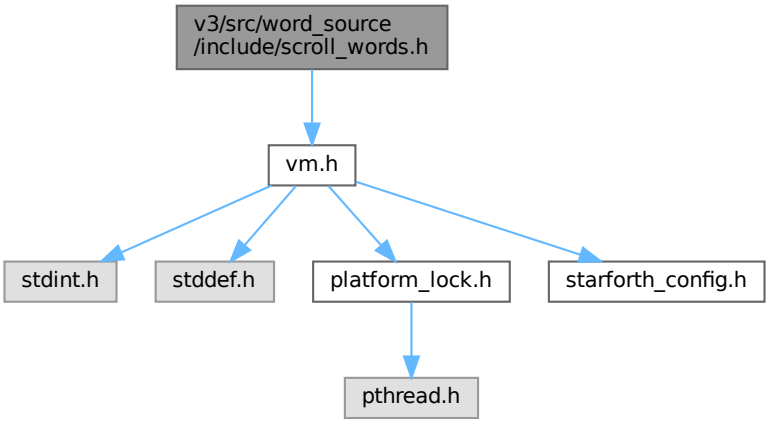
Here is the caller graph for this function:



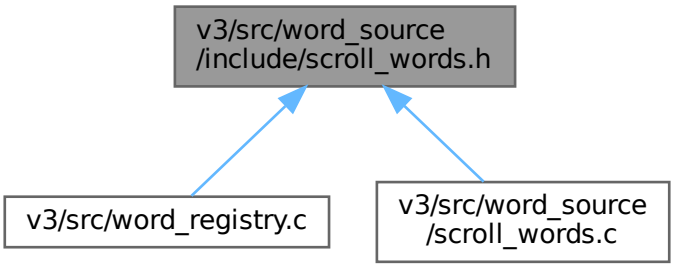
7.390 v3/src/word_source/include/scroll_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for scroll_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_scroll_words (VM *vm)

7.390.1 Function Documentation

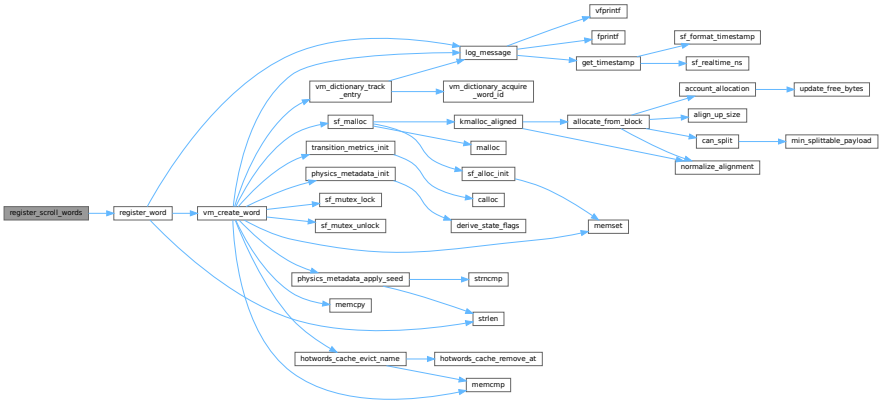
7.390.1.1 register_scroll_words()

```
void register_scroll_words (
            VM * vm )
```

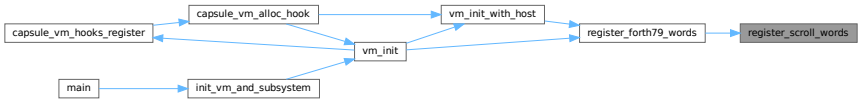
References register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



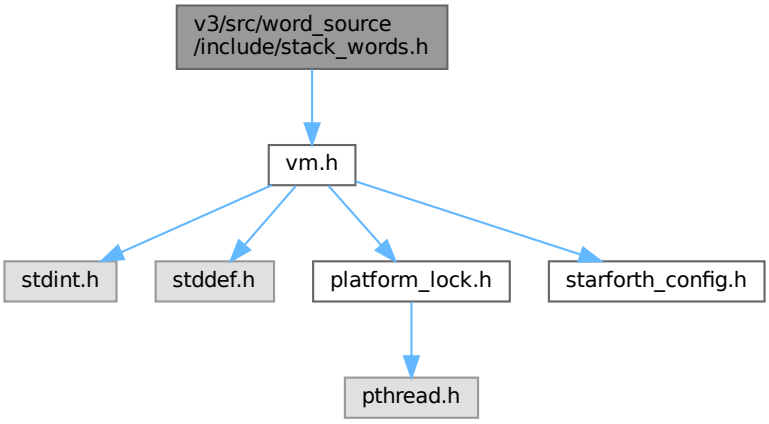
Here is the caller graph for this function:



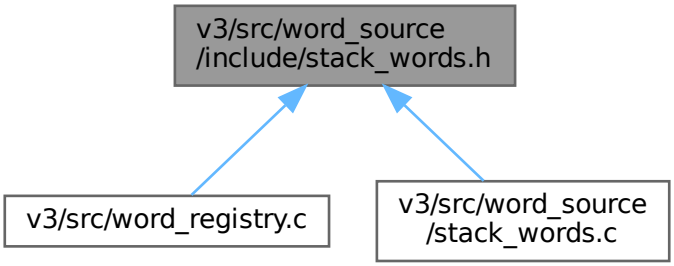
7.391 v3/src/word_source/include/stack_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for stack_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_stack_words (VM *vm)

  Registers all stack operation words with the virtual machine.

7.391.1 Function Documentation

7.391.1.1 register_stack_words()

```
void register_stack_words (
            VM * vm )
```

Registers all stack operation words with the virtual machine.

Parameters

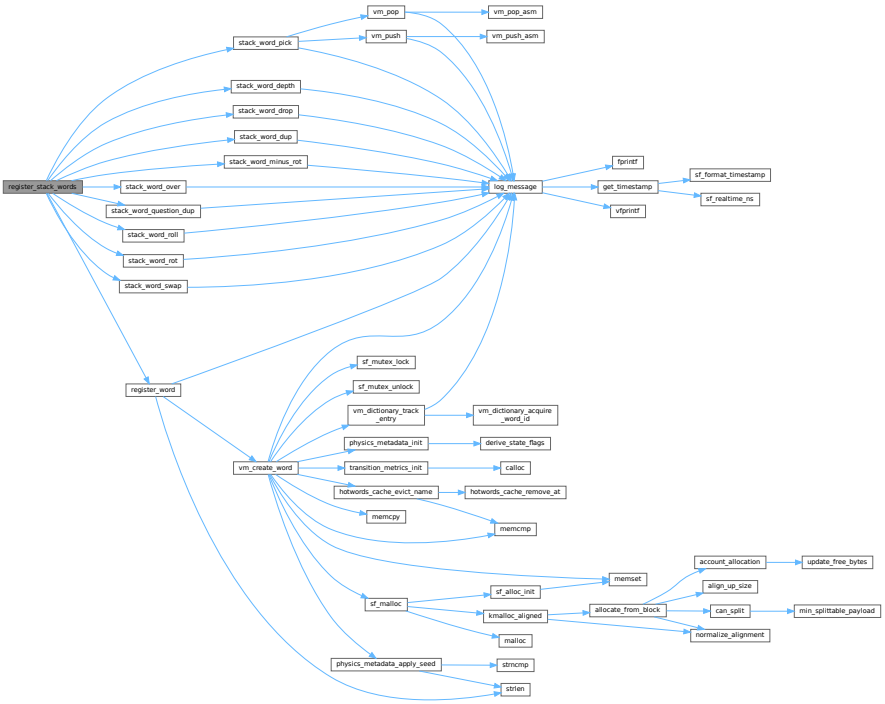
vm	Pointer to the virtual machine state

Registers all FORTH-79 standard stack operation words with the word registry.

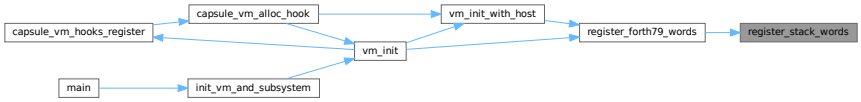
References register_word(), stack_word_depth(), stack_word_drop(), stack_word_dup(), stack_word_minus_rot(), stack_word_over(), stack_word_pick(), stack_word_question_dup(), stack_word_roll(), stack_word_rot(), and stack_word_swap().

Referenced by register_forth79_words().

Here is the call graph for this function:



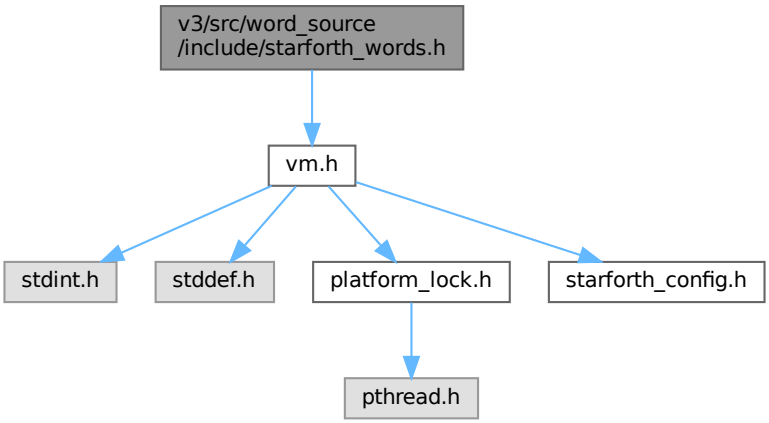
Here is the caller graph for this function:



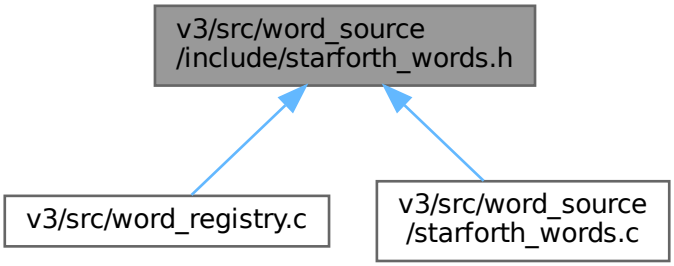
7.392 v3/src/word_source/include/starforth_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for starforth_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_starforth_words (VM *vm)

  Register StarForth vocabulary words with the VM.

StarForth Implementation Vocabulary Words

- void starforth_word_entropy_fetch (VM *vm)

  Fetch entropy value from address.
- void starforth_word_entropy_store (VM *vm)

  Store entropy value to address.
- void starforth_word_word_entropy (VM *vm)

  Calculate word entropy.
- void starforth_word_reset_entropy (VM *vm)

Reset entropy values.

- void starforth_word_top_words (VM *vm)
  Display the N most frequently used words.
- void starforth_word_paren_dash (VM *vm)
  Shebang-style comment word for init.4th metadata.
- void starforth_word_init (VM *vm)
  Initialize system from init.4th configuration file.
- void starforth_word_seed (VM *vm)
  Set the PRNG seed.
- void starforth_word_random (VM *vm)
  Generate bounded random number.
- void starforth_word_wait (VM *vm)
  Wait for specified number of heartbeat ticks.

7.392.1 Function Documentation

7.392.1.1 register_starforth_words()

```
void register_starforth_words (
            VM * vm )
```

Register StarForth vocabulary words with the VM.

Registers all StarForth-specific words and creates the STARFORTH vocabulary

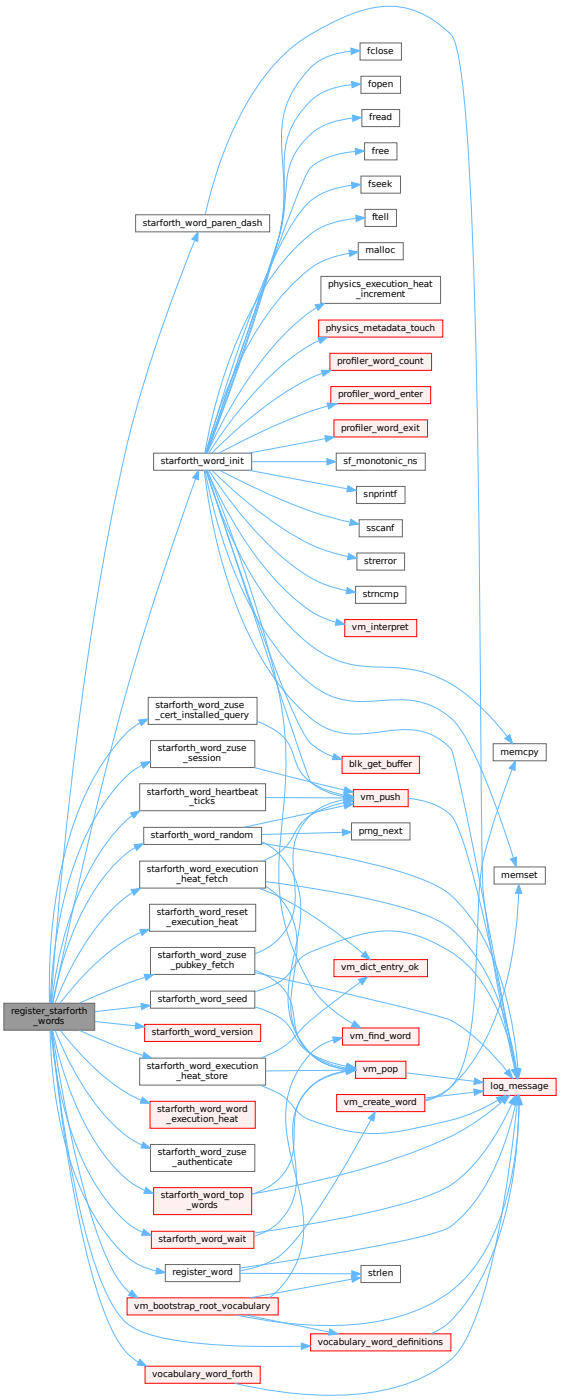
Parameters

vm	Pointer to the VM instance

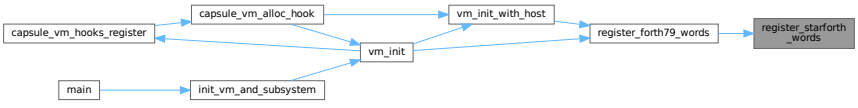
References register_word(), STARFORTH_CHECK_ARENA, starforth_word_execution_heat_fetch(), starforth_word_execution_heat_
starforth_word_heartbeat_ticks(), starforth_word_init(), starforth_word_paren_dash(), starforth_word_random(),
starforth_word_reset_execution_heat(), starforth_word_seed(), starforth_word_top_words(), starforth_word_version(),
starforth_word_wait(), starforth_word_word_execution_heat(), starforth_word_zuse_authenticate(), starforth_word_zuse_cert_installe
starforth_word_zuse_pubkey_fetch(), starforth_word_zuse_session(), vm_bootstrap_root_vocabulary(), vocabulary_word_definitions(
and vocabulary_word_forth().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.392.1.2 starforth_word_entropy_fetch()

```
void starforth_word_entropy_fetch (
            VM * vm )
```

Fetch entropy value from address.

Parameters

vm	Pointer to VM instance @stack (addr – n)

7.392.1.3 starforth_word_entropy_store()

```
void starforth_word_entropy_store (
            VM * vm )
```

Store entropy value to address.

Parameters

vm	Pointer to VM instance @stack (n addr –)

7.392.1.4 starforth_word_init()

```
void starforth_word_init (
            VM * vm )
```

Initialize system from init.4th configuration file.

Reads ./capsules/core/init.4th, parses Block headers, copies blocks sequentially starting at block 1, then executes them. Stack effect: (–)

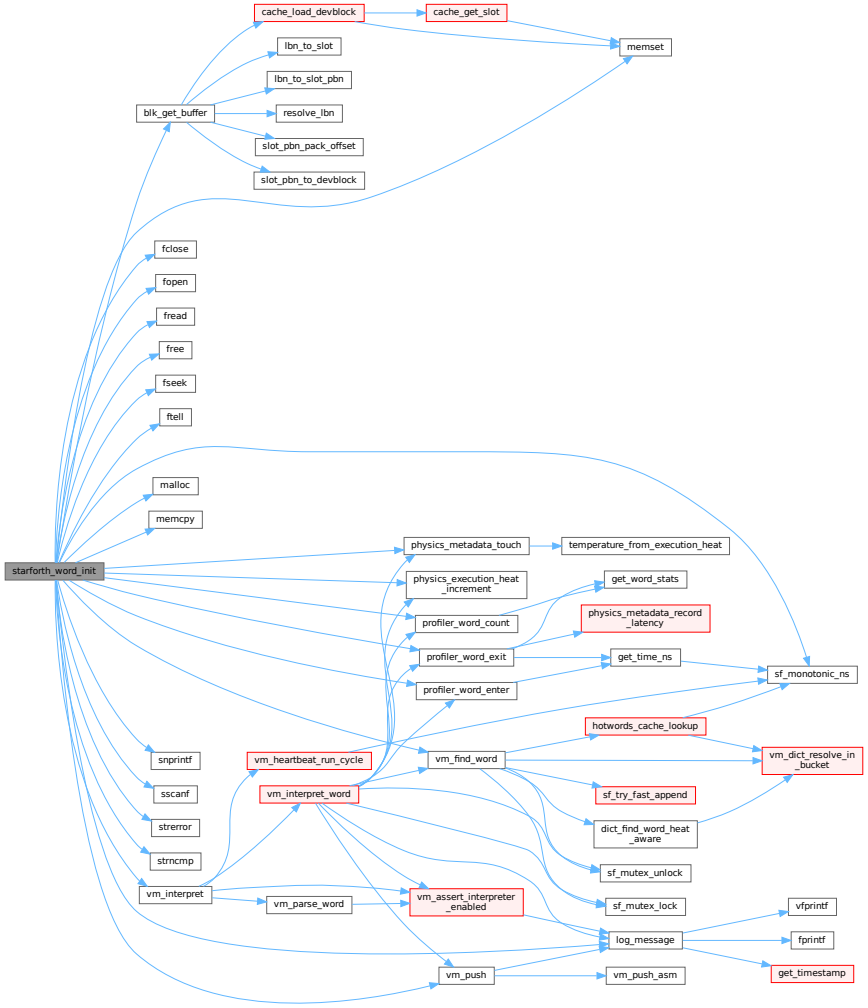
Parameters

vm	Pointer to the VM instance

References blk_get_buffer(), VM::current_executing_entry, ENOENT, errno, VM::error, DictEntry::execution_heat, fclose(), fopen(), fread(), free(), fseek(), ftell(), DictEntry::func, VM::halted, LOG_DEBUG, LOG_ERROR, LOG_INFO, log_message(), LOG_WARN, malloc(), memcpy(), memset(), physics_execution_heat_increment(), physics_metadata_touch(), profiler_word_count(), profiler_word_enter(), profiler_word_exit(), SEEK_END, SEEK_SET, sf_monotonic_ns(), snprintf(), sscanf(), strerror(), strncmp(), vm_find_word(), vm_interpret(), and vm_push().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.392.1.5 starforth_word_paren_dash()

```
void starforth_word_paren_dash (
            VM * vm )
```

Shebang-style comment word for init.4th metadata.

Starts with "(- " and consumes input until closing ")" Used for marking blocks that should be extracted to init.4th Stack effect: (–)

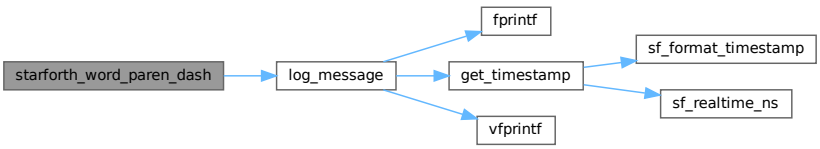
Parameters

vm	Pointer to the VM instance

References VM::input_buffer, VM::input_length, VM::input_pos, LOG_DEBUG, log_message(), and LOG_WARN.

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.392.1.6 starforth_word_random()

```
void starforth_word_random (
            VM * vm )
```

Generate bounded random number.

Parameters

vm	Pointer to VM instance @stack (lo hi – n)

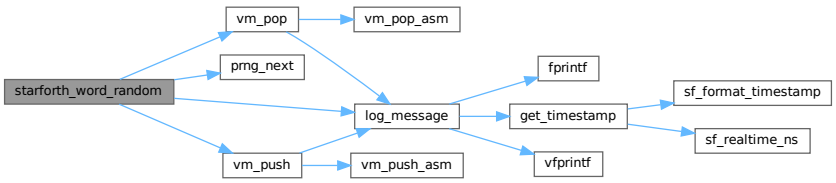
Note

Returns random value in inclusive range [lo, hi]

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), prng_next(), vm_pop(), and vm_push().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.392.1.7 starforth_word_reset_entropy()

```
void starforth_word_reset_entropy (
            VM * vm )
```

Reset entropy values.

Parameters

vm	Pointer to VM instance @stack (–)

7.392.1.8 starforth_word_seed()

```
void starforth_word_seed (
            VM * vm )
```

Set the PRNG seed.

Stack effect: (n –) Sets the internal PRNG state for reproducible random sequences.

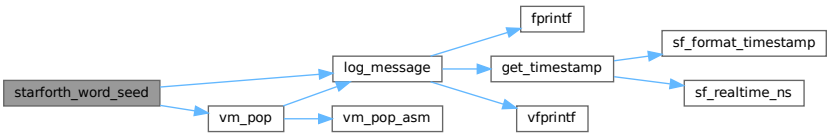
Parameters

vm	Pointer to the VM instance

References VM::dsp, VM::error, g_prng_state, LOG_DEBUG, LOG_ERROR, log_message(), and vm_pop().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.392.1.9 starforth_word_top_words()

```
void starforth_word_top_words (
            VM * vm )
```

Display the N most frequently used words.

Stack effect: (n –)

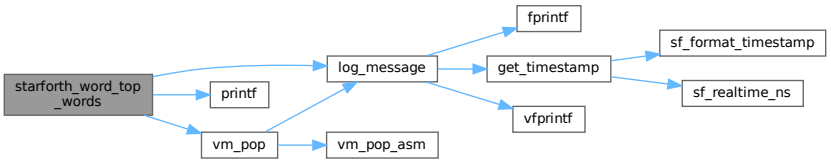
Parameters

vm	Pointer to the VM instance

References VM::dsp, entry, VM::error, execution_heat, VM::latest, LOG_ERROR, log_message(), printf(), vm_pop(), and word_count.

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.392.1.10 starforth_word_wait()

```
void starforth_word_wait (
            VM * vm )
```

Wait for specified number of heartbeat ticks.

Stack effect: (n –) Counts n heartbeat ticks by calling vm_tick() n times. Time is relative — we count heartbeats, not wall-clock milliseconds. This is architecture-independent: works on amd64, aarch64, riscv64 without depending on any platform timer.

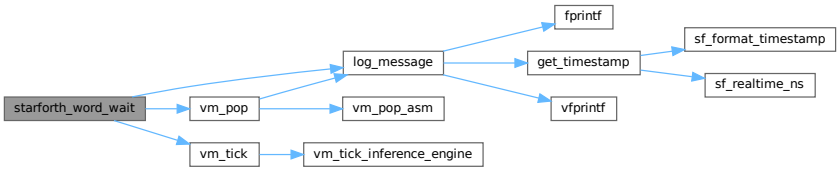
Parameters

vm	Pointer to the VM instance

References VM::dsp, VM::error, LOG_ERROR, log_message(), vm_pop(), and vm_tick().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.392.1.11 starforth_word_word_entropy()

```
void starforth_word_word_entropy (
            VM * vm )
```

Calculate word entropy.

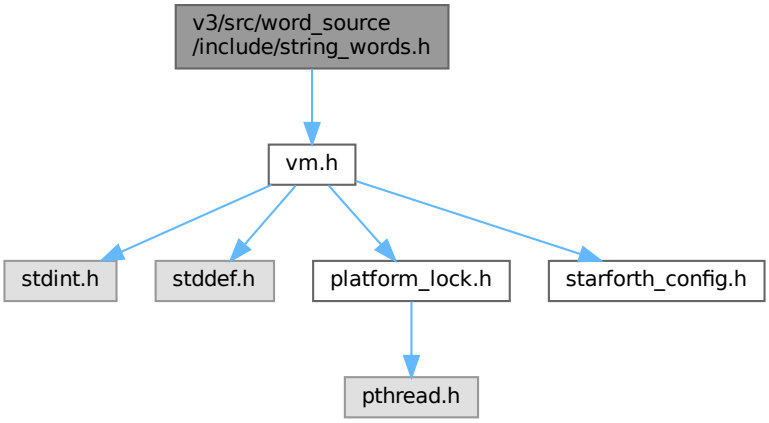
Parameters

vm	Pointer to VM instance @stack (–)

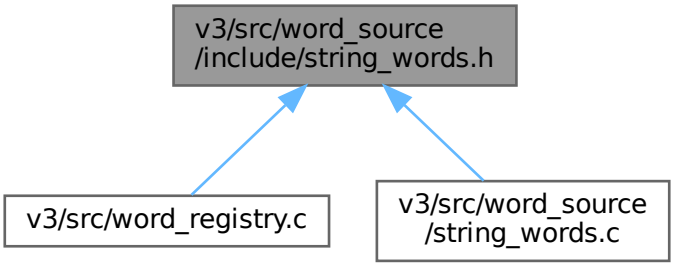
7.393 v3/src/word_source/include/string_words.h File Reference

FORTH-79 Standard string and text processing words implementation.

`#include "vm.h"`
Include dependency graph for string_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_string_words (VM *vm)

  COUNT (addr1 – addr2 u) Get string length and address Returns the character count and address of a counted string.

7.393.1 Detailed Description

FORTH-79 Standard string and text processing words implementation.

This file contains the implementation of FORTH-79 standard string and text processing words. These words provide functionality for string manipulation, input handling, and parsing.

7.393.2 Function Documentation

7.393.2.1 register_string_words()

```
void register_string_words (
            VM * vm )
```

COUNT (addr1 – addr2 u) Get string length and address Returns the character count and address of a counted string.

EXPECT (addr u –) Accept input line Accepts up to u characters of input and stores them at addr

SPAN (– addr) Variable: number of characters input Contains the count of characters actually received by EXPECT

QUERY (–) Accept input into TIB Accepts a line of input into the Terminal Input Buffer

TIB (– addr) Terminal input buffer address Returns the address of the Terminal Input Buffer

WORD (c – addr) Parse next word, delimited by c Parses the next word from input stream using c as delimiter

>IN (– addr) Input stream pointer Contains the offset into the input buffer

SOURCE (– addr u) Input source specification Returns current input buffer address and length

BL (– c) ASCII space character (32) Pushes ASCII space character value onto stack

FIND (addr – addr flag) Find word in dictionary Searches dictionary for word, returns status flag

' (– xt) Get execution token of next word Returns execution token of next word in input stream

['] (– xt) Compile execution token Compiles execution token as literal

LITERAL (n –) Compile literal number Compiles top stack value as literal

[LITERAL] (n –) Compile literal at compile time Compiles literal during compilation

CONVERT (d1 addr1 – d2 addr2) Convert string to number Converts string starting at addr1 to number

NUMBER (addr – n flag) Convert string to single number Converts string to single precision number

ENCLOSE (addr c – addr1 n1 n2 n3) Parse for delimiter Parses string for delimiter character

Register all string processing words with VM

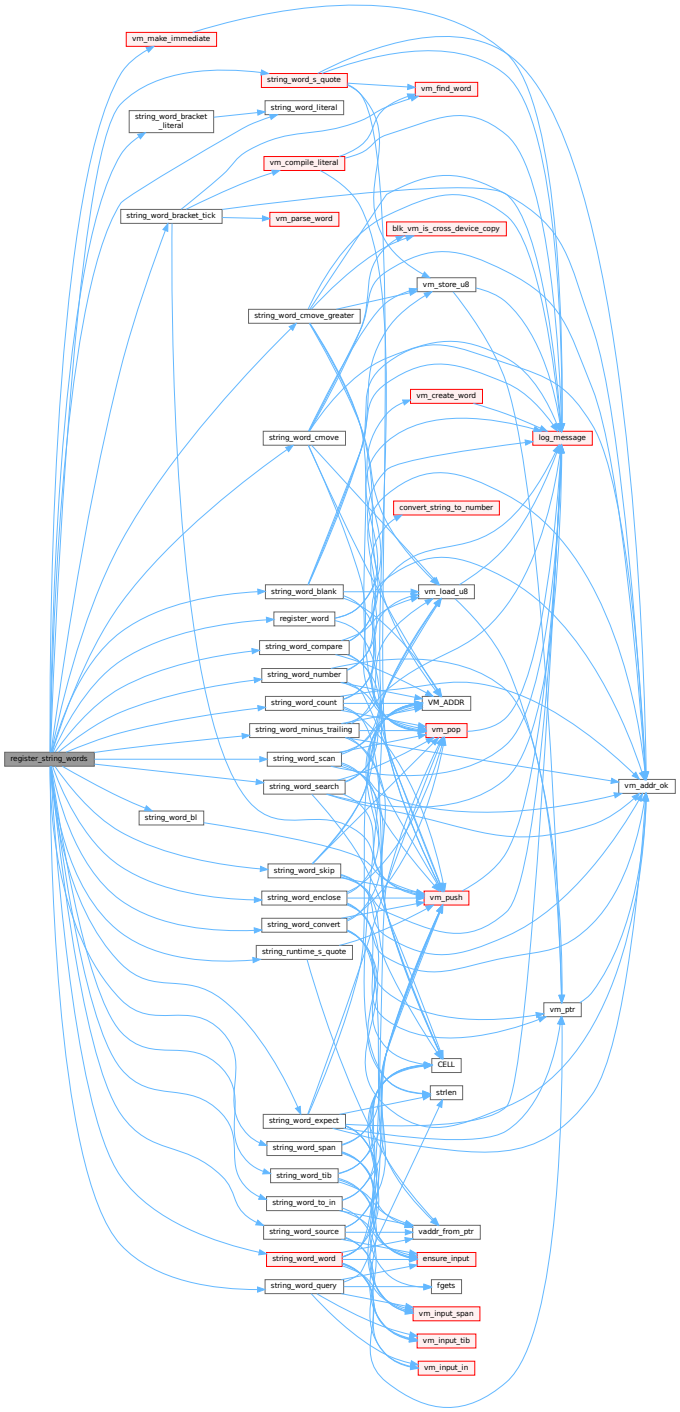
Parameters

vm	Pointer to VM instance where words will be registered

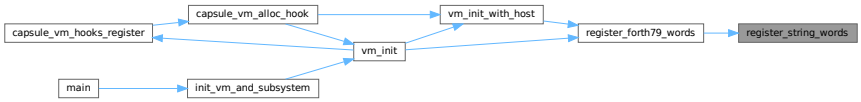
References register_word(), string_runtime_s_quote(), string_word_bl(), string_word_blank(), string_word_bracket_literal(), string_word_bracket_tick(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_literal(), string_word_minus_trailing(), string_word_number(), string_word_query(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), string_word_source(), string_word_span(), string_word_tib(), string_word_to_in(), string_word_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:

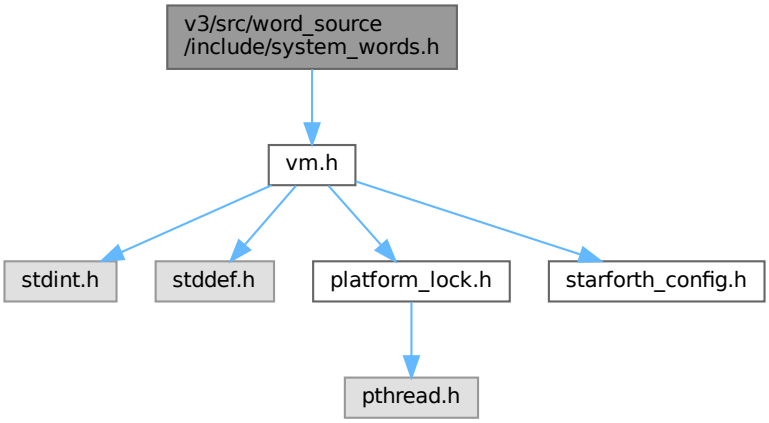


Here is the caller graph for this function:

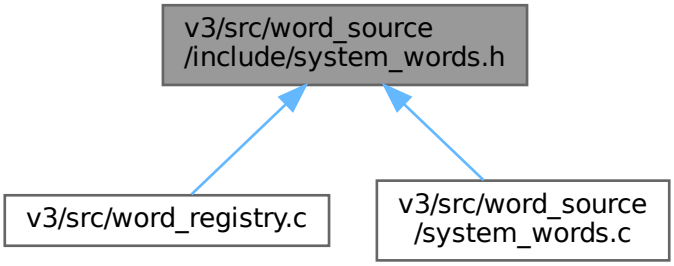


7.394 v3/src/word_source/include/system_words.h File Reference

`#include "vm.h"`
Include dependency graph for system_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_system_words (VM *vm)

  Registers all system words with the virtual machine.

7.394.1 Function Documentation

7.394.1.1 register_system_words()

```
void register_system_words (
            VM * vm )
```

Registers all system words with the virtual machine.

Parameters

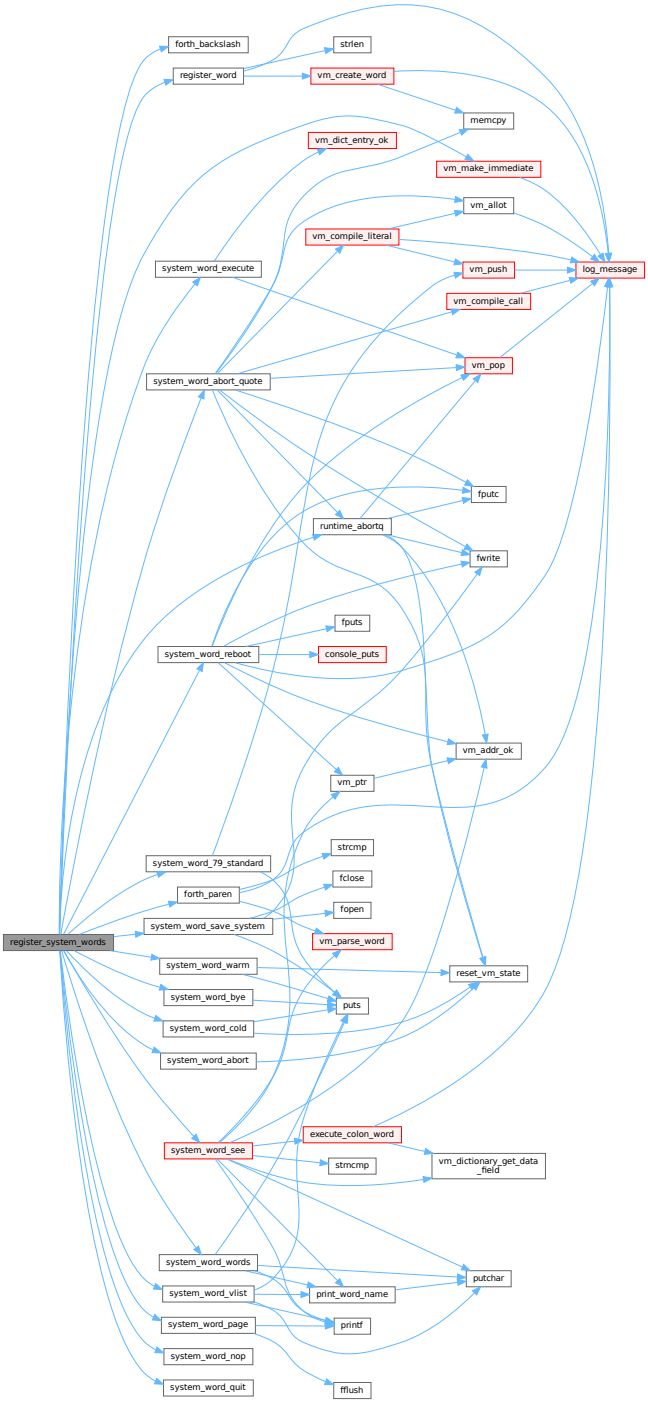
vm	Pointer to the virtual machine instance

This function registers all FORTH-79 system and environment words with the specified virtual machine instance, making them available for use in the FORTH environment.

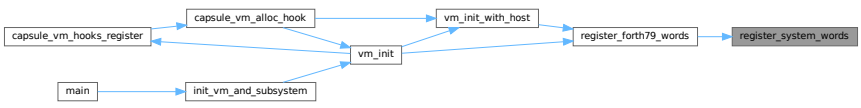
References forth_backslash(), forth_paren(), register_word(), runtime_abortq(), system_word_79_standard(), system_word_abort(), system_word_abort_quote(), system_word_bye(), system_word_cold(), system_word_execute(), system_word_nop(), system_word_page(), system_word_quit(), system_word_reboot(), system_word_save_system(), system_word_see(), system_word_vlist(), system_word_warm(), system_word_words(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



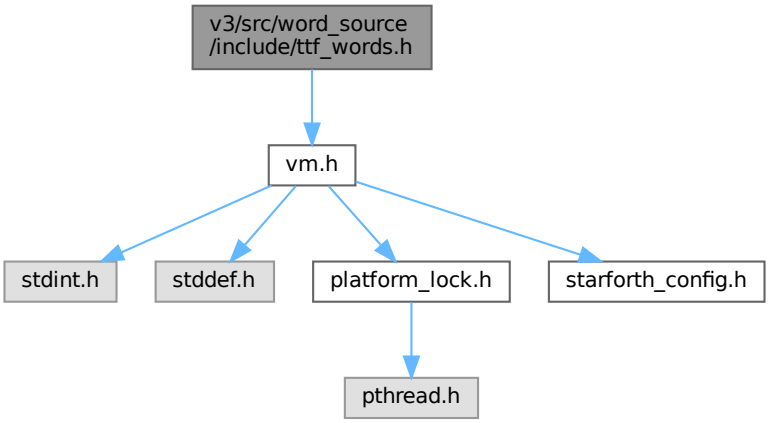
Here is the caller graph for this function:



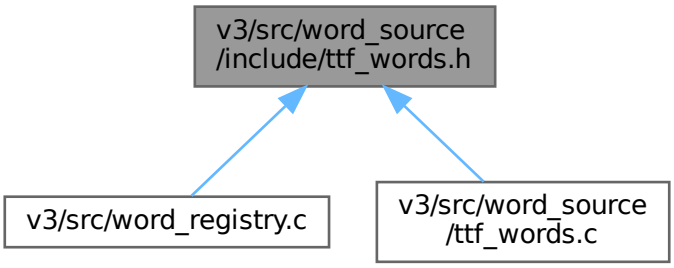
7.395 v3/src/word_source/include/ttf_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for ttf_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_ttf_words (VM *vm)

7.395.1 Function Documentation

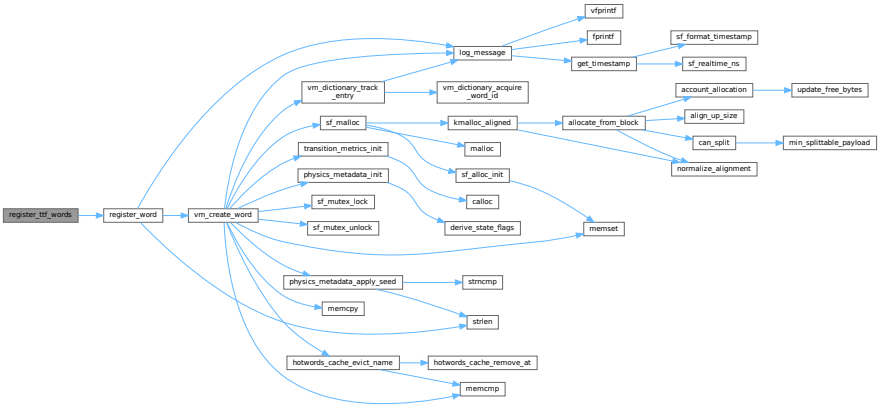
7.395.1.1 register_ttf_words()

```
void register_ttf_words (
            VM * vm )
```

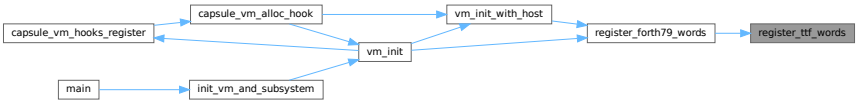
References register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



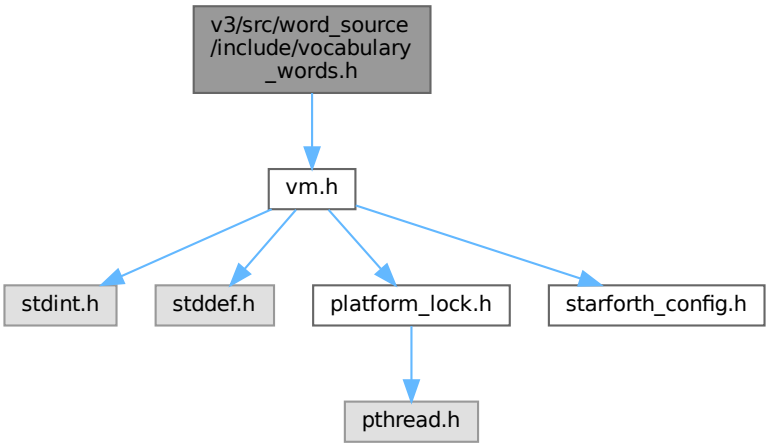
Here is the caller graph for this function:



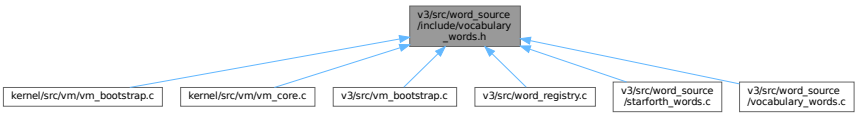
7.396 v3/src/word_source/include/vocabulary_words.h File Reference

```
#include "vm.h"
```

Include dependency graph for vocabulary_words.h:



This graph shows which files directly or indirectly include this file:



Functions

- void register_vocabulary_words (VM *vm)

  Register all FORTH-79 vocabulary words.
- void vocabulary_create_vocabulary_direct (VM *vm, const char *name)

  Direct vocabulary creation (kernel bootstrap)
- void vocabulary_word_definitions (VM *vm)

  DEFINITIONS (–) Set CURRENT to CONTEXT.
- void vocabulary_word_forth (VM *vm)

  FORTH (–) Select FORTH vocabulary.

7.396.1 Function Documentation

7.396.1.1 register_vocabulary_words()

```
void register_vocabulary_words (
            VM * vm )
```

Register all FORTH-79 vocabulary words.

Registers VOCABULARY, DEFINITIONS, CONTEXT, CURRENT, FORTH, ORDER, and (FIND)

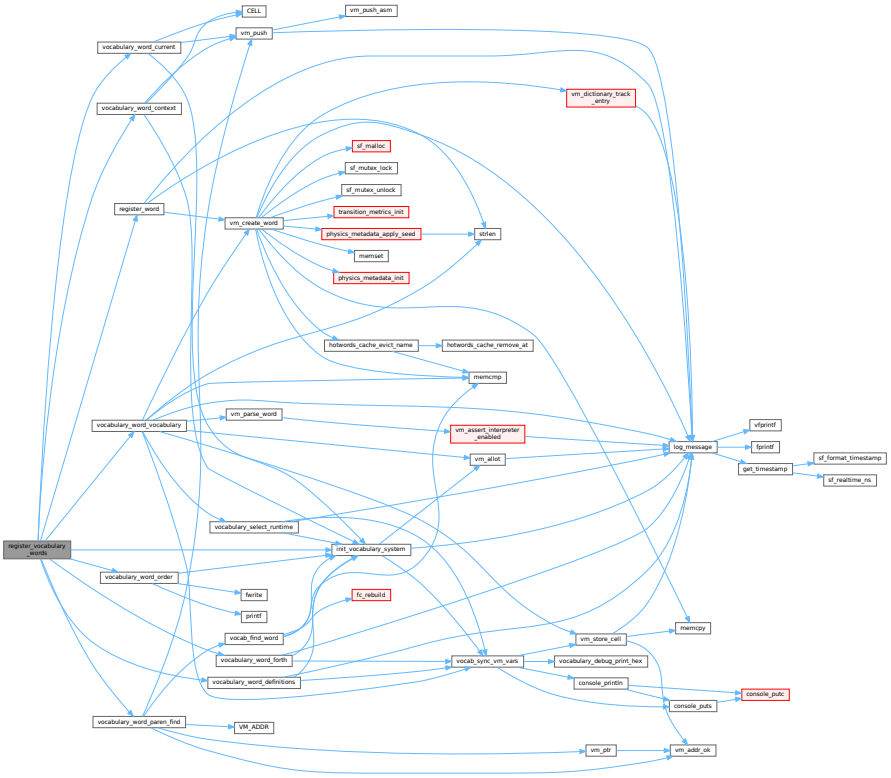
Parameters

vm	Pointer to VM instance

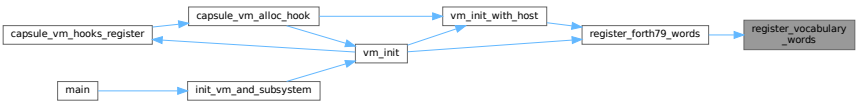
References init_vocabulary_system(), register_word(), vocabulary_word_context(), vocabulary_word_current(), vocabulary_word_definitions(), vocabulary_word_forth(), vocabulary_word_order(), vocabulary_word_paren_find(), and vocabulary_word_vocabulary().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.396.1.2 vocabulary_create_vocabulary_direct()

```
void vocabulary_create_vocabulary_direct (
            VM * vm,
            const char * name )
```

Direct vocabulary creation (kernel bootstrap)

Parameters

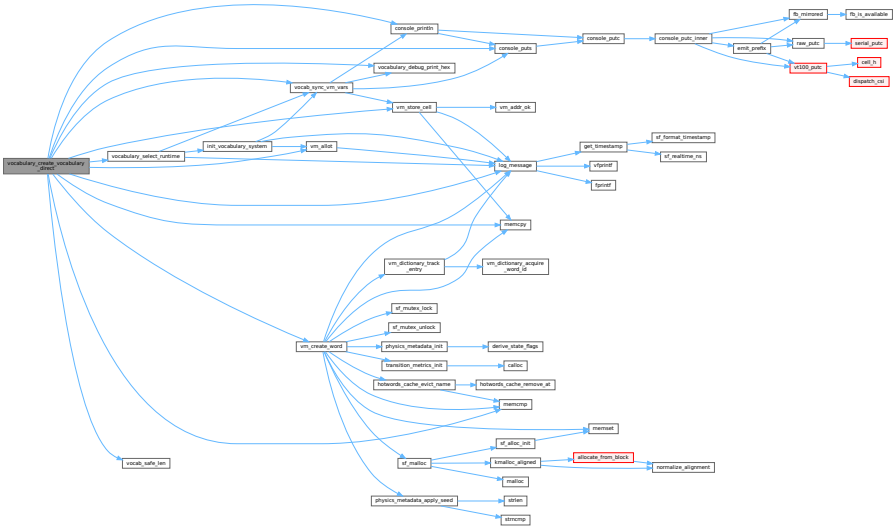
vm	Pointer to the virtual machine instance
name	Vocabulary name

References console_println(), console_puts(), entry, VM::error, VM::here, VM::latest, LOG_DEBUG, LOG_ERROR,

log_message(), memcmp(), memcpy(), VM::memory, name, vm_allot(), vm_create_word(), vm_store_cell(), vocab_safe_len(), vocab_sync_vm_vars(), vocabulary_debug_print_hex(), and vocabulary_select_runtime().

Referenced by vm_bootstrap_root_vocabulary().

Here is the call graph for this function:



Here is the caller graph for this function:

7.396.1.3 vocabulary_word_definitions()

```
void vocabulary_word_definitions (
            VM * vm )
```

DEFINITIONS (–) Set CURRENT to CONTEXT.

Makes new words be defined in the CONTEXT vocabulary

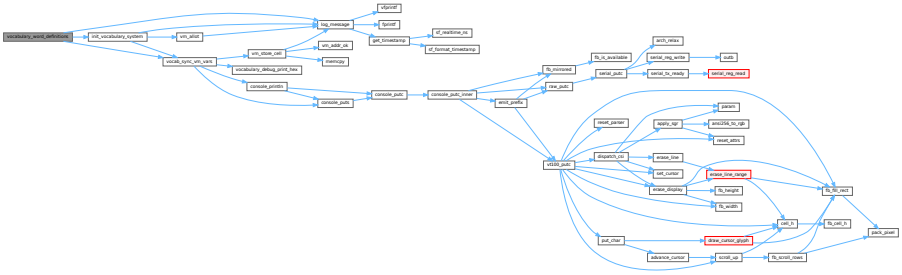
Parameters

vm	Pointer to VM instance

References VocabularyState::context_vocab, VocabularyState::current_vocab, init_vocabulary_system(), LOG_DEBUG, log_message(), VM::vocab, and vocab_sync_vm_vars().

Referenced by register_starforth_words(), register_vocabulary_words(), and vm_bootstrap_root_vocabulary().

Here is the call graph for this function:



Here is the caller graph for this function:

7.396.1.4 vocabulary_word_forth()

```
void vocabulary_word_forth (
            VM * vm )
```

FORTH (–) Select FORTH vocabulary.

Makes FORTH the CONTEXT vocabulary

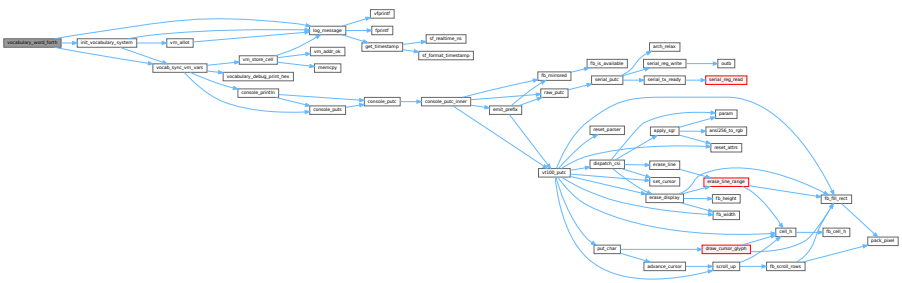
Parameters

vm	Pointer to VM instance

References VocabularyState::context_vocab, VocabularyState::forth_vocab, init_vocabulary_system(), LOG_DEBUG, log_message(), VM::vocab, and vocab_sync_vm_vars().

Referenced by register_starforth_words(), and register_vocabulary_words().

Here is the call graph for this function:

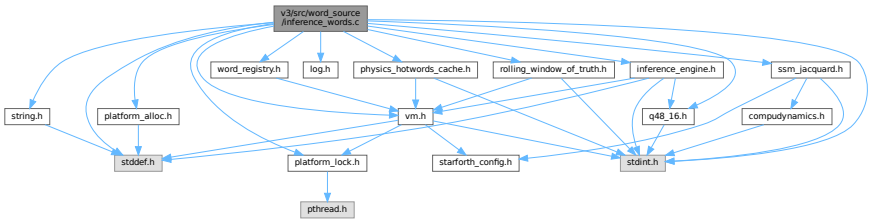


Here is the caller graph for this function:

7.397 v3/src/word_source/inference_words.c File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "vm.h"
#include "word_registry.h"
#include "log.h"
#include "q48_16.h"
#include "inference_engine.h"
#include "ssm_jacquard.h"
#include "rolling_window_of_truth.h"
#include "physics_hotwords_cache.h"
#include "platform_alloc.h"
#include "platform_lock.h"
```

Include dependency graph for inference_words.c:



Functions

- static q48_16_t q48_pop_inf (VM *vm)

  inference_words.c — Module 26: SSM inference engine + Jacquard FORTH words
- static void q48_push_inf (VM *vm, q48_16_t q)
- static double q48_to_dbl (q48_16_t q)
- static const uint64_t * array_ptr (VM *vm, vaddr_t addr, cell_t u)
- static void infer_word_q_variance (VM *vm)
- static void infer_word_decay_slope (VM *vm)
- static void infer_word_window_width (VM *vm)
- static void infer_word_window_diversity (VM *vm)
- static void infer_word_run (VM *vm)
- static void infer_word_window_fetch (VM *vm)
- static void infer_word_decay_fetch (VM *vm)
- static void infer_word_variance_fetch (VM *vm)
- static void infer_word_fit_fetch (VM *vm)
- static void infer_word_early_exit_fetch (VM *vm)

- static void infer_word_l8_mode (VM *vm)
- static void infer_word_l8_update (VM *vm)
- static void infer_word_l8_apply (VM *vm)
- static void infer_word_l8_table_force (VM *vm)
- static BayesianLatencyPosterior cache_posterior (VM *vm)
- static BayesianLatencyPosterior bucket_posterior (VM *vm)
- static void infer_word_bayes_cache_mean (VM *vm)
- static void infer_word_bayes_cache_lower (VM *vm)
- static void infer_word_bayes_cache_upper (VM *vm)
- static void infer_word_bayes_bucket_mean (VM *vm)
- static void infer_word_bayes_bucket_lower (VM *vm)
- static void infer_word_bayes_bucket_upper (VM *vm)
- void register_inference_words (VM *vm)

inference_words.h — Module 26: SSM inference engine + Jacquard FORTH words

7.397.1 Function Documentation

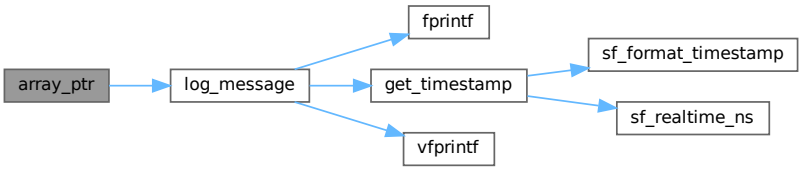
7.397.1.1 array_ptr()

```
static const uint64_t * array_ptr (
            VM * vm,
            vaddr_t addr,
            cell_t u )  [static]
```

References bytes, VM::error, LOG_ERROR, log_message(), VM::memory, and VM_MEMORY_SIZE.

Referenced by infer_word_decay_slope(), infer_word_q_variance(), and infer_word_window_width().

Here is the call graph for this function:



Here is the caller graph for this function:

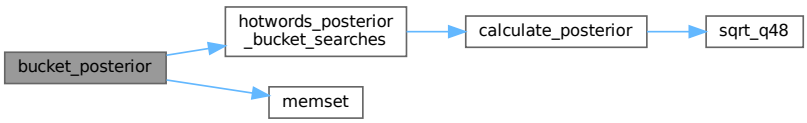
7.397.1.2 bucket_posterior()

```
static BayesianLatencyPosterior bucket_posterior (
            VM * vm )  [static]
```

References VM::hotwords_cache, hotwords_posterior_bucket_searches(), memset(), HotwordsCache_s::stats, and zero.

Referenced by infer_word_bayes_bucket_lower(), infer_word_bayes_bucket_mean(), and infer_word_bayes_bucket_upper().

Here is the call graph for this function:



Here is the caller graph for this function:

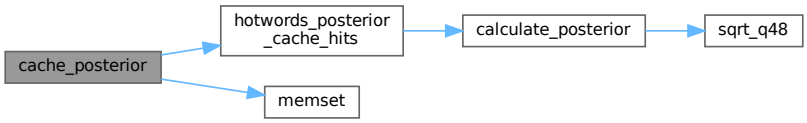
7.397.1.3 cache_posterior()

```
static BayesianLatencyPosterior cache_posterior (
            VM * vm )  [static]
```

References VM::hotwords_cache, hotwords_posterior_cache_hits(), memset(), HotwordsCache_s::stats, and zero.

Referenced by infer_word_bayes_cache_lower(), infer_word_bayes_cache_mean(), and infer_word_bayes_cache_upper().

Here is the call graph for this function:



Here is the caller graph for this function:

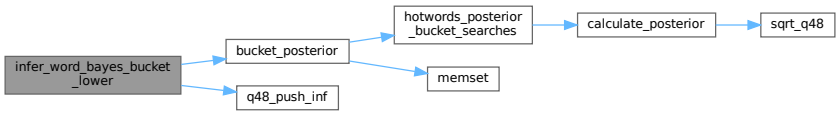
7.397.1.4 infer_word_bayes_bucket_lower()

```
static void infer_word_bayes_bucket_lower (
            VM * vm )  [static]
```

References bucket_posterior(), and q48_push_inf().

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

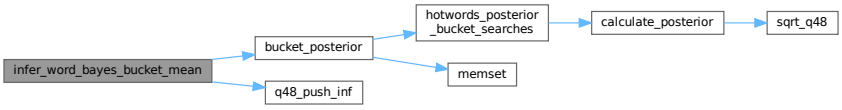
7.397.1.5 infer_word_bayes_bucket_mean()

```
static void infer_word_bayes_bucket_mean (
            VM * vm )  [static]
```

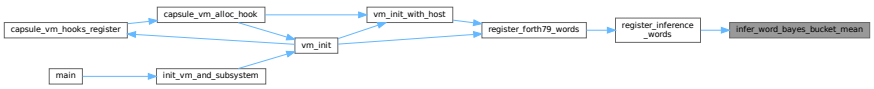
References bucket_posterior(), and q48_push_inf().

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:



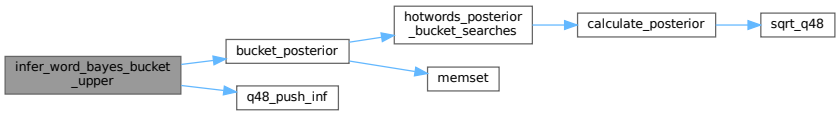
7.397.1.6 infer_word_bayes_bucket_upper()

```
static void infer_word_bayes_bucket_upper (
            VM * vm )  [static]
```

References bucket_posterior(), and q48_push_inf().

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

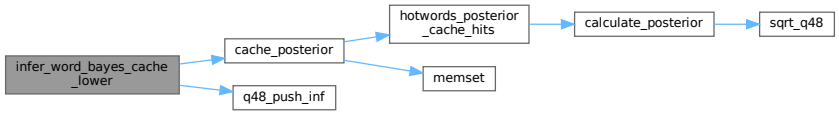
7.397.1.7 infer_word_bayes_cache_lower()

```
static void infer_word_bayes_cache_lower (
            VM * vm )  [static]
```

References cache_posterior(), and q48_push_inf().

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

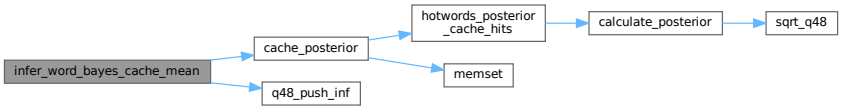
7.397.1.8 infer_word_bayes_cache_mean()

```
static void infer_word_bayes_cache_mean (
            VM * vm )  [static]
```

References cache_posterior(), and q48_push_inf().

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

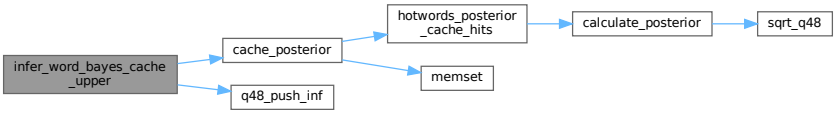
7.397.1.9 infer_word_bayes_cache_upper()

```
static void infer_word_bayes_cache_upper (
            VM * vm )  [static]
```

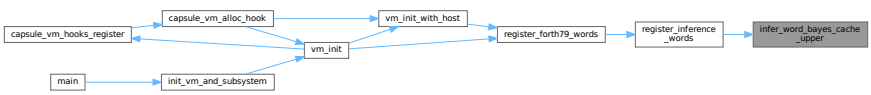
References cache_posterior(), and q48_push_inf().

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.397.1.10 infer_word_decay_fetch()

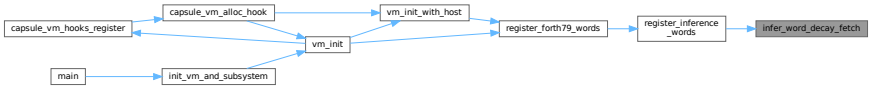
```
static void infer_word_decay_fetch (
            VM * vm )  [static]
```

References InferenceOutputs::adaptive_decay_slope, VM::last_inference_outputs, and q48_push_inf().

Referenced by register_inference_words().

Here is the call graph for this function:

Here is the caller graph for this function:



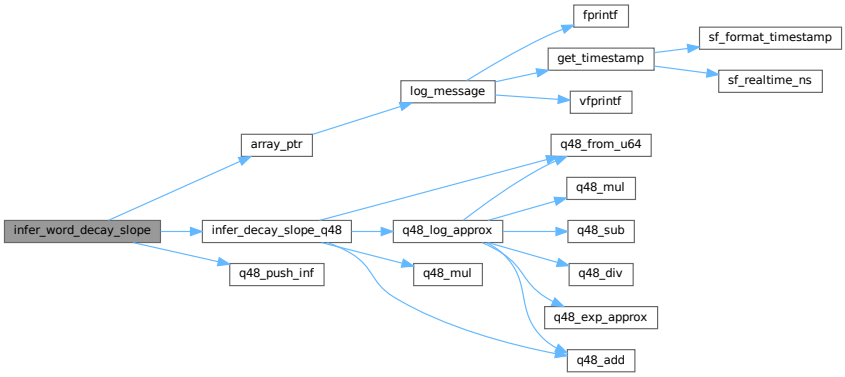
7.397.1.11 infer_word_decay_slope()

```
static void infer_word_decay_slope (
            VM * vm )  [static]
```

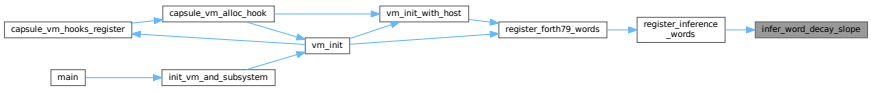
References array_ptr(), infer_decay_slope_q48(), q48_push_inf(), and VM_POP.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.397.1.12 infer_word_early_exit_fetch()

```
static void infer_word_early_exit_fetch (
            VM * vm )  [static]
```

References InferenceOutputs::early_exited, VM::last_inference_outputs, and VM_PUSH.

Referenced by register_inference_words().

Here is the caller graph for this function:

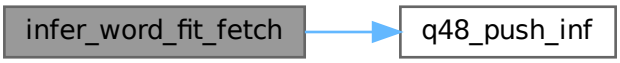
7.397.1.13 infer_word_fit_fetch()

```
static void infer_word_fit_fetch (
            VM * vm )  [static]
```

References VM::last_inference_outputs, q48_push_inf(), and InferenceOutputs::slope_fit_quality_q48.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.397.1.14 infer_word_l8_apply()

```
static void infer_word_l8_apply (
            VM * vm )  [static]
```

References ssm_apply_mode(), VM::ssm_config, and VM::ssm_l8_state.

Referenced by register_inference_words().

Here is the call graph for this function:

Here is the caller graph for this function:

7.397.1.15 infer_word_l8_mode()

```
static void infer_word_l8_mode (
            VM * vm )  [static]
```

References ssm_l8_state_t::current_mode, VM::ssm_l8_state, and VM_PUSH.

Referenced by register_inference_words().

Here is the caller graph for this function:

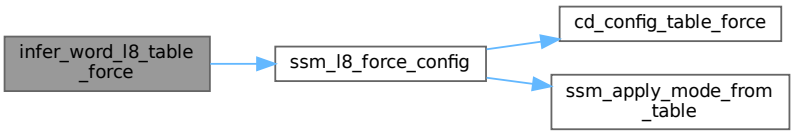
7.397.1.16 infer_word_l8_table_force()

```
static void infer_word_l8_table_force (
            VM * vm )  [static]
```

References VM::ssm_config, ssm_l8_force_config(), VM::ssm_l8_state, and VM_POP.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

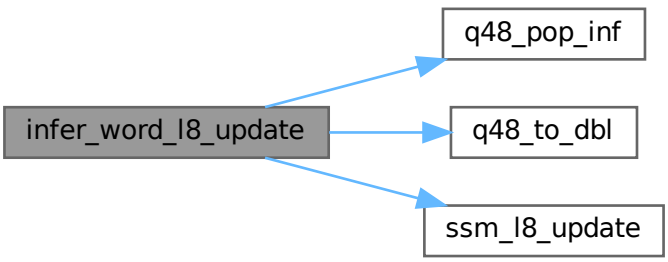
7.397.1.17 infer_word_l8_update()

```
static void infer_word_l8_update (
            VM * vm )  [static]
```

References ssm_l8_metrics_t::cv, ssm_l8_metrics_t::entropy, q48_pop_inf(), q48_to_dbl(), VM::ssm_l8_state, ssm_l8_update(), ssm_l8_metrics_t::stability_score, ssm_l8_metrics_t::temporal_decay, and VM_POP.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

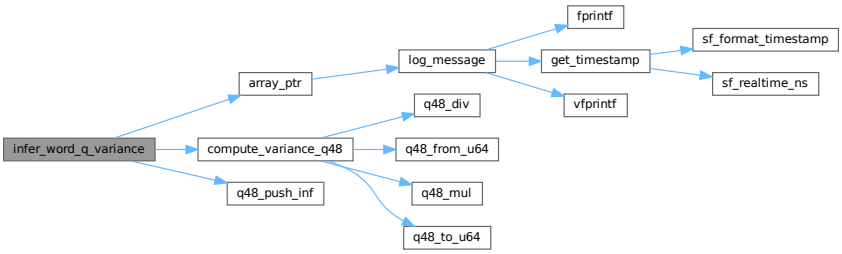
7.397.1.18 infer_word_q_variance()

```
static void infer_word_q_variance (
            VM * vm )  [static]
```

References array_ptr(), compute_variance_q48(), q48_push_inf(), and VM_POP.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.397.1.19 infer_word_run()

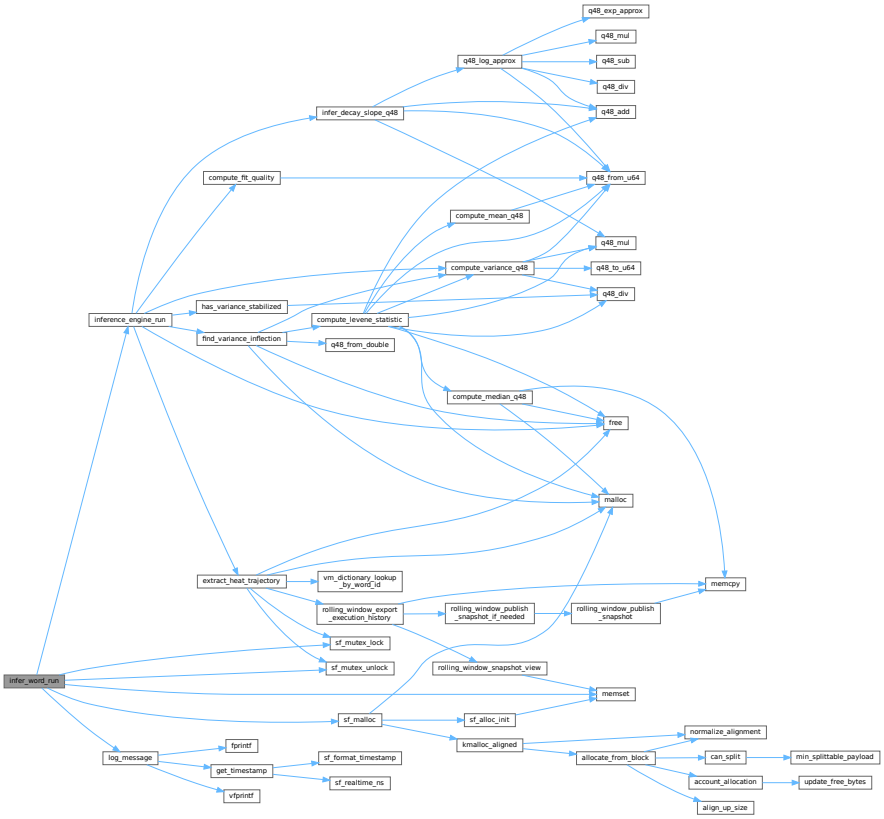
```
static void infer_word_run (
            VM * vm )  [static]
```

References VM::dict_lock, VM::error, DictEntry::execution_heat, InferenceInputs::hot_word_count, HOTWORDS_EXECUTION_HEAT
inference_engine_run(), VM::last_inference_outputs, InferenceInputs::last_stale_count, InferenceInputs::last_total_heat,
VM::latest, DictEntry::link, LOG_ERROR, log_message(), memset(), VM::pipeline_metrics, InferenceInputs::prefetch_attempts,
PipelineGlobalMetrics::prefetch_attempts, InferenceInputs::prefetch_hits, PipelineGlobalMetrics::prefetch_hits,
VM::rolling_window, sf_malloc(), sf_mutex_lock(), sf_mutex_unlock(), InferenceInputs::stale_word_count,
VM::stale_word_count_at_check, RollingWindowOfTruth::total_executions, InferenceInputs::total_heat, VM::total_heat_at_last_check

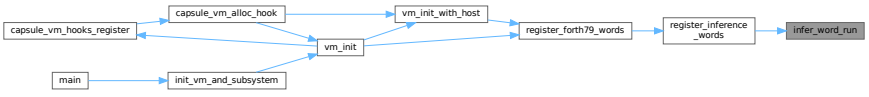
InferenceInputs::trajectory_length, InferenceInputs::vm, InferenceInputs::window, RollingWindowOfTruth::window_pos, InferenceInputs::word_count, and word_count.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:



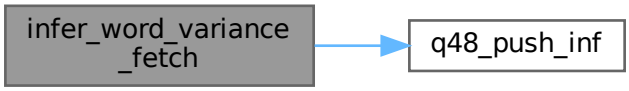
7.397.1.20 infer_word_variance_fetch()

```
static void infer_word_variance_fetch (
            VM * vm )  [static]
```

References VM::last_inference_outputs, q48_push_inf(), and InferenceOutputs::window_variance_q48.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

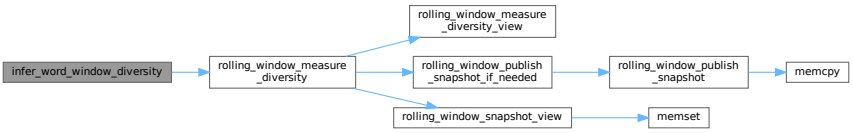
7.397.1.21 infer_word_window_diversity()

```
static void infer_word_window_diversity (
            VM * vm )  [static]
```

References VM::rolling_window, rolling_window_measure_diversity(), and VM_PUSH.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.397.1.22 infer_word_window_fetch()

```
static void infer_word_window_fetch (
            VM * vm )  [static]
```

References InferenceOutputs::adaptive_window_width, VM::last_inference_outputs, and VM_PUSH.

Referenced by register_inference_words().

Here is the caller graph for this function:

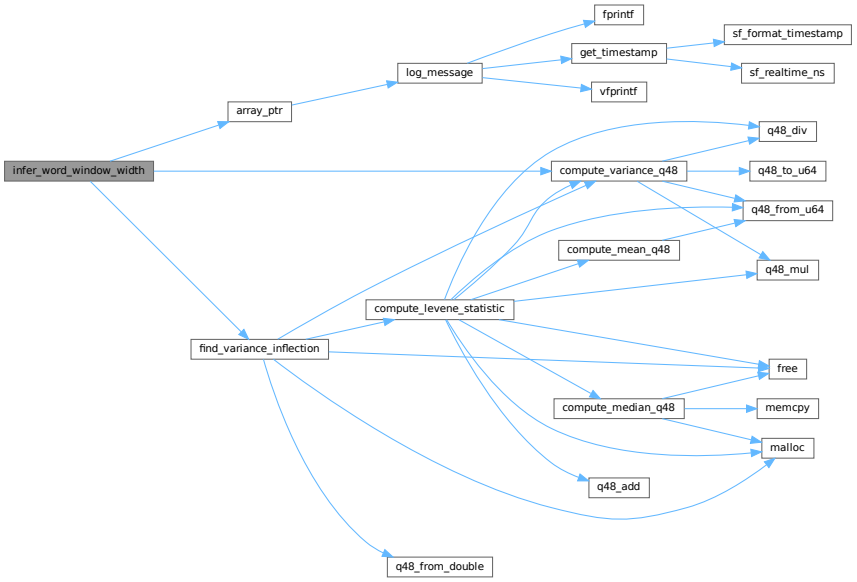
7.397.1.23 infer_word_window_width()

```
static void infer_word_window_width (
            VM * vm )  [static]
```

References array_ptr(), compute_variance_q48(), find_variance_inflection(), VM_POP, and VM_PUSH.

Referenced by register_inference_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.397.1.24 q48_pop_inf()

```
static q48_16_t q48_pop_inf (
            VM * vm )  [inline], [static]
```

inference_words.c — Module 26: SSM inference engine + Jacquard FORTH words

Exposes the SSM physics engine to FORTH:

- Variance / decay-slope / window-width inference on arbitrary arrays
- Full inference engine run on this VM's rolling window
- L8 Jacquard mode selector: update, query, apply
- Bayesian latency posteriors for cache-hit and bucket-search latencies
- Rolling window diversity (entropy metric)
- Readable inference output fields from vm->last_inference_outputs

All Q48.16 values pushed as cell_t (int64_t), reinterpreted as uint64_t.

Words registered: Q.VARIANCE (addr u – q) variance of u cells at addr (Q48.16) INFER-DECAY-SLOPE (addr u – q) exponential decay slope (Q48.16) INFER-WINDOW-WIDTH (addr u – n) optimal window width from inflection WINDOW-DIVERSITY (– u) rolling window diversity (entropy) INFER-RUN (–) run full inference, update vm->last_inference_outputs INFER-WINDOW@ (– u) last adaptive_window_width INFER-DECAY@ (– q) last adaptive_decay_slope (Q48.16) INFER-VARIANCE@ (– q) last window_variance_q48 INFER-FIT@ (– q) last slope_fit_quality_q48 INFER-EARLY-EXIT@ (– flag) 1 if last INFER-RUN used ANOVA early-exit L8-MODE (– n) current Jacquard mode (0-15) L8-UPDATE (entropy_q cv_q temporal_q stability_q –) L8-APPLY (–) apply current mode to vm->ssm_config (legacy 16-mode) L8-TABLE-FORCE (config_idx –) force the adaptive table onto config_idx (0-127) BAYES-CACHE-MEAN (– q) Bayesian mean latency for cache hits (Q48.16) BAYES-CACHE-LOWER (– q) 95% credible lower bound, cache hits BAYES-CACHE-UPPER (– q) 95% credible upper bound, cache hits BAYES-BUCKET-MEAN (– q) Bayesian mean latency for bucket searches BAYES-BUCKET-LOWER (– q) 95% credible lower bound, bucket searches BAYES-BUCKET-UPPER (– q) 95% credible upper bound, bucket searches

References VM_POP.

Referenced by infer_word_l8_update().

Here is the caller graph for this function:

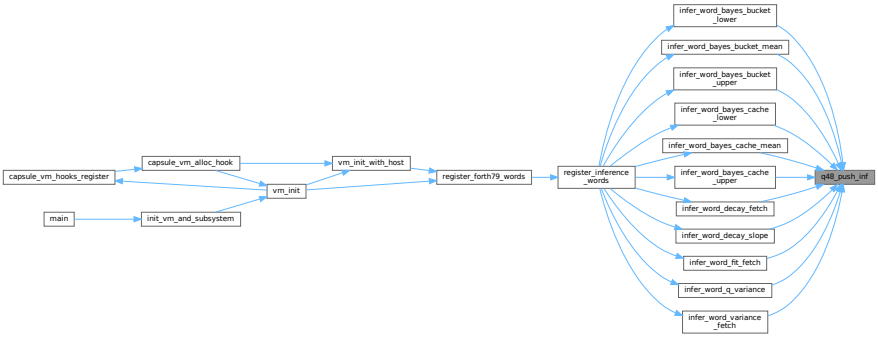
7.397.1.25 q48_push_inf()

```
static void q48_push_inf (
            VM * vm,
            q48_16_t q )  [inline], [static]
```

References VM_PUSH.

Referenced by infer_word_bayes_bucket_lower(), infer_word_bayes_bucket_mean(), infer_word_bayes_bucket_upper(), infer_word_bayes_cache_lower(), infer_word_bayes_cache_mean(), infer_word_bayes_cache_upper(), infer_word_decay_fetch(), infer_word_decay_slope(), infer_word_fit_fetch(), infer_word_q_variance(), and infer_word_variance_fetch().

Here is the caller graph for this function:

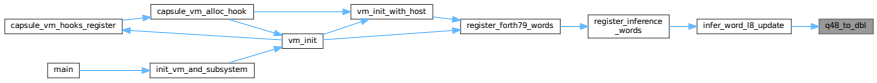


7.397.1.26 q48_to_dbl()

```
static double q48_to_dbl (
            q48_16_t q )  [inline], [static]
```

Referenced by infer_word_l8_update().

Here is the caller graph for this function:



7.397.1.27 register_inference_words()

```
void register_inference_words (
            VM * vm )
```

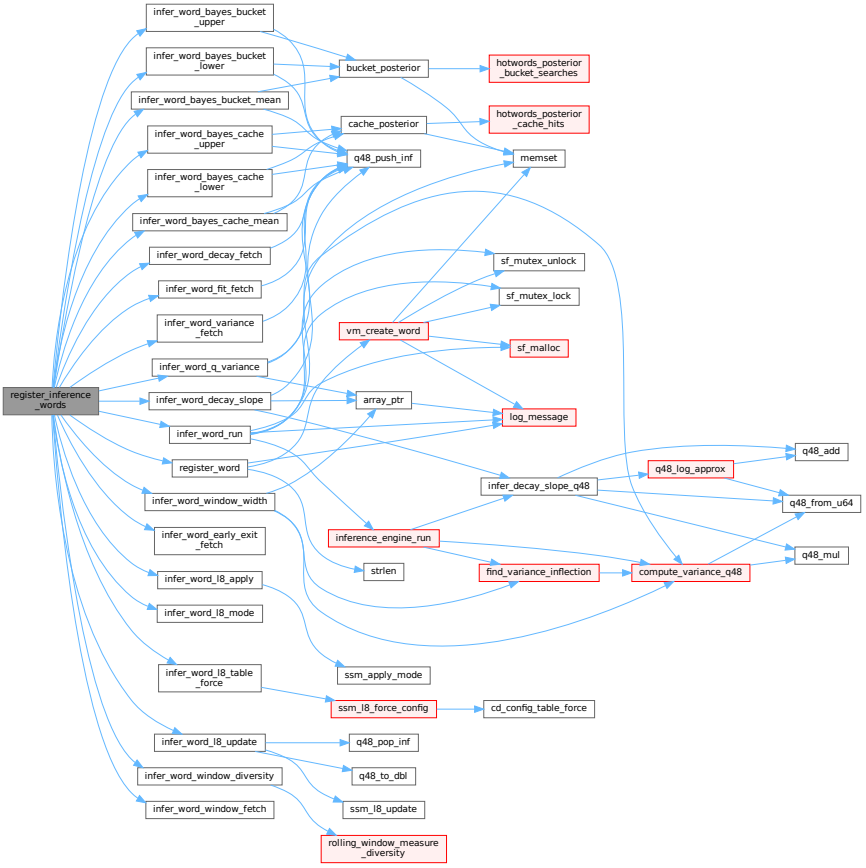
inference_words.h — Module 26: SSM inference engine + Jacquard FORTH words

Register all inference engine / Jacquard FORTH words with the VM. Module 26 in register_forth79_words().

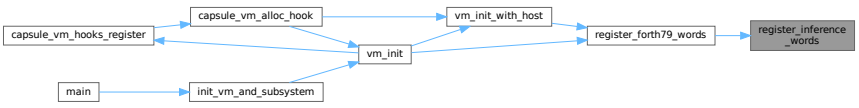
References infer_word_bayes_bucket_lower(), infer_word_bayes_bucket_mean(), infer_word_bayes_bucket_upper(), infer_word_bayes_cache_lower(), infer_word_bayes_cache_mean(), infer_word_bayes_cache_upper(), infer_word_decay_fetch(), infer_word_decay_slope(), infer_word_early_exit_fetch(), infer_word_fit_fetch(), infer_word_l8_apply(), infer_word_l8_mode(), infer_word_l8_table_force(), infer_word_l8_update(), infer_word_q_variance(), infer_word_run(), infer_word_variance_fetch(), infer_word_window_diversity(), infer_word_window_fetch(), infer_word_window_width(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

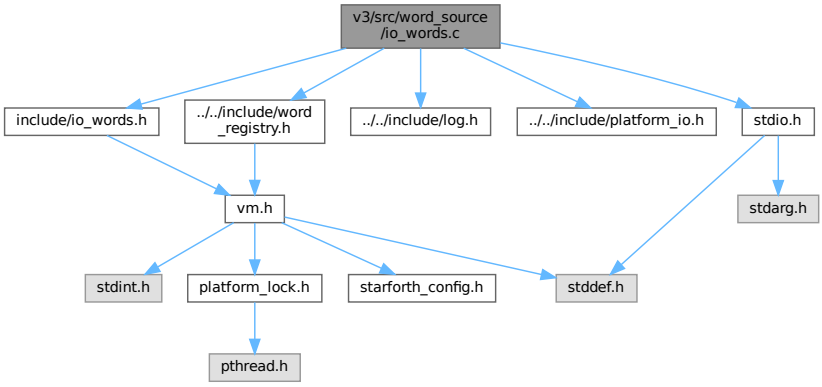


7.398 v3/src/word_source/io_words.c File Reference

```
#include "include/io_words.h"
#include "../../include/word_registry.h"
```

```
#include "../../include/log.h"
#include "../../include/platform_io.h"
#include <stdio.h>
```

Include dependency graph for io_words.c:



Functions

- static void io_word_emit (VM *vm)

  FORTH word EMIT - Output character to terminal.
- static void io_word_cr (VM *vm)

  FORTH word CR - Output carriage return.
- static void io_word_key (VM *vm)

  FORTH word KEY - Input character from terminal.
- static void io_word_question_terminal (VM *vm)

  FORTH word ?TERMINAL - Check if input is available.
- static void io_word_type (VM *vm)

  FORTH word TYPE - Output string of characters.
- static void io_word_space (VM *vm)

  FORTH word SPACE - Output one space character.
- static void io_word_spaces (VM *vm)

  FORTH word SPACES - Output multiple spaces.
- static void io_runtime_do_string (VM *vm)
- static void io_word_dot_quote (VM *vm)
- void register_io_words (VM *vm)

  Registers all I/O related FORTH words with the virtual machine.

7.398.1 Function Documentation

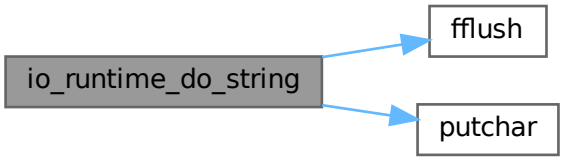
7.398.1.1 io_runtime_do_string()

```
static void io_runtime_do_string (
            VM * vm )  [static]
```

References VM::error, fflush(), putchar(), VM::return_stack, VM::rsp, and stdout.

Referenced by register_io_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.398.1.2 io_word_cr()

```
static void io_word_cr (
            VM * vm )  [static]
```

FORTH word CR - Output carriage return.

Parameters

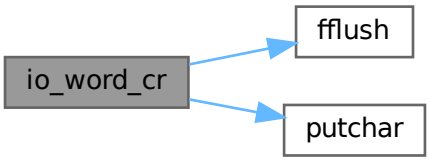
vm	Pointer to the VM structure

Stack effect: (–) Outputs a newline character to the terminal

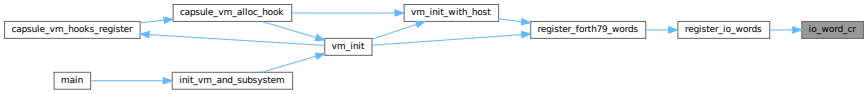
References fflush(), putchar(), and stdout.

Referenced by register_io_words().

Here is the call graph for this function:



Here is the caller graph for this function:



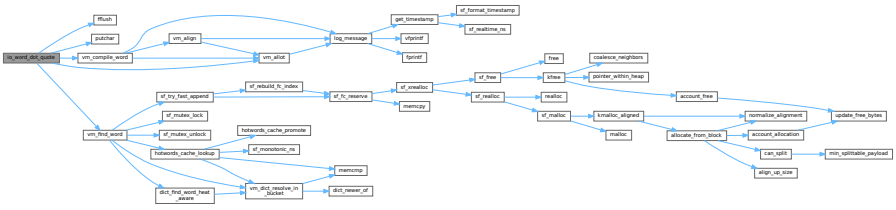
7.398.1.3 io_word_dot_quote()

```
static void io_word_dot_quote (
            VM * vm )  [static]
```

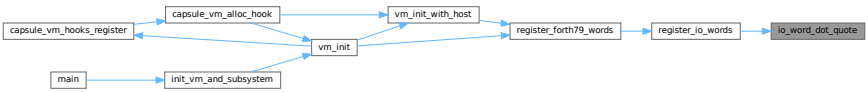
References VM::error, fflush(), VM::input_buffer, VM::input_length, VM::input_pos, VM::mode, MODE_INTERPRET, putchar(), stdout, vm_allot(), vm_compile_word(), and vm_find_word().

Referenced by register_io_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.398.1.4 io_word_emit()

```
static void io_word_emit (
            VM * vm )  [static]
```

FORTH word EMIT - Output character to terminal.

Parameters

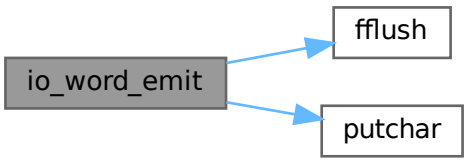
vm	Pointer to the VM structure

Stack effect: (c –) Outputs the character from top of stack to the terminal

References VM::data_stack, VM::dsp, VM::error, fflush(), putchar(), and stdout.

Referenced by register_io_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.398.1.5 io_word_key()

```
static void io_word_key (
            VM * vm )  [static]
```

FORTH word KEY - Input character from terminal.

Parameters

vm	Pointer to the VM structure

Stack effect: (– c) Reads one character from terminal and pushes it to stack

References VM::data_stack, VM::dsp, VM::error, getchar(), and STACK_SIZE.

Referenced by register_io_words().

Here is the call graph for this function:

Here is the caller graph for this function:

7.398.1.6 io_word_question_terminal()

```
static void io_word_question_terminal (
            VM * vm )  [static]
```

FORTH word ?TERMINAL - Check if input is available.

Parameters

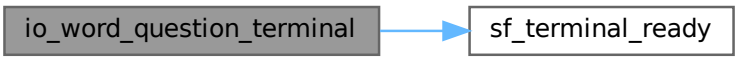
vm	Pointer to the VM structure

Stack effect: (– flag) Pushes true if input is available, false otherwise

References VM::data_stack, VM::dsp, VM::error, sf_terminal_ready(), and STACK_SIZE.

Referenced by register_io_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.398.1.7 io_word_space()

```
static void io_word_space (
            VM * vm )  [static]
```

FORTH word SPACE - Output one space character.

Parameters

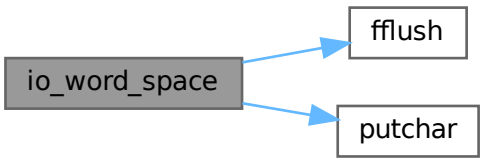
vm	Pointer to the VM structure

Stack effect: (–) Outputs a single space character to terminal

References fflush(), putchar(), and stdout.

Referenced by register_io_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.398.1.8 io_word_spaces()

```
static void io_word_spaces (
            VM * vm )  [static]
```

FORTH word SPACES - Output multiple spaces.

Parameters

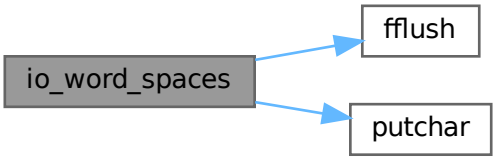
vm	Pointer to the VM structure

Stack effect: (n –) Outputs n space characters to terminal

References count, VM::data_stack, VM::dsp, VM::error, fflush(), putchar(), and stdout.

Referenced by register_io_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.398.1.9 io_word_type()

```
static void io_word_type (
            VM * vm )  [static]
```

FORTH word TYPE - Output string of characters.

Parameters

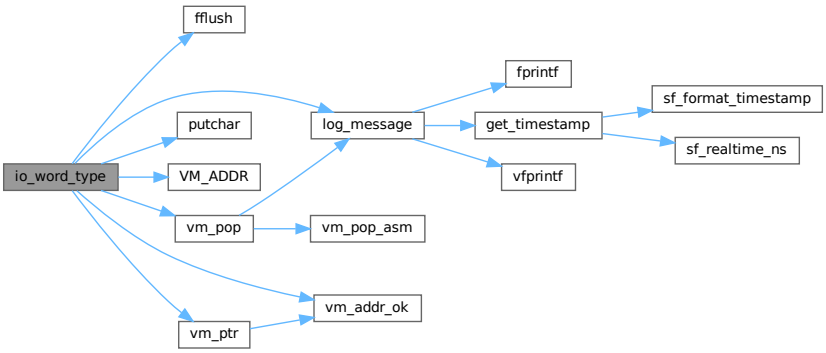
vm	Pointer to the VM structure

Stack effect: (addr u –) Outputs u characters from memory starting at addr

References count, VM::dsp, VM::error, fflush(), LOG_DEBUG, LOG_ERROR, log_message(), putchar(), stdout, VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_ptr().

Referenced by register_io_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.398.1.10 register_io_words()

```
void register_io_words (
            VM * vm )
```

Registers all I/O related FORTH words with the virtual machine.

Parameters

vm	Pointer to the virtual machine instance

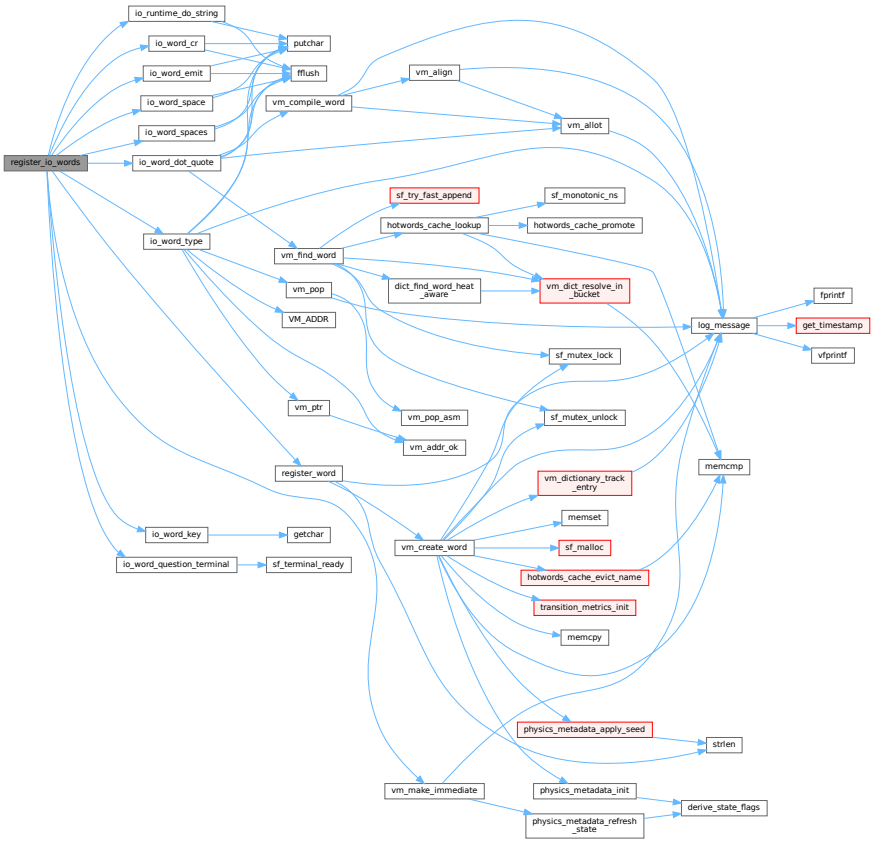
Registers the following FORTH-79 I/O & Terminal Words:

- EMIT (c –) Output character c
- CR (–) Output carriage return
- KEY (– c) Input character from keyboard
- ?TERMINAL (– flag) True if key pressed
- TYPE (addr u –) Output u characters from addr
- SPACE (–) Output one space
- SPACES (n –) Output n spaces

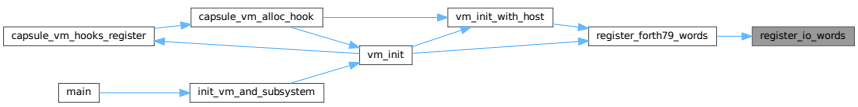
References io_runtime_do_string(), io_word_cr(), io_word_dot_quote(), io_word_emit(), io_word_key(), io_word_question_terminal(), io_word_space(), io_word_spaces(), io_word_type(), register_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



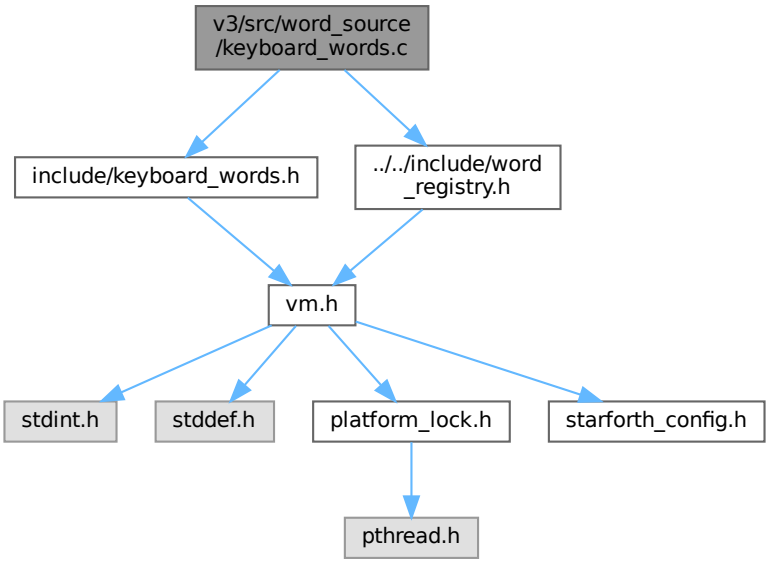
Here is the caller graph for this function:



7.399 v3/src/word_source/keyboard_words.c File Reference

```
#include "include/keyboard_words.h"
#include "../../include/word_registry.h"
```

Include dependency graph for keyboard_words.c:



Functions

- static void kbw_scan (VM *vm)
- static void kbw_debug (VM *vm)
- static void kbw_vevent (VM *vm)
- static void kbw_vdebug (VM *vm)
- int sk_key_event_poll (uint16_t *keycode, int *pressed)

  FABRIC-0.md item 4.4v: shared C-level poll, one converged event stream (Linux input keycode namespace, all three architectures) – see KEY-EVENT's doc comment above for the full per-arch translation rationale.
- static void kbw_key_event (VM *vm)
- static void kbw_alt_tab (VM *vm)
- void register_keyboard_words (VM *vm)

Variables

- volatile uint32_t g_i8042_isr_count
- volatile uint32_t g_spurious_count
- volatile uint32_t g_virtio_input_isr_count

7.399.1 Function Documentation

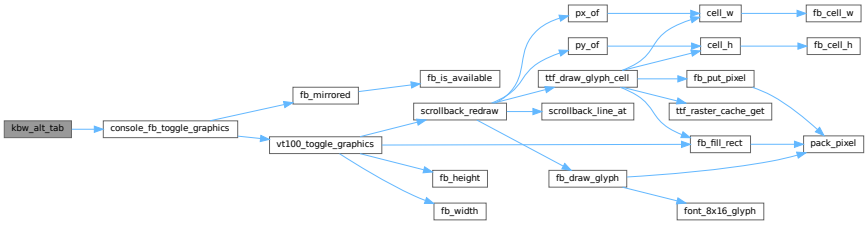
7.399.1.1 kbw_alt_tab()

```
static void kbw_alt_tab (
            VM * vm )  [static]
```

References console_fb_toggle_graphics().

Referenced by register_keyboard_words().

Here is the call graph for this function:



Here is the caller graph for this function:

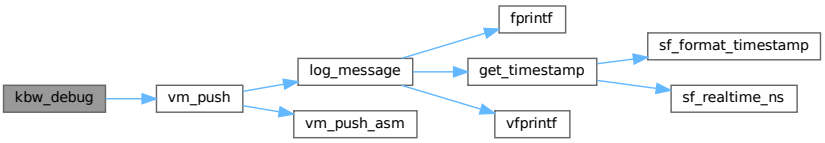
7.399.1.2 kbw_debug()

```
static void kbw_debug (
            VM * vm )  [static]
```

References g_i8042_isr_count, g_spurious_count, and vm_push().

Referenced by register_keyboard_words().

Here is the call graph for this function:



Here is the caller graph for this function:

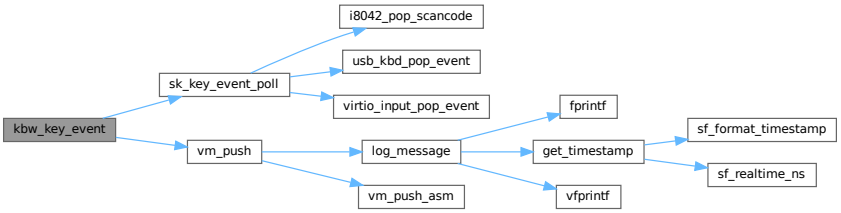
7.399.1.3 kbw_key_event()

```
static void kbw_key_event (
            VM * vm )  [static]
```

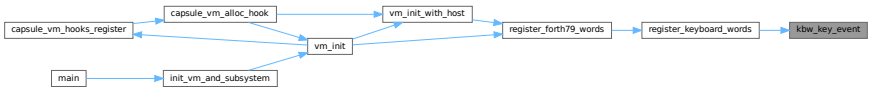
References keycode, pressed, sk_key_event_poll(), and vm_push().

Referenced by register_keyboard_words().

Here is the call graph for this function:



Here is the caller graph for this function:



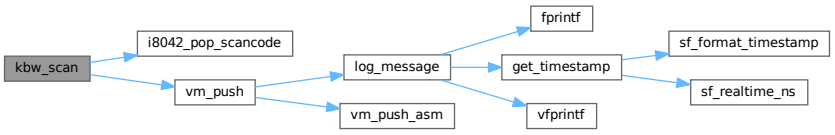
7.399.1.4 kbw_scan()

```
static void kbw_scan (
            VM * vm )  [static]
```

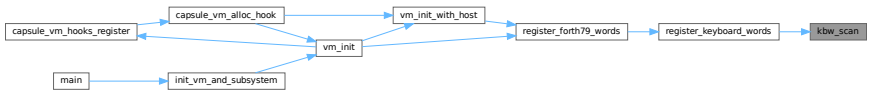
References i8042_pop_scancode(), and vm_push().

Referenced by register_keyboard_words().

Here is the call graph for this function:



Here is the caller graph for this function:



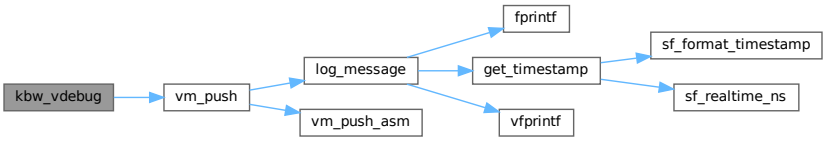
7.399.1.5 kbw_vdebug()

```
static void kbw_vdebug (
            VM * vm )  [static]
```

References g_virtio_input_isr_count, and vm_push().

Referenced by register_keyboard_words().

Here is the call graph for this function:



Here is the caller graph for this function:

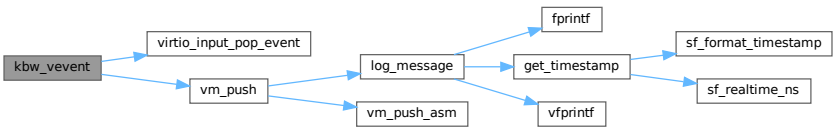
7.399.1.6 kbw_vevent()

```
static void kbw_vevent (
            VM * vm )  [static]
```

References virtio_input_pop_event(), and vm_push().

Referenced by register_keyboard_words().

Here is the call graph for this function:



Here is the caller graph for this function:

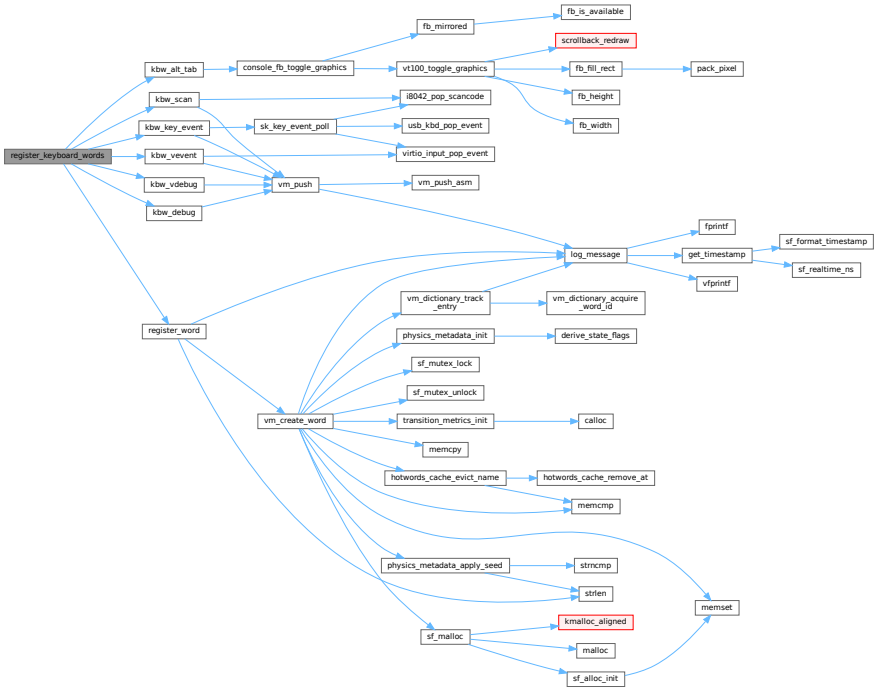
7.399.1.7 register_keyboard_words()

```
void register_keyboard_words (
            VM * vm )
```

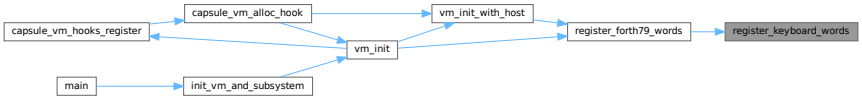
References kbw_alt_tab(), kbw_debug(), kbw_key_event(), kbw_scan(), kbw_vdebug(), kbw_vevent(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.399.1.8 sk_key_event_poll()

```
int sk_key_event_poll (
            uint16_t * keycode,
            int * pressed )
```

FABRIC-0.md item 4.4v: shared C-level poll, one converged event stream (Linux input keycode namespace, all three architectures) – see KEY-EVENT's doc comment above for the full per-arch translation rationale.

Used by both kbw_key_event() (the FORTH word) and the REPL keyboard bridge (repl.c), so there is exactly one implementation.

Parameters

keycode	Written with the Linux input keycode on success.
pressed	Written with 1 (press/repeat) or 0 (release) on success.

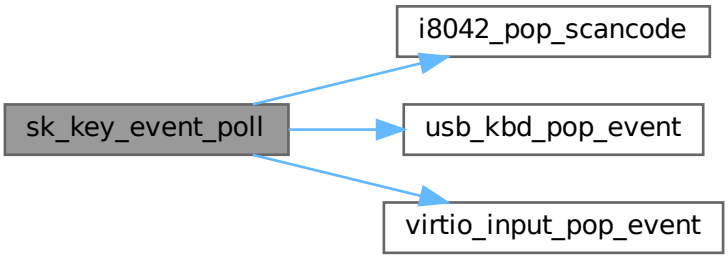
Returns

1 if an event was available, 0 otherwise.

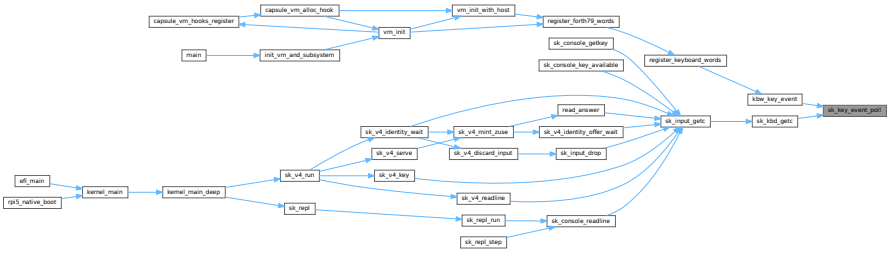
References i8042_pop_scancode(), keycode, pressed, usb_kbd_pop_event(), and virtio_input_pop_event().

Referenced by kbw_key_event(), and sk_kbd_getc().

Here is the call graph for this function:



Here is the caller graph for this function:



7.399.2 Variable Documentation

7.399.2.1 g_i8042_isr_count

```
volatile uint32_t g_i8042_isr_count  [extern]
```

Referenced by kbw_debug().

7.399.2.2 g_spurious_count

```
volatile uint32_t g_spurious_count  [extern]
```

Referenced by isr_common_handler(), and kbw_debug().

7.399.2.3 g_virtio_input_isr_count

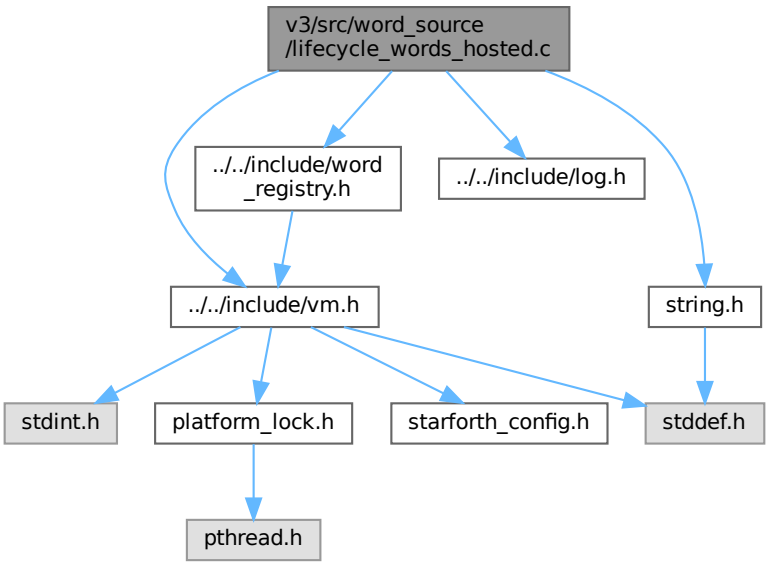
```
volatile uint32_t g_virtio_input_isr_count  [extern]
```

Referenced by kbw_vdebug(), and virtio_input_isr().

7.400 v3/src/word_source/lifecycle_words_hosted.c File Reference

```
#include "../../include/vm.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include <string.h>
```

Include dependency graph for lifecycle_words_hosted.c:



Macros

- #define NAME_MAX_LEN 64

  lifecycle_words_hosted.c - BIRTH KILL PAUSE RESUME USE for hosted Linux build

Functions

- static void extract_name (VM ∗vm, cell_t caddr, cell_t u, char ∗out, int max)
- static void hosted_lifecycle_birth (VM ∗vm)
- static void hosted_lifecycle_kill (VM ∗vm)
- static void hosted_lifecycle_pause (VM ∗vm)
- static void hosted_lifecycle_resume (VM ∗vm)
- static void hosted_lifecycle_use (VM ∗vm)
- void register_lifecycle_words (VM ∗vm)

7.400.1 Macro Definition Documentation

7.400.1.1 NAME_MAX_LEN

```
#define NAME_MAX_LEN 64
```

lifecycle_words_hosted.c - BIRTH KILL PAUSE RESUME USE for hosted Linux build

Hosted-build counterpart to src/starkernel/capsule/lifecycle_words.c. All five words are (c-addr u –), stack-clean, errors logged not pushed.

In the hosted build there is no capsule blob or kernel VM registry, so these implementations consume their arguments and log intent. The kernel build (compiled with -D__STARKERNEL__) skips this file entirely because the Makefile glob does not include src/word_source/ for that target.

7.400.2 Function Documentation

7.400.2.1 extract_name()

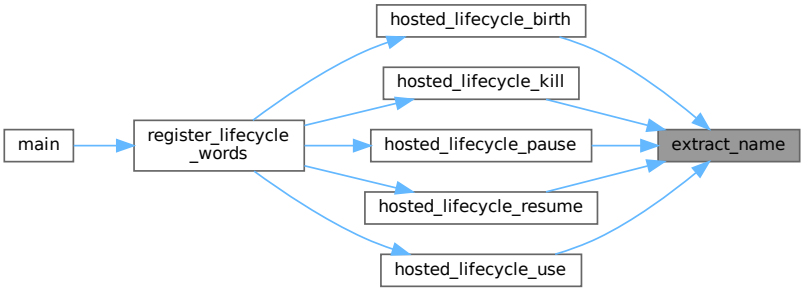
```
static void extract_name (
            VM * vm,
            cell_t caddr,
            cell_t u,
            char * out,
            int max )  [static]
```

References memcpy(), VM::memory, and VM_MEMORY_SIZE.

Referenced by hosted_lifecycle_birth(), hosted_lifecycle_kill(), hosted_lifecycle_pause(), hosted_lifecycle_resume(), and hosted_lifecycle_use().

Here is the call graph for this function:

Here is the caller graph for this function:



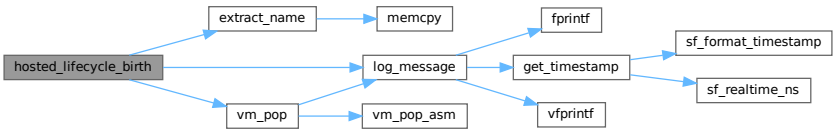
7.400.2.2 hosted_lifecycle_birth()

```
static void hosted_lifecycle_birth (
            VM * vm )  [static]
```

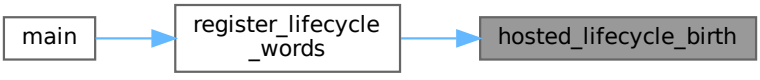
References extract_name(), LOG_INFO, log_message(), name, NAME_MAX_LEN, and vm_pop().

Referenced by register_lifecycle_words().

Here is the call graph for this function:



Here is the caller graph for this function:



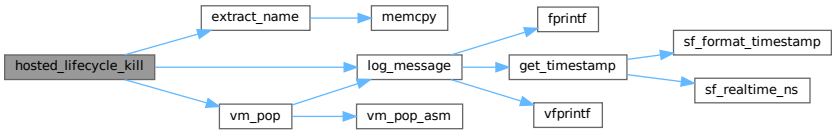
7.400.2.3 hosted_lifecycle_kill()

```
static void hosted_lifecycle_kill (
            VM * vm )  [static]
```

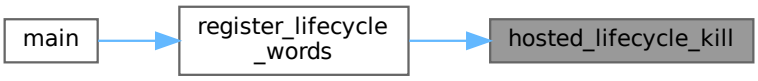
References extract_name(), LOG_INFO, log_message(), name, NAME_MAX_LEN, and vm_pop().

Referenced by register_lifecycle_words().

Here is the call graph for this function:



Here is the caller graph for this function:



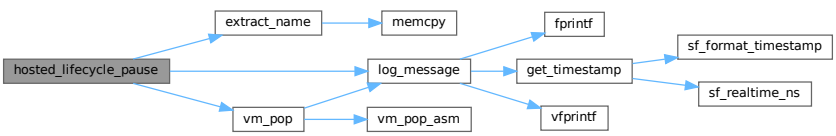
7.400.2.4 hosted_lifecycle_pause()

```
static void hosted_lifecycle_pause (
            VM * vm )  [static]
```

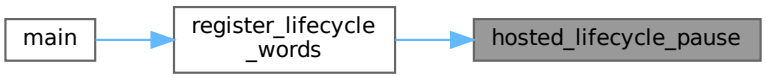
References extract_name(), LOG_INFO, log_message(), name, NAME_MAX_LEN, and vm_pop().

Referenced by register_lifecycle_words().

Here is the call graph for this function:



Here is the caller graph for this function:



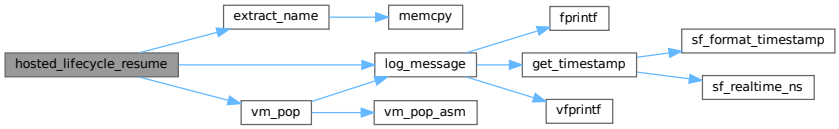
7.400.2.5 hosted_lifecycle_resume()

```
static void hosted_lifecycle_resume (
            VM * vm )  [static]
```

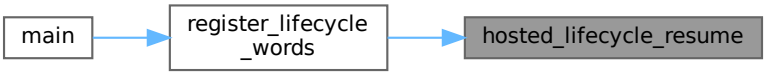
References extract_name(), LOG_INFO, log_message(), name, NAME_MAX_LEN, and vm_pop().

Referenced by register_lifecycle_words().

Here is the call graph for this function:



Here is the caller graph for this function:



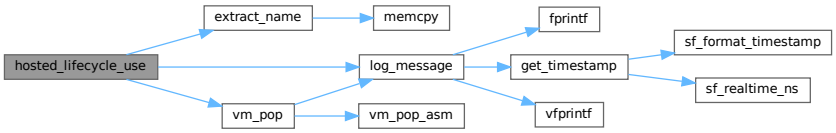
7.400.2.6 hosted_lifecycle_use()

```
static void hosted_lifecycle_use (
            VM * vm )  [static]
```

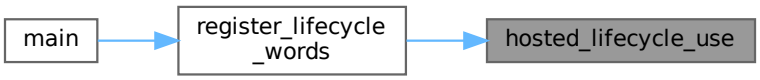
References extract_name(), LOG_INFO, log_message(), name, NAME_MAX_LEN, and vm_pop().

Referenced by register_lifecycle_words().

Here is the call graph for this function:



Here is the caller graph for this function:



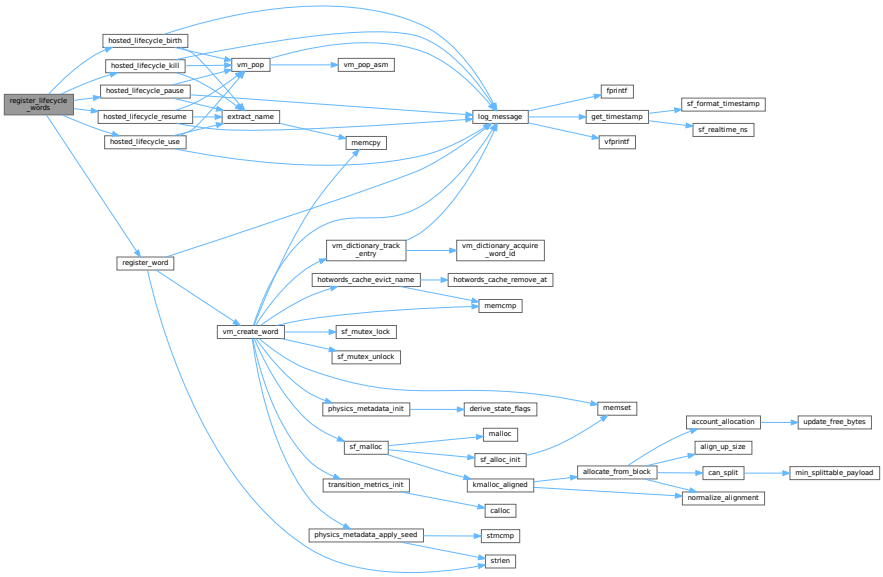
7.400.2.7 register_lifecycle_words()

```
void register_lifecycle_words (
            VM * vm )
```

References hosted_lifecycle_birth(), hosted_lifecycle_kill(), hosted_lifecycle_pause(), hosted_lifecycle_resume(), hosted_lifecycle_use(), and register_word().

Referenced by main().

Here is the call graph for this function:

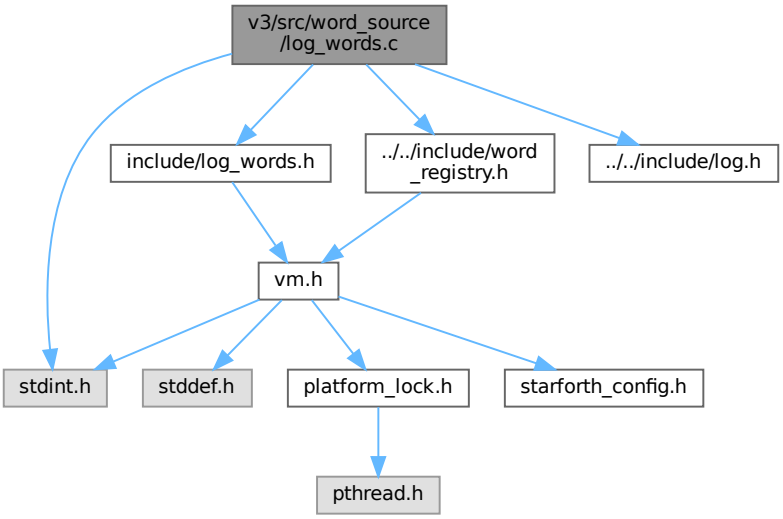


Here is the caller graph for this function:

7.401 v3/src/word_source/log_words.c File Reference

```
#include "include/log_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include <stdint.h>
```

Include dependency graph for log_words.c:



Functions

- static void log_word_error (VM *vm)

  LOG-ERROR (– n) Push the `LOG_ERROR` level constant.
- static void log_word_warn (VM *vm)

  LOG-WARN (– n) Push the `LOG_WARN` level constant.
- static void log_word_info (VM *vm)

  LOG-INFO (– n) Push the `LOG_INFO` level constant.
- static void log_word_test (VM *vm)

LOG-TEST (– n) Push the `LOG_TEST` level constant.

- static void log_word_debug (VM *vm)

  LOG-DEBUG (– n) Push the `LOG_DEBUG` level constant.
- static void log_word_set_level (VM *vm)

  LOG-LEVEL! (n –)
- static void log_word_get_level (VM *vm)

  LOG-LEVEL@ (– n)
- static void log_do_emit (VM *vm, LogLevel level)
- static void log_do_error (VM *vm)
- static void log_do_warn (VM *vm)
- static void log_do_info (VM *vm)
- static void log_do_test (VM *vm)
- static void log_do_debug (VM *vm)
- static void log_emit_string (VM *vm, LogLevel level, const char *rt_name, int rt_nlen)
- static void log_word_error_quote (VM *vm)

  LOG-ERROR" ("ccc<quote>" –) Log string at ERROR level.
- static void log_word_warn_quote (VM *vm)

  LOG-WARN" ("ccc<quote>" –) Log string at WARN level.
- static void log_word_info_quote (VM *vm)

  LOG-INFO" ("ccc<quote>" –) Log string at INFO level.
- static void log_word_test_quote (VM *vm)

  LOG-TEST" ("ccc<quote>" –) Log string at TEST level.
- static void log_word_debug_quote (VM *vm)

  LOG-DEBUG" ("ccc<quote>" –) Log string at DEBUG level.
- static void log_str_emit (VM *vm, LogLevel level)
- static void log_word_error_str (VM *vm)

  LOG-ERROR-STR (c-addr u –) Log string from stack at ERROR level.
- static void log_word_warn_str (VM *vm)

  LOG-WARN-STR (c-addr u –) Log string from stack at WARN level.
- static void log_word_info_str (VM *vm)

  LOG-INFO-STR (c-addr u –) Log string from stack at INFO level.
- static void log_word_test_str (VM *vm)

  LOG-TEST-STR (c-addr u –) Log string from stack at TEST level.
- static void log_word_debug_str (VM *vm)

  LOG-DEBUG-STR (c-addr u –) Log string from stack at DEBUG level.
- void register_log_words (VM *vm)

  Register all log words with the VM dictionary.

7.401.1 Function Documentation

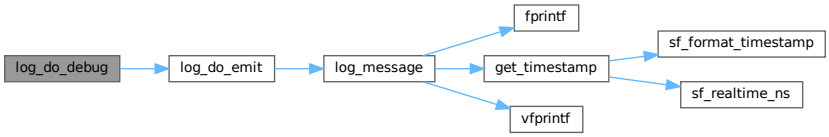
7.401.1.1 log_do_debug()

```
static void log_do_debug (
            VM * vm )  [static]
```

References LOG_DEBUG, and log_do_emit().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

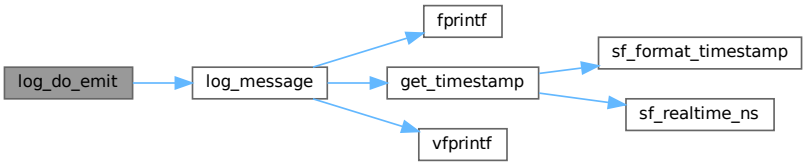
7.401.1.2 log_do_emit()

```
static void log_do_emit (
            VM * vm,
            LogLevel level )  [static]
```

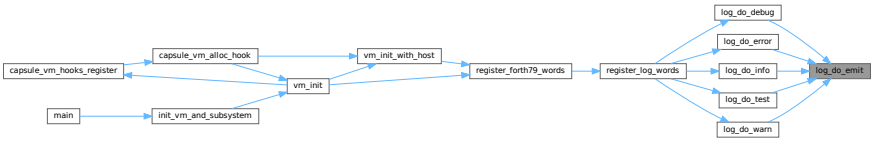
References VM::error, level, LOG_ERROR, log_message(), VM::return_stack, and VM::rsp.

Referenced by log_do_debug(), log_do_error(), log_do_info(), log_do_test(), and log_do_warn().

Here is the call graph for this function:



Here is the caller graph for this function:



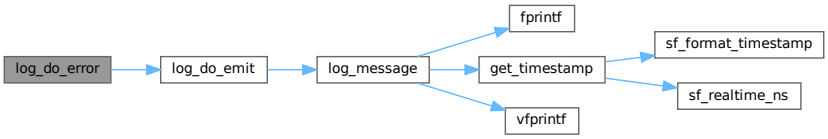
7.401.1.3 log_do_error()

```
static void log_do_error (
            VM * vm )  [static]
```

References log_do_emit(), and LOG_ERROR.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

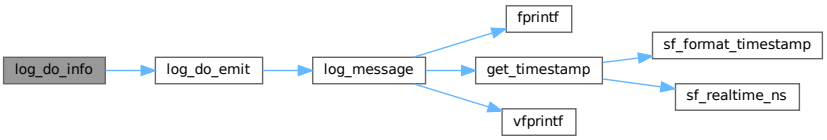
7.401.1.4 log_do_info()

```
static void log_do_info (
            VM * vm )  [static]
```

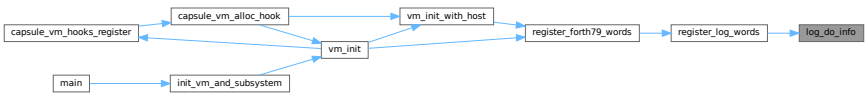
References log_do_emit(), and LOG_INFO.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:



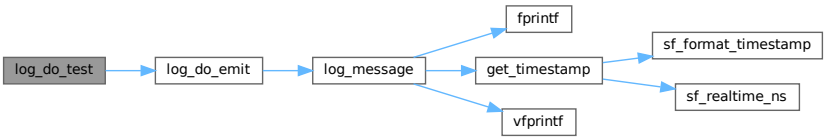
7.401.1.5 log_do_test()

```
static void log_do_test (
            VM * vm )  [static]
```

References log_do_emit(), and LOG_TEST.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

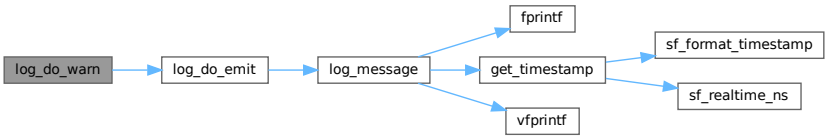
7.401.1.6 log_do_warn()

```
static void log_do_warn (
            VM * vm )  [static]
```

References log_do_emit(), and LOG_WARN.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

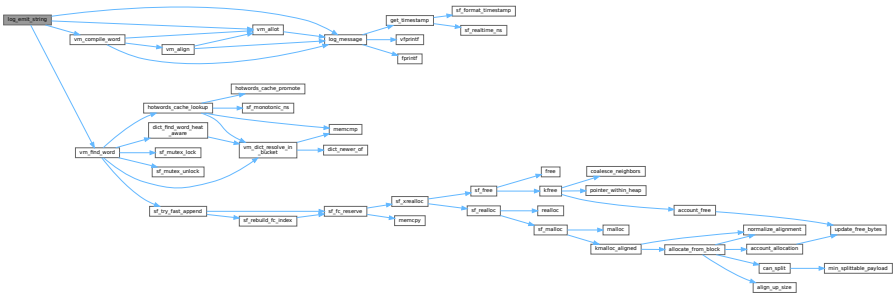
7.401.1.7 log_emit_string()

```
static void log_emit_string (
            VM * vm,
            LogLevel level,
            const char * rt_name,
            int rt_nlen )  [static]
```

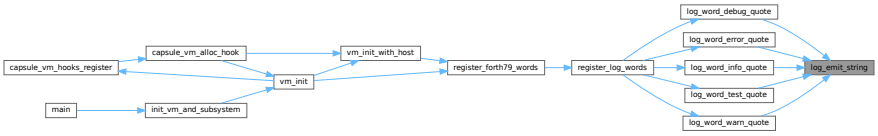
References VM::error, VM::input_buffer, VM::input_length, VM::input_pos, level, LOG_ERROR, log_message(), VM::mode, MODE_INTERPRET, vm_allot(), vm_compile_word(), and vm_find_word().

Referenced by log_word_debug_quote(), log_word_error_quote(), log_word_info_quote(), log_word_test_quote(), and log_word_warn_quote().

Here is the call graph for this function:



Here is the caller graph for this function:



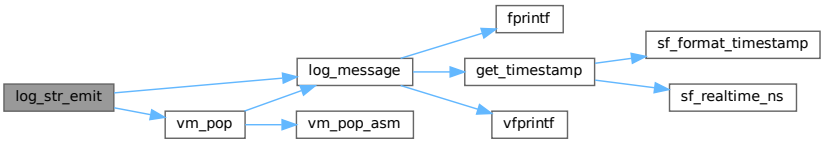
7.401.1.8 log_str_emit()

```
static void log_str_emit (
            VM * vm,
            LogLevel level )  [static]
```

References VM::dsp, VM::error, level, LOG_ERROR, LOG_LINE_MAX, log_message(), VM::memory, VM_MEMORY_SIZE, and vm_pop().

Referenced by log_word_debug_str(), log_word_error_str(), log_word_info_str(), log_word_test_str(), and log_word_warn_str().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.9 log_word_debug()

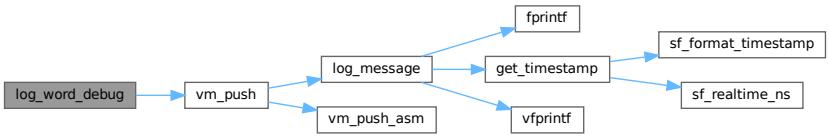
```
static void log_word_debug (
            VM * vm )  [static]
```

LOG-DEBUG (– n) Push the `LOG_DEBUG` level constant.

References LOG_DEBUG, and vm_push().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.10 log_word_debug_quote()

```
static void log_word_debug_quote (
            VM * vm )  [static]
```

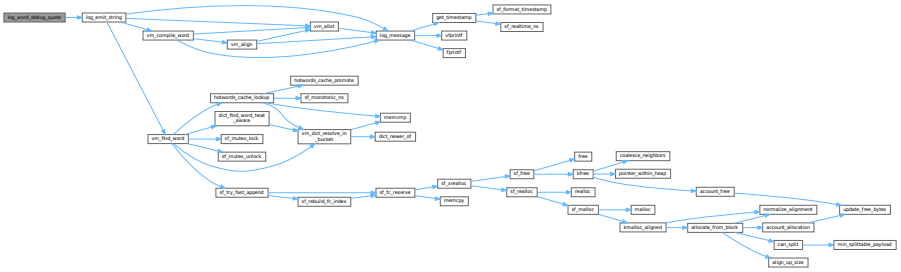
LOG-DEBUG" ("ccc<quote>" –) Log string at DEBUG level.

IMMEDIATE.

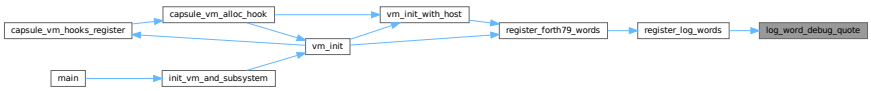
References LOG_DEBUG, and log_emit_string().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.401.1.11 log_word_debug_str()

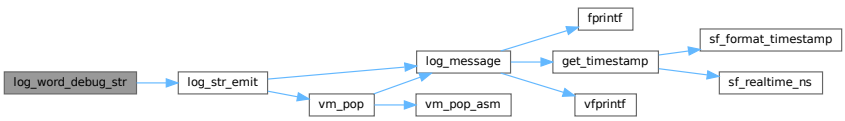
```
static void log_word_debug_str (
            VM * vm )  [static]
```

LOG-DEBUG-STR (c-addr u –) Log string from stack at DEBUG level.

References LOG_DEBUG, and log_str_emit().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.12 log_word_error()

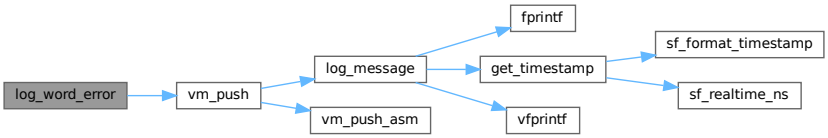
```
static void log_word_error (
            VM * vm )  [static]
```

LOG-ERROR (– n) Push the LOG_ERROR level constant.

References LOG_ERROR, and vm_push().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.13 log_word_error_quote()

```
static void log_word_error_quote (
            VM * vm )  [static]
```

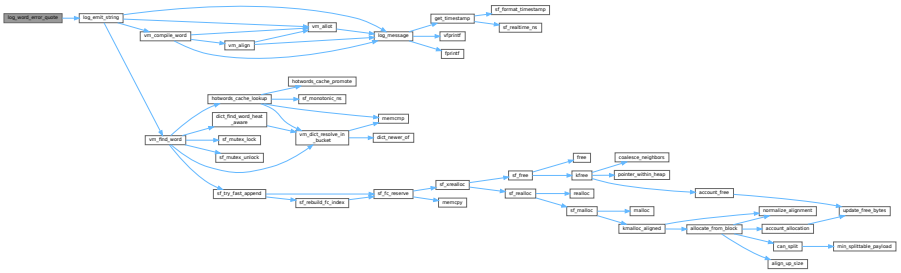
LOG-ERROR" ("ccc<quote>" –) Log string at ERROR level.

IMMEDIATE.

References log_emit_string(), and LOG_ERROR.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.14 log_word_error_str()

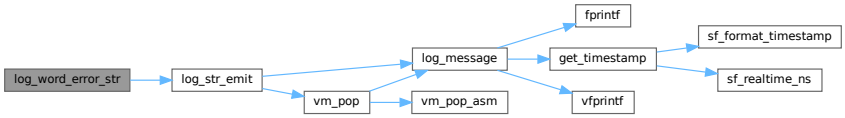
```
static void log_word_error_str (
            VM * vm )  [static]
```

LOG-ERROR-STR (c-addr u –) Log string from stack at ERROR level.

References LOG_ERROR, and log_str_emit().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.15 log_word_get_level()

```
static void log_word_get_level (
            VM * vm )  [static]
```

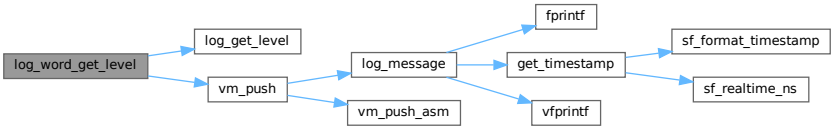
LOG-LEVEL@ (– n)

Pushes the current log level onto the stack.

References log_get_level(), and vm_push().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.16 log_word_info()

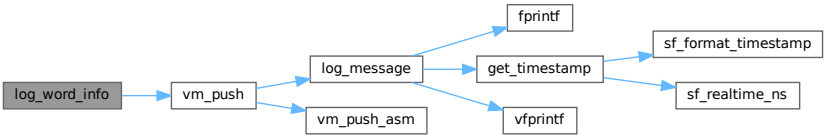
```
static void log_word_info (
            VM * vm )  [static]
```

LOG-INFO (– n) Push the `LOG_INFO` level constant.

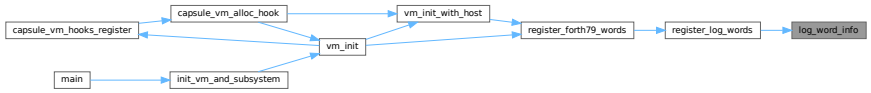
References LOG_INFO, and vm_push().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.401.1.17 log_word_info_quote()

```
static void log_word_info_quote (
            VM * vm )  [static]
```

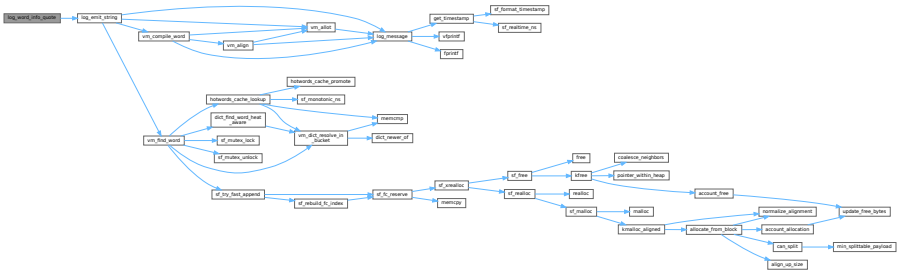
LOG-INFO" ("ccc<quote>" –) Log string at INFO level.

IMMEDIATE.

References log_emit_string(), and LOG_INFO.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.18 log_word_info_str()

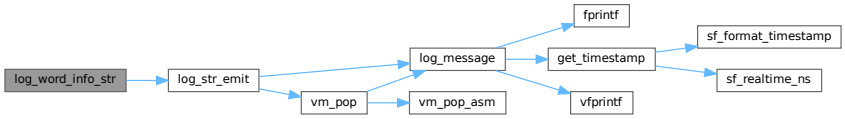
```
static void log_word_info_str (
            VM * vm )  [static]
```

LOG-INFO-STR (c-addr u –) Log string from stack at INFO level.

References LOG_INFO, and log_str_emit().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.19 log_word_set_level()

```
static void log_word_set_level (
            VM * vm )  [static]
```

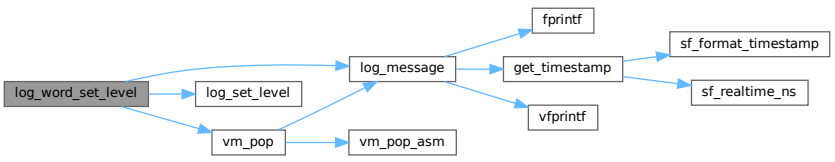
LOG-LEVEL! (n –)

Pops `n` from the stack and calls `log_set_level()` to update the active log filter. Values below `LOG_ERROR` are clamped to `LOG_ERROR`; values above `LOG_DEBUG` are clamped to `LOG_DEBUG`.

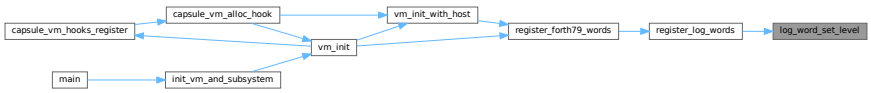
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), log_set_level(), and vm_pop().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.401.1.20 log_word_test()

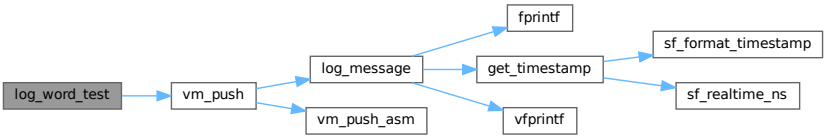
```
static void log_word_test (
            VM * vm )  [static]
```

LOG-TEST (– n) Push the LOG_TEST level constant.

References LOG_TEST, and vm_push().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.21 log_word_test_quote()

```
static void log_word_test_quote (
            VM * vm )  [static]
```

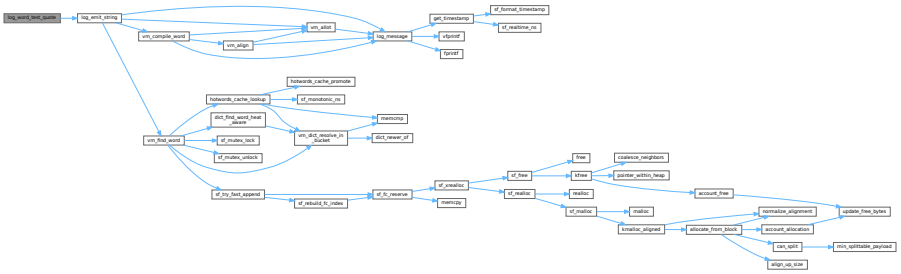
LOG-TEST" ("ccc<quote>" –) Log string at TEST level.

IMMEDIATE.

References log_emit_string(), and LOG_TEST.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.22 log_word_test_str()

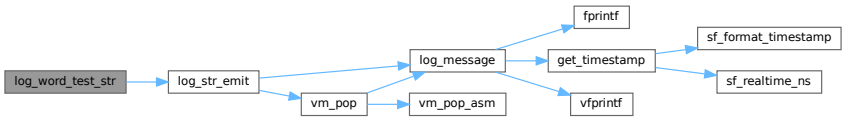
```
static void log_word_test_str (
            VM * vm )  [static]
```

LOG-TEST-STR (c-addr u –) Log string from stack at TEST level.

References log_str_emit(), and LOG_TEST.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.23 log_word_warn()

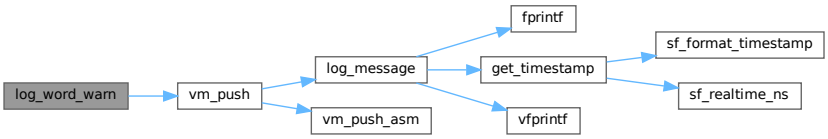
```
static void log_word_warn (
            VM * vm )  [static]
```

LOG-WARN (– n) Push the LOG_WARN level constant.

References LOG_WARN, and vm_push().

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.24 log_word_warn_quote()

```
static void log_word_warn_quote (
            VM * vm )  [static]
```

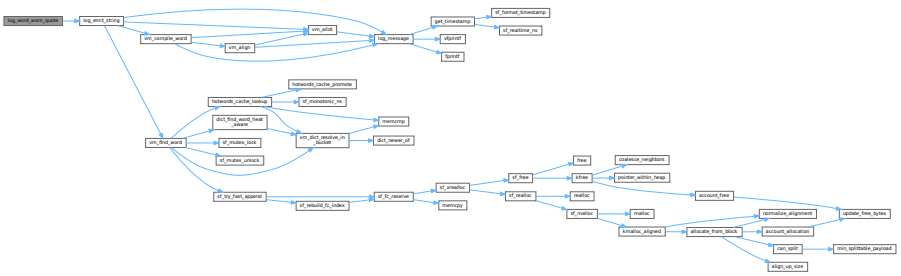
LOG-WARN" ("ccc<quote>" –) Log string at WARN level.

IMMEDIATE.

References log_emit_string(), and LOG_WARN.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.25 log_word_warn_str()

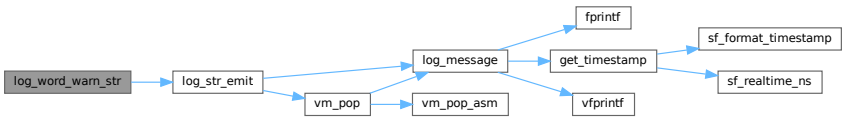
```
static void log_word_warn_str (
            VM * vm )  [static]
```

LOG-WARN-STR (c-addr u –) Log string from stack at WARN level.

References log_str_emit(), and LOG_WARN.

Referenced by register_log_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.401.1.26 register_log_words()

```
void register_log_words (
            VM * vm )
```

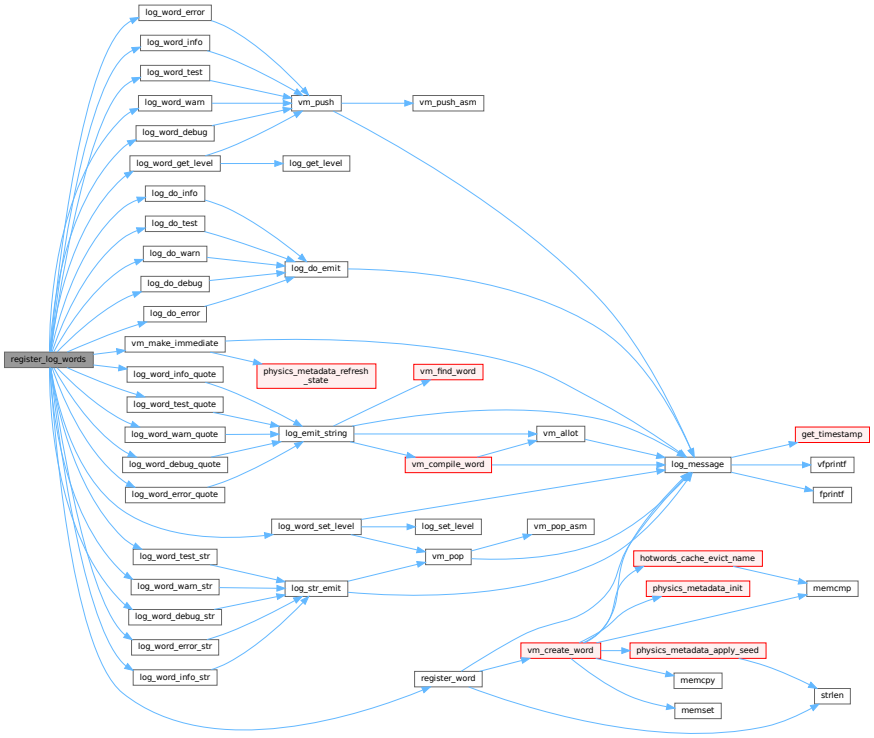
Register all log words with the VM dictionary.

Level constants: LOG-ERROR LOG-WARN LOG-INFO LOG-TEST LOG-DEBUG Level control: LOG-LEVEL! LOG-LEVEL@ Runtime (hidden): (do-log-error) (do-log-warn) (do-log-info) (do-log-test) (do-log-debug) String literals: LOG-ERROR" LOG-WARN" LOG-INFO" LOG-TEST" LOG-DEBUG" Stack string: LOG-ERROR-STR LOG-WARN-STR LOG-INFO-STR LOG-TEST-STR LOG-DEBUG-STR Kernel only: (LOG-APPEND-RAW) LOG-FLUSH

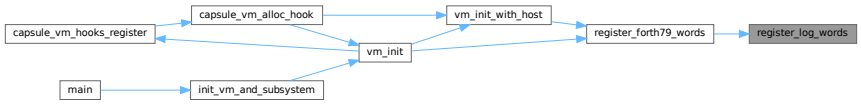
References log_do_debug(), log_do_error(), log_do_info(), log_do_test(), log_do_warn(), log_word_debug(), log_word_debug_quote(), log_word_debug_str(), log_word_error(), log_word_error_quote(), log_word_error_str(), log_word_get_level(), log_word_info(), log_word_info_quote(), log_word_info_str(), log_word_set_level(), log_word_test(), log_word_test_quote(), log_word_test_str(), log_word_warn(), log_word_warn_quote(), log_word_warn_str(), register_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

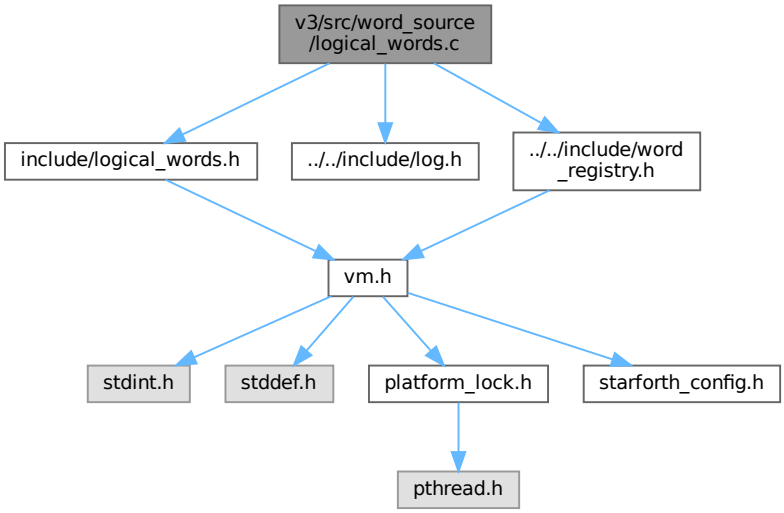


7.402 v3/src/word_source/logical_words.c File Reference

```
#include "include/logical_words.h"
#include "../../include/log.h"
```

```
#include "../../include/word_registry.h"
```

Include dependency graph for logical_words.c:



Macros

- #define FORTH_TRUE ((cell_t)-1)

  FORTH-79 TRUE value (-1, all bits set)
- #define FORTH_FALSE ((cell_t)0)

  FORTH-79 FALSE value (0)

Functions

- static void logical_word_and (VM *vm)

  FORTH word: AND (n1 n2 – n3)
- static void logical_word_or (VM *vm)
- static void logical_word_xor (VM *vm)
- static void logical_word_not (VM *vm)
- static void logical_word_invert (VM *vm)
- static void logical_word_lshift (VM *vm)
- static void logical_word_rshift (VM *vm)
- static void logical_word_zero_equals (VM *vm)
- static void logical_word_zero_less (VM *vm)
- static void logical_word_zero_greater (VM *vm)
- static void logical_word_equals (VM *vm)
- static void logical_word_not_equals (VM *vm)
- static void logical_word_less_than (VM *vm)
- static void logical_word_greater_than (VM *vm)
- static void logical_word_greater_equal (VM *vm)
- static void logical_word_less_equal (VM *vm)
- static void logical_word_u_less_than (VM *vm)
- static void logical_word_u_greater_than (VM *vm)

- static void logical_word_within (VM *vm)
- static void logical_word_true (VM *vm)
- static void logical_word_false (VM *vm)
- void logical_word_zero_not_equal (VM *vm)
- void register_logical_words (VM *vm)

  Registers all FORTH-79 logical and comparison words with the VM.

7.402.1 Macro Definition Documentation

7.402.1.1 FORTH_FALSE

```
#define FORTH_FALSE ((cell_t)0)
```

FORTH-79 FALSE value (0)

7.402.1.2 FORTH_TRUE

```
#define FORTH_TRUE ((cell_t)-1)
```

FORTH-79 TRUE value (-1, all bits set)

7.402.2 Function Documentation

7.402.2.1 logical_word_and()

```
static void logical_word_and (
            VM * vm )  [static]
```

FORTH word: AND (n1 n2 – n3)

Parameters

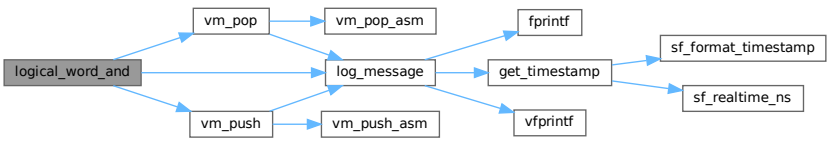
vm	Pointer to the virtual machine instance

Performs bitwise AND operation on top two stack values. Stack effect: (n1 n2 – n3) where n3 = n1 AND n2

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

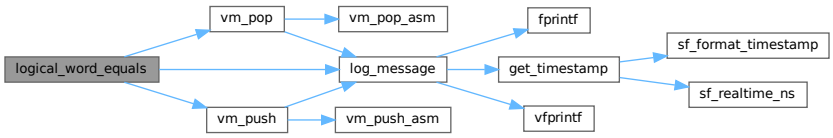
7.402.2.2 logical_word_equals()

```
static void logical_word_equals (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

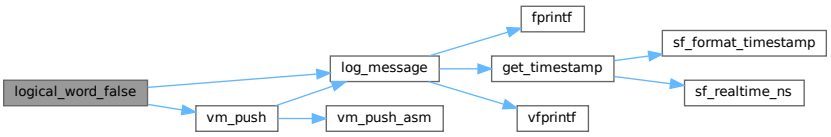
7.402.2.3 logical_word_false()

```
static void logical_word_false (
            VM * vm )  [static]
```

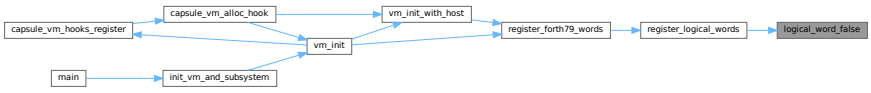
References FORTH_FALSE, LOG_DEBUG, log_message(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:



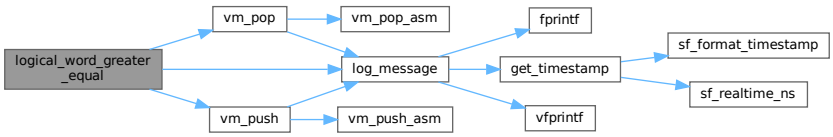
7.402.2.4 logical_word_greater_equal()

```
static void logical_word_greater_equal (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

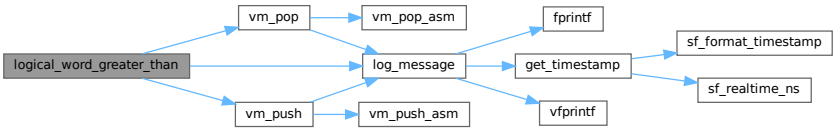
7.402.2.5 logical_word_greater_than()

```
static void logical_word_greater_than (
            VM * vm )  [static]
```

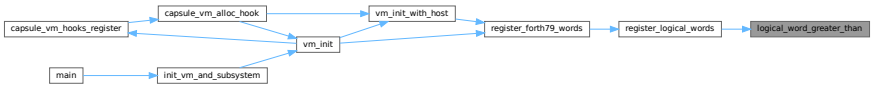
References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:



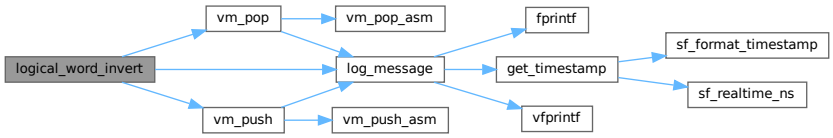
7.402.2.6 logical_word_invert()

```
static void logical_word_invert (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

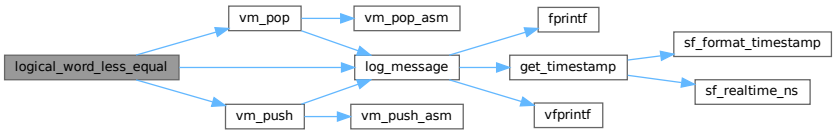
7.402.2.7 logical_word_less_equal()

```
static void logical_word_less_equal (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

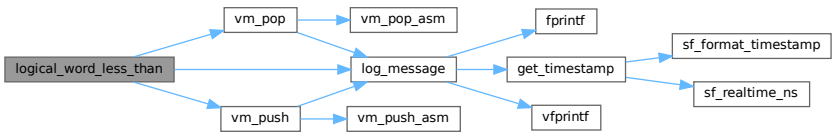
7.402.2.8 logical_word_less_than()

```
static void logical_word_less_than (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

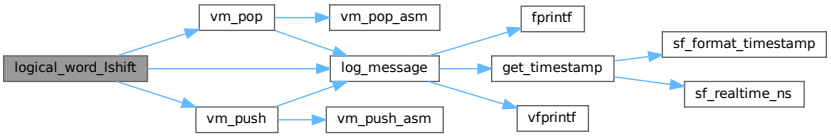
7.402.2.9 logical_word_lshift()

```
static void logical_word_lshift (
            VM * vm )  [static]
```

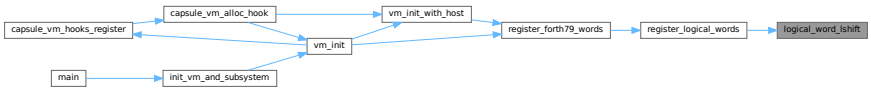
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:



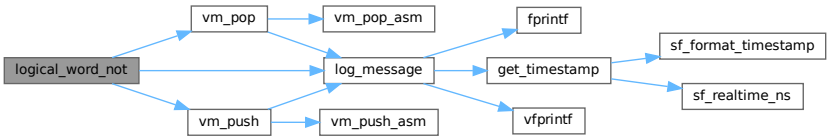
7.402.2.10 logical_word_not()

```
static void logical_word_not (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

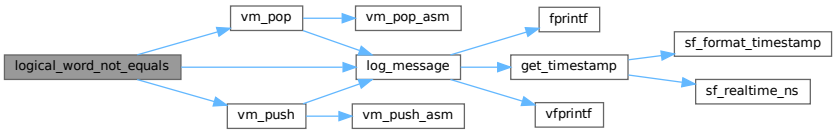
7.402.2.11 logical_word_not_equals()

```
static void logical_word_not_equals (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

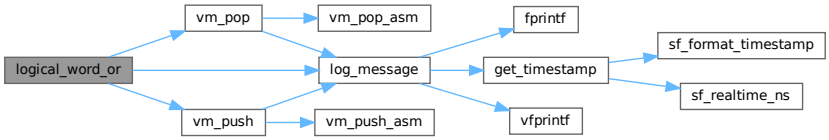
7.402.2.12 logical_word_or()

```
static void logical_word_or (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

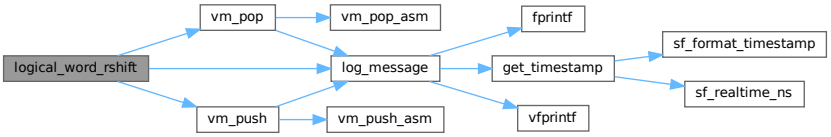
7.402.2.13 logical_word_rshift()

```
static void logical_word_rshift (
            VM * vm )  [static]
```

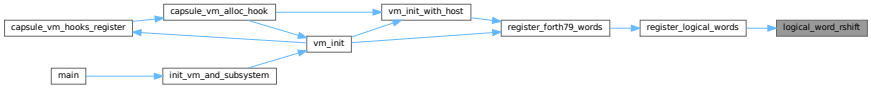
References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:



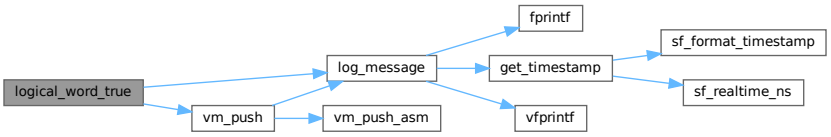
7.402.2.14 logical_word_true()

```
static void logical_word_true (
            VM * vm )  [static]
```

References FORTH_TRUE, LOG_DEBUG, log_message(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

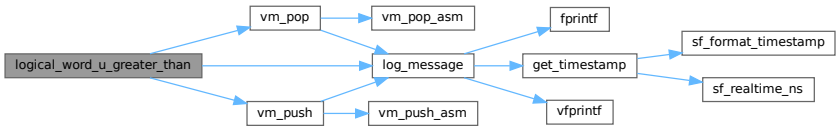
7.402.2.15 logical_word_u_greater_than()

```
static void logical_word_u_greater_than (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

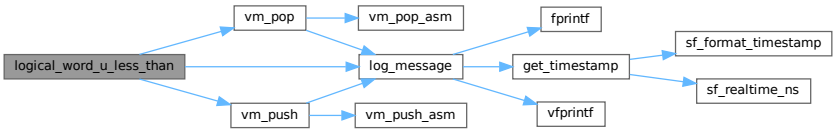
7.402.2.16 logical_word_u_less_than()

```
static void logical_word_u_less_than (
            VM * vm )  [static]
```

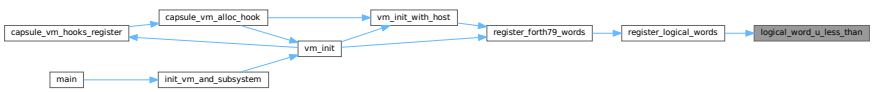
References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:



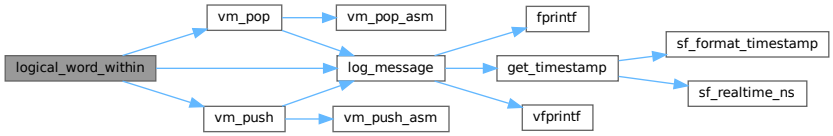
7.402.2.17 logical_word_within()

```
static void logical_word_within (
            VM * vm )  [static]
```

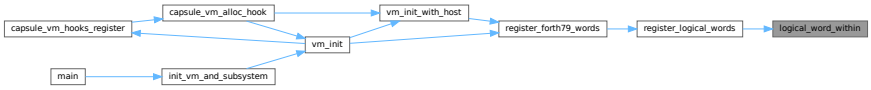
References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:



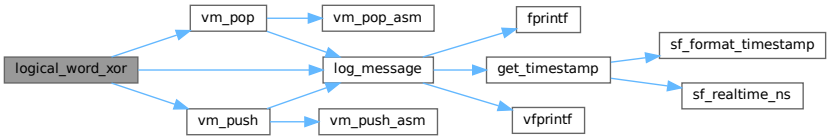
7.402.2.18 logical_word_xor()

```
static void logical_word_xor (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

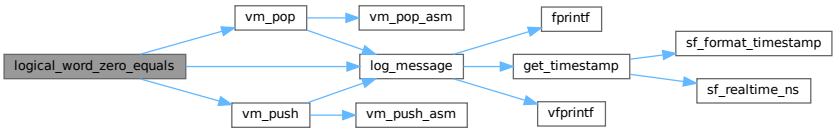
7.402.2.19 logical_word_zero_equals()

```
static void logical_word_zero_equals (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

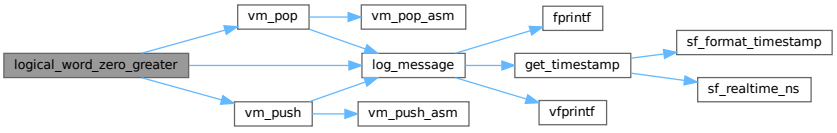
7.402.2.20 logical_word_zero_greater()

```
static void logical_word_zero_greater (
            VM * vm )  [static]
```

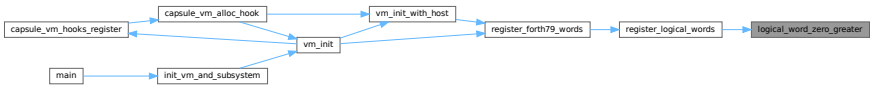
References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:



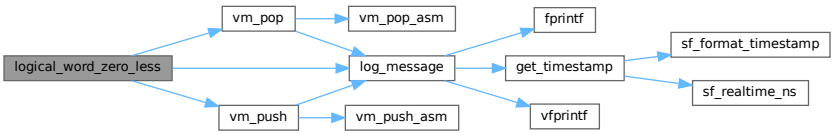
7.402.2.21 logical_word_zero_less()

```
static void logical_word_zero_less (
            VM * vm )  [static]
```

References VM::dsp, VM::error, FORTH_FALSE, FORTH_TRUE, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

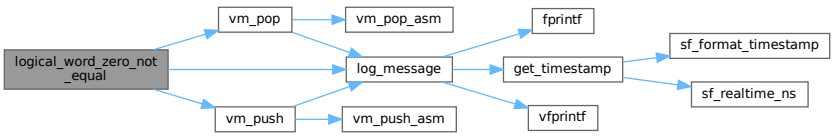
7.402.2.22 logical_word_zero_not_equal()

```
void logical_word_zero_not_equal (
            VM * vm )
```

References VM::dsp, VM::error, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_logical_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.402.2.23 register_logical_words()

```
void register_logical_words (
            VM * vm )
```

Registers all FORTH-79 logical and comparison words with the VM.

Parameters

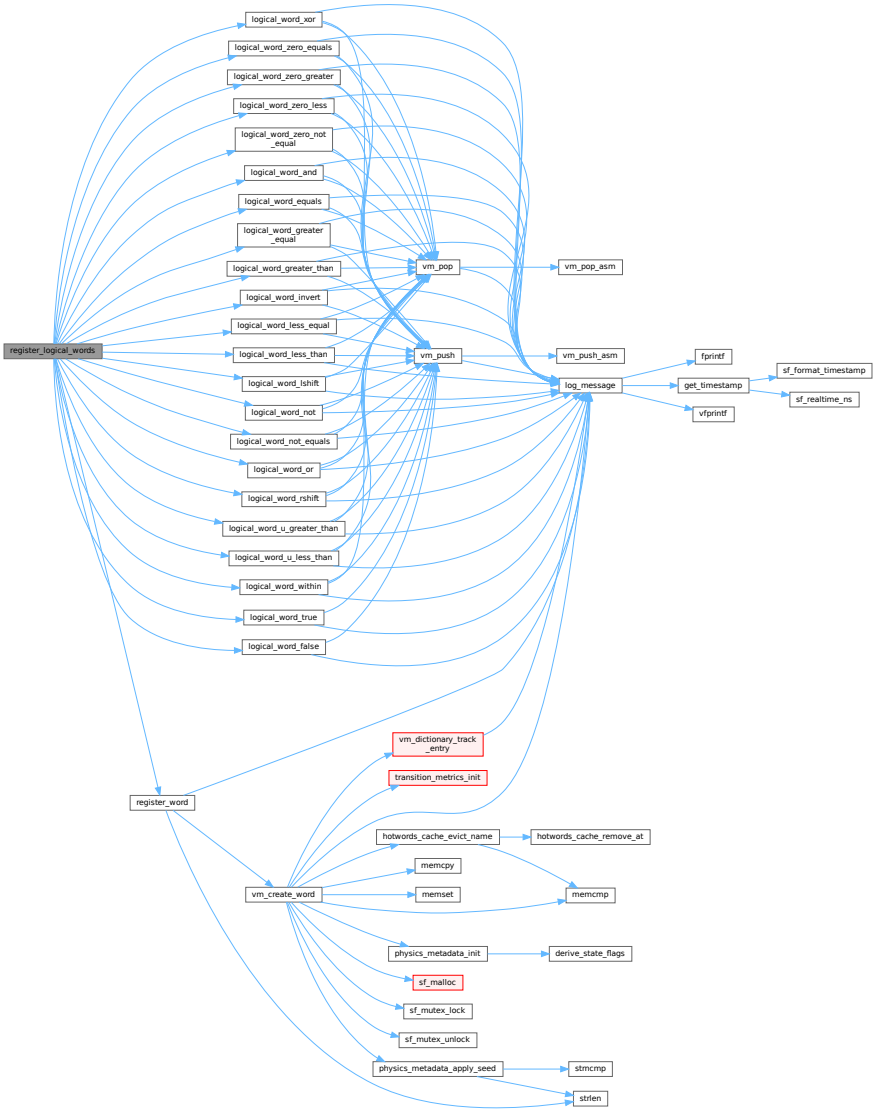
vm	Pointer to the virtual machine instance

Registers logical operations (AND, OR, XOR, NOT), comparison operations (=, <>, <, >, etc.), and constant words (TRUE, FALSE) with the virtual machine.

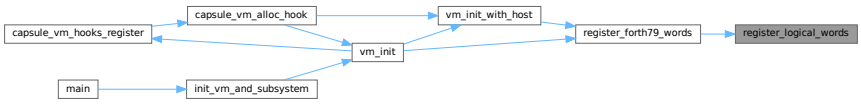
References logical_word_and(), logical_word_equals(), logical_word_false(), logical_word_greater_equal(), logical_word_greater_than(), logical_word_invert(), logical_word_less_equal(), logical_word_less_than(), logical_word_lshift(), logical_word_not(), logical_word_not_equals(), logical_word_or(), logical_word_rshift(), logical_word_true(), logical_word_u_greater_than(), logical_word_u_less_than(), logical_word_within(), logical_word_xor(), logical_word_zero_equals(), logical_word_zero_greater(), logical_word_zero_less(), logical_word_zero_not_equal(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

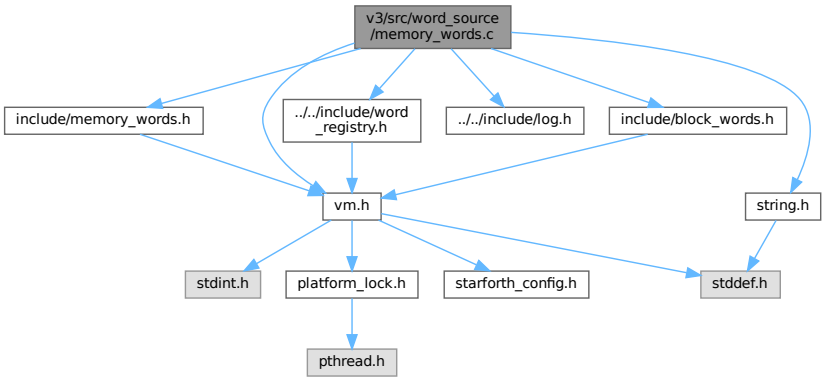


7.403 v3/src/word_source/memory_words.c File Reference

```
#include "include/memory_words.h"
#include "../../include/word_registry.h"
#include "vm.h"
```

```
#include "../../include/log.h"
#include "include/block_words.h"
#include <string.h>
```

Include dependency graph for memory_words.c:



Functions

- void memory_word_fetch (VM *vm)

  Implements FORTH word '@' - Fetch cell from VM memory.
- void memory_word_store (VM *vm)

  Implements FORTH word '!' - Store cell into VM memory.
- void memory_word_cfetch (VM *vm)

  Implements FORTH word 'C@' - Fetch byte from VM memory.
- void memory_word_cstore (VM *vm)

  Implements FORTH word 'C!' - Store byte into VM memory.
- void memory_word_plus_store (VM *vm)

  Implements FORTH word '+!' - Add to contents of cell.
- void memory_word_minus_store (VM *vm)

  Implements FORTH word '-!' - Subtract from contents of cell.
- void memory_word_fill (VM *vm)

  Implements FORTH word 'FILL' - Fill memory with byte value.
- void memory_word_move (VM *vm)

  Implements FORTH word 'MOVE' - Copy memory region.
- void memory_word_erase (VM *vm)

  Implements FORTH word 'ERASE' - Zero memory region.
- void memory_word_2fetch (VM *vm)

  Implements FORTH word '2@' - Fetch double cell.
- void memory_word_2store (VM *vm)

  Implements FORTH word '2!' - Store double cell.
- static void memory_word_cells (VM *vm)

  Implements FORTH word 'CELLS' - Scale by cell size.
- void register_memory_words (VM *vm)

  Register all memory manipulation words with the VM.

7.403.1 Function Documentation

7.403.1.1 memory_word_2fetch()

```
void memory_word_2fetch (
            VM * vm )
```

Implements FORTH word '2@' - Fetch double cell.

Stack effect: (addr – x_low x_high) Fetches two consecutive cells from memory, pushing low cell then high cell.

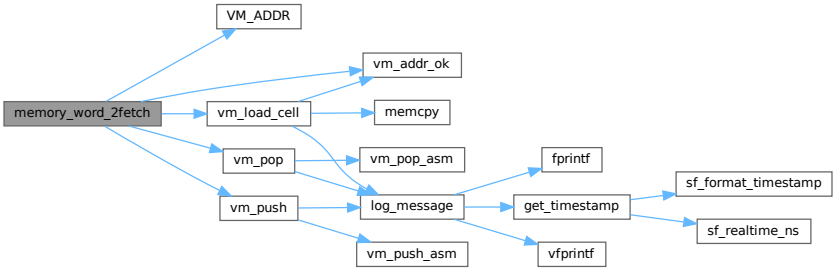
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_load_cell(), vm_pop(), and vm_push().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.2 memory_word_2store()

```
void memory_word_2store (
            VM * vm )
```

Implements FORTH word '2!' - Store double cell.

Stack effect: (x_low x_high addr –) Stores two consecutive cells to memory, low cell at addr, high cell at addr+cell.

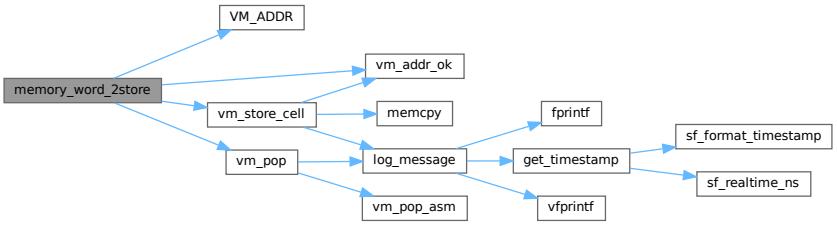
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_store_cell().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.3 memory_word_cells()

```
static void memory_word_cells (
            VM * vm )  [static]
```

Implements FORTH word 'CELLS' - Scale by cell size.

Stack effect: (n – n') Multiplies top of stack by cell size in bytes.

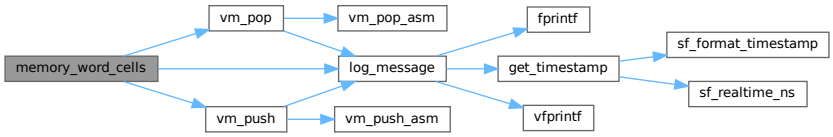
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.4 memory_word_cfetch()

```
void memory_word_cfetch (
            VM * vm )
```

Implements FORTH word 'C@' - Fetch byte from VM memory.

Stack effect: (addr – c) Fetches a byte value from the given memory address in VM memory.

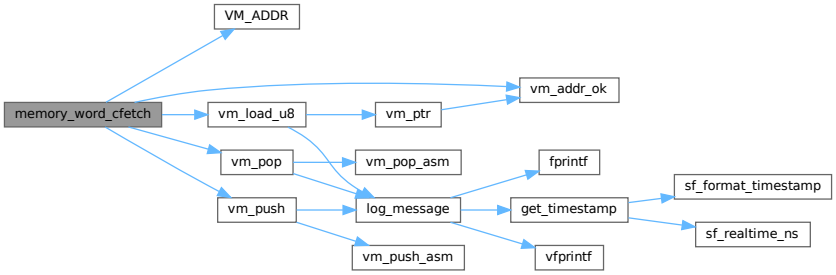
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_push().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.5 memory_word_cstore()

```
void memory_word_cstore (
            VM * vm )
```

Implements FORTH word 'C!' - Store byte into VM memory.

Stack effect: (c addr –) Stores a byte value at the given memory address in VM memory.

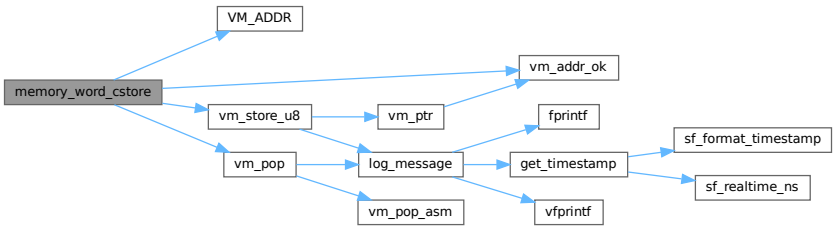
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_store_u8().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.6 memory_word_erase()

```
void memory_word_erase (
            VM * vm )
```

Implements FORTH word 'ERASE' - Zero memory region.

Stack effect: (addr len –) Sets len bytes to zero starting at addr.

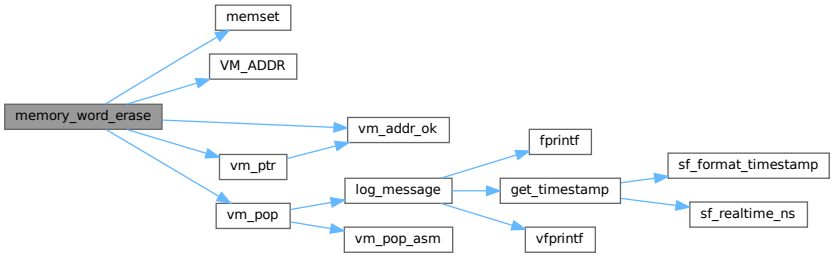
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, memset(), VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_ptr().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.7 memory_word_fetch()

```
void memory_word_fetch (
            VM * vm )
```

Implements FORTH word '@' - Fetch cell from VM memory.

Stack effect: (addr – n) Fetches a cell value from the given memory address in VM memory.

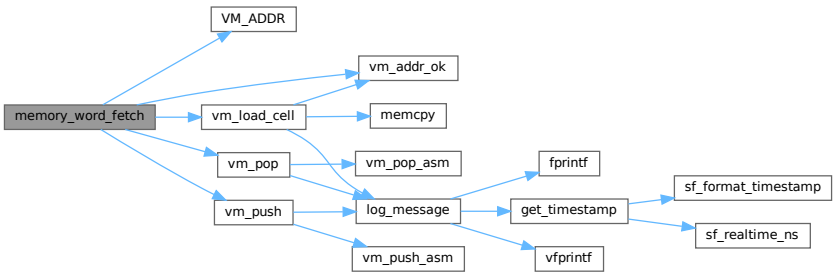
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_load_cell(), vm_pop(), and vm_push().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.8 memory_word_fill()

```
void memory_word_fill (
            VM * vm )
```

Implements FORTH word 'FILL' - Fill memory with byte value.

Stack effect: (addr len c –) Fills len bytes of memory starting at addr with the value c.

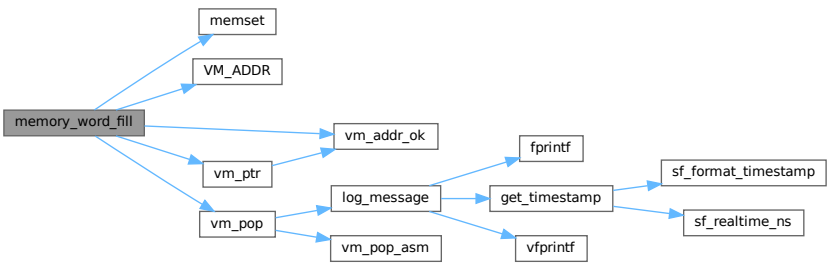
Parameters

vm	Pointer to the VM structure

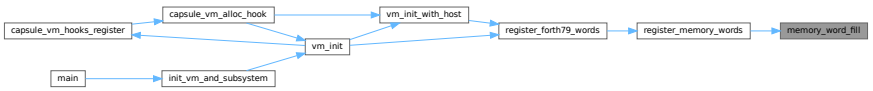
References VM::dsp, VM::error, memset(), VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_ptr().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.403.1.9 memory_word_minus_store()

```
void memory_word_minus_store (
            VM * vm )
```

Implements FORTH word '-!' - Subtract from contents of cell.

Stack effect: (n addr –) Subtracts n from the contents of the cell at the given address.

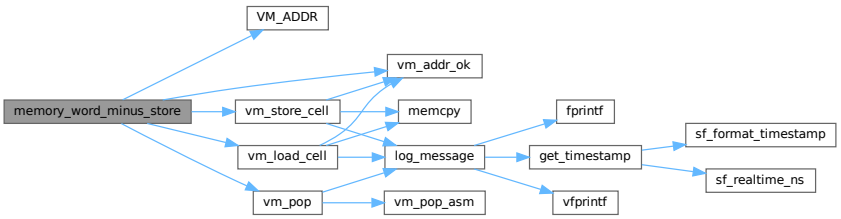
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_load_cell(), vm_pop(), and vm_store_cell().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.10 memory_word_move()

```
void memory_word_move (
            VM * vm )
```

Implements FORTH word 'MOVE' - Copy memory region.

Stack effect: (addr1 addr2 len –) Copies len bytes from addr1 to addr2, handling overlapping regions correctly.

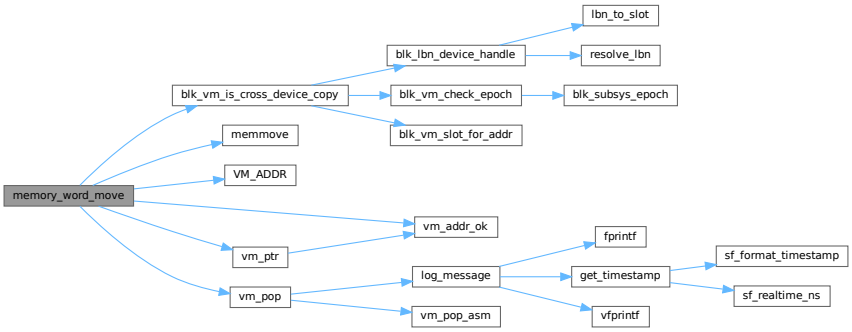
Parameters

vm	Pointer to the VM structure

References blk_vm_is_cross_device_copy(), VM::dsp, VM::error, memmove(), VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_ptr().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.11 memory_word_plus_store()

```
void memory_word_plus_store (
            VM * vm )
```

Implements FORTH word '+!' - Add to contents of cell.

Stack effect: (n addr –) Adds n to the contents of the cell at the given address.

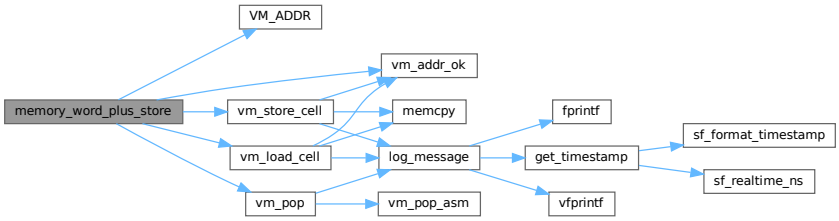
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_load_cell(), vm_pop(), and vm_store_cell().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.12 memory_word_store()

```
void memory_word_store (
            VM * vm )
```

Implements FORTH word '!' - Store cell into VM memory.

Stack effect: (n addr –) Stores a cell value at the given memory address in VM memory.

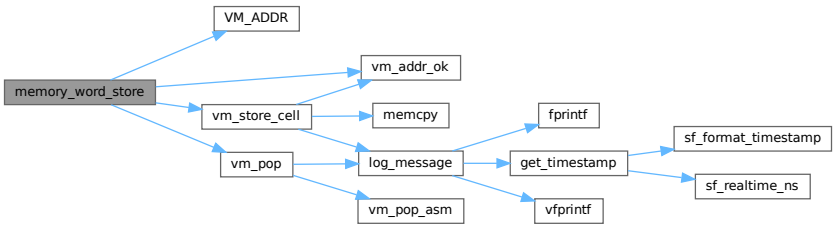
Parameters

vm	Pointer to the VM structure

References VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_pop(), and vm_store_cell().

Referenced by register_memory_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.403.1.13 register_memory_words()

```
void register_memory_words (
            VM * vm )
```

Register all memory manipulation words with the VM.

Registers all standard memory words (@, !, C@, C!, etc.) with the VM's word dictionary.

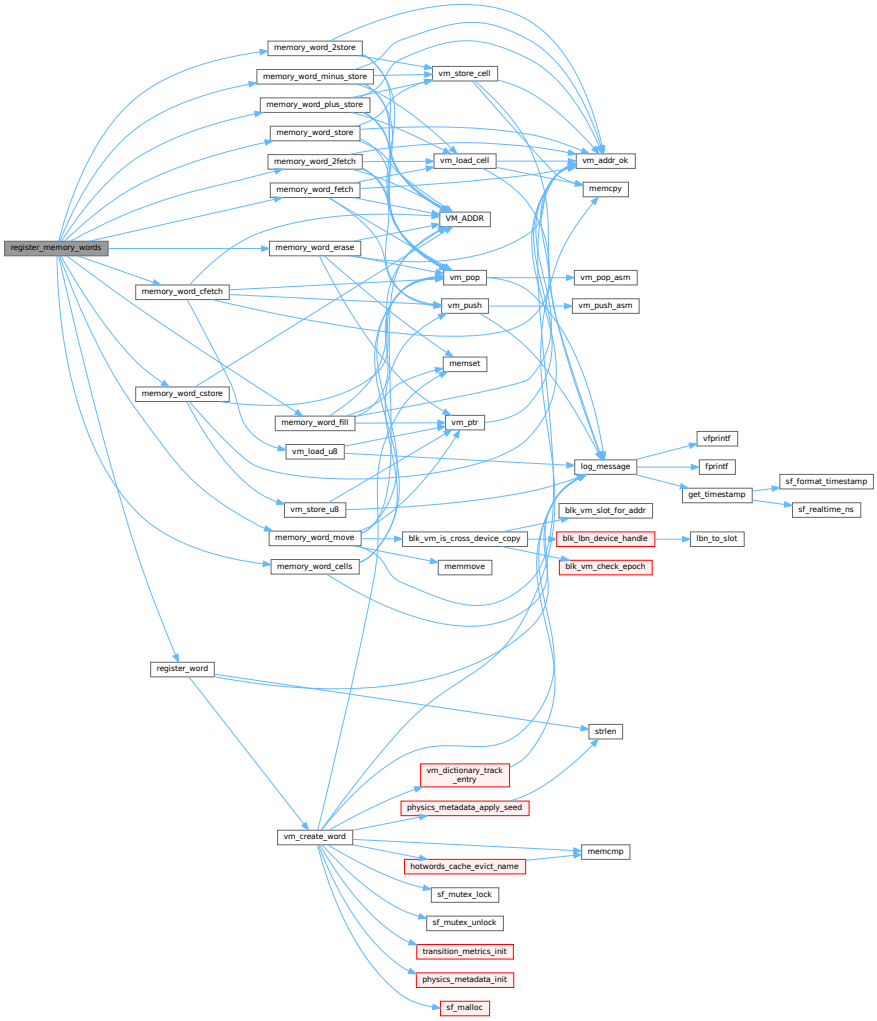
Parameters

vm	Pointer to the VM structure

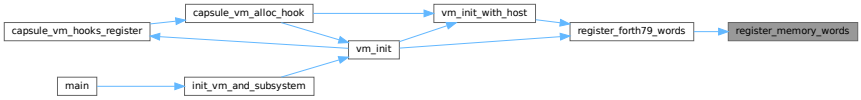
References memory_word_2fetch(), memory_word_2store(), memory_word_cells(), memory_word_cfetch(), memory_word_cstore(), memory_word_erase(), memory_word_fetch(), memory_word_fill(), memory_word_minus_store(), memory_word_move(), memory_word_plus_store(), memory_word_store(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

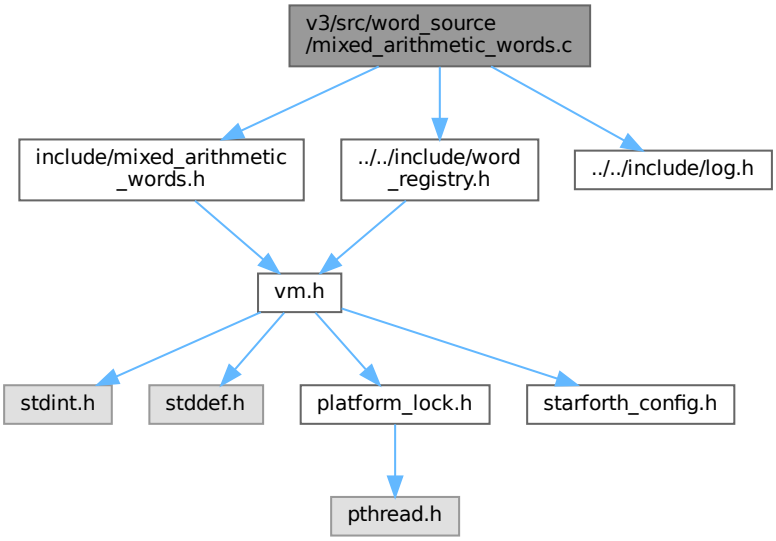


7.404 v3/src/word_source/mixed_arithmetic_words.c File Reference

```
#include "include/mixed_arithmetic_words.h"
#include "../../include/word_registry.h"
```

```
#include "../../include/log.h"
```

Include dependency graph for mixed_arithmetic_words.c:



Functions

- void mixed_math_word_m_plus (VM *vm)

  M+ (d n – d)
- void mixed_math_word_m_minus (VM *vm)

  M- (d n – d)
- void mixed_math_word_m_star (VM *vm)

  M (n1 n2 – d)*
- void mixed_math_word_m_slash_mod (VM *vm)

  M/MOD (d n – rem quot)
- void mixed_math_word_mod (VM *vm)

  MOD (n1 n2 – r)
- void mixed_math_word_slash_mod (VM *vm)

  /MOD (n1 n2 – rem quot)
- void mixed_math_word_star_slash (VM *vm)

  *STAR-SLASH (FORTH word: */) — multiply then divide.*
- void mixed_math_word_star_slash_mod (VM *vm)

  *STAR-SLASH-MOD (FORTH word: */MOD) — multiply then divide, leaving remainder.*
- void register_mixed_arithmetic_words (VM *vm)

  Register all FORTH-79 mixed arithmetic words with the VM dictionary.

7.404.1 Function Documentation

7.404.1.1 mixed_math_word_m_minus()

```
void mixed_math_word_m_minus (
            VM * vm )
```

M- (d n – d)

Subtracts the single-cell signed integer `n` from the double-cell value `d`, producing a new double-cell result. Mirrors `M+` carry logic: a positive `n` that causes the low cell to wrap below zero decrements the high cell, and vice versa.

Stack effect: (d_high d_low n – d_high' d_low')

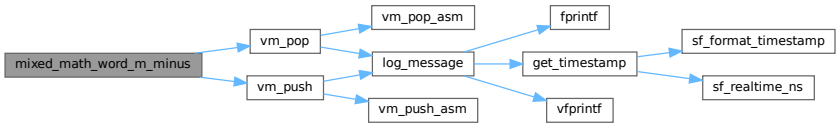
Parameters

vm	Active VM; sets `vm->error` = 1 on stack underflow

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_mixed_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.404.1.2 mixed_math_word_m_plus()

```
void mixed_math_word_m_plus (
            VM * vm )
```

M+ (d n – d)

Adds the single-cell signed integer `n` to the double-cell value `d`, producing a new double-cell result. Handles carry propagation: if the low cell addition overflows a positive `n`, the high cell is incremented; if it underflows a negative `n`, the high cell is decremented.

Stack effect: (d_high d_low n – d_high' d_low')

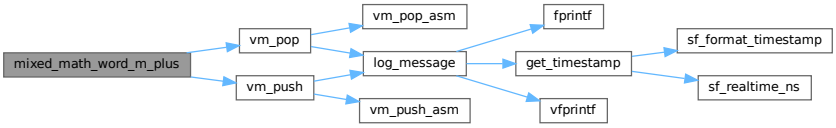
Parameters

vm	Active VM; sets `vm->error` = 1 on stack underflow

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_mixed_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.404.1.3 mixed_math_word_m_slash_mod()

```
void mixed_math_word_m_slash_mod (
            VM * vm )
```

M/MOD (d n – rem quot)

Divides the double-cell dividend `d` by the single-cell divisor `n`, pushing the remainder deeper and the quotient on TOS. A "double" here is two full `cell_t-width` cells (128 bits total on a 64-bit build) – same convention as M∗ (see that word's own doc comment for the full story). The previous implementation reconstructed the dividend as a plain 64-bit `long` long from `(dhigh << 32) | (dlow & 0xFFFFFFFF)`, the same wrong "32-bit halves of a 64-bit value" assumption M∗ had – silently discarding the upper 32 bits of `dlow` and losing `dhigh` entirely once it wasn't just 0 or -1. Latent for any dividend whose true magnitude fit in 32 bits (never exercised otherwise until M∗'s own fix started producing genuine wide doubles to feed it – FABRIC-3.md ?XII.4).

Fixed via a hand-rolled unsigned 128-bit-by-64-bit bit-serial (restoring) long division, not `__int128's` own `//%` operators: those need libgcc's `__udivti3/__umodti3`, unavailable in this freestanding, `-nostdlib` build (confirmed in `src/starkernel/arch/amd64/timer.c`'s own doc comment – `__int128` multiply/shift-by-constant/compare/subtract all compile cleanly with zero undefined symbols on all three toolchains, only division doesn't). Shift-by-1, compare, and subtract on `unsigned __int128` are all in that safe set, so the classic 128-iteration shift-and-subtract algorithm works without any libgcc dependency. Operates on magnitudes (converted via unsigned negation from 0, which is well-defined even for the extreme negative edge, unlike signed negation); signs are reapplied afterward matching C99 truncating-toward-zero division (quotient sign = XOR of operand signs, remainder sign = the dividend's own sign) – the same convention the previous, narrower implementation already used via plain C `//%`, unchanged here.

On 32-bit builds, only handles the case where the high cell is zero; sets `vm->error` = 1 otherwise.

Stack effect: (d_high d_low n – remainder quotient) TOS = quotient

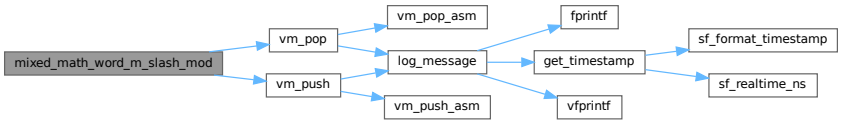
Parameters

vm	Active VM; sets `vm->error` = 1 on underflow or division by zero

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_mixed_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.404.1.4 mixed_math_word_m_star()

```
void mixed_math_word_m_star (
            VM * vm )
```

M∗ (n1 n2 – d)

Multiplies two single-cell signed integers, producing a double-cell result. A "double" here is two full `cell_t-width` cells (128 bits total on a 64-bit build), matching what D+/D-/D./etc. all expect – NOT the low/high 32-bit halves of a single 64-bit product. The previous implementation confused the two: it computed an ordinary 64-bit `long` long product (already truncated/overflowed for any operand pair whose true product doesn't fit in 64 bits) and then split THAT into 32-bit low/high halves, as if `cell_t` were 32-bit. For a small negative product like `-123 456 M∗` (-56088, which fits entirely in one 64-bit cell with sign extension), that produced a positive, zero-extended `dlow` paired with a correctly-looking `dhigh=-1` – a malformed double that D.'s overflow check (correctly) rejected as DOUBLE-OVERFLOW on every architecture, every time a negative product was printed (FABRIC-3.md ?XII.4 bug 1).

Fixed via `__int128` (a genuine 64x64->128-bit multiply, no truncation) – an established, safe pattern in this codebase for exactly this operation: see src/starkernel/crypto/fe25519.c and scalar25519.c, and src/starkernel/arch/amd64/timer.c's own doc comment confirming __int128 multiply/shift-by-constant compile cleanly with zero undefined symbols on all three target toolchains (only *division* needs libgcc support unavailable in this freestanding build – not used here).

Stack effect: (n1 n2 – d_low d_high) TOS = high

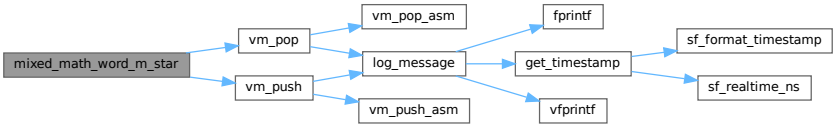
Parameters

vm	Active VM; sets `vm->error` = 1 on stack underflow

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_mixed_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.404.1.5 mixed_math_word_mod()

```
void mixed_math_word_mod (
            VM * vm )
```

MOD (n1 n2 – r)

Computes `n1` % `n2`, leaving the remainder on the stack. Sets `vm->error` = 1 if `n2` is zero (division by zero guard).

Stack effect: (n1 n2 – remainder)

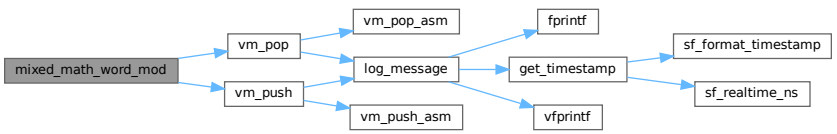
Parameters

vm	Active VM; sets `vm->error` = 1 on underflow or zero divisor

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_mixed_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.404.1.6 mixed_math_word_slash_mod()

```
void mixed_math_word_slash_mod (
            VM * vm )
```

/MOD (n1 n2 – rem quot)

Divides `n1` by `n2`, pushing the remainder deeper and the quotient on TOS. Sets `vm->error` = 1 if `n2` is zero.

Stack effect: (n1 n2 – remainder quotient) TOS = quotient

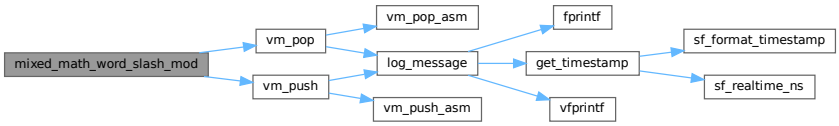
Parameters

vm	Active VM; sets `vm->error` = 1 on underflow or zero divisor

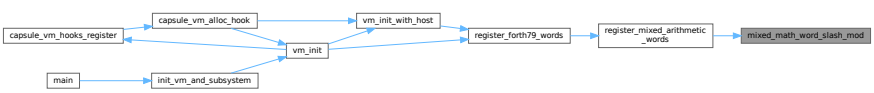
References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_mixed_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.404.1.7 mixed_math_word_star_slash()

```
void mixed_math_word_star_slash (
            VM * vm )
```

STAR-SLASH (FORTH word: `*-/`) — multiply then divide.

(n1 n2 n3 – n4)

Computes (n1 * n2) / n3 using a `long` long intermediate on 64-bit builds to avoid intermediate overflow. On 32-bit builds falls back to `long` double. Sets `vm->error` = 1 if `n3` is zero.

Stack effect: (n1 n2 n3 – quotient)

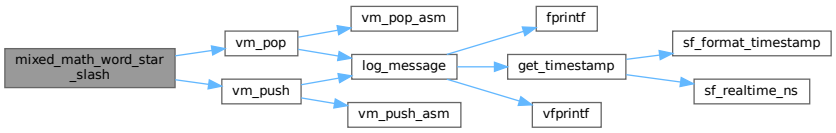
Parameters

vm	Active VM; sets `vm->error` = 1 on underflow or zero divisor

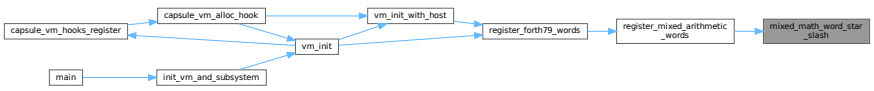
References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_mixed_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.404.1.8 mixed_math_word_star_slash_mod()

```
void mixed_math_word_star_slash_mod (
            VM * vm )
```

STAR-SLASH-MOD (FORTH word: `*-/MOD`) — multiply then divide, leaving remainder.

(n1 n2 n3 – rem quot)

Computes (n1 * n2) / n3 and leaves both the remainder (deeper) and quotient (TOS). Uses `long` long on 64-bit builds; falls back to `long` double with integer remainder reconstruction on 32-bit builds. Sets `vm->error` = 1 if `n3` is zero.

Stack effect: (n1 n2 n3 – remainder quotient) TOS = quotient

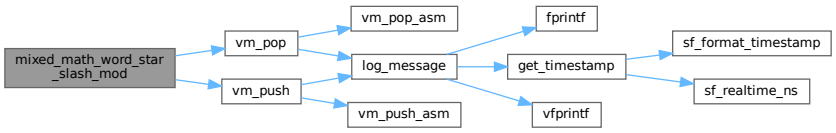
Parameters

vm	Active VM; sets `vm->error` = 1 on underflow or zero divisor

References VM::dsp, VM::error, vm_pop(), and vm_push().

Referenced by register_mixed_arithmetic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.404.1.9 register_mixed_arithmetic_words()

```
void register_mixed_arithmetic_words (
            VM * vm )
```

Register all FORTH-79 mixed arithmetic words with the VM dictionary.

Registers: `M+`, `M-`, `M*`, `M/MOD`, `MOD`, `/MOD`, `*\/`, `*\/MOD`. Called during VM bootstrap.

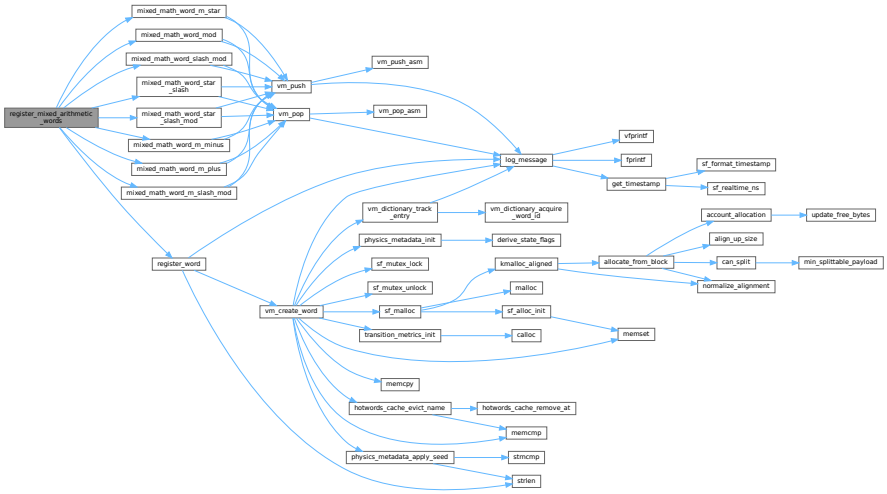
Parameters

vm	Active VM to register words into

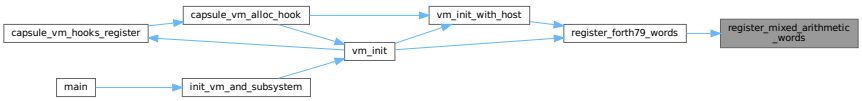
References mixed_math_word_m_minus(), mixed_math_word_m_plus(), mixed_math_word_m_slash_mod(), mixed_math_word_m_star(), mixed_math_word_mod(), mixed_math_word_slash_mod(), mixed_math_word_star_slash(), mixed_math_word_star_slash_mod(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



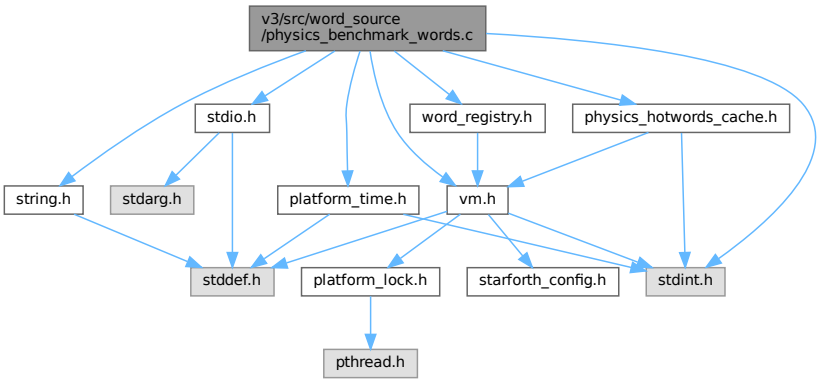
Here is the caller graph for this function:



7.405 v3/src/word_source/physics_benchmark_words.c File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "vm.h"
#include "physics_hotwords_cache.h"
#include "platform_time.h"
#include "word_registry.h"
```

Include dependency graph for physics_benchmark_words.c:



Functions

- void forth_BENCH_DICT_LOOKUP (VM *vm)

  BENCH-DICT-LOOKUP (iterations –)
- void forth_PHYSICS_CACHE_STATS (VM *vm)

  PHYSICS-CACHE-STATS (–)
- void forth_PHYSICS_TOGGLE_CACHE (VM *vm)

  PHYSICS-TOGGLE-CACHE (–)
- void forth_PHYSICS_RESET_STATS (VM *vm)

  PHYSICS-RESET-STATS (–)
- void forth_PHYSICS_BUILD_INFO (VM *vm)

  PHYSICS-BUILD-INFO (–)
- void forth_PHYSICS_BAYESIAN_REPORT (VM *vm)

  PHYSICS-BAYESIAN-REPORT (addr_baseline –)
- void register_physics_benchmark_words (VM *vm)

  Register all physics benchmark words with the VM.

Variables

- DictEntry ** sf_fc_list [256]
- size_t sf_fc_count [256]

7.405.1 Function Documentation

7.405.1.1 forth_BENCH_DICT_LOOKUP()

```
void forth_BENCH_DICT_LOOKUP (
            VM * vm )
```

BENCH-DICT-LOOKUP (iterations –)

Benchmark dictionary lookup performance with statistically valid sample sizes.

STATISTICAL VALIDITY:

- Minimum recommended: 10,000 iterations (95% confidence interval)
- Standard test: 100,000 iterations (tight confidence intervals)
- Stress test: 1,000,000 iterations (ultra-precise measurements)

FOR BAYESIAN INFERENCE:

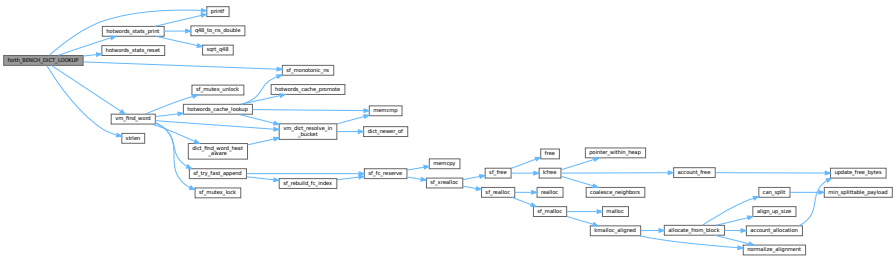
- All latencies recorded in 64-bit Q48.16 fixed-point format
- Variance accumulators enable prior/posterior calculations
- Sample sizes affect confidence in ML decisions

USAGE: 100000 BENCH-DICT-LOOKUP (statistically valid baseline) 1000000 BENCH-DICT-LOOKUP (stress test, ∼30sec on modern CPUs)

References VM::data_stack, VM::dsp, ENABLE_HOTWORDS_CACHE, VM::error, VM::hotwords_cache, hotwords_stats_print(), hotwords_stats_reset(), printf(), sf_monotonic_ns(), HotwordsCache_s::stats, strlen(), and vm_find_word().

Referenced by register_physics_benchmark_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.405.1.2 forth_PHYSICS_BAYESIAN_REPORT()

```
void forth_PHYSICS_BAYESIAN_REPORT (
            VM * vm )
```

PHYSICS-BAYESIAN-REPORT (addr_baseline –)

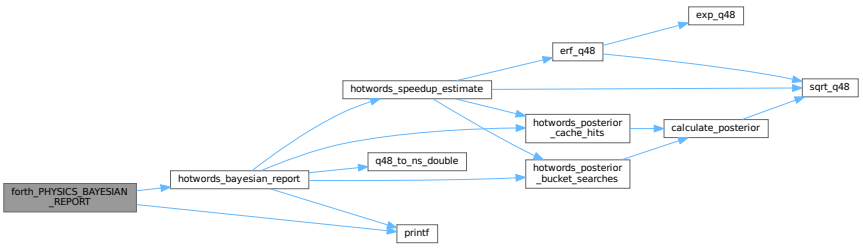
Generate Bayesian inference report comparing current cache stats against baseline stats stored at given address

Usage: PHYSICS-RESET-STATS [run baseline benchmark without cache] [save stats somewhere] PHYSICS-↩ RESET-STATS [run benchmark with cache] PHYSICS-BAYESIAN-REPORT

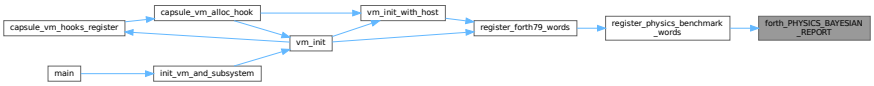
References HotwordsStats::bucket_hits, HotwordsStats::bucket_search_samples, HotwordsStats::bucket_search_total_ns_q48, HotwordsStats::bucket_search_variance_sum_q48, HotwordsStats::cache_hits, VM::error, hotwords_bayesian_report(), VM::hotwords_cache, HotwordsStats::max_bucket_search_ns, HotwordsStats::min_bucket_search_ns, printf(), and HotwordsCache_s::stats.

Referenced by register_physics_benchmark_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.405.1.3 forth_PHYSICS_BUILD_INFO()

```
void forth_PHYSICS_BUILD_INFO (
            VM * vm )
```

PHYSICS-BUILD-INFO (–)

Show current build configuration for this variant

References HotwordsCache_s::cache_count, ENABLE_HOTWORDS_CACHE, HotwordsCache_s::enabled, VM::error, VM::hotwords_cache, HOTWORDS_CACHE_SIZE, HOTWORDS_EXECUTION_HEAT_DELTA_THRESHOLD, HOTWORDS_EXECUTION_HEAT_THRESHOLD, and printf().

Referenced by register_physics_benchmark_words().

Here is the call graph for this function:

Here is the caller graph for this function:

7.405.1.4 forth_PHYSICS_CACHE_STATS()

```
void forth_PHYSICS_CACHE_STATS (
            VM * vm )
```

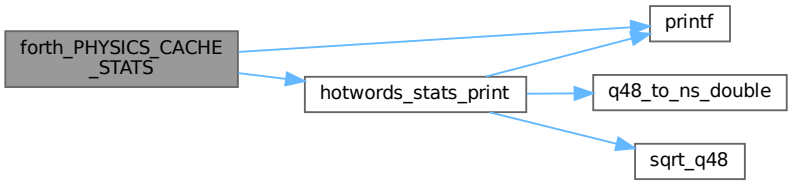
PHYSICS-CACHE-STATS (–)

Display detailed hot-words cache statistics

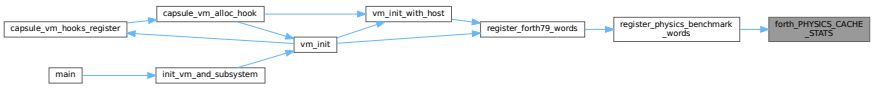
References HotwordsCache_s::cache, HotwordsCache_s::cache_count, ENABLE_HOTWORDS_CACHE, VM::error, DictEntry::execution_heat, VM::hotwords_cache, hotwords_stats_print(), DictEntry::name, DictEntry::physics, printf(), HotwordsCache_s::stats, and DictPhysics::temperature_q8.

Referenced by register_physics_benchmark_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.405.1.5 forth_PHYSICS_RESET_STATS()

```
void forth_PHYSICS_RESET_STATS (
            VM * vm )
```

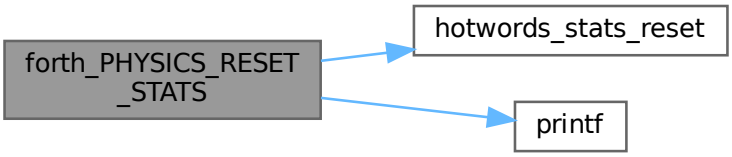
PHYSICS-RESET-STATS (–)

Reset cache statistics for clean before/after comparison

References VM::error, VM::hotwords_cache, hotwords_stats_reset(), VM::pipeline_metrics, PipelineGlobalMetrics::prefetch_attempts, PipelineGlobalMetrics::prefetch_hits, printf(), HotwordsCache_s::stats, and PipelineGlobalMetrics::window_tuning_checks.

Referenced by register_physics_benchmark_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.405.1.6 forth_PHYSICS_TOGGLE_CACHE()

```
void forth_PHYSICS_TOGGLE_CACHE (
            VM * vm )
```

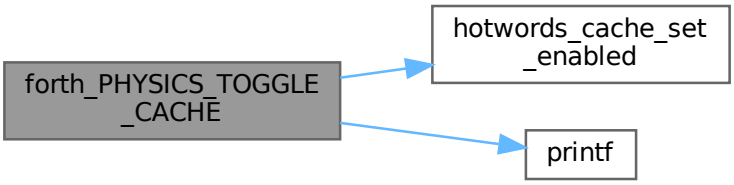
PHYSICS-TOGGLE-CACHE (–)

Enable/disable hot-words cache at runtime (for A/B testing)

References ENABLE_HOTWORDS_CACHE, HotwordsCache_s::enabled, VM::error, VM::hotwords_cache, hotwords_cache_set_enabled(), and printf().

Referenced by register_physics_benchmark_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.405.1.7 register_physics_benchmark_words()

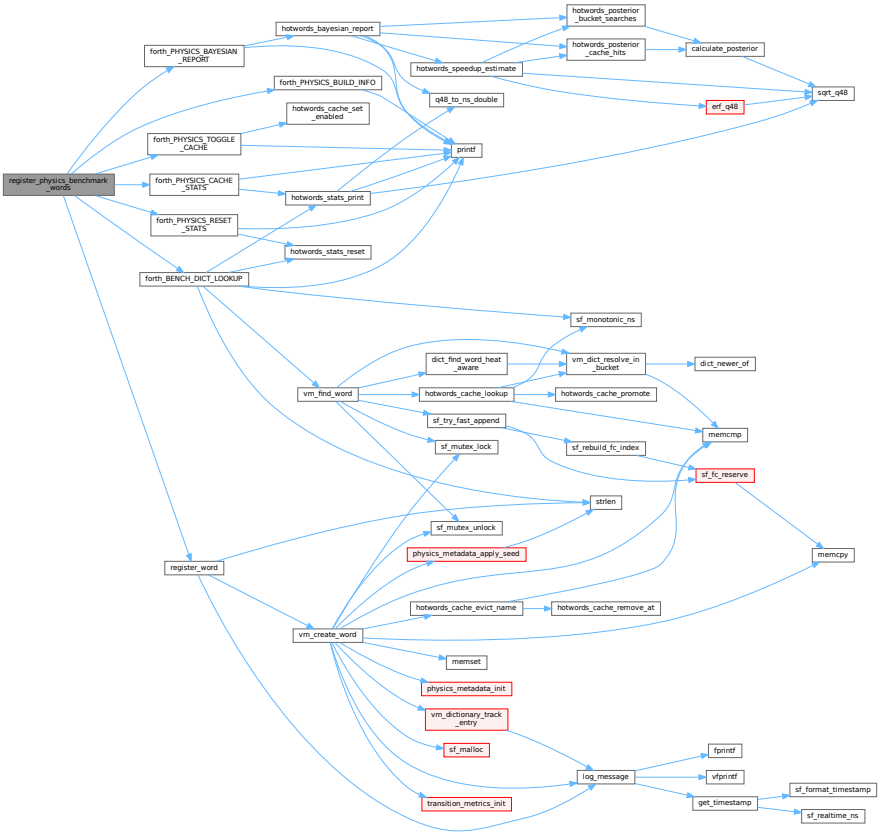
```
void register_physics_benchmark_words (
            VM * vm )
```

Register all physics benchmark words with the VM.

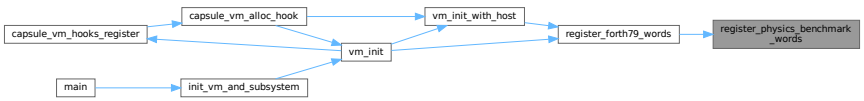
References forth_BENCH_DICT_LOOKUP(), forth_PHYSICS_BAYESIAN_REPORT(), forth_PHYSICS_BUILD_INFO(), forth_PHYSICS_CACHE_STATS(), forth_PHYSICS_RESET_STATS(), forth_PHYSICS_TOGGLE_CACHE(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.405.2 Variable Documentation

7.405.2.1 sf_fc_count

```
size_t sf_fc_count[256]  [extern]
```

Referenced by sf_fc_clear_all(), sf_fc_reserve(), sf_rebuild_fc_index(), sf_try_fast_append(), and vm_find_word().

7.405.2.2 sf_fc_list

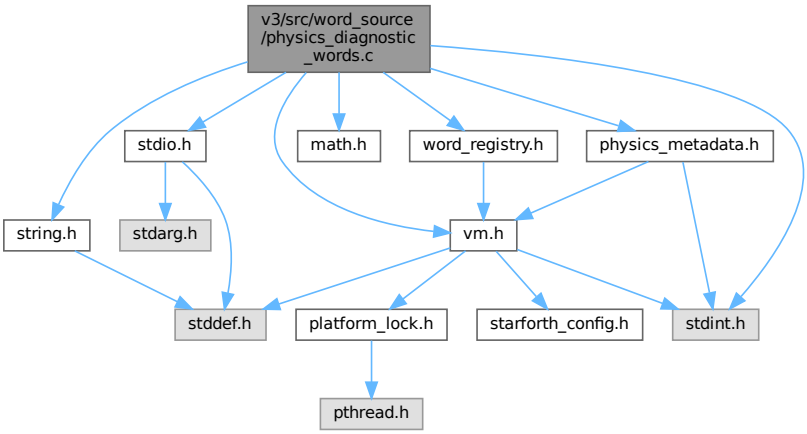
```
DictEntry** sf_fc_list[256]  [extern]
```

Referenced by sf_fc_clear_all(), sf_fc_reserve(), sf_rebuild_fc_index(), sf_try_fast_append(), and vm_find_word().

7.406 v3/src/word_source/physics_diagnostic_words.c File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <math.h>
#include "vm.h"
#include "physics_metadata.h"
#include "word_registry.h"
```

Include dependency graph for physics_diagnostic_words.c:



Functions

- void forth_PHYSICS_WORD_METRICS (VM *vm)

  PHYSICS-WORD-METRICS (–)
- void forth_PHYSICS_CALC_KNOBS (VM *vm)

  PHYSICS-CALC-KNOBS (–)
- void forth_PHYSICS_BURN (VM *vm)

  PHYSICS-BURN (n –)
- void forth_PHYSICS_SHOW_FEEDBACK (VM *vm)

  PHYSICS-SHOW-FEEDBACK (–)
- void register_physics_diagnostic_words (VM *vm)

  Register all physics diagnostic words with the VM.

7.406.1 Function Documentation

7.406.1.1 forth_PHYSICS_BURN()

```
void forth_PHYSICS_BURN (
            VM * vm )
```

PHYSICS-BURN (n –)

Execute a word repeatedly (burn) and show thermal feedback.

Execute the top-of-stack word n times and show thermal feedback.

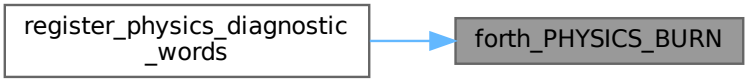
Usage: : HOTTEST 100 0 DO 1 2 + 3 * LOOP ; HOTTEST 10 PHYSICS-BURN (execute 10 times) PHYSICS-↩ SHOW-FEEDBACK

References VM::data_stack, VM::dsp, VM::error, DictEntry::execution_heat, DictEntry::func, DictPhysics::last_active_ns, VM::latest, DictEntry::name, DictEntry::physics, printf(), and DictPhysics::temperature_q8.

Referenced by register_physics_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



7.406.1.2 forth_PHYSICS_CALC_KNOBS()

```
void forth_PHYSICS_CALC_KNOBS (
            VM * vm )
```

PHYSICS-CALC-KNOBS (–)

Calculate and show knob adjustments based on current thermal metrics.

Calculate what knobs should be adjusted based on current word metrics. Shows:

- Thermal pressure (0.0 to 1.0)
- Recommended priority adjustment
- Recommended sampling rate
- Stack depth limit

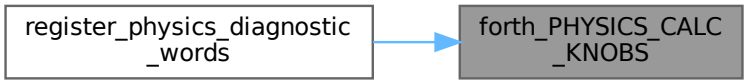
- Core affinity preference

References VM::error, DictPhysics::last_active_ns, VM::latest, DictEntry::name, DictEntry::physics, printf(), and DictPhysics::temperature_q8.

Referenced by register_physics_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



7.406.1.3 forth_PHYSICS_SHOW_FEEDBACK()

```
void forth_PHYSICS_SHOW_FEEDBACK (
            VM * vm )
```

PHYSICS-SHOW-FEEDBACK (–)

Display the complete feedback loop: inputs → math → knobs → effect.

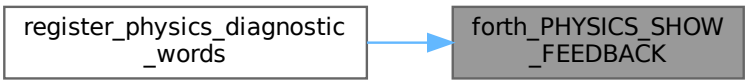
Display the complete feedback loop for the most recent word. Shows: Metrics → Calculations → Knobs → Effect

References DictPhysics::avg_latency_ns, VM::error, DictEntry::execution_heat, DictPhysics::last_active_ns, VM::latest, DictPhysics::mass_bytes, DictEntry::name, DictEntry::physics, printf(), and DictPhysics::temperature_q8.

Referenced by register_physics_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



7.406.1.4 forth_PHYSICS_WORD_METRICS()

```
void forth_PHYSICS_WORD_METRICS (
            VM * vm )
```

PHYSICS-WORD-METRICS (–)

Display detailed physics metrics for the most recently executed word.

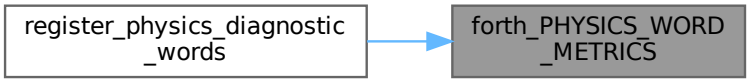
Display detailed physics metrics for a word. Looks up the most recently executed word and shows its state.

References DictPhysics::avg_latency_ns, VM::error, DictEntry::execution_heat, DictPhysics::last_active_ns, VM::latest, DictPhysics::mass_bytes, DictEntry::name, DictEntry::physics, printf(), and DictPhysics::temperature_q8.

Referenced by register_physics_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



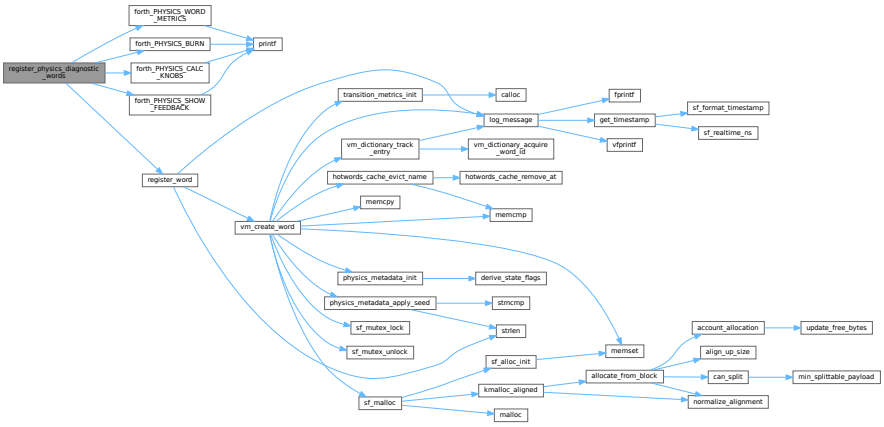
7.406.1.5 register_physics_diagnostic_words()

```
void register_physics_diagnostic_words (
            VM * vm )
```

Register all physics diagnostic words with the VM.

References forth_PHYSICS_BURN(), forth_PHYSICS_CALC_KNOBS(), forth_PHYSICS_SHOW_FEEDBACK(), forth_PHYSICS_WORD_METRICS(), and register_word().

Here is the call graph for this function:

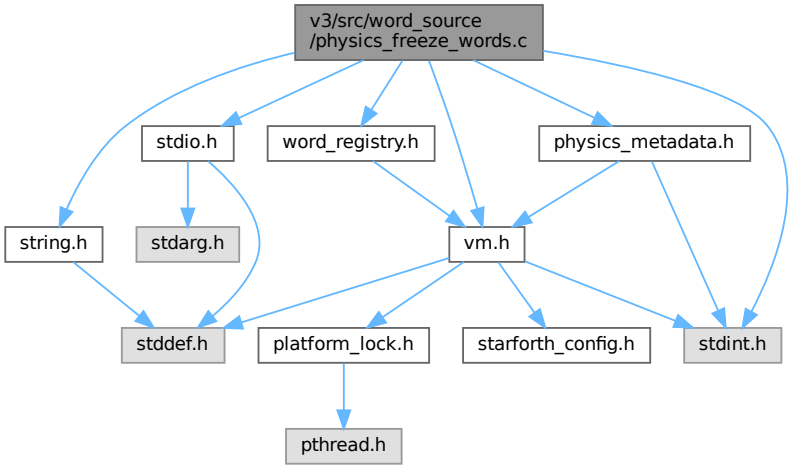


7.407 v3/src/word_source/physics_freeze_words.c File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "vm.h"
#include "physics_metadata.h"
```

```
#include "word_registry.h"
```

Include dependency graph for physics_freeze_words.c:



Functions

- void forth_FREEZE_WORD (VM *vm)

  FREEZE-WORD (caddr u –)
- void forth_UNFREEZE_WORD (VM *vm)

  UNFREEZE-WORD (caddr u –)
- void forth_FROZEN_QUERY (VM *vm)

  FROZEN? (caddr u – flag)
- void forth_HEAT_STORE (VM *vm)

  HEAT! (heat caddr u –)
- void forth_HEAT_FETCH (VM *vm)

  HEAT@ (caddr u – heat)
- void forth_SHOW_HEAT (VM *vm)

  SHOW-HEAT (caddr u –)
- void forth_ALL_HEATS (VM *vm)

  ALL-HEATS (–)
- void forth_DECAY_RATE_FETCH (VM *vm)

  DECAY-RATE@ (– rate)
- void forth_FREEZE_CRITICAL (VM *vm)

  FREEZE-CRITICAL (–)
- void register_physics_freeze_words (VM *vm)

  Register all Phase 2 freeze/decay control words with the VM dictionary.

7.407.1 Function Documentation

7.407.1.1 forth_ALL_HEATS()

```
void forth_ALL_HEATS (
            VM * vm )
```

ALL-HEATS (–)

Iterates the entire dictionary, collects up to 1024 entries, sorts them by descending `execution_heat` using an $O(n^2)$ bubble sort, then prints a formatted table to `stdout` with columns for word name, heat value, and status (frozen/pinned). Intended for interactive diagnostics; the $O(n^2)$ sort is acceptable for dictionary sizes below 1000 words.

Stack effect: (–)

Parameters

vm	Active VM; no-op if `vm->error` is set

References count, VM::error, execution_heat, DictEntry::execution_heat, DictEntry::flags, VM::latest, DictEntry::name, DictEntry::name_len, printf(), WORD_FROZEN, and WORD_PINNED.

Referenced by register_physics_freeze_words().

Here is the call graph for this function:

Here is the caller graph for this function:

7.407.1.2 forth_DECAY_RATE_FETCH()

```
void forth_DECAY_RATE_FETCH (
            VM * vm )
```

DECAY-RATE@ (– rate)

Pushes the compile-time constant `DECAY_RATE_PER_US_Q16` as a Q48.16 fixed-point value. This is the base heat decay rate per microsecond used by Loop #3 before adaptive tuning by Loop #6. Intended for introspection and diagnostics.

Stack effect: (– rate)

Parameters

vm	Active VM; sets `vm->error` = 1 on stack overflow

References VM::data_stack, DECAY_RATE_PER_US_Q16, VM::dsp, VM::error, and STACK_SIZE.

Referenced by register_physics_freeze_words().

Here is the caller graph for this function:

7.407.1.3 forth_FREEZE_CRITICAL()

```
void forth_FREEZE_CRITICAL (
            VM * vm )
```

FREEZE-CRITICAL (–)

Sets `WORD_FROZEN` on a hard-coded list of 21 system-critical words (DUP, DROP, SWAP, OVER, ROT, @, !, C@, C!, EXECUTE, IF, THEN, ELSE, DO, LOOP, BEGIN, UNTIL, REPEAT, ., EMIT, CR). Silently skips any word not found in the current dictionary. Intended to be called once at startup to prevent essential words from decaying out of the hot-words cache.

Stack effect: (–)

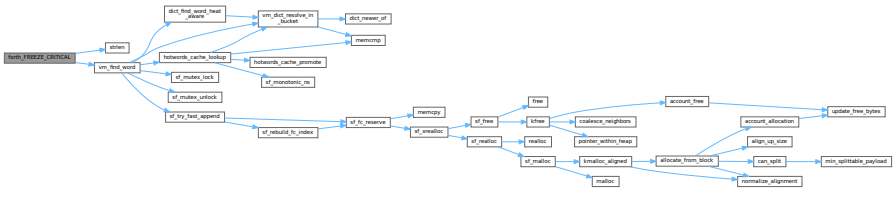
Parameters

vm	Active VM; no-op if `vm->error` is set

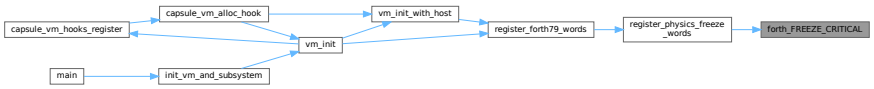
References entry, VM::error, name, strlen(), vm_find_word(), and WORD_FROZEN.

Referenced by register_physics_freeze_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.407.1.4 forth_FREEZE_WORD()

```
void forth_FREEZE_WORD (
            VM * vm )
```

FREEZE-WORD (caddr u –)

Set the `WORD_FROZEN` flag on the named word, preventing `physics_metadata_apply_linear_decay()` from reducing its `execution_heat`. Frozen words stay hot across OS context switches indefinitely. Silently succeeds if the word is not found (lenient policy).

Stack effect: (caddr u –)

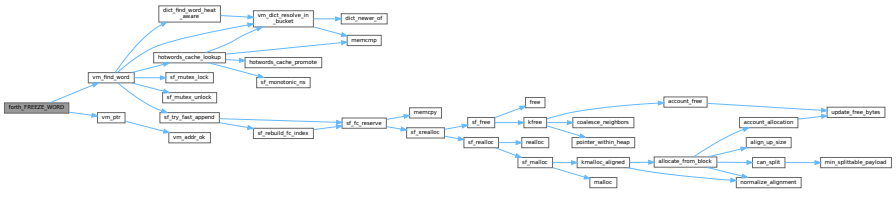
Parameters

vm	Active VM; `vm->error` is set and returns early on underflow

References VM::data_stack, VM::dsp, entry, VM::error, name, vm_find_word(), vm_ptr(), WORD_FROZEN, and WORD_NAME_MAX.

Referenced by register_physics_freeze_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.407.1.5 forth_FROZEN_QUERY()

```
void forth_FROZEN_QUERY (
            VM * vm )
```

FROZEN? (caddr u – flag)

Tests whether the WORD_FROZEN flag is set on the named word. Pushes -1 (FORTH true) if frozen, 0 otherwise. Also pushes 0 when the word is not found or u is out of range, so callers need not distinguish "not frozen" from "not found".

Stack effect: (caddr u – flag)

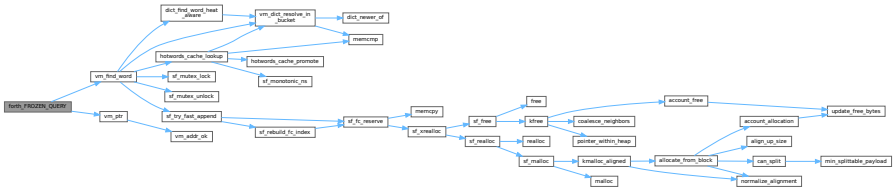
Parameters

vm	Active VM; vm->error is set and returns early on underflow

References VM::data_stack, VM::dsp, entry, VM::error, name, vm_find_word(), vm_ptr(), WORD_FROZEN, and WORD_NAME_MAX.

Referenced by register_physics_freeze_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.407.1.6 forth_HEAT_FETCH()

```
void forth_HEAT_FETCH (
            VM * vm )
```

HEAT@ (caddr u – heat)

Pushes the current execution_heat of the named word. Pushes 0 when the word is not found, u is out of range, or the word genuinely has zero heat.

Stack effect: (caddr u – heat)

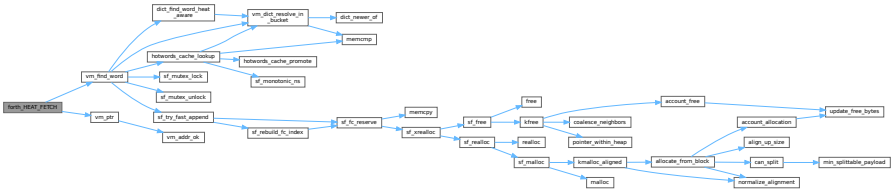
Parameters

vm	Active VM; `vm->error` is set and returns early on underflow

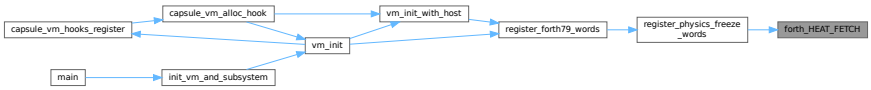
References VM::data_stack, VM::dsp, entry, VM::error, name, vm_find_word(), vm_ptr(), and WORD_NAME_MAX.

Referenced by register_physics_freeze_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.407.1.7 forth_HEAT_STORE()

```
void forth_HEAT_STORE (
            VM * vm )
```

HEAT! (heat caddr u –)

Directly writes `heat` into `entry->execution_heat` for the named word. Intended for diagnostics and testing; bypasses Loop #1 accumulation and Loop #3 decay. Silently succeeds if the word is not found or `u` is out of range.

Stack effect: (heat caddr u –)

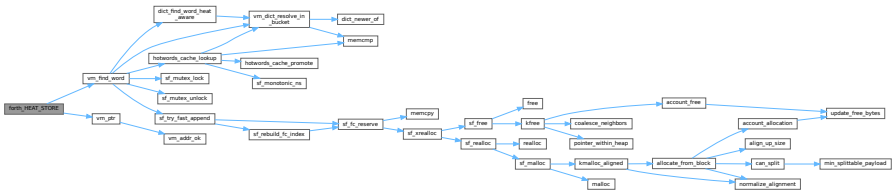
Parameters

vm	Active VM; `vm->error` is set and returns early on underflow

References VM::data_stack, VM::dsp, entry, VM::error, name, vm_find_word(), vm_ptr(), and WORD_NAME_MAX.

Referenced by register_physics_freeze_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.407.1.8 forth_SHOW_HEAT()

```
void forth_SHOW_HEAT (
                VM * vm )
```

SHOW-HEAT (caddr u –)

Prints the `execution_heat` of the named word to `stdout` along with its freeze and pinned status flags. Output format: `"NAME: HEAT (frozen) (pinned)"`. Prints `"Word not found: NAME"` when the dictionary lookup fails. Intended for interactive diagnostics.

Stack effect: (caddr u –)

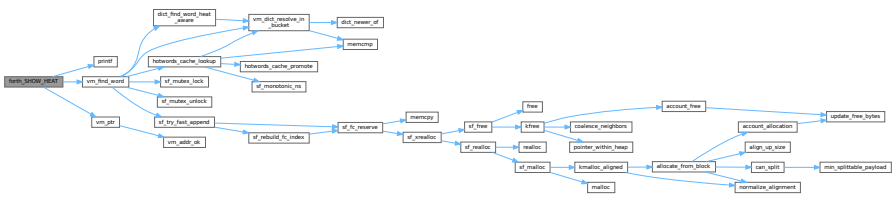
Parameters

vm	Active VM; `vm->error` is set and returns early on underflow

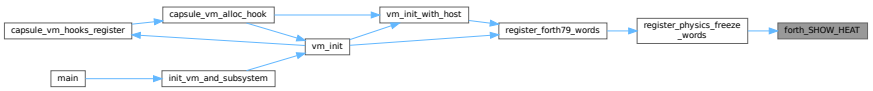
References VM::data_stack, VM::dsp, entry, VM::error, name, printf(), vm_find_word(), vm_ptr(), WORD_FROZEN, WORD_NAME_MAX, and WORD_PINNED.

Referenced by register_physics_freeze_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.407.1.9 forth_UNFREEZE_WORD()

```
void forth_UNFREEZE_WORD (
            VM * vm )
```

UNFREEZE-WORD (caddr u –)

Clears the `WORD_FROZEN` flag on the named word, allowing its `execution_heat` to decay normally via Loop #3. Does not alter `WORD_PINNED`. Silently succeeds if the word is not found or `u` is out of range.

Stack effect: (caddr u –)

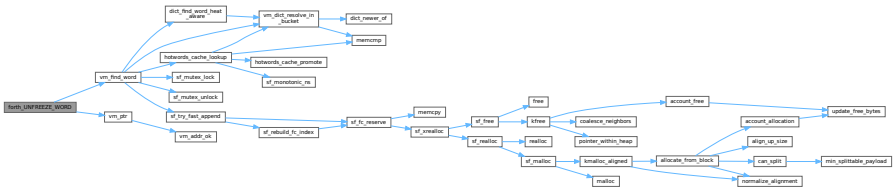
Parameters

vm	Active VM; `vm->error` is set and returns early on underflow

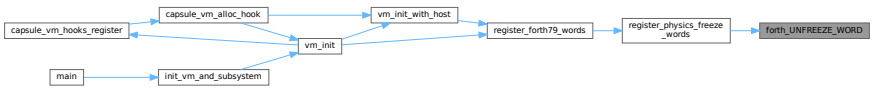
References VM::data_stack, VM::dsp, entry, VM::error, name, vm_find_word(), vm_ptr(), and WORD_NAME_MAX.

Referenced by register_physics_freeze_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.407.1.10 register_physics_freeze_words()

```
void register_physics_freeze_words (
            VM * vm )
```

Register all Phase 2 freeze/decay control words with the VM dictionary.

Register all Phase 2 freeze/decay control words with the VM.

Registers: `FREEZE-WORD`, `UNFREEZE-WORD`, `FROZEN?`, `HEAT!`, `HEAT@`, `SHOW-HEAT`, `ALL-HEATS`, `DECAY-RATE@`, `FREEZE-CRITICAL`. Called during VM bootstrap by the word registration subsystem.

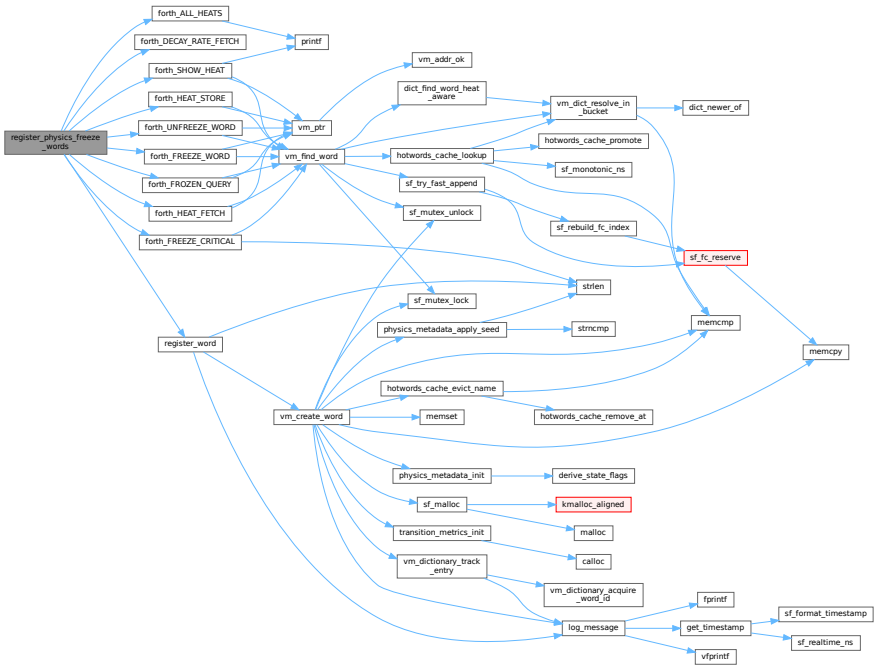
Parameters

vm	Active VM to register words into

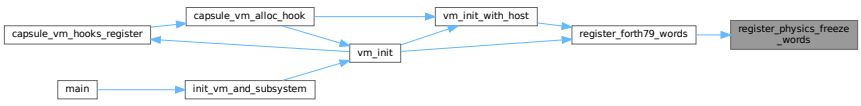
References forth_ALL_HEATS(), forth_DECAY_RATE_FETCH(), forth_FREEZE_CRITICAL(), forth_FREEZE_WORD(), forth_FROZEN_QUERY(), forth_HEAT_FETCH(), forth_HEAT_STORE(), forth_SHOW_HEAT(), forth_UNFREEZE_WORD(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



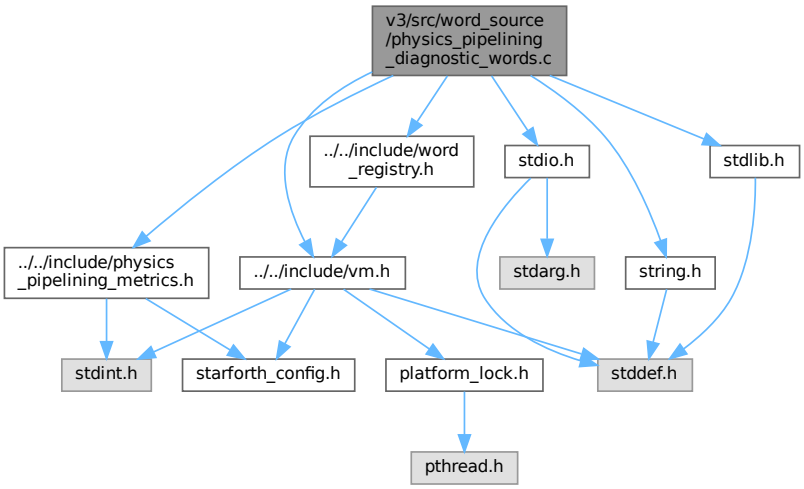
Here is the caller graph for this function:



7.408 v3/src/word_source/physics_pipelining_diagnostic_words.c File Reference

```
#include "../../include/vm.h"
#include "../../include/physics_pipelining_metrics.h"
#include "../../include/word_registry.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for physics_pipelining_diagnostic_words.c:



Functions

- static void forth_PIPELINING_SHOW_STATS (VM *vm)

  PIPELINING-SHOW-STATS (<wordname> –)
- static void forth_PIPELINING_SHOW_TOP_TRANSITIONS (VM *vm)

  PIPELINING-SHOW-TOP-TRANSITIONS (<wordname> N –)
- static void forth_PIPELINING_RESET_ALL (VM *vm)

  PIPELINING-RESET-ALL (–)
- static void forth_PIPELINING_ENABLE (VM *vm)

  PIPELINING-ENABLE (–)
- static void forth_PIPELINING_STATS (VM *vm)

  PIPELINING-STATS (–)
- static void forth_PIPELINING_ANALYZE_WORD (VM *vm)

  PIPELINING-ANALYZE-WORD (<wordname> –)
- void register_physics_pipelining_diagnostic_words (VM *vm)

  Register all pipelining diagnostic words.

7.408.1 Function Documentation

7.408.1.1 forth_PIPELINING_ANALYZE_WORD()

```
static void forth_PIPELINING_ANALYZE_WORD (
            VM * vm )  [static]
```

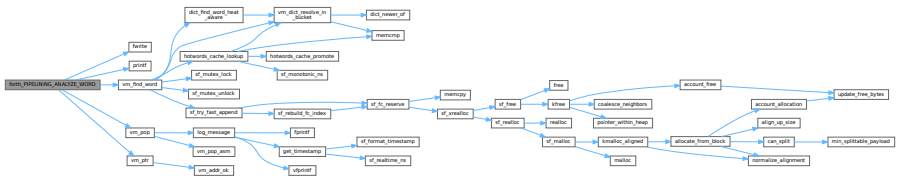
PIPELINING-ANALYZE-WORD (<wordname> –)

Comprehensive analysis of a single word's transition pattern. Shows raw data and interpretation hints.

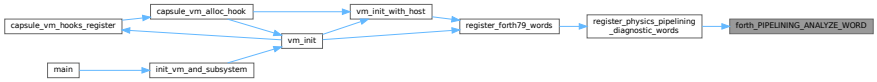
References DICTIONARY_SIZE, VM::dsp, entry, VM::error, fwrite(), WordTransitionMetrics::max_transition_probability_q48, WordTransitionMetrics::misprediction_cost_q48, WordTransitionMetrics::most_likely_next_word_id, WordTransitionMetrics::prefetch_a WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_latency_saved_q48, printf(), stdout, WordTransitionMetrics::total_transitions, WordTransitionMetrics::transition_heat, vm_find_word(), vm_pop(), and vm_ptr().

Referenced by register_physics_pipelining_diagnostic_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.408.1.2 forth_PIPELINING_ENABLE()

```
static void forth_PIPELINING_ENABLE (
            VM * vm )  [static]
```

PIPELINING-ENABLE (–)

Enable pipelining metrics collection (sets ENABLE_PIPELINING flag). Note: This is compile-time on/off for now; this word is a placeholder.

References printf().

Referenced by register_physics_pipelining_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:

7.408.1.3 forth_PIPELINING_RESET_ALL()

```
static void forth_PIPELINING_RESET_ALL (
            VM * vm )  [static]
```

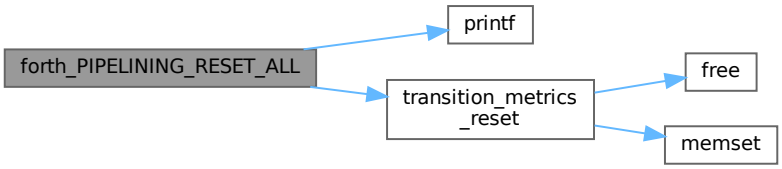
PIPELINING-RESET-ALL (–)

Clear all transition metrics from the dictionary. Useful for starting fresh measurements.

References count, entry, VM::latest, printf(), and transition_metrics_reset().

Referenced by register_physics_pipelining_diagnostic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.408.1.4 forth_PIPELINING_SHOW_STATS()

```
static void forth_PIPELINING_SHOW_STATS (
            VM * vm )  [static]
```

PIPELINING-SHOW-STATS (<wordname> –)

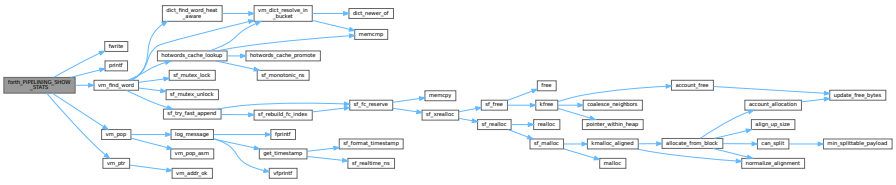
Display transition metrics for a given word. Typical usage: PIPELINING-SHOW-STATS EXIT (show where EXIT typically goes) PIPELINING-SHOW-STATS BENCH-DICT (show where BENCH-DICT transitions)

Requires: word must exist in dictionary Effect: Prints transition statistics to stdout

References VM::dsp, entry, VM::error, fwrite(), WordTransitionMetrics::max_transition_probability_q48, WordTransitionMetrics::mispre WordTransitionMetrics::most_likely_next_word_id, WordTransitionMetrics::prefetch_attempts, WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_latency_saved_q48, WordTransitionMetrics::prefetch_misses, printf(), stdout, WordTransitionMetrics::total_transitions, vm_find_word(), vm_pop(), and vm_ptr().

Referenced by register_physics_pipelining_diagnostic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.408.1.5 forth_PIPELINING_SHOW_TOP_TRANSITIONS()

```
static void forth_PIPELINING_SHOW_TOP_TRANSITIONS (
            VM * vm )  [static]
```

PIPELINING-SHOW-TOP-TRANSITIONS (<wordname> N –)

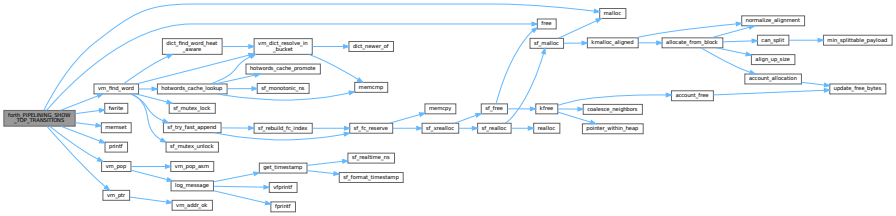
Display the top N words that typically follow a given word. Useful for understanding hot execution paths.

Stack: <wordname> N – Example: " EXIT" 5 PIPELINING-SHOW-TOP-TRANSITIONS (Show top 5 words that usually follow EXIT)

References count, DICTIONARY_SIZE, VM::dsp, entry, VM::error, free(), fwrite(), malloc(), memset(), printf(), stdout, WordTransitionMetrics::total_transitions, WordTransitionMetrics::transition_heat, vm_find_word(), vm_pop(), vm_ptr(), and word_id.

Referenced by register_physics_pipelining_diagnostic_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.408.1.6 forth_PIPELINING_STATS()

```
static void forth_PIPELINING_STATS (
            VM * vm ) [static]
```

PIPELINING-STATS (–)

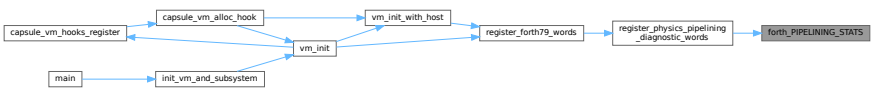
Display aggregate pipelining statistics across entire dictionary.

References entry, VM::latest, WordTransitionMetrics::prefetch_attempts, WordTransitionMetrics::prefetch_hits, WordTransitionMetrics::prefetch_misses, printf(), WordTransitionMetrics::total_transitions, and word_count.

Referenced by register_physics_pipelining_diagnostic_words().

Here is the call graph for this function:

Here is the caller graph for this function:



7.408.1.7 register_physics_pipelining_diagnostic_words()

```
void register_physics_pipelining_diagnostic_words (
            VM * vm )
```

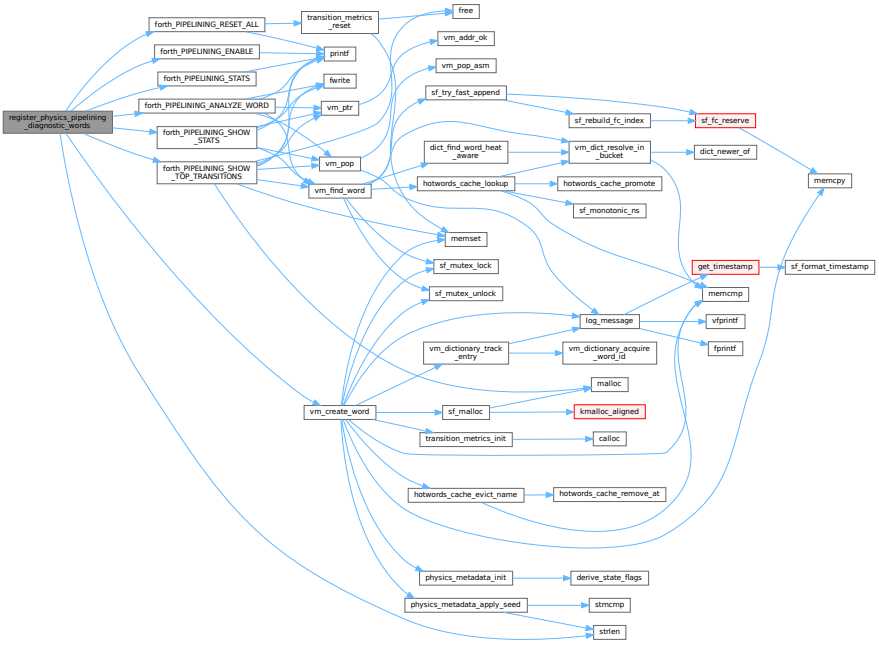
Register all pipelining diagnostic words.

Register all pipelining diagnostic words Called during VM initialization.

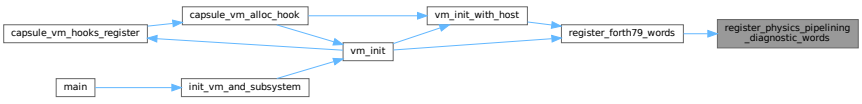
References forth_PIPELINING_ANALYZE_WORD(), forth_PIPELINING_ENABLE(), forth_PIPELINING_RESET_ALL(), forth_PIPELINING_SHOW_STATS(), forth_PIPELINING_SHOW_TOP_TRANSITIONS(), forth_PIPELINING_STATS(), strlen(), and vm_create_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



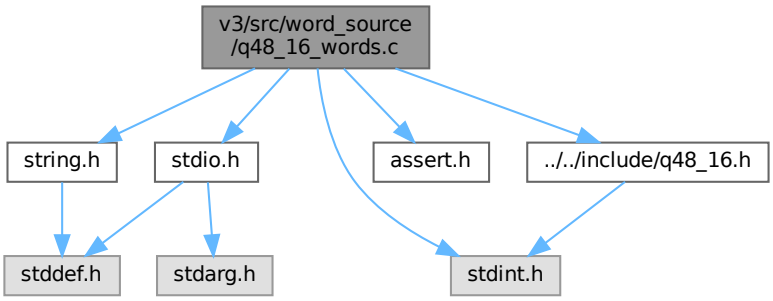
Here is the caller graph for this function:



7.409 v3/src/word_source/q48_16_words.c File Reference

```
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include <assert.h>
#include "../../include/q48_16.h"
```

Include dependency graph for q48_16_words.c:



Functions

- q48_16_t q48_mul (q48_16_t a, q48_16_t b)

  Multiply two Q48.16 fixed-point values.
- q48_16_t q48_div (q48_16_t a, q48_16_t b)

  Divide two Q48.16 fixed-point values.
- q48_16_t q48_from_double (double d)

  Convert a `double` to Q48.16 format.
- double q48_to_double (q48_16_t q)

  Convert a Q48.16 value to `double`.
- q48_16_t q48_log_approx (uint64_t x)

  Compute an approximation of `ln(x)` in Q48.16 format (integer-only).
- q48_16_t q48_exp_approx (q48_16_t q)

  Compute an approximation of `e^q` in Q48.16 format (Taylor series).
- q48_16_t q48_sqrt_approx (q48_16_t q)

  Compute an approximation of `sqrt(q)` in Q48.16 format (Newton-Raphson).
- static int64_t q48_reduce_angle (int64_t x)

  Reduce a signed Q48.16 angle into [-PI_Q48, PI_Q48].
- q48_16_t q48_sin_approx (q48_16_t q)

  Compute an approximation of `sin(q)` in Q48.16 format (radians).
- q48_16_t q48_cos_approx (q48_16_t q)

  Compute an approximation of `cos(q)` in Q48.16 format (radians).
- const char * q48_to_string (q48_16_t q)

  Format a Q48.16 value as a decimal string `"INTEGER.FRAC"`.
- int q48_is_valid (q48_16_t q)

  Test whether a Q48.16 value is valid.

7.409.1 Function Documentation

7.409.1.1 q48_cos_approx()

```
q48_16_t q48_cos_approx (
            q48_16_t q )
```

Compute an approximation of `cos(q)` in Q48.16 format (radians).

Range-reduces into [-pi, pi] via `q48_reduce_angle()`, then sums the Taylor series `1` - x^2/2! + x^4/4! - ... for terms n = 2,4,6,8,10, with an early exit when a term falls below 10. Cosine is even (`cos`(-x) = cos(x)), so no sign correction is needed – `x^2` already absorbs the sign of `x`.

Parameters

q	Angle in Q48.16 format (radians, any magnitude)

Returns

`cos(q)` in Q48.16 format

7.409.1.2 q48_div()

```
q48_16_t q48_div (
            q48_16_t a,
            q48_16_t b )
```

Divide two Q48.16 fixed-point values.

Computes (a << 16) / b. Returns 0 if `b` == 0 (safe, but callers should guard). Returns `UINT64_MAX` as a saturation sentinel when `a` exceeds 2^48, which would overflow the 16-bit left-shift.

Parameters

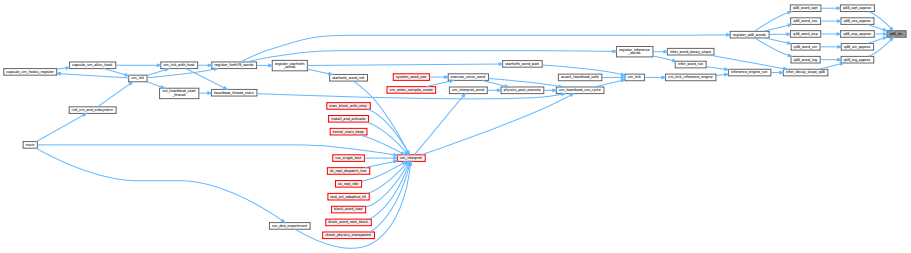
a	Dividend in Q48.16 format
b	Divisor in Q48.16 format; 0 returns 0

Returns

Quotient in Q48.16 format, or UINT64_MAX on saturation

Referenced by q48_cos_approx(), q48_exp_approx(), q48_log_approx(), q48_sin_approx(), and q48_sqrt_approx().

Here is the caller graph for this function:



7.409.1.3 q48_exp_approx()

```
q48_16_t q48_exp_approx (
            q48_16_t q )
```

Compute an approximation of e^q in Q48.16 format (Taylor series).

Supporting function for q48_log_approx() Newton iterations. Uses the Taylor series 1 + x + x^2/2! + ... up to 10 terms, with an early-exit when a term falls below 50 (sub-LSB in Q48.16). Handles negative q by computing e^|q| then returning 1/result. Returns 1.0 (65536) for q == 0, UINT64_MAX for q >= 16.0, and 0 for q <= -16.0.

Parameters

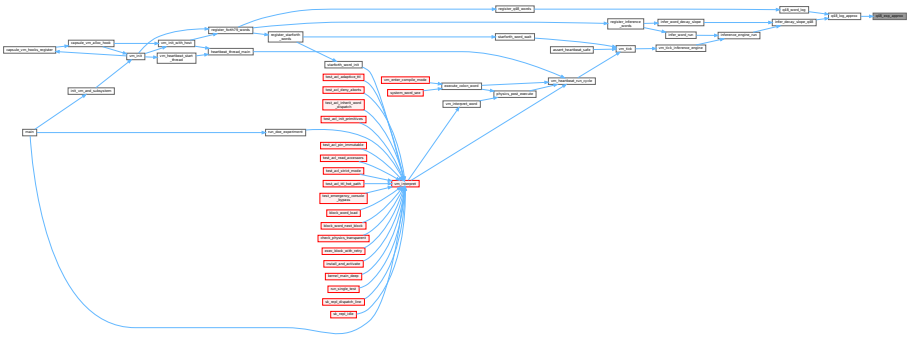
q	Exponent in Q48.16 format (signed interpretation)

Returns

e^q in Q48.16 format

Referenced by q48_log_approx().

Here is the caller graph for this function:



7.409.1.4 q48_from_double()

```
q48_16_t q48_from_double (
            double d )
```

Convert a `double` to Q48.16 format.

Multiplies `d` by 65536.0 and rounds to nearest. Intended for testing and diagnostics only — not called in the inference engine hot path to avoid floating-point dependency.

Parameters

d	Value to convert

Returns

Nearest Q48.16 representation of `d`

7.409.1.5 q48_is_valid()

```
int q48_is_valid (
            q48_16_t q )
```

Test whether a Q48.16 value is valid.

Currently accepts any `uint64_t` value as valid — all bit patterns are representable in Q48.16. A more sophisticated implementation could check for saturation sentinels or range limits. Returns 1 always.

Parameters

q	Value to validate

Returns

1 always (placeholder for future range validation)

7.409.1.6 q48_log_approx()

```
q48_16_t q48_log_approx (
            uint64_t x )
```

Compute an approximation of `ln(x)` in Q48.16 format (integer-only).

Used by `infer_decay_slope_q48()` (Loop #6) for log-linear OLS regression. Algorithm: range-reduces `x` to the interval [1, 2) by tracking the exponent `k` (so `x` = 2^k * m), applies 6 Newton-Raphson iterations on `e^y` = m starting from the first-order approximation `y_0` = m - 1, then assembles the result as `ln(x)` = k*ln(2) + ln(m). Returns 0 for `x` == 0 (ln undefined) and for `x` == 65536 (ln(1.0) == 0.0 exactly). `LN2_Q48` = 45426 (0.693147... × 65536).

Parameters

x	Input value in Q48.16 format (uint64_t; value of 65536 = 1.0)

Returns

`ln(x)` in Q48.16 format, or 0 for x==0 or x==65536

7.409.1.7 q48_mul()

```
q48_16_t q48_mul (
            q48_16_t a,
            q48_16_t b )
```

Multiply two Q48.16 fixed-point values.

Computes (a * b) >> 16 to maintain the Q48.16 scale. Uses `__uint128_t` on GCC/Clang (128-bit intermediate, no precision loss) and falls back to a manual 64-bit four-partial-product path with carry tracking on platforms that lack `__SIZEOF_INT128__`.

Parameters

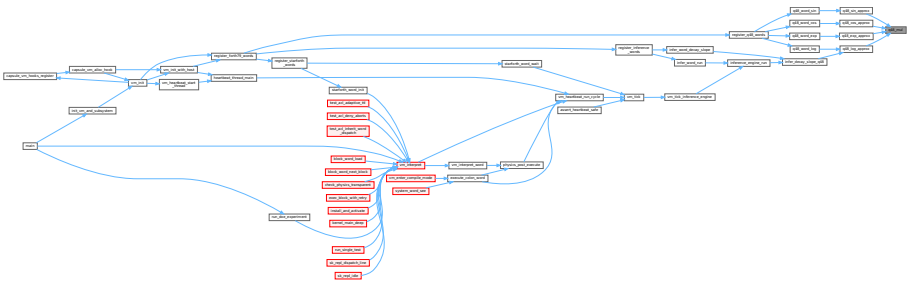
a	First operand in Q48.16 format
b	Second operand in Q48.16 format

Returns

Product in Q48.16 format (truncated, not rounded)

Referenced by q48_cos_approx(), q48_exp_approx(), q48_log_approx(), and q48_sin_approx().

Here is the caller graph for this function:



7.409.1.8 q48_reduce_angle()

```
static int64_t q48_reduce_angle (
            int64_t x )  [static]
```

Reduce a signed Q48.16 angle into [-PI_Q48, PI_Q48].

A single integer division on the raw Q48.16 representations gives the correct (scale-independent) quotient of how many full 2∗pi turns to remove, then a bounded fix-up loop (at most one or two iterations) handles the remainder landing just outside the target interval.

Parameters

x	Signed angle in Q48.16 format (any magnitude)

Returns

Equivalent angle in Q48.16 format, within [-PI_Q48, PI_Q48]

Referenced by q48_cos_approx(), and q48_sin_approx().

Here is the caller graph for this function:

7.409.1.9 q48_sin_approx()

```
q48_16_t q48_sin_approx (
            q48_16_t q )
```

Compute an approximation of `sin(q)` in Q48.16 format (radians).

Range-reduces into [-pi, pi] via `q48_reduce_angle()`, then sums the Taylor series `x` - x^3/3! + x^5/5! - ... for terms n = 3,5,7,9,11, with an early exit when a term falls below 10 (sub-LSB in Q48.16). Sine is odd (`sin`(-x) = -sin(x)), so the sign is factored out before the series and reapplied to the result.

Parameters

q	Angle in Q48.16 format (radians, any magnitude)

Returns

`sin(q)` in Q48.16 format

7.409.1.10 q48_sqrt_approx()

```
q48_16_t q48_sqrt_approx (
            q48_16_t q )
```

Compute an approximation of `sqrt(q)` in Q48.16 format (Newton-Raphson).

Iterates `x_{n+1}` = (x_n + q/x_n) / 2 starting from (q >> 1) + 16384, for up to 8 iterations. Terminates early when the delta between successive estimates falls below 10 (approximately 0.00015 in Q48.16). Used for R^2 computation in the inference engine. Returns 0 for `q` == 0, and 1.0 (65536) for `q` == 65536.

Parameters

q	Input value in Q48.16 format

Returns

`sqrt(q)` in Q48.16 format

7.409.1.11 q48_to_double()

```
double q48_to_double (
            q48_16_t q )
```

Convert a Q48.16 value to `double`.

Divides `q` by 65536.0. Used for logging and RStudio dashboard output; not called in the inference engine hot path.

Parameters

q	Value in Q48.16 format

Returns

Equivalent `double` value

7.409.1.12 q48_to_string()

```
const char * q48_to_string (
            q48_16_t q )
```

Format a Q48.16 value as a decimal string `"INTEGER.FRAC"`.

Writes into a static buffer — not thread-safe and not re-entrant. The fractional part is expressed as 5 decimal digits (`frac_part` * 100000 / 65536). Intended for diagnostic logging.

Parameters

q	Value in Q48.16 format

Returns

Pointer to a static buffer containing the formatted string

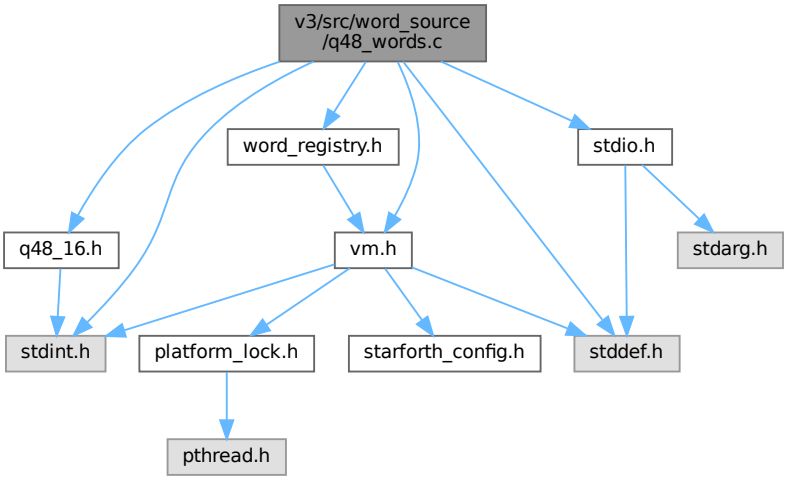
References snprintf().

Here is the call graph for this function:

7.410 v3/src/word_source/q48_words.c File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "q48_16.h"
#include "vm.h"
#include "word_registry.h"
#include <stdio.h>
```

Include dependency graph for q48_words.c:



Functions

- static q48_16_t q48_pop (VM *vm)

  q48_words.c — Module 25: Q48.16 fixed-point FORTH words
- static void q48_push (VM *vm, q48_16_t q)
- static void q48_word_add (VM *vm)
- static void q48_word_sub (VM *vm)
- static void q48_word_mul (VM *vm)
- static void q48_word_div (VM *vm)
- static void q48_word_abs (VM *vm)

- static void q48_word_neg (VM *vm)
- static void q48_word_log (VM *vm)
- static void q48_word_exp (VM *vm)
- static void q48_word_sqrt (VM *vm)
- static void q48_word_sin (VM *vm)
- static void q48_word_cos (VM *vm)
- static void q48_word_from_int (VM *vm)
- static void q48_word_to_int (VM *vm)
- static void q48_word_one (VM *vm)
- static void q48_word_zero (VM *vm)
- static void q48_word_scale (VM *vm)
- static void q48_word_eq (VM *vm)
- static void q48_word_lt (VM *vm)
- static void q48_word_gt (VM *vm)
- static void q48_word_zero_eq (VM *vm)
- static void q48_word_max (VM *vm)
- static void q48_word_min (VM *vm)
- static void q48_word_print (VM *vm)
- void register_q48_words (VM *vm)

q48_words.h — Module 25: Q48.16 fixed-point FORTH words

7.410.1 Function Documentation

7.410.1.1 q48_pop()

```
static q48_16_t q48_pop (
            VM * vm )  [inline], [static]
```

q48_words.c — Module 25: Q48.16 fixed-point FORTH words

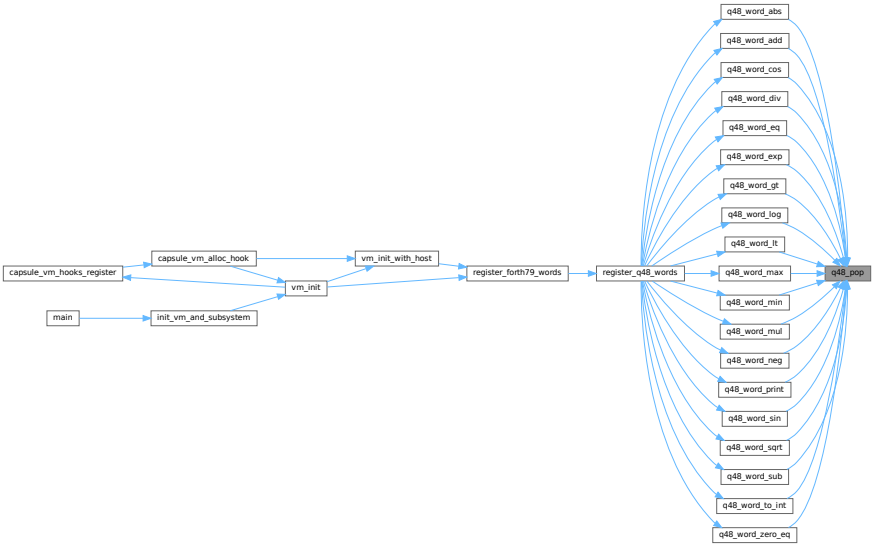
FORTH word set for Q48.16 fixed-point arithmetic. All stack values are cell_t (int64_t) reinterpreted as uint64_t Q48.16. 1.0 = 0x10000 = 65536. Fractional resolution = 1/65536 ≈ 0.0000153.

Words registered: Q.+ (q1 q2 – q_sum) Q.- (q1 q2 – q_diff) Q.* (q1 q2 – q_prod) Q./ (q1 q2 – q_quot) division by zero → q=0 Q.ABS (q – |q|) Q.NEG (q – -q) Q.LOG (q – ln_q) natural log, Newton-Raphson Q.EXP (q – e^q) Taylor series Q.SQRT (q – sqrt_q) Newton-Raphson Q.SIN (q – sin_q) Taylor series, radians, any magnitude Q.COS (q – cos_q) Taylor series, radians, any magnitude Q.FROM-INT (n – q) integer → Q48.16 (n << 16) Q.TO-INT (q – n) Q48.16 → integer (q >> 16, truncate) Q.1 (– 65536) 1.0 in Q48.16 Q.0 (– 0) 0.0 in Q48.16 Q.SCALE (– 65536) scale factor (alias of Q.1) Q.= (q1 q2 – flag) -1 if equal, 0 otherwise Q.< (q1 q2 – flag) -1 if q1 < q2 Q.> (q1 q2 – flag) -1 if q1 > q2 Q.0= (q – flag) -1 if q = 0 Q.MAX (q1 q2 – q_max) Q.MIN (q1 q2 – q_min) Q.PRINT (q –) print as "integer.frac" to console

References VM_POP.

Referenced by q48_word_abs(), q48_word_add(), q48_word_cos(), q48_word_div(), q48_word_eq(), q48_word_exp(), q48_word_gt(), q48_word_log(), q48_word_lt(), q48_word_max(), q48_word_min(), q48_word_mul(), q48_word_neg(), q48_word_print(), q48_word_sin(), q48_word_sqrt(), q48_word_sub(), q48_word_to_int(), and q48_word_zero_eq().

Here is the caller graph for this function:



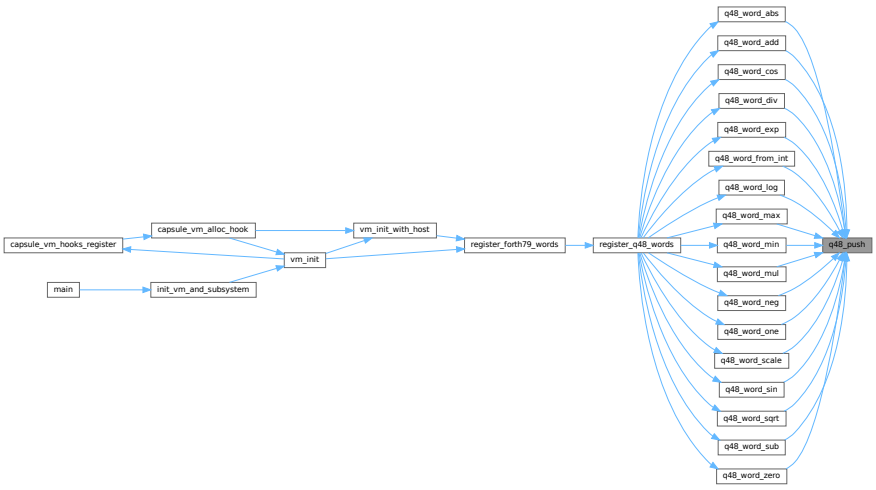
7.410.1.2 q48_push()

```
static void q48_push (
            VM * vm,
            q48_16_t q )  [inline], [static]
```

References VM_PUSH.

Referenced by q48_word_abs(), q48_word_add(), q48_word_cos(), q48_word_div(), q48_word_exp(), q48_word_from_int(), q48_word_log(), q48_word_max(), q48_word_min(), q48_word_mul(), q48_word_neg(), q48_word_one(), q48_word_scale(), q48_word_sin(), q48_word_sqrt(), q48_word_sub(), and q48_word_zero().

Here is the caller graph for this function:



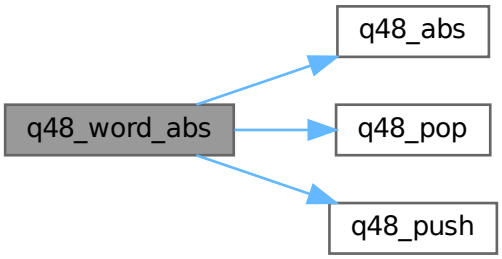
7.410.1.3 q48_word_abs()

```
static void q48_word_abs (
            VM * vm )  [static]
```

References q48_abs(), q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

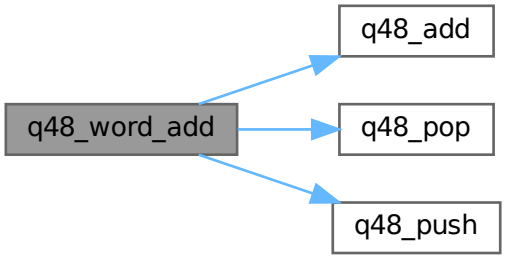
7.410.1.4 q48_word_add()

```
static void q48_word_add (
            VM * vm )  [static]
```

References q48_add(), q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

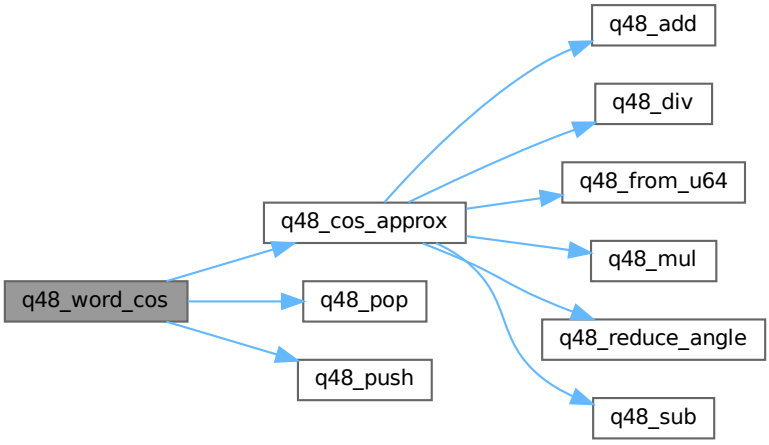
7.410.1.5 q48_word_cos()

```
static void q48_word_cos (
            VM * vm )  [static]
```

References q48_cos_approx(), q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

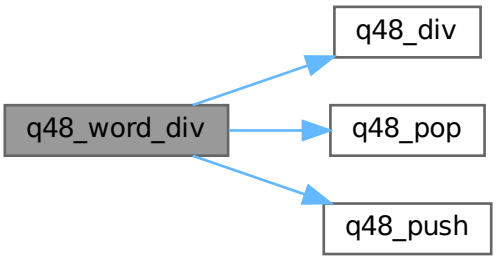
7.410.1.6 q48_word_div()

```
static void q48_word_div (
            VM * vm )  [static]
```

References q48_div(), q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.410.1.7 q48_word_eq()

```
static void q48_word_eq (
            VM * vm )  [static]
```

References q48_pop(), and VM_PUSH.

Referenced by register_q48_words().

Here is the call graph for this function:

Here is the caller graph for this function:

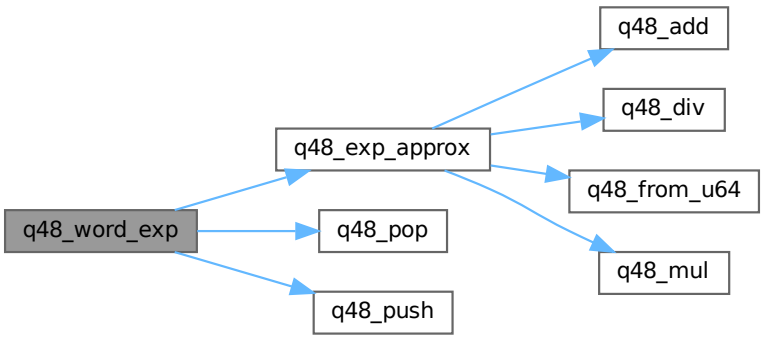
7.410.1.8 q48_word_exp()

```
static void q48_word_exp (
            VM * vm )  [static]
```

References q48_exp_approx(), q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

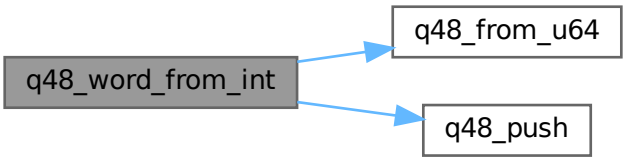
7.410.1.9 q48_word_from_int()

```
static void q48_word_from_int (
            VM * vm )  [static]
```

References q48_from_u64(), q48_push(), and VM_POP.

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.410.1.10 q48_word_gt()

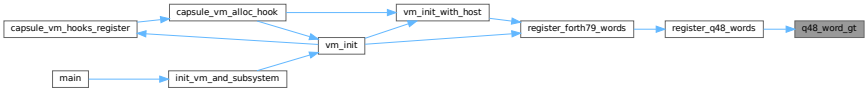
```
static void q48_word_gt (
            VM * vm )  [static]
```

References q48_pop(), and VM_PUSH.

Referenced by register_q48_words().

Here is the call graph for this function:

Here is the caller graph for this function:



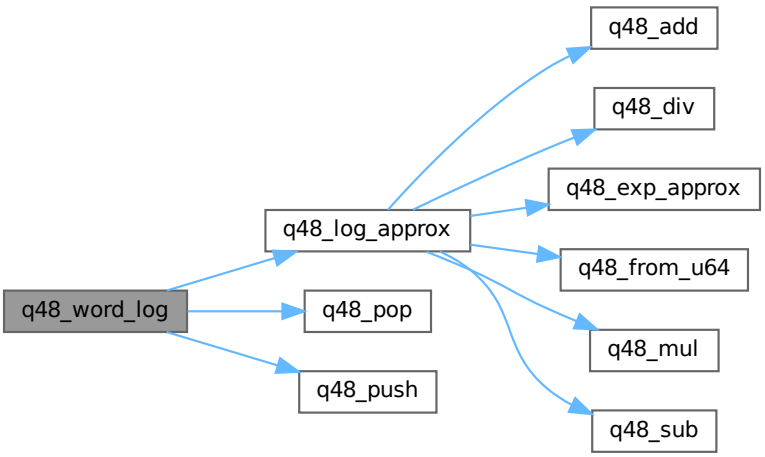
7.410.1.11 q48_word_log()

```
static void q48_word_log (
            VM * vm )  [static]
```

References q48_log_approx(), q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.410.1.12 q48_word_lt()

```
static void q48_word_lt (
            VM * vm )  [static]
```

References q48_pop(), and VM_PUSH.

Referenced by register_q48_words().

Here is the call graph for this function:

Here is the caller graph for this function:

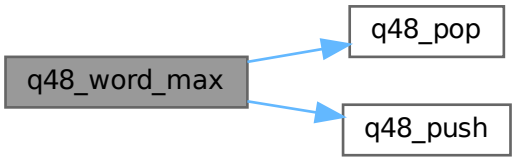
7.410.1.13 q48_word_max()

```
static void q48_word_max (
            VM * vm )  [static]
```

References q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

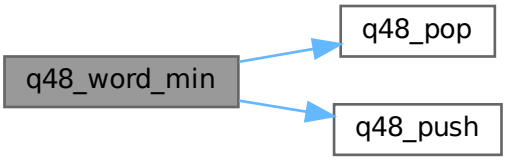
7.410.1.14 q48_word_min()

```
static void q48_word_min (
            VM * vm )  [static]
```

References q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

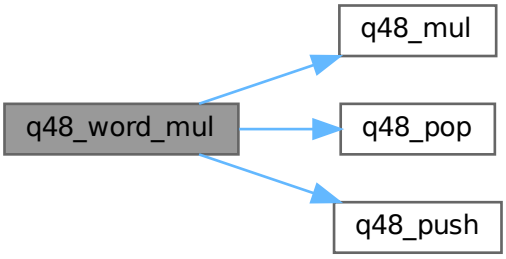
7.410.1.15 q48_word_mul()

```
static void q48_word_mul (
            VM * vm )  [static]
```

References q48_mul(), q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

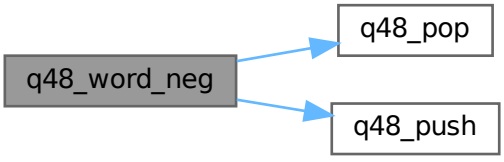
7.410.1.16 q48_word_neg()

```
static void q48_word_neg (
            VM * vm )  [static]
```

References q48_pop(), and q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.410.1.17 q48_word_one()

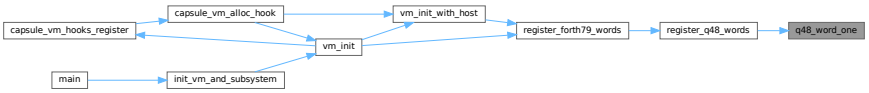
```
static void q48_word_one (
            VM * vm )  [static]
```

References q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:

Here is the caller graph for this function:



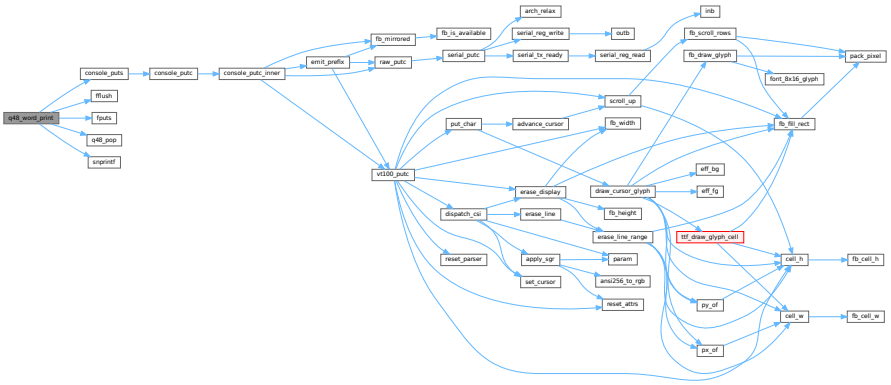
7.410.1.18 q48_word_print()

```
static void q48_word_print (
            VM * vm )  [static]
```

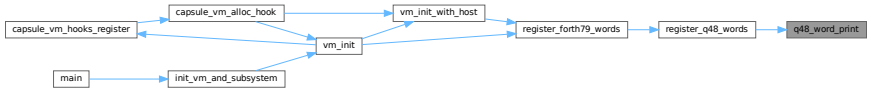
References console_puts(), fflush(), fputs(), q48_pop(), snprintf(), and stdout.

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.410.1.19 q48_word_scale()

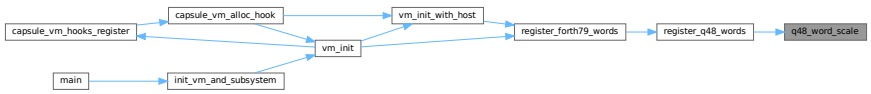
```
static void q48_word_scale (
            VM * vm )  [static]
```

References q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:

Here is the caller graph for this function:



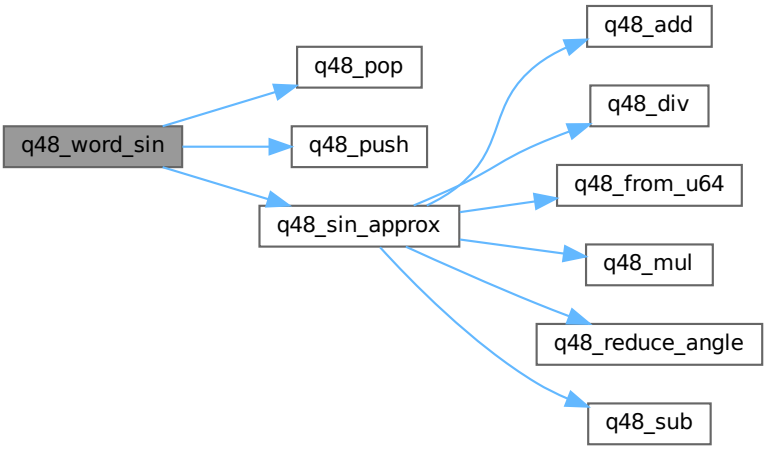
7.410.1.20 q48_word_sin()

```
static void q48_word_sin (
            VM * vm )  [static]
```

References q48_pop(), q48_push(), and q48_sin_approx().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

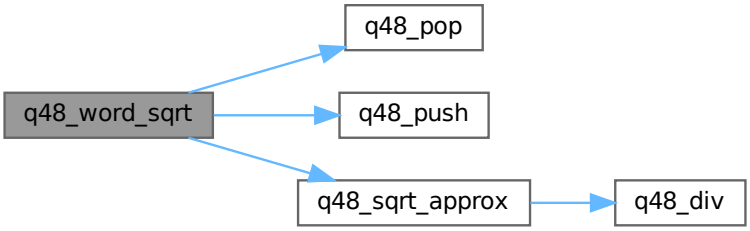
7.410.1.21 q48_word_sqrt()

```
static void q48_word_sqrt (
            VM * vm )  [static]
```

References q48_pop(), q48_push(), and q48_sqrt_approx().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

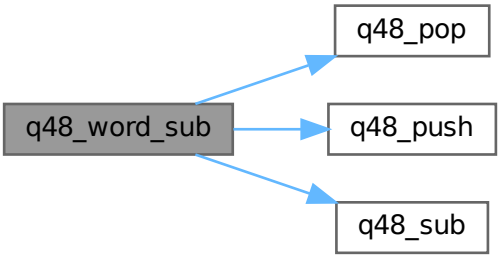
7.410.1.22 q48_word_sub()

```
static void q48_word_sub (
            VM * vm )  [static]
```

References q48_pop(), q48_push(), and q48_sub().

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

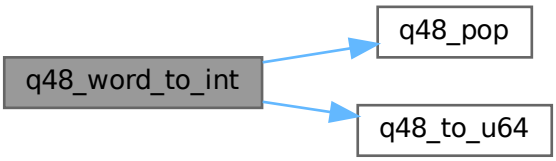
7.410.1.23 q48_word_to_int()

```
static void q48_word_to_int (
            VM * vm )  [static]
```

References q48_pop(), q48_to_u64(), and VM_PUSH.

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.410.1.24 q48_word_zero()

```
static void q48_word_zero (
            VM * vm )  [static]
```

References q48_push().

Referenced by register_q48_words().

Here is the call graph for this function:

Here is the caller graph for this function:

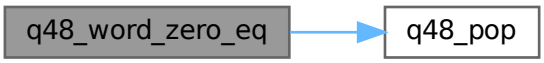
7.410.1.25 q48_word_zero_eq()

```
static void q48_word_zero_eq (
            VM * vm )  [static]
```

References q48_pop(), and VM_PUSH.

Referenced by register_q48_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.410.1.26 register_q48_words()

```
void register_q48_words (
            VM * vm )
```

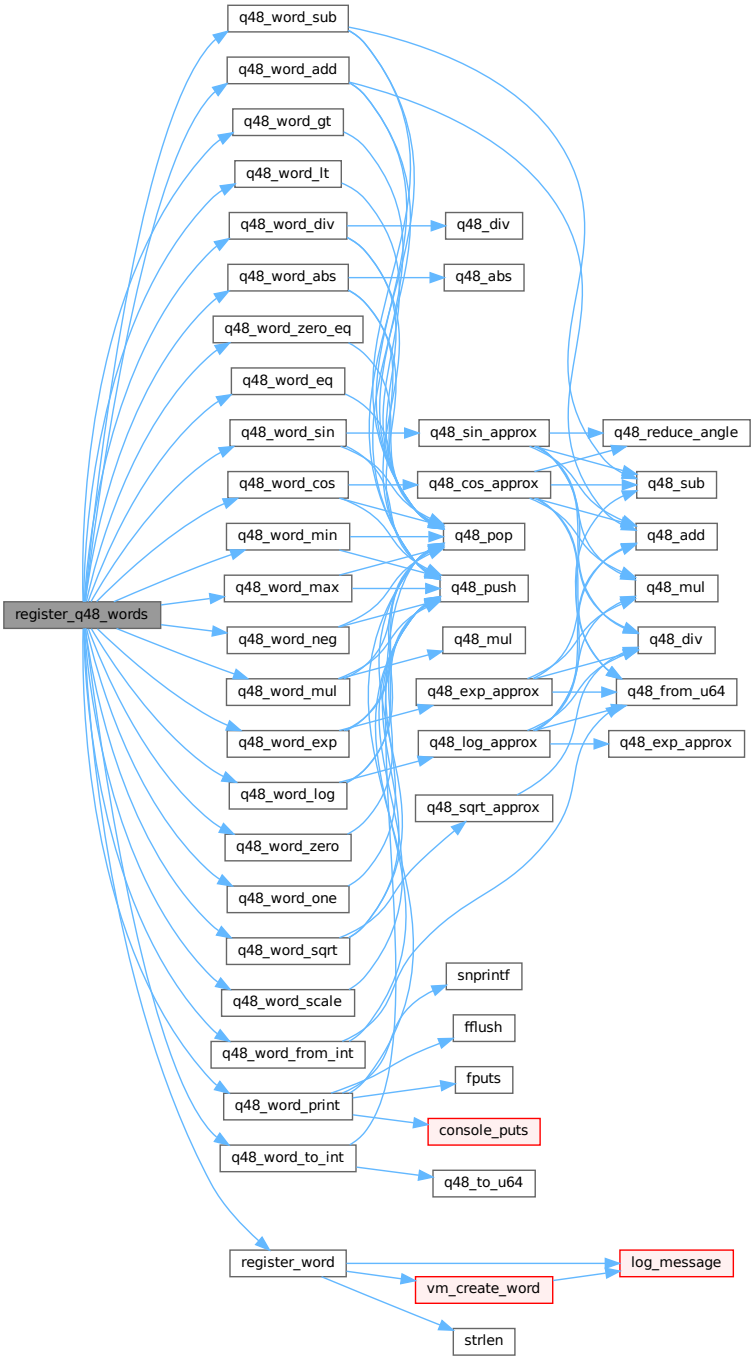
q48_words.h — Module 25: Q48.16 fixed-point FORTH words

Register all Q48.16 FORTH words with the VM. Module 25 in register_forth79_words().

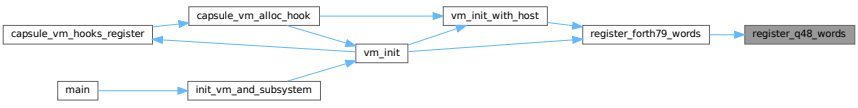
References q48_word_abs(), q48_word_add(), q48_word_cos(), q48_word_div(), q48_word_eq(), q48_word_exp(), q48_word_from_int(), q48_word_gt(), q48_word_log(), q48_word_lt(), q48_word_max(), q48_word_min(), q48_word_mul(), q48_word_neg(), q48_word_one(), q48_word_print(), q48_word_scale(), q48_word_sin(), q48_word_sqrt(), q48_word_sub(), q48_word_to_int(), q48_word_zero(), q48_word_zero_eq(), and register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:

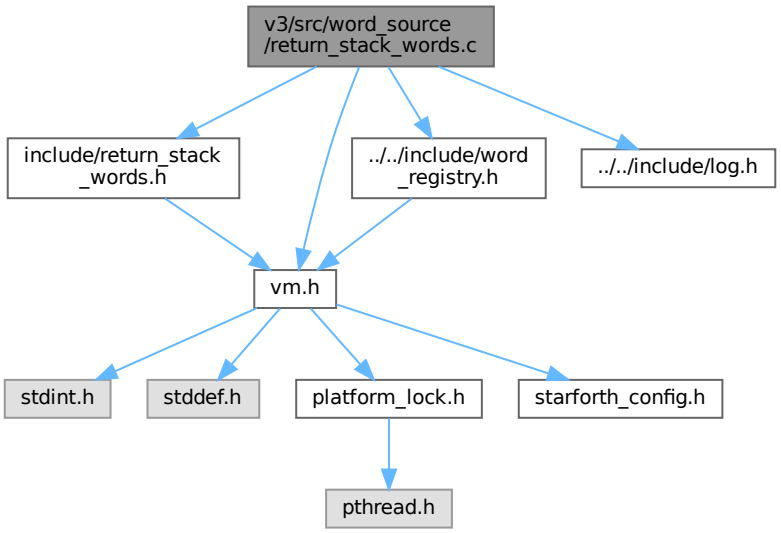


7.411 v3/src/word_source/return_stack_words.c File Reference

Implementation of FORTH-79 standard return stack words.

```
#include "include/return_stack_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include "../../include/vm.h"
```

Include dependency graph for return_stack_words.c:



Functions

- static void return_stack_word_to_r (VM *vm)

  Move TOS from data stack to return stack (>R word)
- static void return_stack_word_r_from (VM *vm)

  Move TOS from return stack back to data stack (R> word)
- static void return_stack_word_r_fetch (VM *vm)

  Copy TOS of return stack to data stack (R@ word)
- void register_return_stack_words (VM *vm)

  Register all FORTH-79 return stack manipulation words.

7.411.1 Detailed Description

Implementation of FORTH-79 standard return stack words.

StarForth - FORTH-79 Standard and ANSI C99 ONLY Last modified - 2025-09-10 See LICENSE file — No warranty.

FORTH-79 compliance:

- Provide only >R, R>, R@ for return stack transfer.
- No direct stack addressing (no RP!, RP@) — forbidden by FORTH-79.

7.411.2 Function Documentation

7.411.2.1 register_return_stack_words()

```
void register_return_stack_words (
            VM * vm )
```

Register all FORTH-79 return stack manipulation words.

Parameters

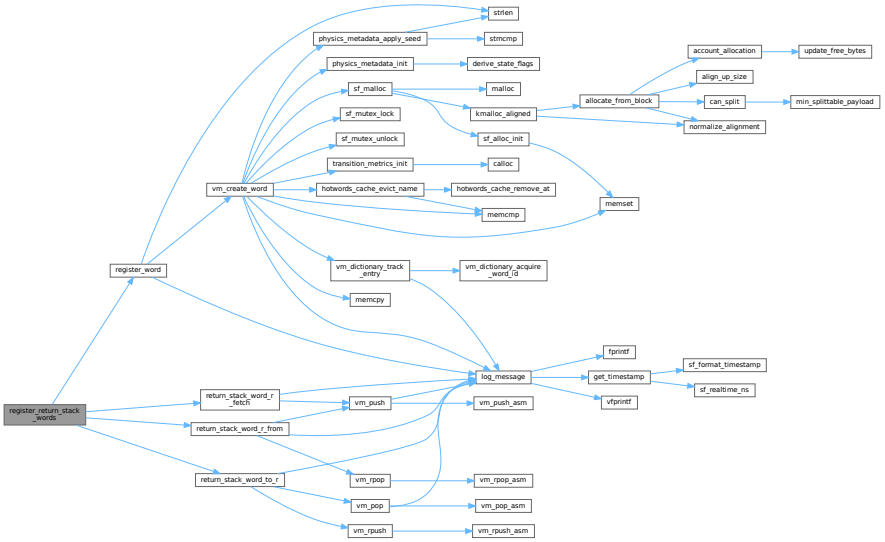
vm	Pointer to VM instance

Registers >R, R>, and R@ words. Direct stack addressing words (RP!, RP@) are deliberately omitted for FORTH-79 compliance.

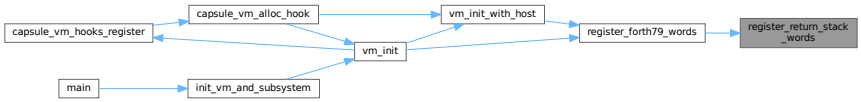
References register_word(), return_stack_word_r_fetch(), return_stack_word_r_from(), and return_stack_word_to_r().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.411.2.2 return_stack_word_r_fetch()

```
static void return_stack_word_r_fetch (
            VM * vm )  [static]
```

Copy TOS of return stack to data stack (R@ word)

Parameters

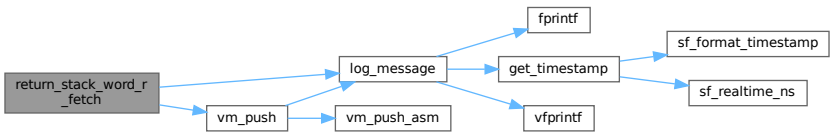
vm	Pointer to VM instance

Stack effect: (– x)

References VM::dsp, VM::ecw_nesting, VM::error, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, STACK_SIZE, and vm_push().

Referenced by register_return_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.411.2.3 return_stack_word_r_from()

```
static void return_stack_word_r_from (
            VM * vm )  [static]
```

Move TOS from return stack back to data stack (R> word)

Parameters

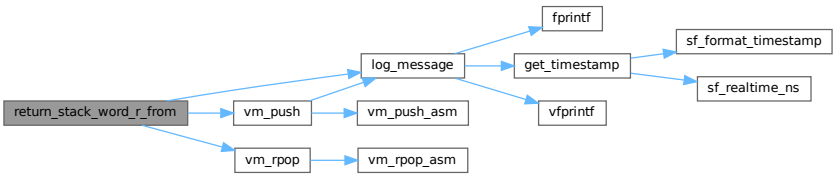
vm	Pointer to VM instance

Stack effect: (– x)

References VM::dsp, VM::ecw_nesting, VM::error, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, STACK_SIZE, vm_push(), and vm_rpop().

Referenced by register_return_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.411.2.4 return_stack_word_to_r()

```
static void return_stack_word_to_r (
            VM * vm )  [static]
```

Move TOS from data stack to return stack (>R word)

Parameters

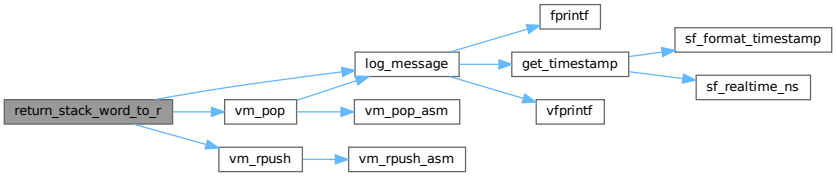
vm	Pointer to VM instance

Stack effect: (x –)

References VM::dsp, VM::ecw_nesting, VM::error, LOG_ERROR, log_message(), VM::return_stack, VM::rsp, STACK_SIZE, vm_pop(), and vm_rpush().

Referenced by register_return_stack_words().

Here is the call graph for this function:

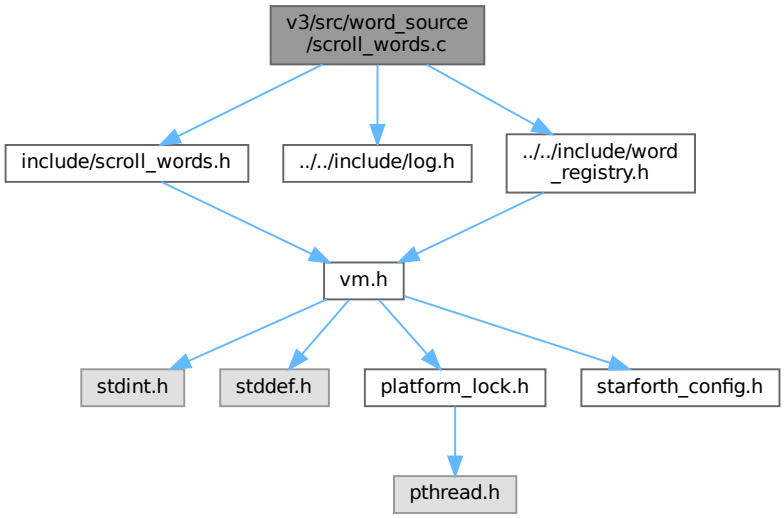


Here is the caller graph for this function:

7.412 v3/src/word_source/scroll_words.c File Reference

```
#include "include/scroll_words.h"
#include "../../include/log.h"
#include "../../include/word_registry.h"
```

Include dependency graph for scroll_words.c:



Functions

- void register_scroll_words (VM *vm)

7.412.1 Function Documentation

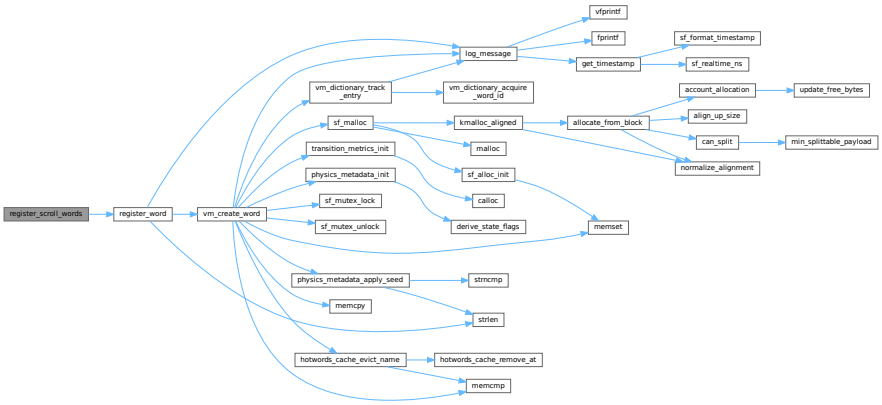
7.412.1.1 register_scroll_words()

```
void register_scroll_words (
            VM * vm )
```

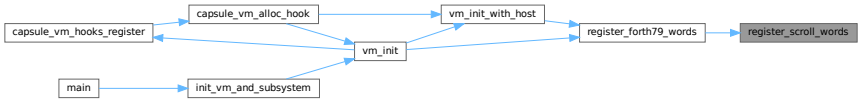
References register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



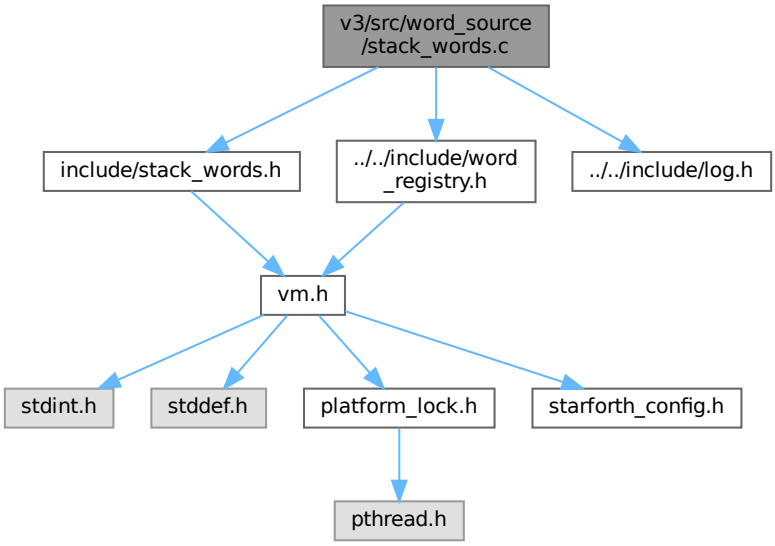
Here is the caller graph for this function:



7.413 v3/src/word_source/stack_words.c File Reference

```
#include "include/stack_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
```

Include dependency graph for stack_words.c:



Functions

- static void stack_word_drop (VM *vm)

  Implements DROP (n –) - Removes top stack item.
- static void stack_word_dup (VM *vm)

  Implements DUP (n – n n) - Duplicates top stack item.
- static void stack_word_question_dup (VM *vm)

  Implements ?DUP (n – n n | n – 0) - Conditionally duplicates top stack item.
- static void stack_word_swap (VM *vm)

  Implements SWAP (n1 n2 – n2 n1) - Exchanges top two stack items.
- static void stack_word_over (VM *vm)

  Implements OVER (n1 n2 – n1 n2 n1) - Copies second stack item to top.
- static void stack_word_rot (VM *vm)

  Implements ROT (n1 n2 n3 – n2 n3 n1) - Rotates top three stack items.
- static void stack_word_minus_rot (VM *vm)

  Implements -ROT (n1 n2 n3 – n3 n1 n2) - Reverse rotates top three items.
- static void stack_word_depth (VM *vm)

  Implements DEPTH (– n) - Returns number of stack items.
- static void stack_word_pick (VM *vm)

  Implements PICK (n – stack[n]) - Copies nth stack item to top.
- static void stack_word_roll (VM *vm)

  Implements ROLL (n –) - Moves nth stack item to top.
- void register_stack_words (VM *vm)

  Registers all stack operation words with the virtual machine.

7.413.1 Function Documentation

7.413.1.1 register_stack_words()

```
void register_stack_words (
            VM * vm )
```

Registers all stack operation words with the virtual machine.

Parameters

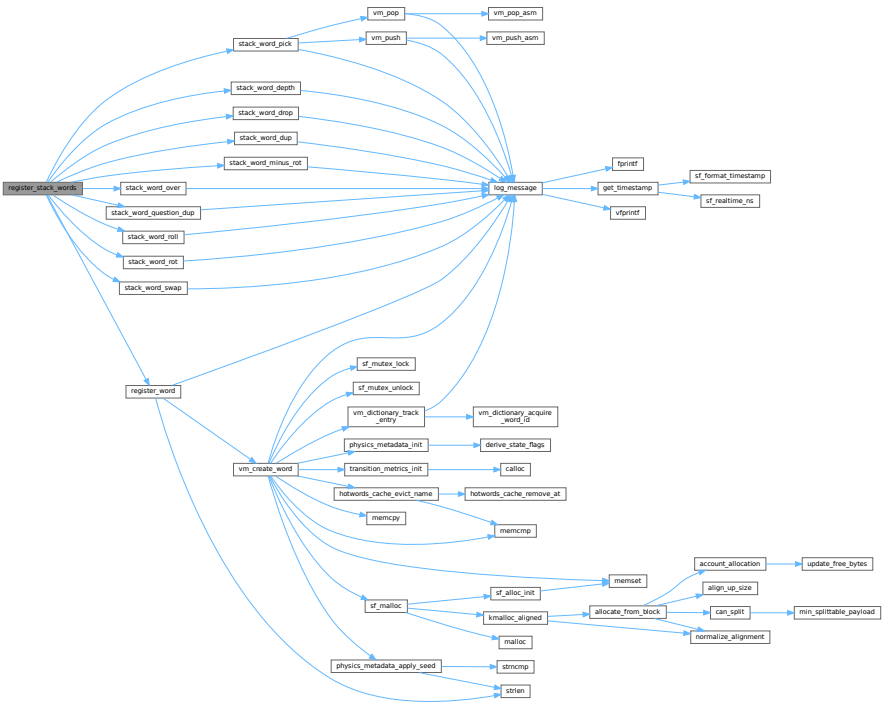
vm	Pointer to the virtual machine state

Registers all FORTH-79 standard stack operation words with the word registry.

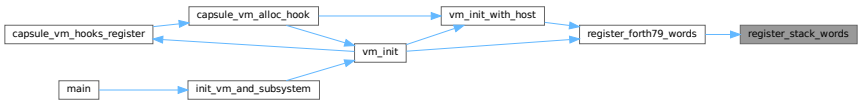
References register_word(), stack_word_depth(), stack_word_drop(), stack_word_dup(), stack_word_minus_rot(), stack_word_over(), stack_word_pick(), stack_word_question_dup(), stack_word_roll(), stack_word_rot(), and stack_word_swap().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.413.1.2 stack_word_depth()

```
static void stack_word_depth (
            VM * vm )  [static]
```

Implements DEPTH (– n) - Returns number of stack items.

Parameters

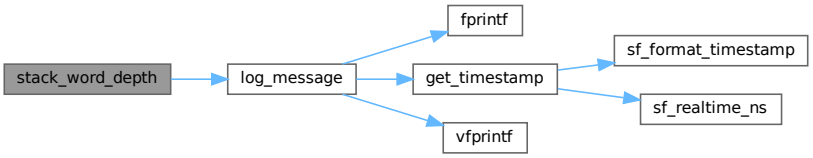
vm	Pointer to the virtual machine state

Pushes the current number of items on the data stack onto the stack.

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), and STACK_SIZE.

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.413.1.3 stack_word_drop()

```
static void stack_word_drop (
            VM * vm )  [static]
```

Implements DROP (n –) - Removes top stack item.

Parameters

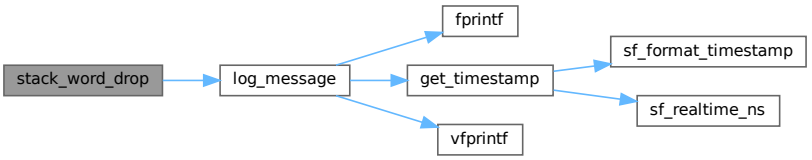
vm	Pointer to the virtual machine state

Removes the top item from the data stack. Generates an error on stack underflow.

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, and log_message().

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.413.1.4 stack_word_dup()

```
static void stack_word_dup (
            VM * vm )  [static]
```

Implements DUP (n – n n) - Duplicates top stack item.

Parameters

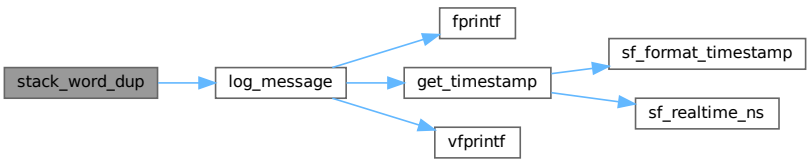
vm	Pointer to the virtual machine state

Duplicates the top item on the data stack. Generates an error on stack underflow/overflow.

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), STACK_SIZE, and UNLIKELY.

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.413.1.5 stack_word_minus_rot()

```
static void stack_word_minus_rot (
            VM * vm )  [static]
```

Implements -ROT (n1 n2 n3 – n3 n1 n2) - Reverse rotates top three items.

Parameters

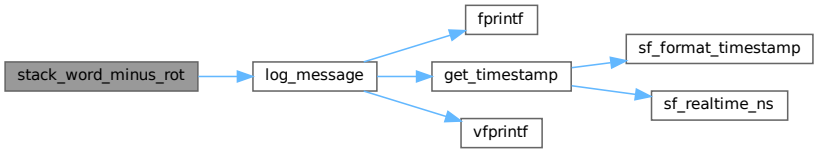
vm	Pointer to the virtual machine state

Performs the reverse rotation of the top three stack items.

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, and log_message().

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.413.1.6 stack_word_over()

```
static void stack_word_over (
            VM * vm )  [static]
```

Implements OVER (n1 n2 – n1 n2 n1) - Copies second stack item to top.

Parameters

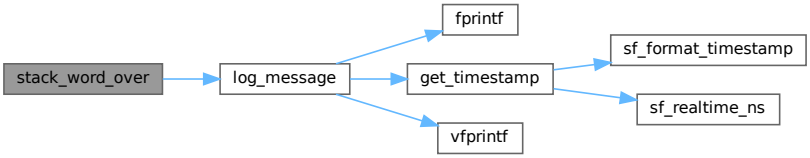
vm	Pointer to the virtual machine state

Copies the second item from the top of the stack to the top.

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), and STACK_SIZE.

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.413.1.7 stack_word_pick()

```
static void stack_word_pick (
            VM * vm )  [static]
```

Implements PICK (n – stack[n]) - Copies nth stack item to top.

Parameters

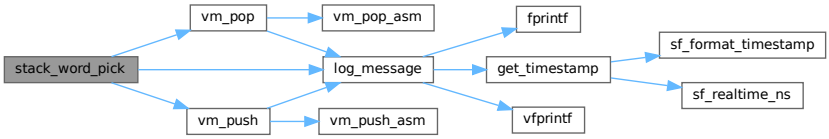
vm	Pointer to the virtual machine state

Copies the nth item from the stack to the top, where n is popped from the stack.

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_pop(), and vm_push().

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.413.1.8 stack_word_question_dup()

```
static void stack_word_question_dup (
            VM * vm )  [static]
```

Implements ?DUP (n – n n | n – 0) - Conditionally duplicates top stack item.

Parameters

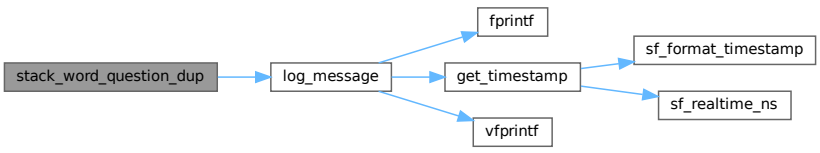
vm	Pointer to the virtual machine state

Duplicates top stack item if it's non-zero, otherwise leaves it unchanged.

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), and STACK_SIZE.

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.413.1.9 stack_word_roll()

```
static void stack_word_roll (
            VM * vm )  [static]
```

Implements ROLL (n –) - Moves nth stack item to top.

Parameters

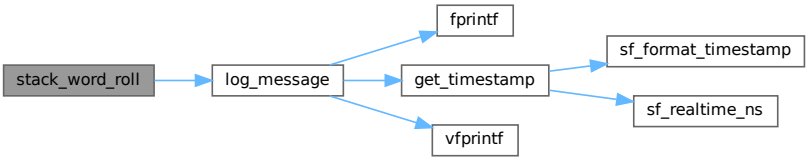
vm	Pointer to the virtual machine state

Moves the nth item from the stack to the top, shifting intermediate items down.

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, and log_message().

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.413.1.10 stack_word_rot()

```
static void stack_word_rot (
            VM * vm )  [static]
```

Implements ROT (n1 n2 n3 – n2 n3 n1) - Rotates top three stack items.

Parameters

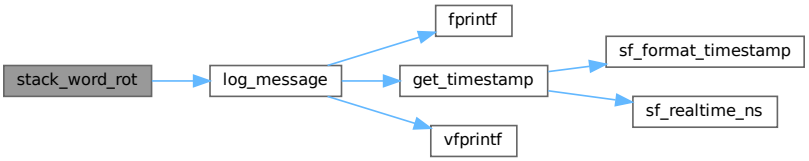
vm	Pointer to the virtual machine state

Rotates the top three items on the stack, moving the third item to the top.

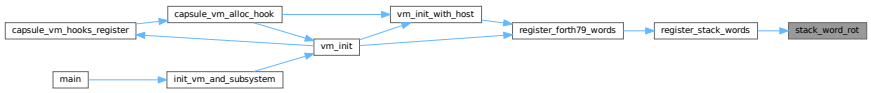
References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, and log_message().

Referenced by register_stack_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.413.1.11 stack_word_swap()

```
static void stack_word_swap (
            VM * vm )  [static]
```

Implements SWAP (n1 n2 – n2 n1) - Exchanges top two stack items.

Parameters

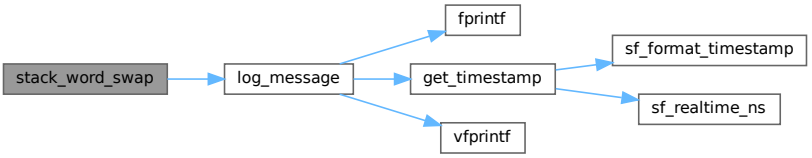
vm	Pointer to the virtual machine state

Exchanges the positions of the top two items on the data stack.

References VM::data_stack, VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), and UNLIKELY.

Referenced by register_stack_words().

Here is the call graph for this function:

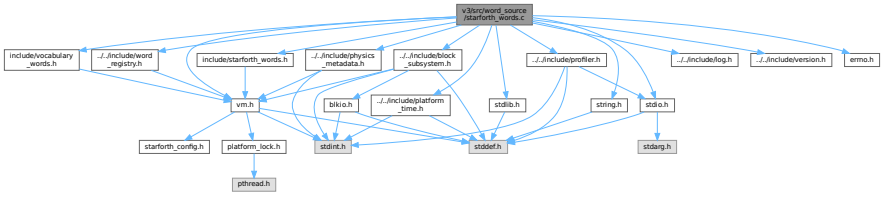


Here is the caller graph for this function:

7.414 v3/src/word_source/starforth_words.c File Reference

```
#include "include/starforth_words.h"
#include "include/vocabulary_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include "../../include/vm.h"
#include "../../include/version.h"
#include "../../include/physics_metadata.h"
#include "../../include/platform_time.h"
#include "../../include/profiler.h"
#include "../../include/block_subsystem.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
```

Include dependency graph for starforth_words.c:



Macros

- #define STARFORTH_CHECK_ARENA(tag) ((void)0)

Functions

- void starforth_word_execution_heat_fetch (VM *vm)
- void starforth_word_execution_heat_store (VM *vm)

  Set execution heat count for word at dictionary address.
- void starforth_word_word_execution_heat (VM *vm)

  Display execution heat statistics for all words in dictionary.
- void starforth_word_reset_execution_heat (VM *vm)

  Reset all execution heat counters to zero.
- void starforth_word_top_words (VM *vm)

  Display the N most frequently used words.
- void starforth_word_paren_dash (VM *vm)

  Shebang-style comment word for init.4th metadata.
- void starforth_word_init (VM *vm)

  Initialize system from init.4th configuration file.
- static uint64_t prng_next (void)

  Internal PRNG step using Linear Congruential Generator.
- void starforth_word_seed (VM *vm)

  Set the PRNG seed.
- void starforth_word_random (VM *vm)

  Generate bounded random number.
- void starforth_word_wait (VM *vm)

  Wait for specified number of heartbeat ticks.
- void starforth_word_version (VM *vm)

  Print StarForth version information (compliance)
- static void starforth_word_zuse_authenticate (VM *vm)
- static void starforth_word_zuse_session (VM *vm)
- static void starforth_word_zuse_pubkey_fetch (VM *vm)
- static void starforth_word_zuse_cert_installed_query (VM *vm)
- static void starforth_word_heartbeat_ticks (VM *vm)

  Read-only accessor for the canonical heartbeat tick counter.
- void register_starforth_words (VM *vm)

  Register StarForth vocabulary words with the VM.

Variables

- static uint64_t g_prng_state = 1

7.414.1 Macro Definition Documentation

7.414.1.1 STARFORTH_CHECK_ARENA

```
#define STARFORTH_CHECK_ARENA(
              tag ) ((void)0)
```

7.414.2 Function Documentation

7.414.2.1 prng_next()

```
static uint64_t prng_next (
            void  )  [static]
```

Internal PRNG step using Linear Congruential Generator.

Uses Numerical Recipes LCG constants for good statistical properties.

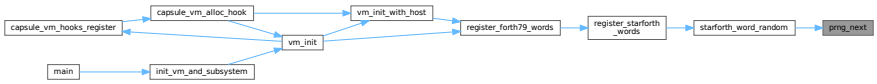
Returns

Next pseudo-random 64-bit value

References g_prng_state.

Referenced by starforth_word_random().

Here is the caller graph for this function:



7.414.2.2 register_starforth_words()

```
void register_starforth_words (
            VM * vm )
```

Register StarForth vocabulary words with the VM.

Registers all StarForth-specific words and creates the STARFORTH vocabulary

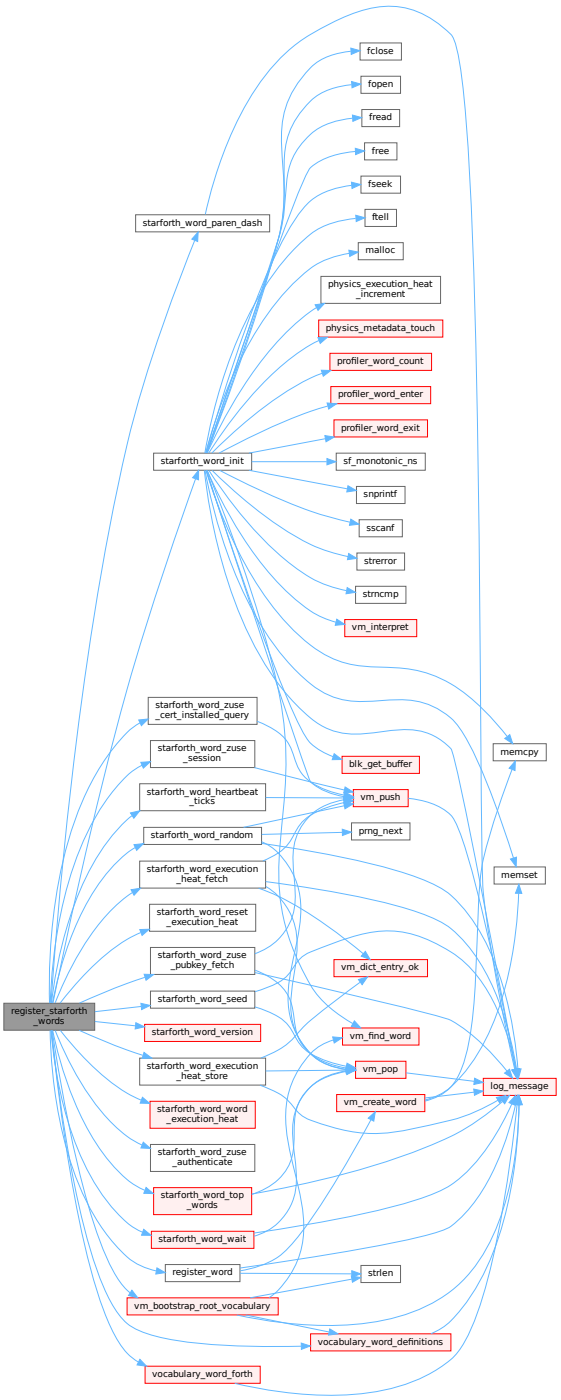
Parameters

vm	Pointer to the VM instance

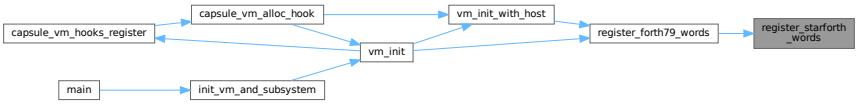
References register_word(), STARFORTH_CHECK_ARENA, starforth_word_execution_heat_fetch(), starforth_word_execution_heat_
starforth_word_heartbeat_ticks(), starforth_word_init(), starforth_word_paren_dash(), starforth_word_random(),
starforth_word_reset_execution_heat(), starforth_word_seed(), starforth_word_top_words(), starforth_word_version(),
starforth_word_wait(), starforth_word_word_execution_heat(), starforth_word_zuse_authenticate(), starforth_word_zuse_cert_installe
starforth_word_zuse_pubkey_fetch(), starforth_word_zuse_session(), vm_bootstrap_root_vocabulary(), vocabulary_word_definitions(
and vocabulary_word_forth().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



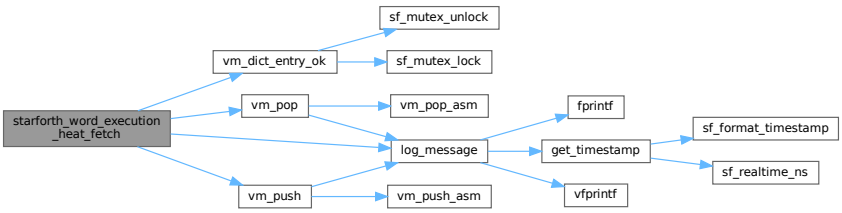
7.414.2.3 starforth_word_execution_heat_fetch()

```
void starforth_word_execution_heat_fetch (
            VM * vm )
```

References VM::dsp, entry, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_dict_entry_ok(), vm_pop(), and vm_push().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.414.2.4 starforth_word_execution_heat_store()

```
void starforth_word_execution_heat_store (
            VM * vm )
```

Set execution heat count for word at dictionary address.

Stack effect: (n addr –) Note: Exposed as ENTROPY! for FORTH compatibility, but sets execution heat.

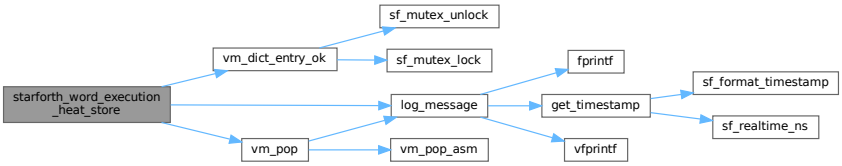
Parameters

vm	Pointer to the VM instance

References VM::dsp, entry, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), vm_dict_entry_ok(), and vm_pop().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.414.2.5 starforth_word_heartbeat_ticks()

```
static void starforth_word_heartbeat_ticks (
            VM * vm )  [static]
```

Read-only accessor for the canonical heartbeat tick counter.

Stack effect: (– n) Pushes vm->heartbeat.tick_count – Loop #7 "Adaptive Heartrate", the one clock this project's timing measurements are supposed to read, not host wall-clock. Read-only: no corresponding store word exists or should exist.

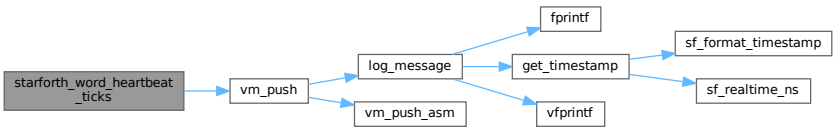
Parameters

vm	Pointer to the VM instance

References VM::heartbeat, HeartbeatState::tick_count, and vm_push().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.414.2.6 starforth_word_init()

```
void starforth_word_init (
            VM * vm )
```

Initialize system from init.4th configuration file.

Reads ./capsules/core/init.4th, parses Block headers, copies blocks sequentially starting at block 1, then executes them. Stack effect: (–)

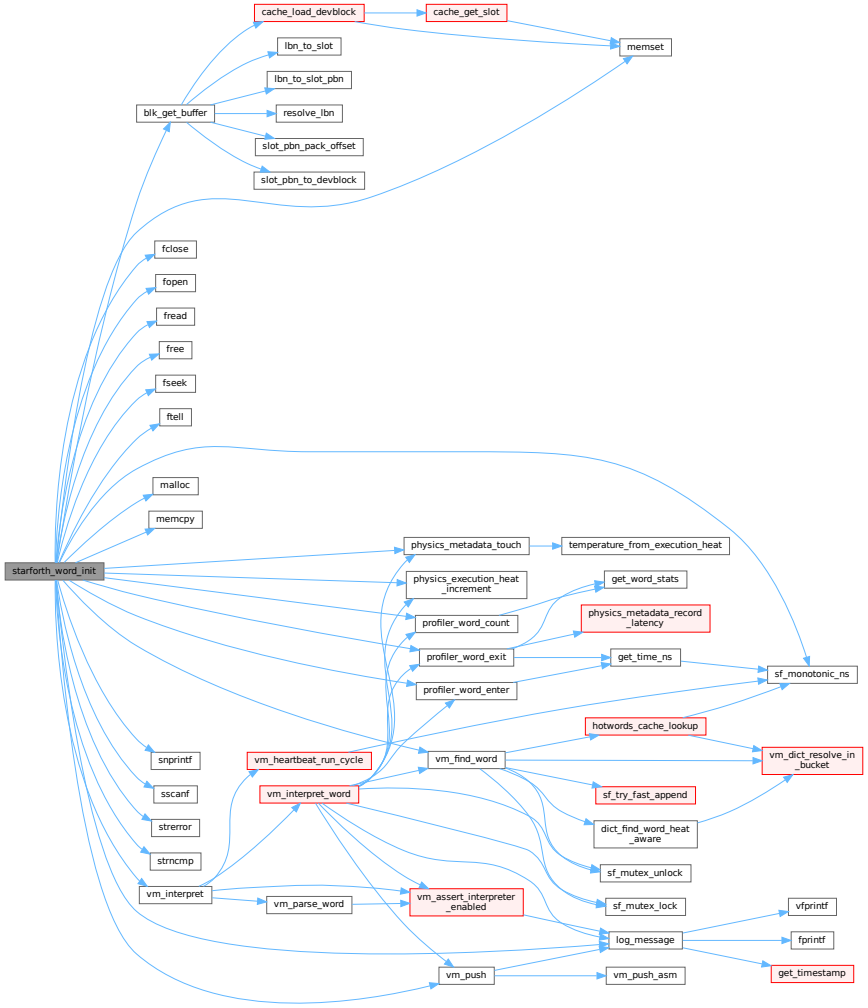
Parameters

vm	Pointer to the VM instance

References blk_get_buffer(), VM::current_executing_entry, ENOENT, errno, VM::error, DictEntry::execution_heat, fclose(), fopen(), fread(), free(), fseek(), ftell(), DictEntry::func, VM::halted, LOG_DEBUG, LOG_ERROR, LOG_INFO, log_message(), LOG_WARN, malloc(), memcpy(), memset(), physics_execution_heat_increment(), physics_metadata_touch(), profiler_word_count(), profiler_word_enter(), profiler_word_exit(), SEEK_END, SEEK_SET, sf_monotonic_ns(), snprintf(), sscanf(), strerror(), strncmp(), vm_find_word(), vm_interpret(), and vm_push().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.414.2.7 starforth_word_paren_dash()

```
void starforth_word_paren_dash (
            VM * vm )
```

Shebang-style comment word for init.4th metadata.

Starts with "(- " and consumes input until closing ")" Used for marking blocks that should be extracted to init.4th Stack effect: (–)

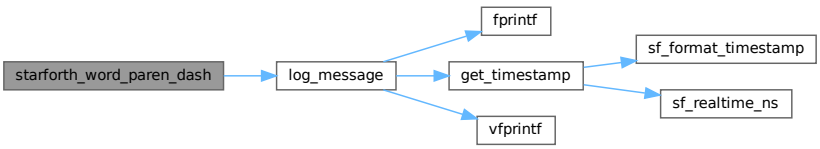
Parameters

vm	Pointer to the VM instance

References VM::input_buffer, VM::input_length, VM::input_pos, LOG_DEBUG, log_message(), and LOG_WARN.

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.414.2.8 starforth_word_random()

```
void starforth_word_random (
            VM * vm )
```

Generate bounded random number.

Parameters

vm	Pointer to VM instance @stack (lo hi – n)

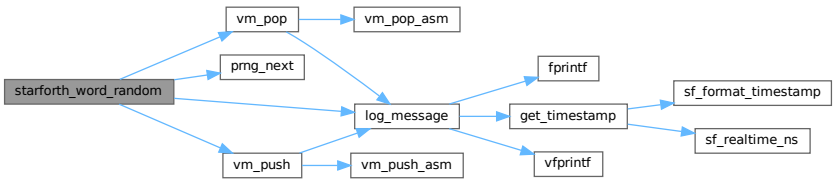
Note

Returns random value in inclusive range [lo, hi]

References VM::dsp, VM::error, LOG_DEBUG, LOG_ERROR, log_message(), prng_next(), vm_pop(), and vm_push().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.414.2.9 starforth_word_reset_execution_heat()

```
void starforth_word_reset_execution_heat (
            VM * vm )
```

Reset all execution heat counters to zero.

Stack effect: (–)

Parameters

vm	Pointer to the VM instance

References entry, and VM::latest.

Referenced by register_starforth_words().

Here is the caller graph for this function:

7.414.2.10 starforth_word_seed()

```
void starforth_word_seed (
            VM * vm )
```

Set the PRNG seed.

Stack effect: (n –) Sets the internal PRNG state for reproducible random sequences.

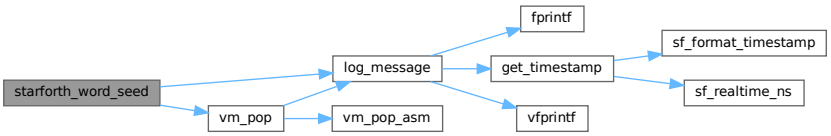
Parameters

vm	Pointer to the VM instance

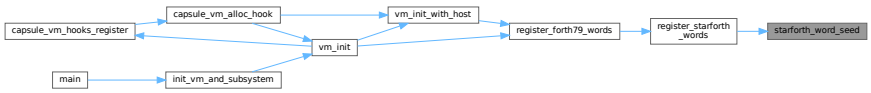
References VM::dsp, VM::error, g_prng_state, LOG_DEBUG, LOG_ERROR, log_message(), and vm_pop().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.414.2.11 starforth_word_top_words()

```
void starforth_word_top_words (
            VM * vm )
```

Display the N most frequently used words.

Stack effect: (n –)

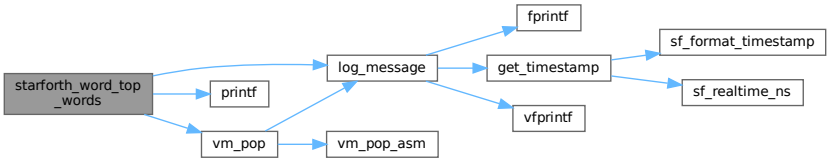
Parameters

vm	Pointer to the VM instance

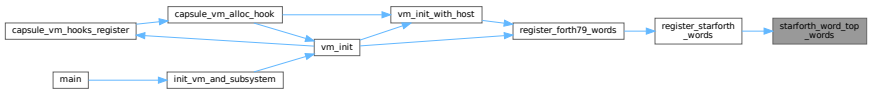
References VM::dsp, entry, VM::error, execution_heat, VM::latest, LOG_ERROR, log_message(), printf(), vm_pop(), and word_count.

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.414.2.12 starforth_word_version()

```
void starforth_word_version (
            VM * vm )
```

Print StarForth version information (compliance)

Prints: StarForth v<version> <architecture> <variant> <timestamp> Stack effect: (–)

Parameters

vm	Pointer to the VM instance

References printf(), and STARFORTH_VERSION_FULL.

Referenced by register_starforth_words().

Here is the call graph for this function:

Here is the caller graph for this function:

7.414.2.13 starforth_word_wait()

```
void starforth_word_wait (
            VM * vm )
```

Wait for specified number of heartbeat ticks.

Stack effect: (n –) Counts n heartbeat ticks by calling vm_tick() n times. Time is relative — we count heartbeats, not wall-clock milliseconds. This is architecture-independent: works on amd64, aarch64, riscv64 without depending on any platform timer.

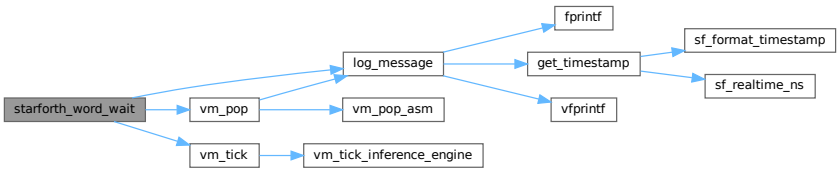
Parameters

vm	Pointer to the VM instance

References VM::dsp, VM::error, LOG_ERROR, log_message(), vm_pop(), and vm_tick().

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.414.2.14 starforth_word_word_execution_heat()

```
void starforth_word_word_execution_heat (
            VM * vm )
```

Display execution heat statistics for all words in dictionary.

Stack effect: (–) Shows execution frequency (execution_heat) for each word.

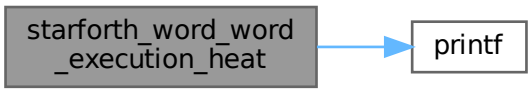
Parameters

vm	Pointer to the VM instance

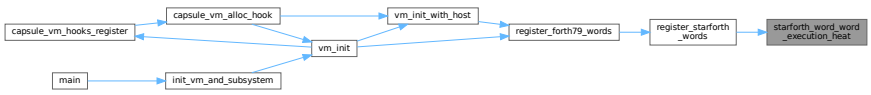
References entry, VM::latest, printf(), and word_count.

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.414.2.15 starforth_word_zuse_authenticate()

```
static void starforth_word_zuse_authenticate (
            VM * vm )  [static]
```

References VM::zuse_session.

Referenced by register_starforth_words().

Here is the caller graph for this function:

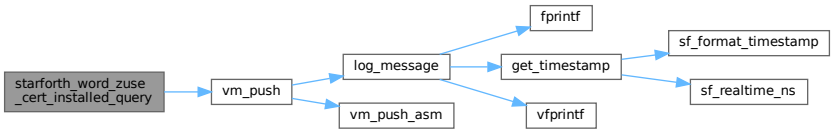
7.414.2.16 starforth_word_zuse_cert_installed_query()

```
static void starforth_word_zuse_cert_installed_query (
            VM * vm )  [static]
```

References vm_push(), and VM::zuse_cert_installed.

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

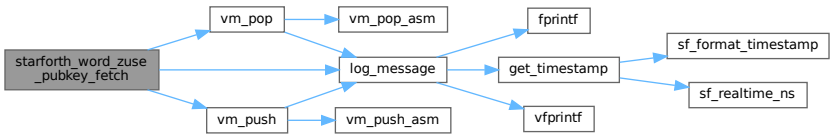
7.414.2.17 starforth_word_zuse_pubkey_fetch()

```
static void starforth_word_zuse_pubkey_fetch (
            VM * vm )  [static]
```

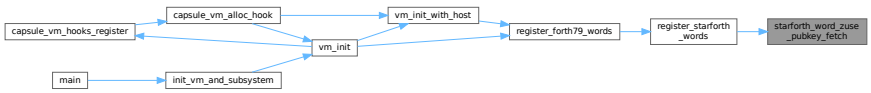
References VM::dsp, VM::error, LOG_ERROR, log_message(), vm_pop(), vm_push(), and VM::zuse_cert_pubkey.

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:



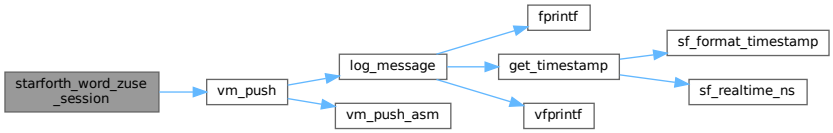
7.414.2.18 starforth_word_zuse_session()

```
static void starforth_word_zuse_session (
            VM * vm )  [static]
```

References vm_push(), and VM::zuse_session.

Referenced by register_starforth_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.414.3 Variable Documentation

7.414.3.1 g_prng_state

```
uint64_t g_prng_state = 1  [static]
```

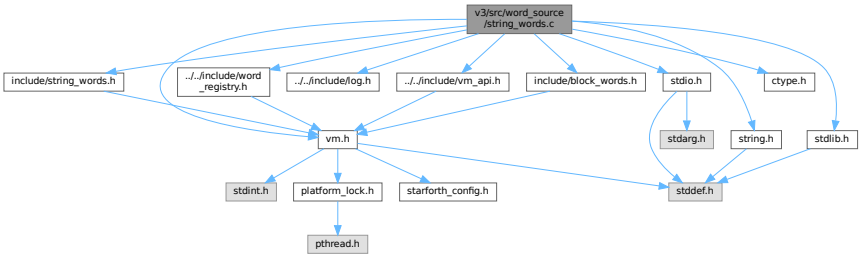
Referenced by prng_next(), and starforth_word_seed().

7.415 v3/src/word_source/string_words.c File Reference

```
#include "include/string_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include "../../include/vm.h"
#include "../../include/vm_api.h"
#include "include/block_words.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>
```

```
#include <stdlib.h>
```

Include dependency graph for string_words.c:



Macros

- #define WORD_SCRATCH_CAP 64

Functions

- static vaddr_t vaddr_from_ptr (VM *vm, const void *p)

  Convert a C pointer that points into vm->memory into a VM address (offset)
- static int ensure_input (VM *vm)

  Ensure input buffers and variables are available.
- static int convert_string_to_number (const char *str, size_t len, cell_t *result)
- void string_word_count (VM *vm)

  COUNT (addr1 – addr2 u)
- void string_word_expect (VM *vm)
- void string_word_span (VM *vm)
- void string_word_query (VM *vm)
- void string_word_tib (VM *vm)
- void string_word_word (VM *vm)
- void string_word_to_in (VM *vm)
- void string_word_source (VM *vm)
- void string_word_bl (VM *vm)
- void string_word_bracket_tick (VM *vm)
- void string_word_literal (VM *vm)
- void string_word_bracket_literal (VM *vm)
- void string_word_convert (VM *vm)
- void string_word_number (VM *vm)
- void string_word_enclose (VM *vm)
- static void string_runtime_s_quote (VM *vm)
- static void string_word_s_quote (VM *vm)
- static void string_word_minus_trailing (VM *vm)
- static void string_word_cmove (VM *vm)
- static void string_word_cmove_greater (VM *vm)
- static void string_word_compare (VM *vm)
- static void string_word_search (VM *vm)
- static void string_word_scan (VM *vm)
- static void string_word_skip (VM *vm)
- static void string_word_blank (VM *vm)
- void register_string_words (VM *vm)

  COUNT (addr1 – addr2 u) Get string length and address Returns the character count and address of a counted string.

Variables

- static vaddr_t word_scratch_addr = 0

7.415.1 Macro Definition Documentation

7.415.1.1 WORD_SCRATCH_CAP

```
#define WORD_SCRATCH_CAP 64
```

7.415.2 Function Documentation

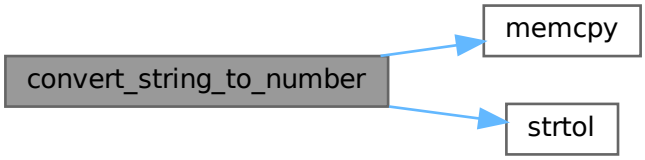
7.415.2.1 convert_string_to_number()

```
static int convert_string_to_number (
            const char * str,
            size_t len,
            cell_t * result )  [static]
```

References memcpy(), and strtol().

Referenced by string_word_number().

Here is the call graph for this function:



Here is the caller graph for this function:

7.415.2.2 ensure_input()

```
static int ensure_input (
            VM * vm )  [inline], [static]
```

Ensure input buffers and variables are available.

Parameters

vm	Pointer to VM instance

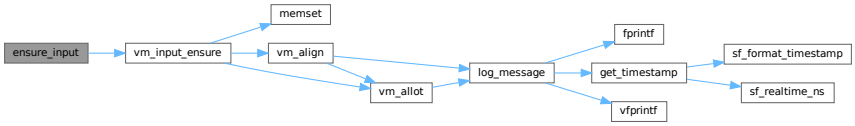
Returns

0 on success, non-zero on error

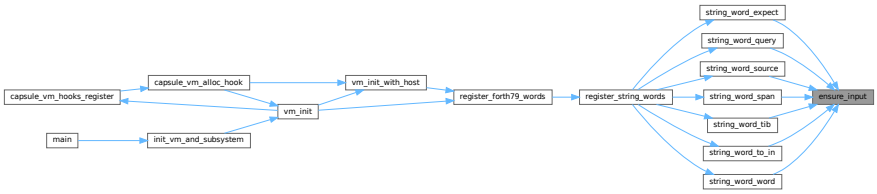
References VM::error, and vm_input_ensure().

Referenced by string_word_expect(), string_word_query(), string_word_source(), string_word_span(), string_word_tib(), string_word_to_in(), and string_word_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.415.2.3 register_string_words()

```
void register_string_words (
            VM * vm )
```

COUNT (addr1 – addr2 u) Get string length and address Returns the character count and address of a counted string.

EXPECT (addr u –) Accept input line Accepts up to u characters of input and stores them at addr

SPAN (– addr) Variable: number of characters input Contains the count of characters actually received by EXPECT

QUERY (–) Accept input into TIB Accepts a line of input into the Terminal Input Buffer

TIB (– addr) Terminal input buffer address Returns the address of the Terminal Input Buffer

WORD (c – addr) Parse next word, delimited by c Parses the next word from input stream using c as delimiter

>IN (– addr) Input stream pointer Contains the offset into the input buffer

SOURCE (– addr u) Input source specification Returns current input buffer address and length

BL (– c) ASCII space character (32) Pushes ASCII space character value onto stack

FIND (addr – addr flag) Find word in dictionary Searches dictionary for word, returns status flag

' (– xt) Get execution token of next word Returns execution token of next word in input stream

['] (– xt) Compile execution token Compiles execution token as literal

LITERAL (n –) Compile literal number Compiles top stack value as literal

[LITERAL] (n –) Compile literal at compile time Compiles literal during compilation

CONVERT (d1 addr1 – d2 addr2) Convert string to number Converts string starting at addr1 to number

NUMBER (addr – n flag) Convert string to single number Converts string to single precision number

ENCLOSE (addr c – addr1 n1 n2 n3) Parse for delimiter Parses string for delimiter character

Register all string processing words with VM

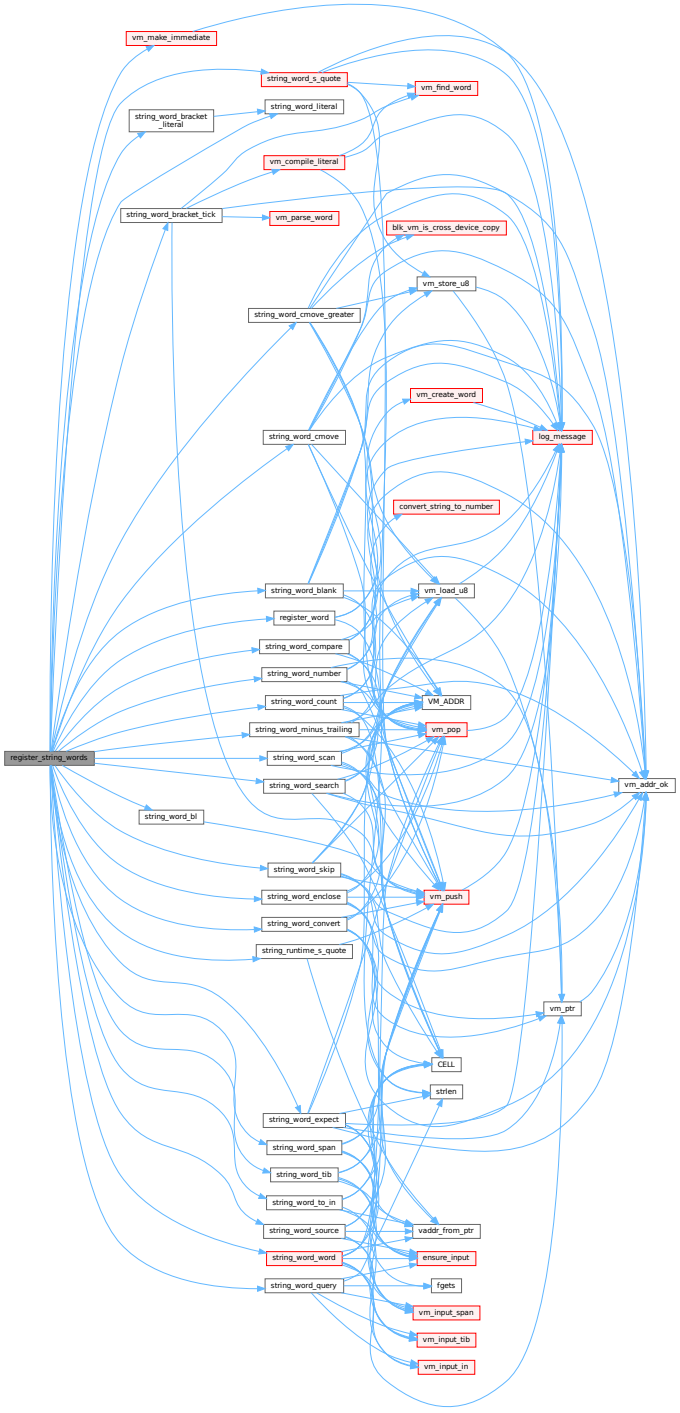
Parameters

vm	Pointer to VM instance where words will be registered

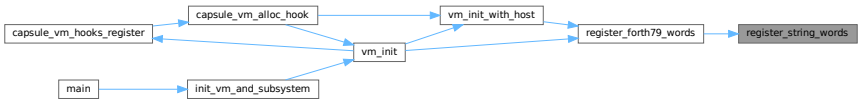
References register_word(), string_runtime_s_quote(), string_word_bl(), string_word_blank(), string_word_bracket_literal(), string_word_bracket_tick(), string_word_cmove(), string_word_cmove_greater(), string_word_compare(), string_word_convert(), string_word_count(), string_word_enclose(), string_word_expect(), string_word_literal(), string_word_minus_trailing(), string_word_number(), string_word_query(), string_word_s_quote(), string_word_scan(), string_word_search(), string_word_skip(), string_word_source(), string_word_span(), string_word_tib(), string_word_to_in(), string_word_word(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



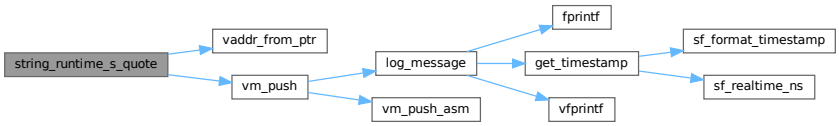
7.415.2.4 string_runtime_s_quote()

```
static void string_runtime_s_quote (
            VM * vm )  [static]
```

References VM::error, VM::return_stack, VM::rsp, vaddr_from_ptr(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

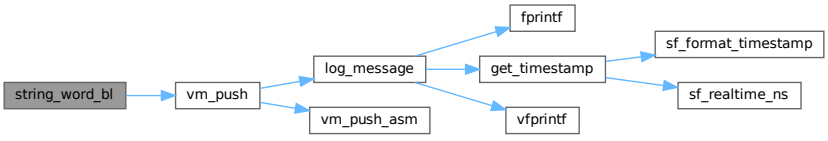
7.415.2.5 string_word_bl()

```
void string_word_bl (
            VM * vm )
```

References vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

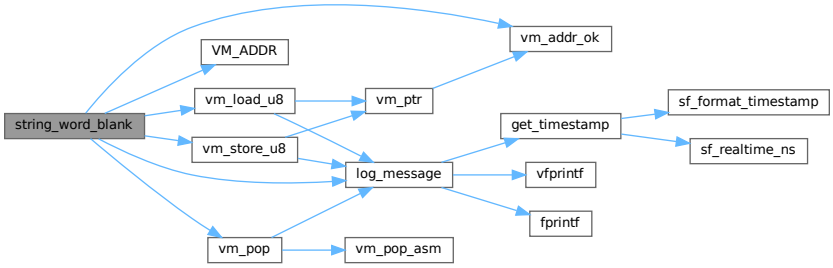
7.415.2.6 string_word_blank()

```
static void string_word_blank (
            VM * vm )  [static]
```

References VM::dsp, VM::error, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_store_u8().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

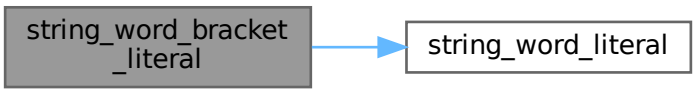
7.415.2.7 string_word_bracket_literal()

```
void string_word_bracket_literal (
            VM * vm )
```

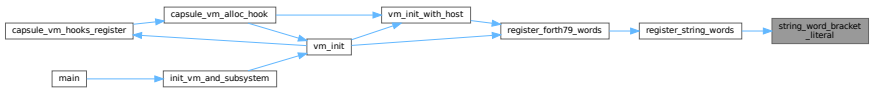
References string_word_literal().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:



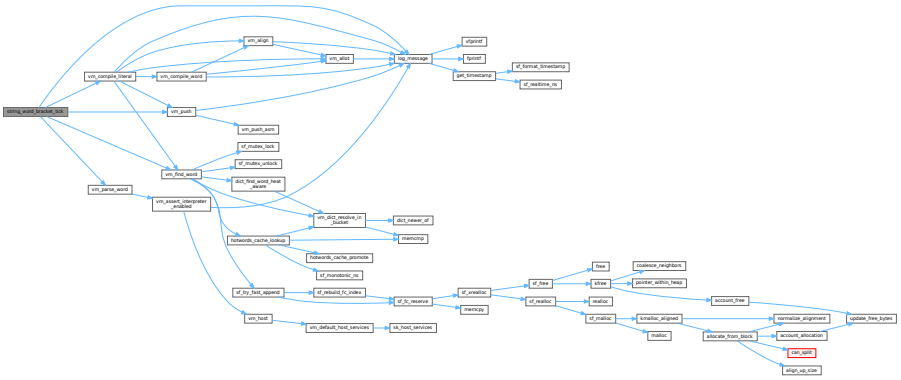
7.415.2.8 string_word_bracket_tick()

```
void string_word_bracket_tick (
            VM * vm )
```

References VM::error, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_compile_literal(), vm_find_word(), vm_parse_word(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

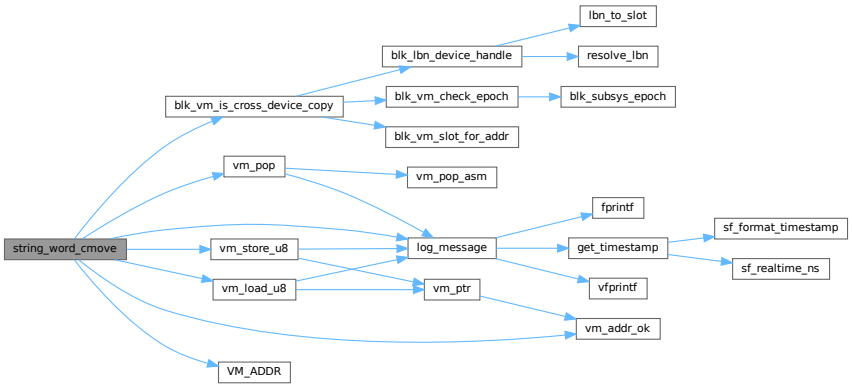
7.415.2.9 string_word_cmove()

```
static void string_word_cmove (
            VM * vm )  [static]
```

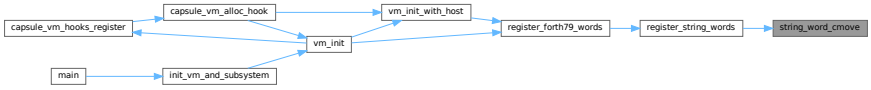
References blk_vm_is_cross_device_copy(), VM::dsp, VM::error, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_store_u8().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:



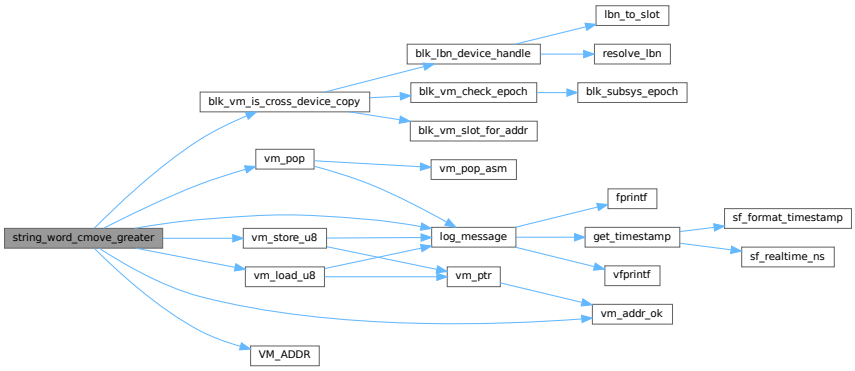
7.415.2.10 string_word_cmove_greater()

```
static void string_word_cmove_greater (
            VM * vm )  [static]
```

References blk_vm_is_cross_device_copy(), VM::dsp, VM::error, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_store_u8().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

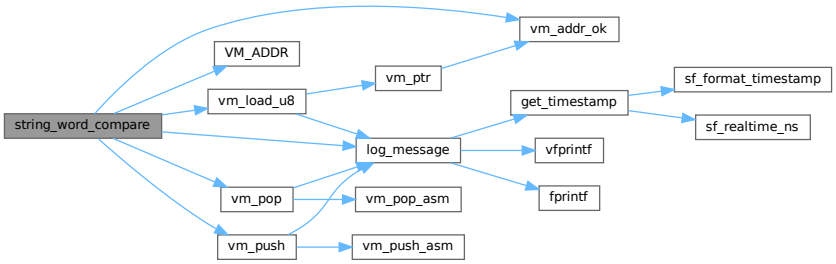
7.415.2.11 string_word_compare()

```
static void string_word_compare (
            VM * vm )  [static]
```

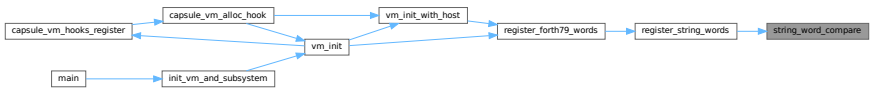
References VM::dsp, VM::error, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:



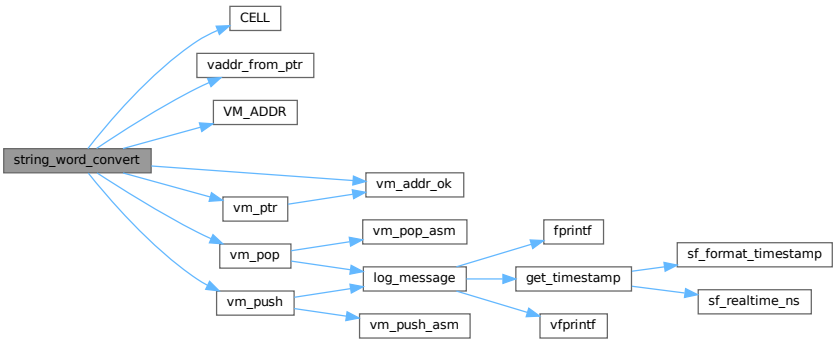
7.415.2.12 string_word_convert()

```
void string_word_convert (
            VM * vm )
```

References CELL(), VM::dsp, VM::error, vaddr_from_ptr(), VM_ADDR(), vm_addr_ok(), vm_pop(), vm_ptr(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.415.2.13 string_word_count()

```
void string_word_count (
            VM * vm )
```

COUNT (addr1 – addr2 u)

Get string length and address. Converts a counted string at addr1 into an address (addr2) and length (u) pair.

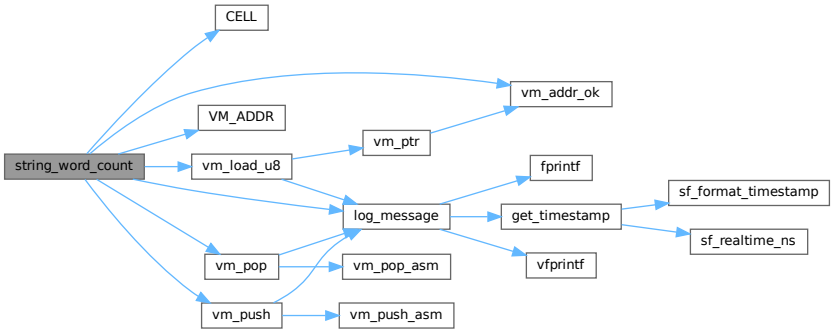
Parameters

vm	Pointer to VM instance

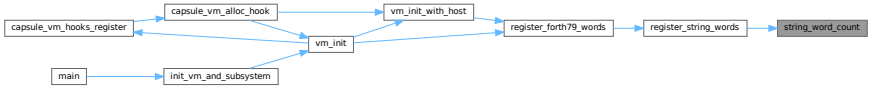
References CELL(), count, VM::dsp, VM::error, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:



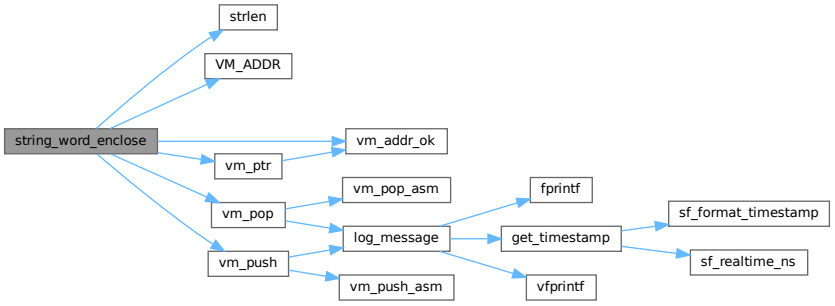
7.415.2.14 string_word_enclose()

```
void string_word_enclose (
            VM * vm )
```

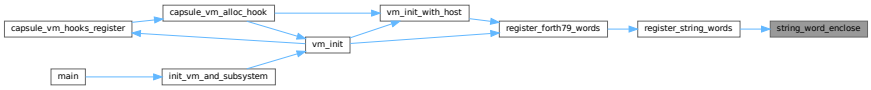
References VM::dsp, VM::error, strlen(), VM_ADDR(), vm_addr_ok(), vm_pop(), vm_ptr(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:



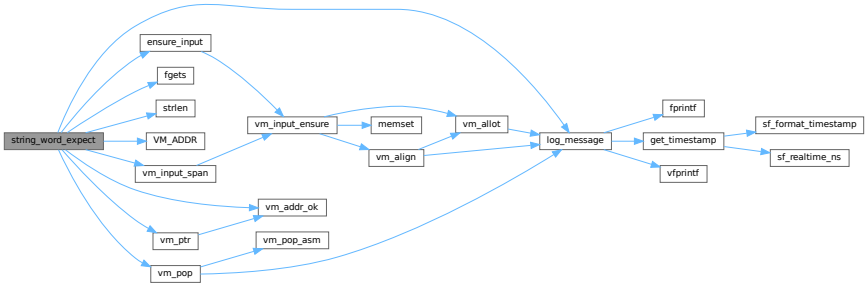
7.415.2.15 string_word_expect()

```
void string_word_expect (
            VM * vm )
```

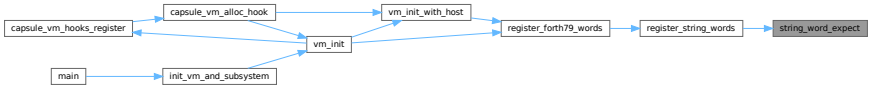
References VM::dsp, ensure_input(), VM::error, fgets(), LOG_ERROR, log_message(), stdin, strlen(), VM_ADDR(), vm_addr_ok(), vm_input_span(), vm_pop(), and vm_ptr().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.415.2.16 string_word_literal()

```
void string_word_literal (
            VM * vm )
```

Referenced by register_string_words(), and string_word_bracket_literal().

Here is the caller graph for this function:

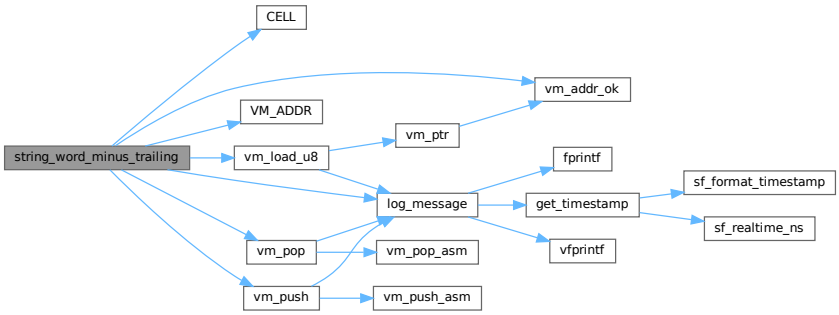
7.415.2.17 string_word_minus_trailing()

```
static void string_word_minus_trailing (
            VM * vm )  [static]
```

References CELL(), VM::dsp, VM::error, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

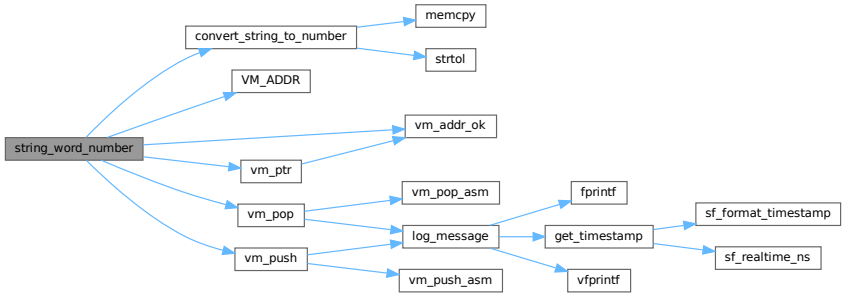
7.415.2.18 string_word_number()

```
void string_word_number (
            VM * vm )
```

References convert_string_to_number(), VM::dsp, VM::error, VM_ADDR(), vm_addr_ok(), vm_pop(), vm_ptr(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

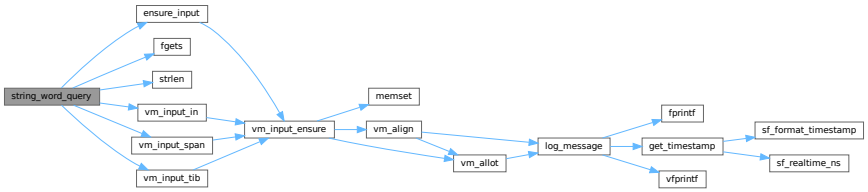
7.415.2.19 string_word_query()

```
void string_word_query (
            VM * vm )
```

References ensure_input(), VM::error, fgets(), INPUT_BUFFER_SIZE, stdin, strlen(), vm_input_in(), vm_input_span(), and vm_input_tib().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

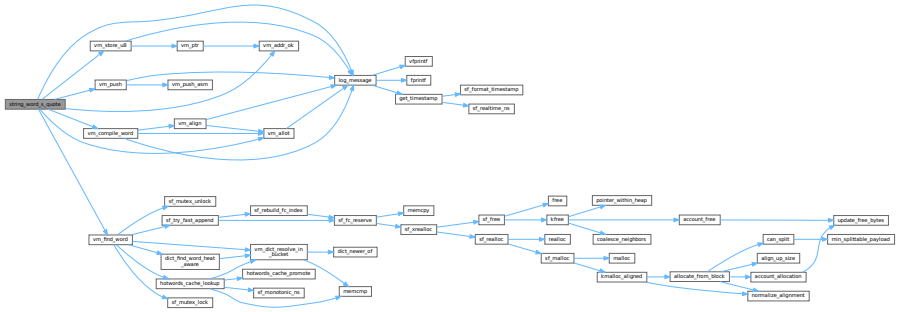
7.415.2.20 string_word_s_quote()

```
static void string_word_s_quote (
            VM * vm )  [static]
```

References VM::error, VM::here, VM::input_buffer, VM::input_length, VM::input_pos, LOG_DEBUG, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, vm_addr_ok(), vm_allot(), vm_compile_word(), vm_find_word(), vm_push(), and vm_store_u8().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

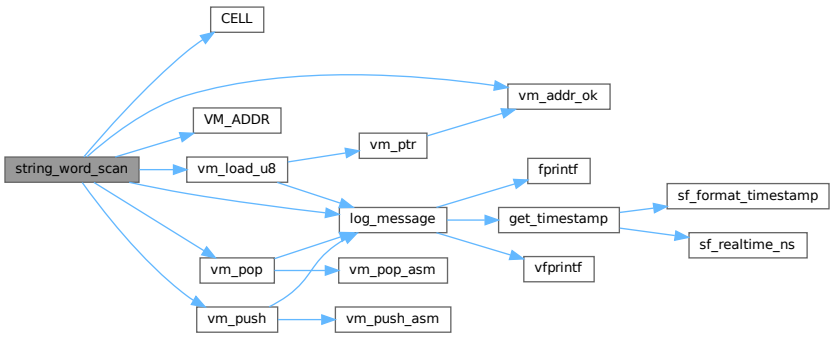
7.415.2.21 string_word_scan()

```
static void string_word_scan (
            VM * vm )  [static]
```

References CELL(), VM::dsp, VM::error, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

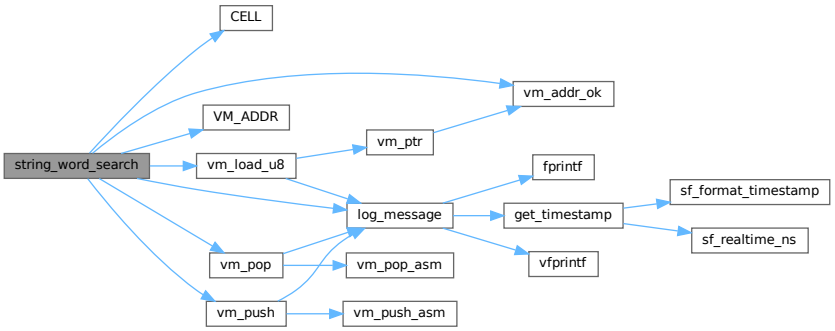
7.415.2.22 string_word_search()

```
static void string_word_search (
            VM * vm )  [static]
```

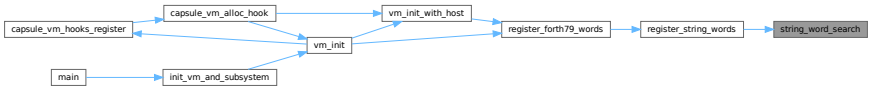
References CELL(), VM::dsp, VM::error, limit, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:



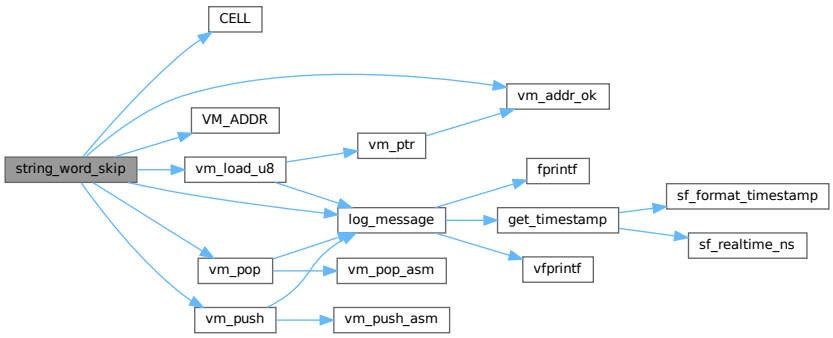
7.415.2.23 string_word_skip()

```
static void string_word_skip (
            VM * vm )  [static]
```

References CELL(), VM::dsp, VM::error, LOG_ERROR, log_message(), VM_ADDR(), vm_addr_ok(), vm_load_u8(), vm_pop(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

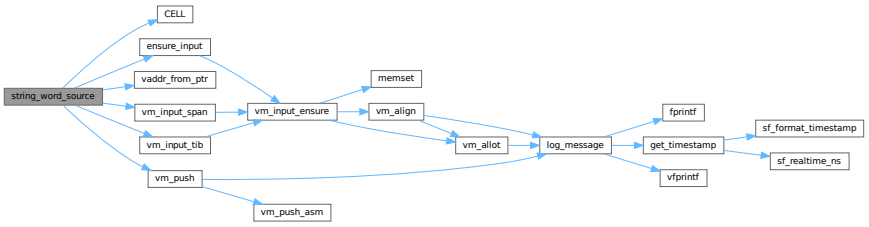
7.415.2.24 string_word_source()

```
void string_word_source (
            VM * vm )
```

References CELL(), ensure_input(), VM::error, vaddr_from_ptr(), vm_input_span(), vm_input_tib(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

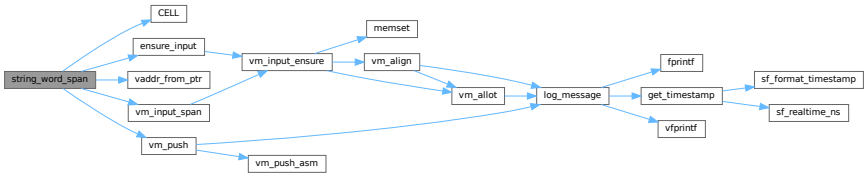
7.415.2.25 string_word_span()

```
void string_word_span (
            VM * vm )
```

References CELL(), ensure_input(), VM::error, vaddr_from_ptr(), vm_input_span(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

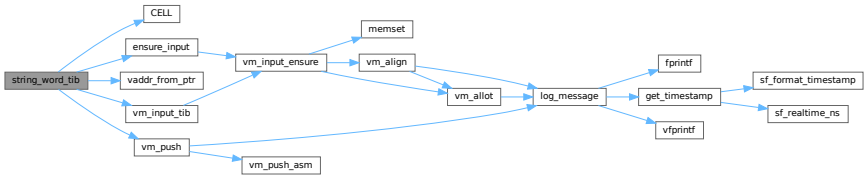
7.415.2.26 string_word_tib()

```
void string_word_tib (
            VM * vm )
```

References CELL(), ensure_input(), VM::error, vaddr_from_ptr(), vm_input_tib(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

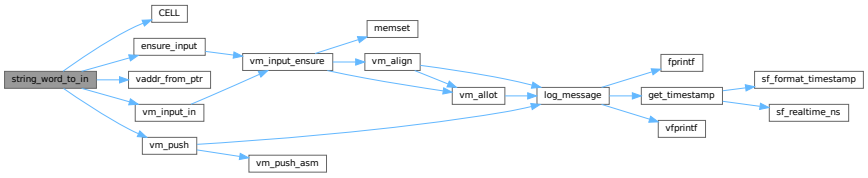
7.415.2.27 string_word_to_in()

```
void string_word_to_in (
            VM * vm )
```

References CELL(), ensure_input(), VM::error, vaddr_from_ptr(), vm_input_in(), and vm_push().

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

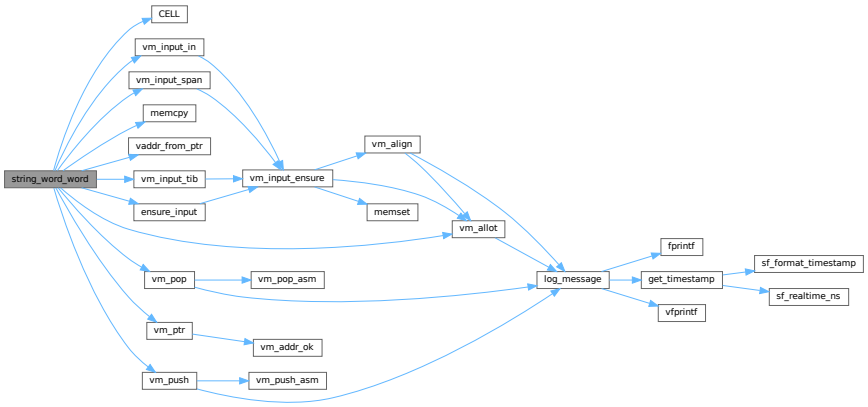
7.415.2.28 string_word_word()

```
void string_word_word (
            VM * vm )
```

References CELL(), VM::dsp, ensure_input(), VM::error, memcpy(), vaddr_from_ptr(), vm_allot(), vm_input_in(), vm_input_span(), vm_input_tib(), vm_pop(), vm_ptr(), vm_push(), word_scratch_addr, and WORD_SCRATCH_CAP.

Referenced by register_string_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.415.2.29 vaddr_from_ptr()

```
static vaddr_t vaddr_from_ptr (
            VM * vm,
            const void * p )  [inline], [static]
```

Convert a C pointer that points into vm->memory into a VM address (offset)

Parameters

vm	Pointer to VM instance
p	Pointer into VM memory space

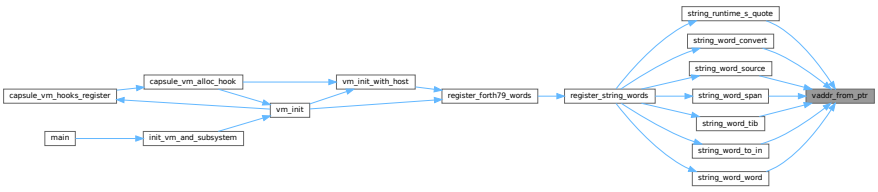
Returns

Virtual address (offset) corresponding to the pointer

References VM::memory.

Referenced by string_runtime_s_quote(), string_word_convert(), string_word_source(), string_word_span(), string_word_tib(), string_word_to_in(), and string_word_word().

Here is the caller graph for this function:



7.415.3 Variable Documentation

7.415.3.1 word_scratch_addr

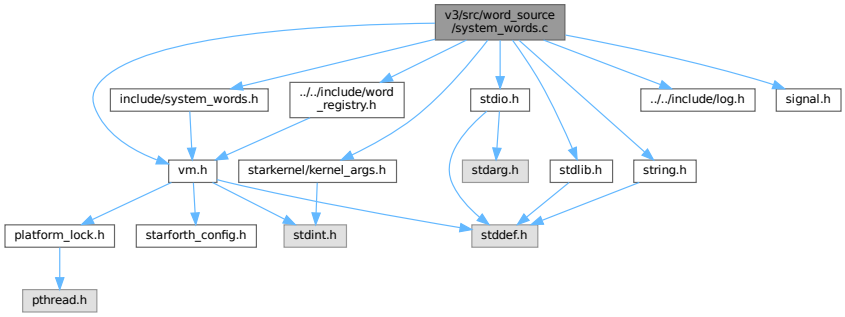
```
vaddr_t word_scratch_addr = 0  [static]
```

Referenced by string_word_word().

7.416 v3/src/word_source/system_words.c File Reference

```
#include "include/system_words.h"
#include "../../include/vm.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include "starkernel/kernel_args.h"
```

Include dependency graph for system_words.c:



Functions

- static void print_word_name (const DictEntry *entry)
- static void reset_vm_state (VM *vm, int cold_start)
- static void forth_paren (VM *vm)
- static void forth_backslash (VM *vm)
- void system_word_cold (VM *vm)
- void system_word_warm (VM *vm)
- void system_word_bye (VM *vm)
- void system_word_save_system (VM *vm)
- void system_word_words (VM *vm)
- void system_word_vlist (VM *vm)
- void system_word_page (VM *vm)
- void system_word_nop (VM *vm)
- void system_word_79_standard (VM *vm)
- static void system_word_quit (VM *vm)
- static void system_word_abort (VM *vm)
- static void runtime_abortq (VM *vm)
- static void system_word_abort_quote (VM *vm)
- static void system_word_see (VM *vm)
- static void system_word_reboot (VM *vm)
- static void system_word_execute (VM *vm)
- int system_is_running (void)
- void set_forth_79_compliance (int enabled)
- void register_system_words (VM *vm)

  Registers all system words with the virtual machine.

Variables

- static int system_running = 1

  Flag indicating if the Forth system is currently running.
- static int forth_79_standard = 1

  Flag controlling FORTH-79 standard compliance mode.

7.416.1 Function Documentation

7.416.1.1 forth_backslash()

```
static void forth_backslash (
            VM * vm )  [static]
```

References VM::input_buffer, VM::input_length, and VM::input_pos.

Referenced by register_system_words().

Here is the caller graph for this function:

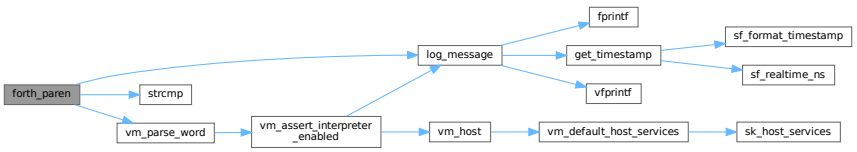
7.416.1.2 forth_paren()

```
static void forth_paren (
            VM * vm )  [static]
```

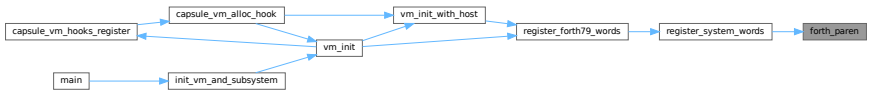
References log_message(), LOG_WARN, strcmp(), and vm_parse_word().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:



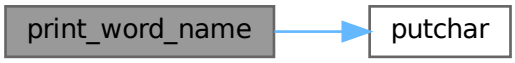
7.416.1.3 print_word_name()

```
static void print_word_name (
            const DictEntry * entry )  [static]
```

References entry, and putchar().

Referenced by system_word_see(), system_word_vlist(), and system_word_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.416.1.4 register_system_words()

```
void register_system_words (
            VM * vm )
```

Registers all system words with the virtual machine.

Parameters

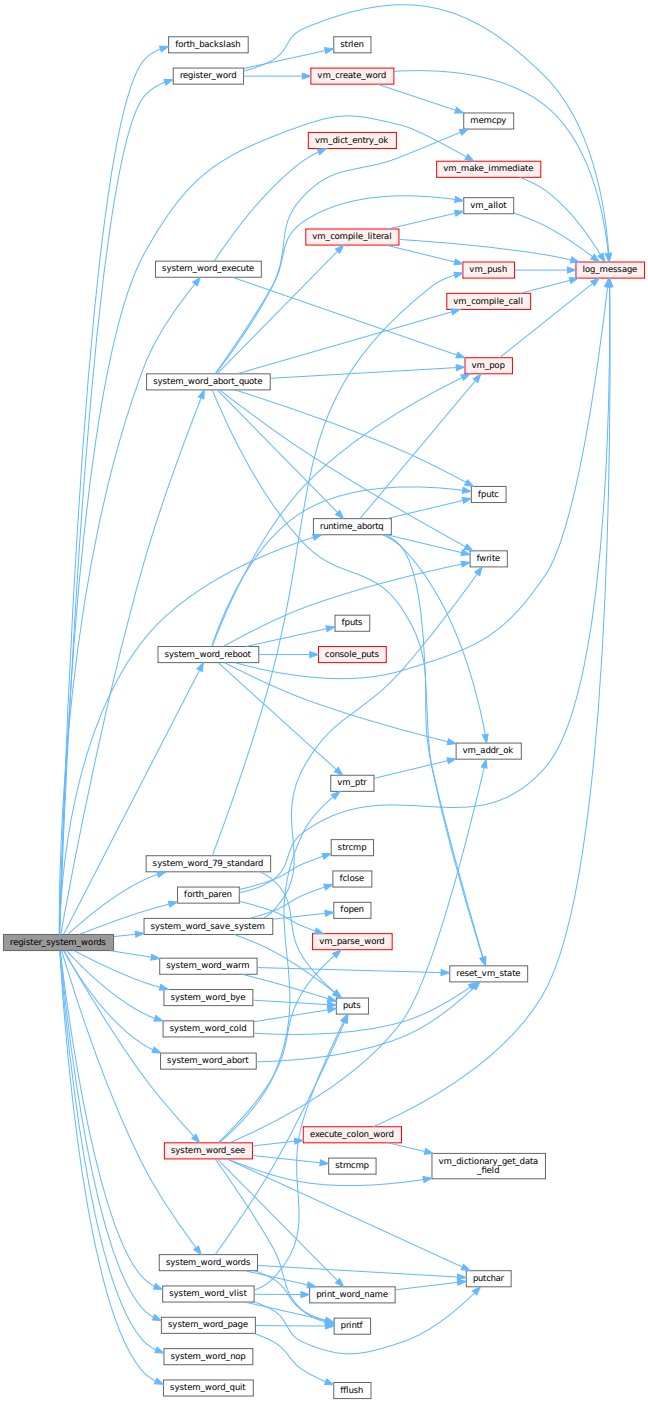
vm	Pointer to the virtual machine instance

This function registers all FORTH-79 system and environment words with the specified virtual machine instance, making them available for use in the FORTH environment.

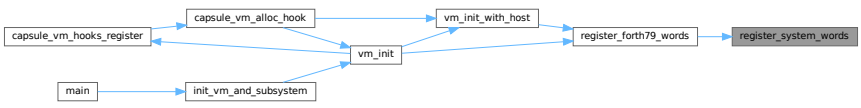
References forth_backslash(), forth_paren(), register_word(), runtime_abortq(), system_word_79_standard(), system_word_abort(), system_word_abort_quote(), system_word_bye(), system_word_cold(), system_word_execute(), system_word_nop(), system_word_page(), system_word_quit(), system_word_reboot(), system_word_save_system(), system_word_see(), system_word_vlist(), system_word_warm(), system_word_words(), and vm_make_immediate().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.416.1.5 reset_vm_state()

```
static void reset_vm_state (
            VM * vm,
            int cold_start )  [static]
```

References VM::dsp, VM::error, VM::here, VM::mode, MODE_INTERPRET, VM::rsp, and system_running.

Referenced by runtime_abortq(), system_word_abort(), system_word_abort_quote(), system_word_cold(), and system_word_warm().

Here is the caller graph for this function:

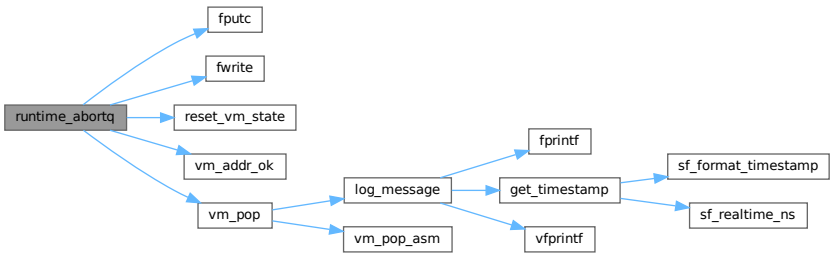
7.416.1.6 runtime_abortq()

```
static void runtime_abortq (
            VM * vm )  [static]
```

References VM::dsp, VM::error, fputc(), fwrite(), VM::memory, reset_vm_state(), stdout, vm_addr_ok(), and vm_pop().

Referenced by register_system_words(), and system_word_abort_quote().

Here is the call graph for this function:



Here is the caller graph for this function:

7.416.1.7 set_forth_79_compliance()

```
void set_forth_79_compliance (
            int enabled )
```

References enabled, and forth_79_standard.

7.416.1.8 system_is_running()

```
int system_is_running (
            void  )
```

References system_running.

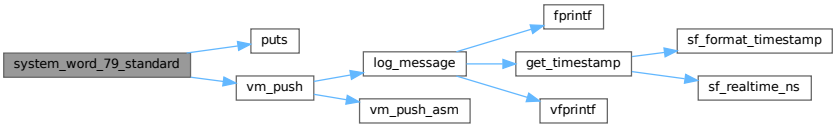
7.416.1.9 system_word_79_standard()

```
void system_word_79_standard (
            VM * vm )
```

References forth_79_standard, puts(), and vm_push().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.416.1.10 system_word_abort()

```
static void system_word_abort (
            VM * vm )  [static]
```

References VM::abort_requested, VM::error, and reset_vm_state().

Referenced by register_system_words().

Here is the call graph for this function:

Here is the caller graph for this function:

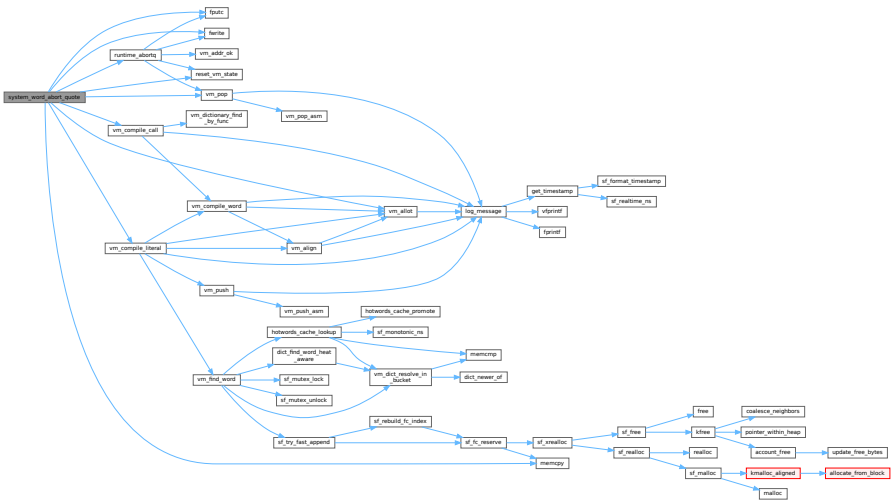
7.416.1.11 system_word_abort_quote()

```
static void system_word_abort_quote (
            VM * vm )  [static]
```

References VM::dsp, VM::error, fputc(), fwrite(), VM::input_buffer, VM::input_length, VM::input_pos, memcpy(), VM::memory, VM::mode, MODE_INTERPRET, reset_vm_state(), runtime_abortq(), stdout, vm_allot(), vm_compile_call(), vm_compile_literal(), and vm_pop().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.416.1.12 system_word_bye()

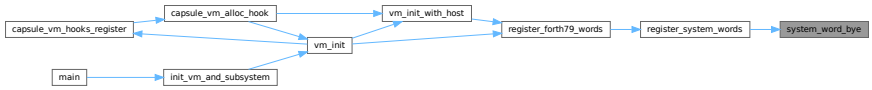
```
void system_word_bye (
            VM * vm )
```

References VM::halted, and puts().

Referenced by register_system_words().

Here is the call graph for this function:

Here is the caller graph for this function:



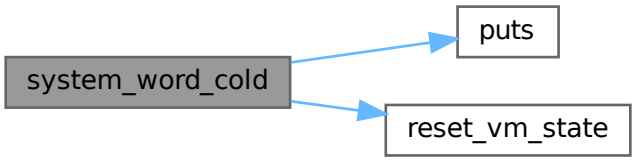
7.416.1.13 system_word_cold()

```
void system_word_cold (
            VM * vm )
```

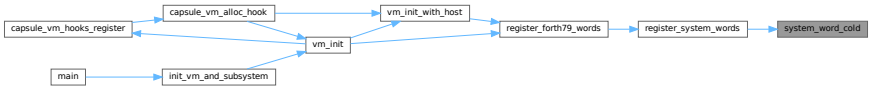
References puts(), and reset_vm_state().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:



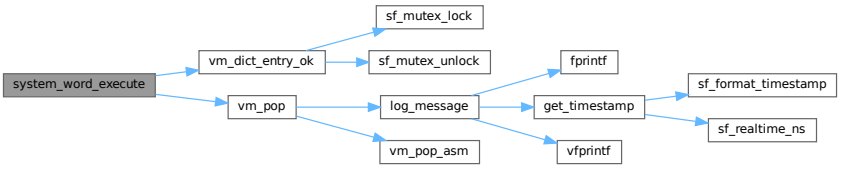
7.416.1.14 system_word_execute()

```
static void system_word_execute (
            VM * vm )  [static]
```

References VM::current_executing_entry, VM::dsp, entry, VM::error, DictEntry::func, vm_dict_entry_ok(), and vm_pop().

Referenced by register_system_words().

Here is the call graph for this function:



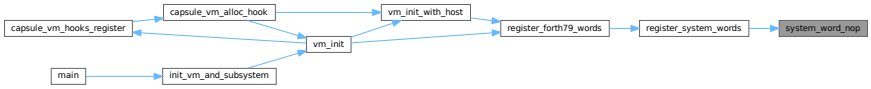
Here is the caller graph for this function:

7.416.1.15 system_word_nop()

```
void system_word_nop (
            VM * vm )
```

Referenced by register_system_words().

Here is the caller graph for this function:



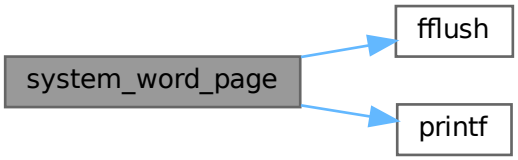
7.416.1.16 system_word_page()

```
void system_word_page (
            VM * vm )
```

References fflush(), printf(), and stdout.

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:

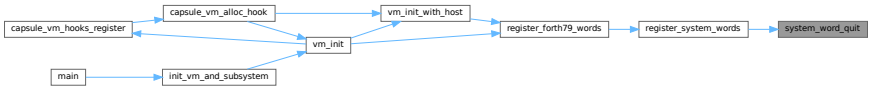
7.416.1.17 system_word_quit()

```
static void system_word_quit (
            VM * vm )  [static]
```

References VM::error, VM::mode, MODE_COMPILE, MODE_INTERPRET, and VM::rsp.

Referenced by register_system_words().

Here is the caller graph for this function:



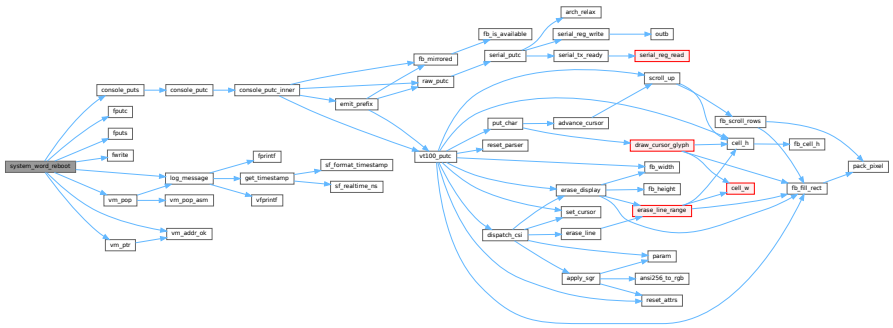
7.416.1.18 system_word_reboot()

```
static void system_word_reboot (
            VM * vm )  [static]
```

References console_puts(), VM::dsp, EFI_SUCCESS, EFI_VARIABLE_BOOTSERVICE_ACCESS, EFI_VARIABLE_NON_VOLATILE EFI_VARIABLE_RUNTIME_ACCESS, EfiResetCold, VM::error, fputc(), fputs(), fwrite(), g_sk_runtime_services, _EFI_RUNTIME_SERVICES::GetVariable, VM::halted, KERNEL_ARGS_CMDLINE_MAX, LOG_ERROR, log_message(), VM::mode, MODE_COMPILE, REBOOT_MAX_TRIES, _EFI_RUNTIME_SERVICES::ResetSystem, _EFI_RUNTIME_SERVICES::SetVariable, SF_VAR_BOOT_ARGS, SF_VAR_REBOOT_TRIES, STARFORTH_VENDOR_GUID, stdout, vm_addr_ok(), vm_pop(), and vm_ptr().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:

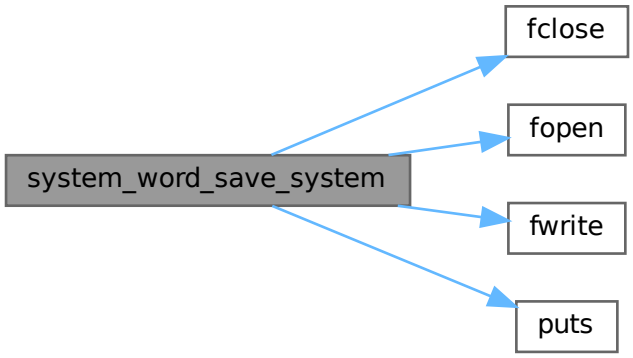
7.416.1.19 system_word_save_system()

```
void system_word_save_system (
            VM * vm )
```

References VM::error, fclose(), fopen(), fwrite(), VM::here, VM::memory, and puts().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:

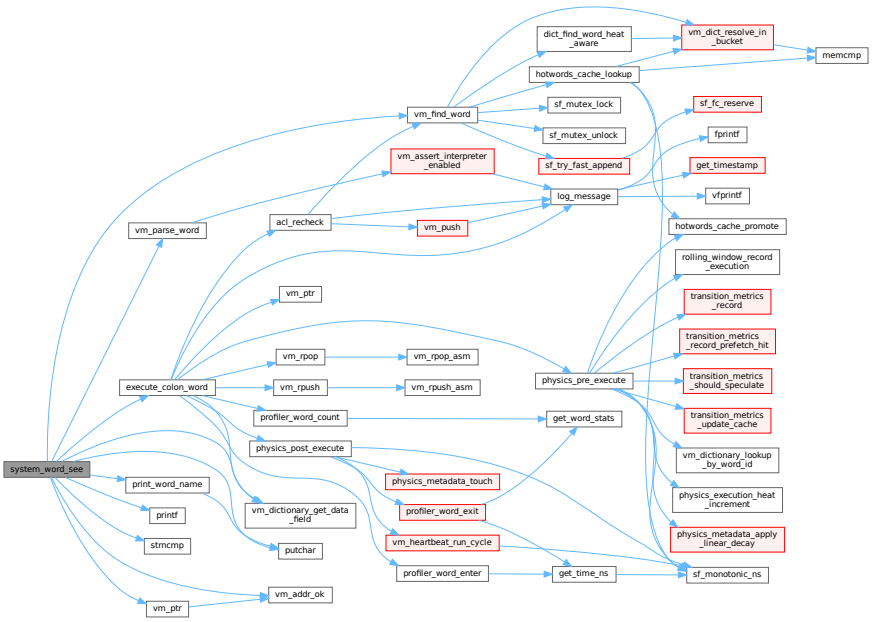
7.416.1.20 system_word_see()

```
static void system_word_see (
            VM * vm )  [static]
```

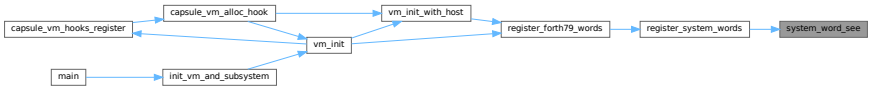
References entry, VM::error, execute_colon_word(), VM::memory, DictEntry::name, DictEntry::name_len, print_word_name(), printf(), putchar(), strncmp(), vm_addr_ok(), vm_dictionary_get_data_field(), vm_find_word(), vm_parse_word(), vm_ptr(), word_count, and WORD_NAME_MAX.

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:



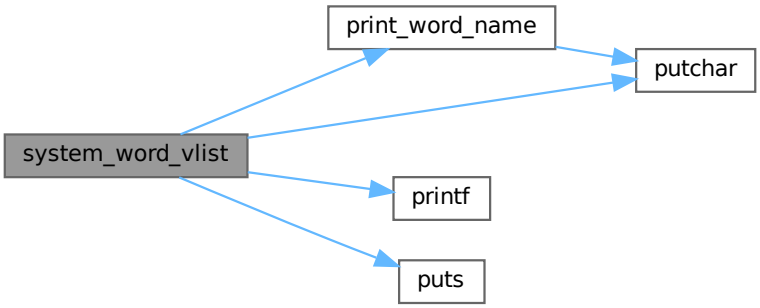
7.416.1.21 system_word_vlist()

```
void system_word_vlist (
            VM * vm )
```

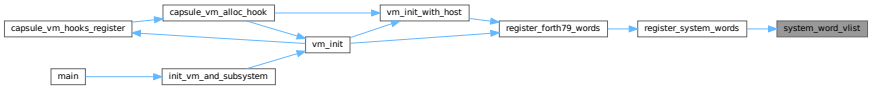
References count, DictEntry::flags, VM::latest, DictEntry::name_len, print_word_name(), printf(), putchar(), and puts().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:



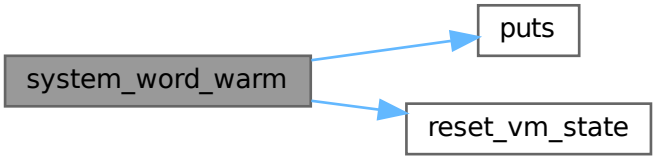
7.416.1.22 system_word_warm()

```
void system_word_warm (
            VM * vm )
```

References puts(), and reset_vm_state().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:

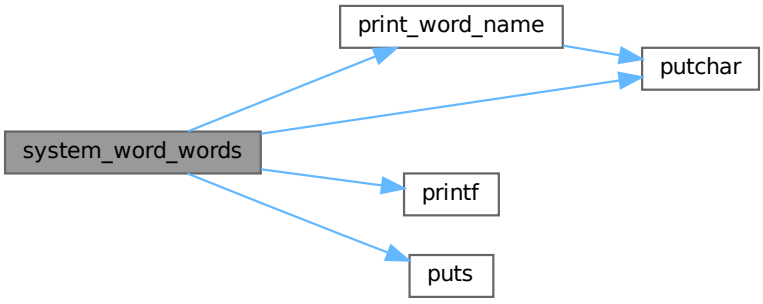
7.416.1.23 system_word_words()

```
void system_word_words (
            VM * vm )
```

References count, VM::latest, DictEntry::link, print_word_name(), printf(), putchar(), and puts().

Referenced by register_system_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.416.2 Variable Documentation

7.416.2.1 forth_79_standard

```
int forth_79_standard = 1  [static]
```

Flag controlling FORTH-79 standard compliance mode.

Referenced by set_forth_79_compliance(), and system_word_79_standard().

7.416.2.2 system_running

```
int system_running = 1  [static]
```

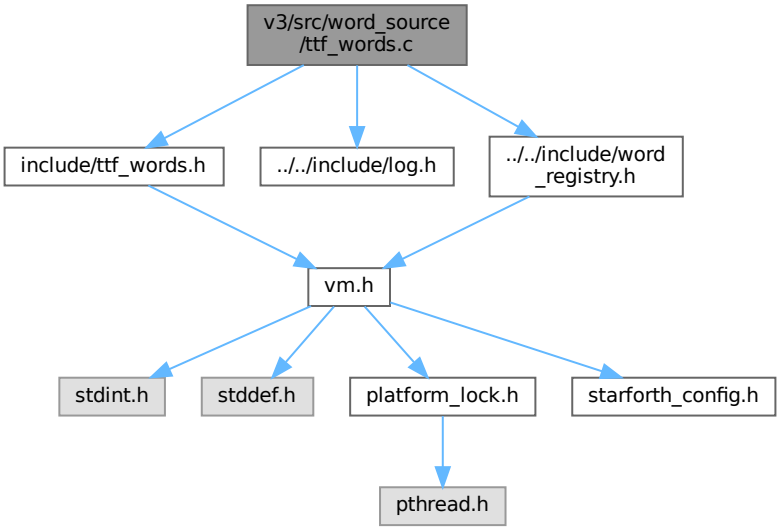
Flag indicating if the Forth system is currently running.

Referenced by reset_vm_state(), and system_is_running().

7.417 v3/src/word_source/ttf_words.c File Reference

```
#include "include/ttf_words.h"
#include "../../include/log.h"
#include "../../include/word_registry.h"
```

Include dependency graph for ttf_words.c:



Functions

- void register_ttf_words (VM *vm)

7.417.1 Function Documentation

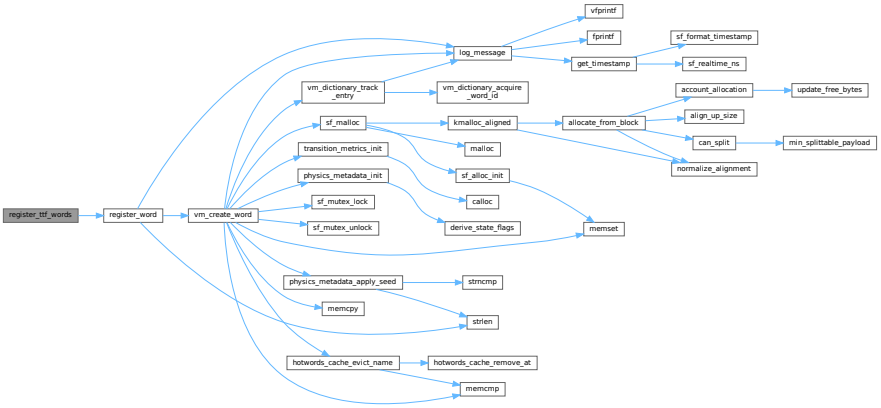
7.417.1.1 register_ttf_words()

```
void register_ttf_words (
            VM * vm )
```

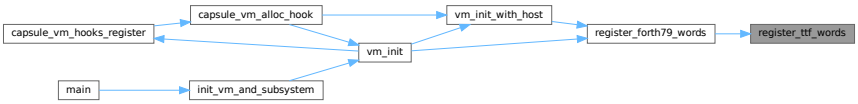
References register_word().

Referenced by register_forth79_words().

Here is the call graph for this function:



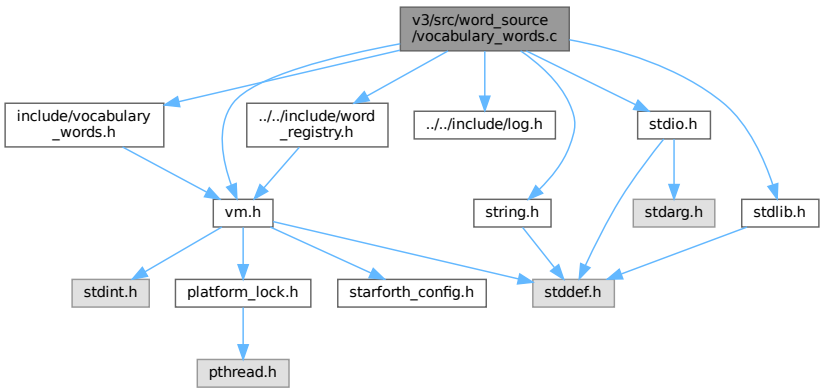
Here is the caller graph for this function:



7.418 v3/src/word_source/vocabulary_words.c File Reference

```
#include "include/vocabulary_words.h"
#include "../../include/word_registry.h"
#include "../../include/log.h"
#include "../../include/vm.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for vocabulary_words.c:



Functions

- static void vocabulary_debug_print_hex (uint64_t value) __attribute__((unused))
- static size_t vocab_safe_len (const char *text, size_t max_len)
- static void fc_free (DictEntry ***lists, size_t *counts)
- static void fc_rebuild (DictEntry *head, DictEntry ***lists, size_t *counts, DictEntry **cached_head)
- static void vocab_sync_vm_vars (VM *vm)
- static void init_vocabulary_system (VM *vm)

  Initialize the FORTH vocabulary system.
- static DictEntry * vocab_find_word (VM *vm, const char *name, size_t len)
- static void vocabulary_select_runtime (VM *vm)
- void vocabulary_word_vocabulary (VM *vm)

  VOCABULARY (–) Create a new vocabulary.
- void vocabulary_create_vocabulary_direct (VM *vm, const char *name)

  Direct vocabulary creation (kernel bootstrap)
- void vocabulary_word_definitions (VM *vm)

  DEFINITIONS (–) Set CURRENT to CONTEXT.
- void vocabulary_word_context (VM *vm)

  CONTEXT (– addr) Get CONTEXT variable address.
- void vocabulary_word_current (VM *vm)

  CURRENT (– addr) Get CURRENT variable address.
- void vocabulary_word_forth (VM *vm)

  FORTH (–) Select FORTH vocabulary.
- void vocabulary_word_paren_find (VM *vm)

  (FIND) (addr – addr flag) Find word in dictionary
- void vocabulary_word_order (VM *vm)

  ORDER (–) Display search order.
- DictEntry * vm_vocabulary_find_word (VM *vm, const char *name, size_t len)

  Find word in vocabulary-aware dictionary.
- void register_vocabulary_words (VM *vm)

  Register all FORTH-79 vocabulary words.

7.418.1 Function Documentation

7.418.1.1 fc_free()

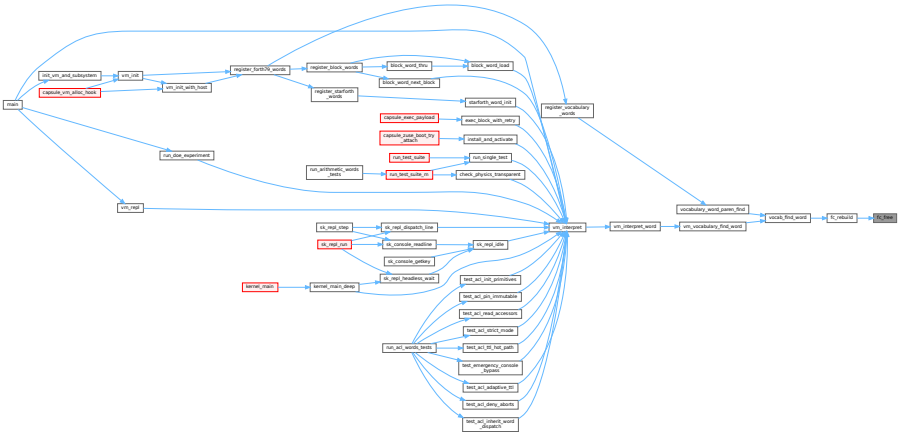
```
static void fc_free (
            DictEntry *** lists,
            size_t * counts )  [static]
```

References free(), and VOCAB_FC_BUCKETS.

Referenced by fc_rebuild().

Here is the call graph for this function:

Here is the caller graph for this function:



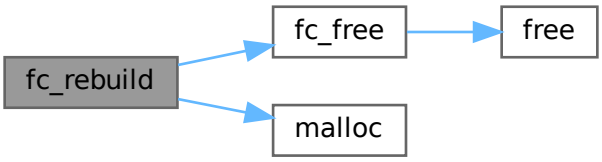
7.418.1.2 fc_rebuild()

```
static void fc_rebuild (
            DictEntry * head,
            DictEntry *** lists,
            size_t * counts,
            DictEntry ** cached_head )  [static]
```

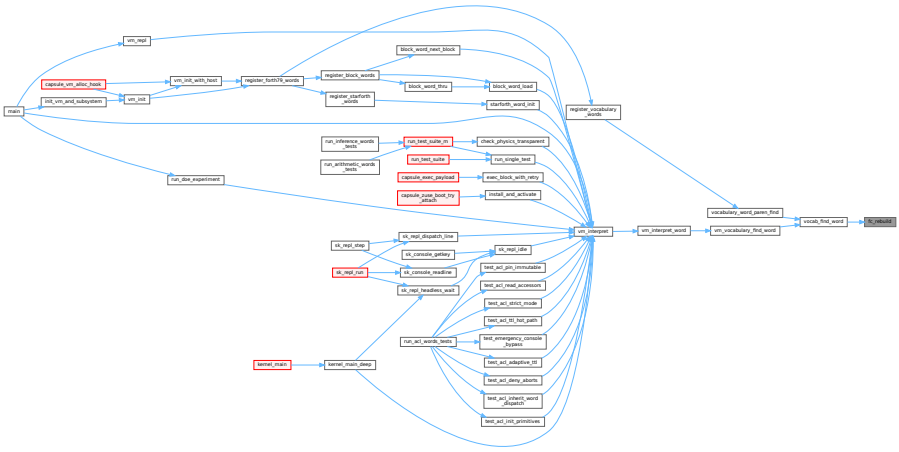
References fc_free(), head, malloc(), DictEntry::name, and VOCAB_FC_BUCKETS.

Referenced by vocab_find_word().

Here is the call graph for this function:



Here is the caller graph for this function:



7.418.1.3 init_vocabulary_system()

```
static void init_vocabulary_system (
            VM * vm )  [static]
```

Initialize the FORTH vocabulary system.

Sets up FORTH as root vocabulary and allocates VM cells for CONTEXT/CURRENT. Per-VM (vm->vocab.↩ initialized): each VM in the fleet seeds its own vocabulary roots from its own dictionary, rather than every VM after the first silently sharing whichever VM ran this first.

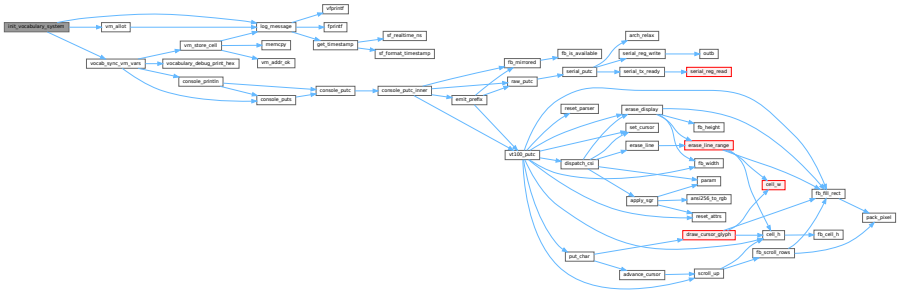
Parameters

vm	Pointer to VM instance

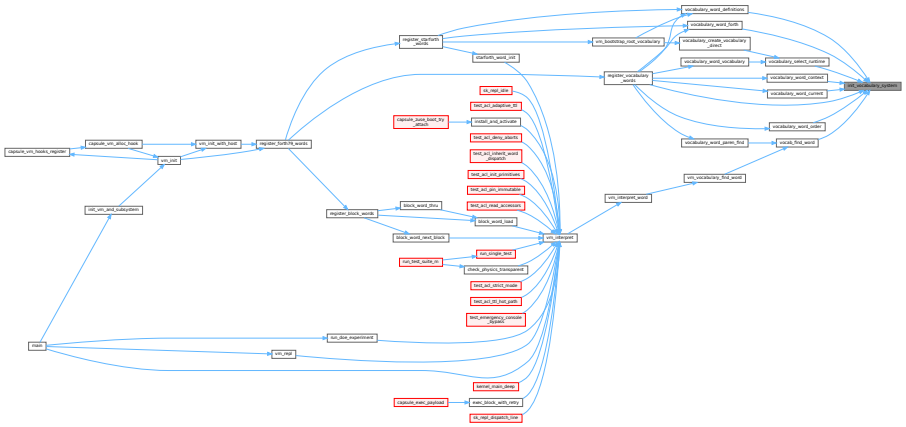
References VocabularyState::context_var_addr, VocabularyState::context_vocab, VocabularyState::current_var_addr, VocabularyState::current_vocab, VM::error, VocabularyState::forth_vocab, VocabularyState::initialized, VM::latest, LOG_ERROR, log_message(), VM::memory, vm_allot(), VM::vocab, and vocab_sync_vm_vars().

Referenced by register_vocabulary_words(), vocab_find_word(), vocabulary_select_runtime(), vocabulary_word_context(), vocabulary_word_current(), vocabulary_word_definitions(), vocabulary_word_forth(), and vocabulary_word_order().

Here is the call graph for this function:



Here is the caller graph for this function:



7.418.1.4 register_vocabulary_words()

```
void register_vocabulary_words (
            VM * vm )
```

Register all FORTH-79 vocabulary words.

Registers VOCABULARY, DEFINITIONS, CONTEXT, CURRENT, FORTH, ORDER, and (FIND)

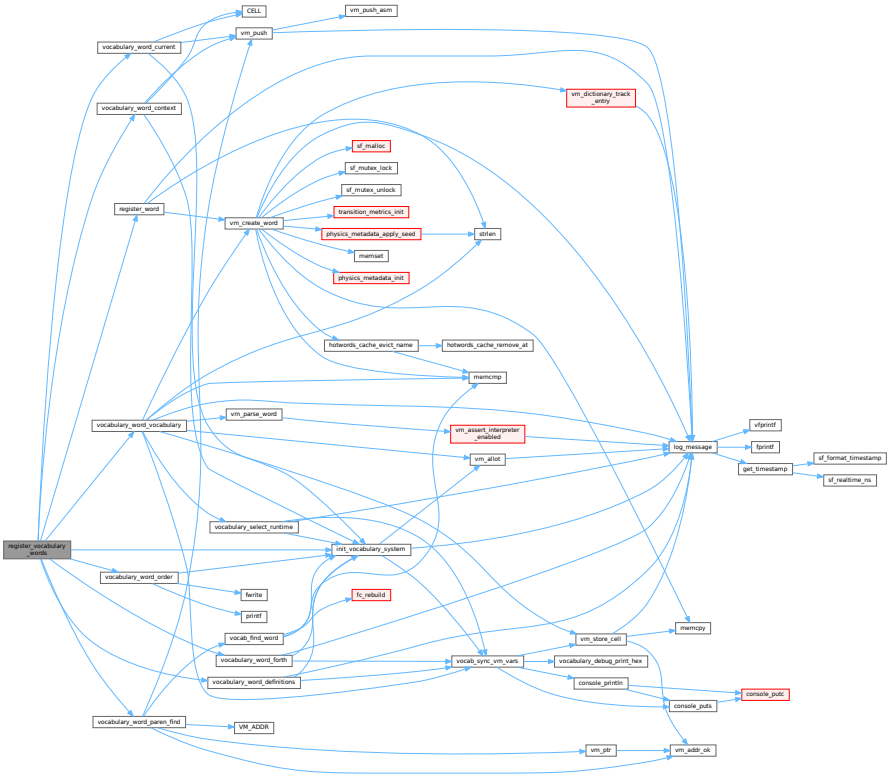
Parameters

vm	Pointer to VM instance

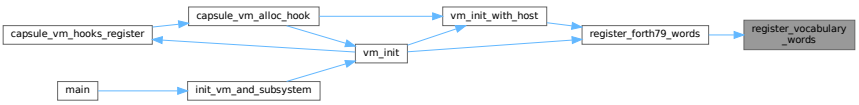
References init_vocabulary_system(), register_word(), vocabulary_word_context(), vocabulary_word_current(), vocabulary_word_definitions(), vocabulary_word_forth(), vocabulary_word_order(), vocabulary_word_paren_find(), and vocabulary_word_vocabulary().

Referenced by register_forth79_words().

Here is the call graph for this function:



Here is the caller graph for this function:



7.418.1.5 vm_vocabulary_find_word()

```
DictEntry * vm_vocabulary_find_word (
            VM * vm,
            const char * name,
            size_t len )
```

Find word in vocabulary-aware dictionary.

Searches CONTEXT then FORTH vocabularies

Parameters

vm	Pointer to VM instance
name	Word name to find
len	Length of word name

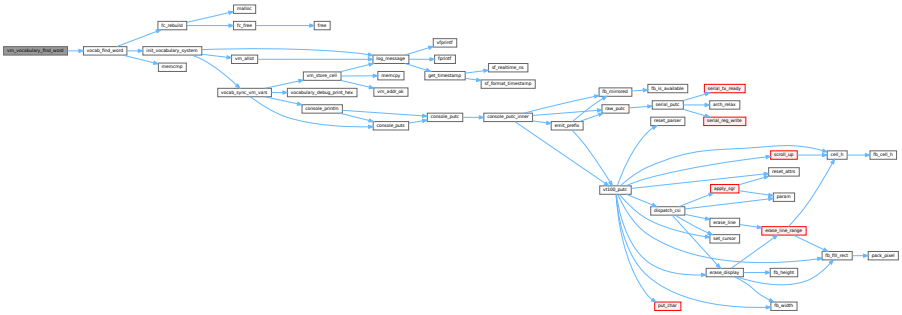
Returns

Pointer to dictionary entry if found, NULL if not found

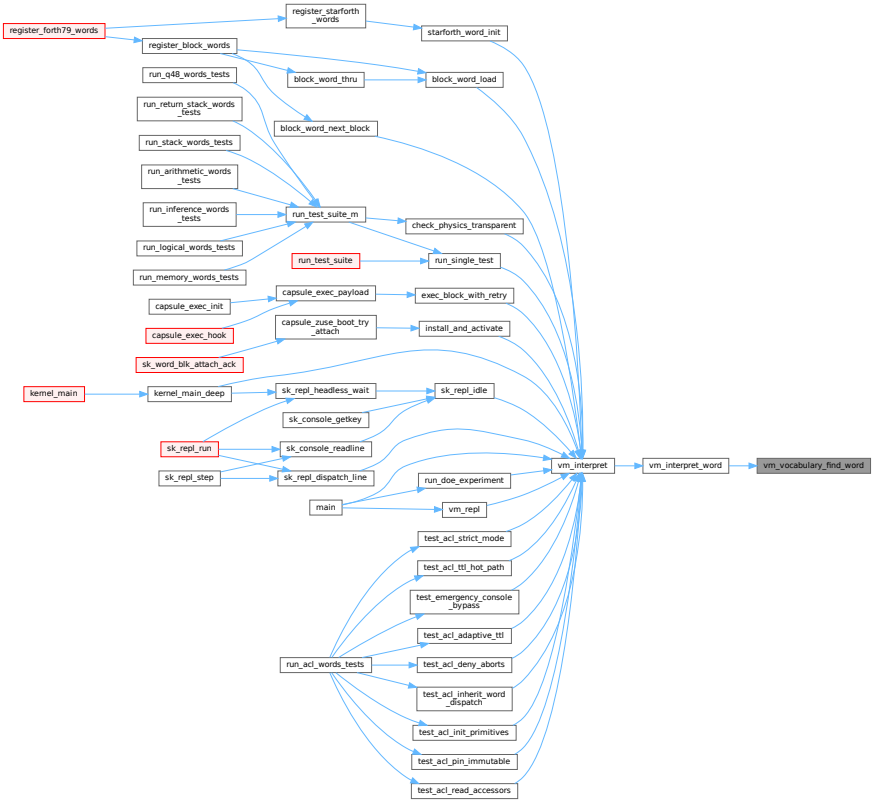
References name, and vocab_find_word().

Referenced by vm_interpret_word().

Here is the call graph for this function:



Here is the caller graph for this function:



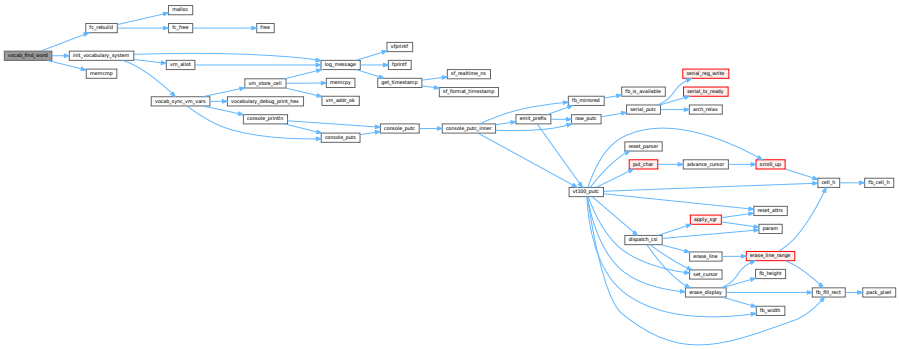
7.418.1.6 vocab_find_word()

```
static DictEntry * vocab_find_word (
            VM * vm,
            const char * name,
            size_t len )  [static]
```

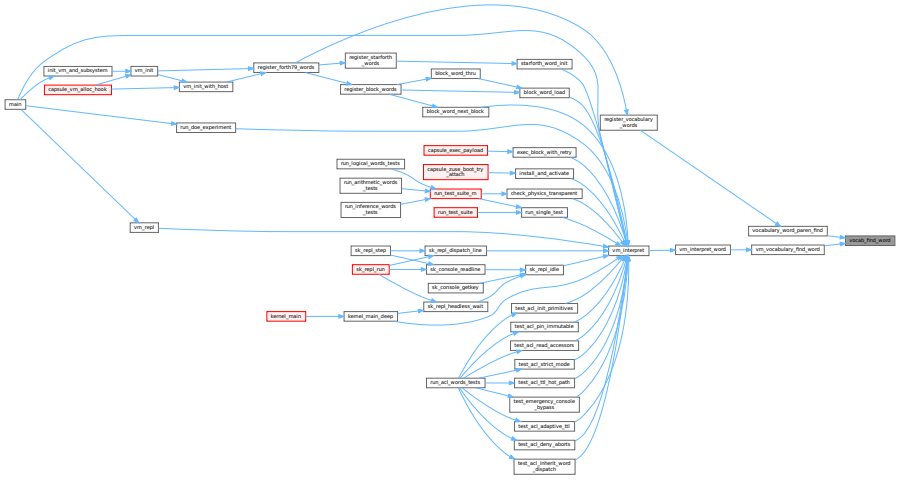
References VocabularyState::context_vocab, VocabularyState::ctx_cached_head, VocabularyState::ctx_fc, VocabularyState::ctx_n, fc_rebuild(), DictEntry::flags, VocabularyState::forth_cached_head, VocabularyState::forth_fc, VocabularyState::forth_n, VocabularyState::forth_vocab, init_vocabulary_system(), memcmp(), name, DictEntry::name, DictEntry::name_len, UNLIKELY, VM::vocab, WORD_HIDDEN, and WORD_SMUDGED.

Referenced by vm_vocabulary_find_word(), and vocabulary_word_paren_find().

Here is the call graph for this function:



Here is the caller graph for this function:

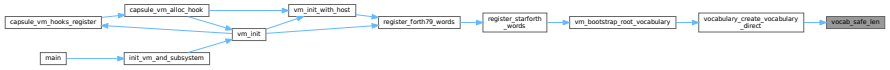


7.418.1.7 vocab_safe_len()

```
static size_t vocab_safe_len (
            const char * text,
            size_t max_len )  [static]
```

Referenced by vocabulary_create_vocabulary_direct().

Here is the caller graph for this function:



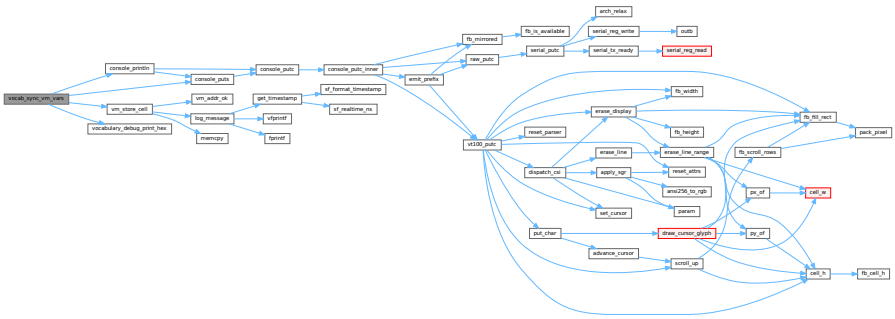
7.418.1.8 vocab_sync_vm_vars()

```
static void vocab_sync_vm_vars (
            VM * vm )  [inline], [static]
```

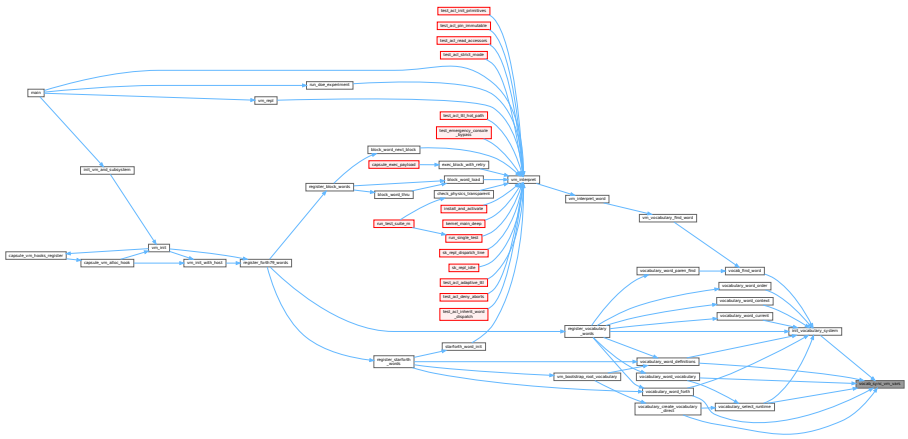
References base, console_println(), console_puts(), VocabularyState::context_var_addr, VocabularyState::context_vocab, VocabularyState::current_var_addr, VocabularyState::current_vocab, VM::memory, vm_store_cell(), VM::vocab, and vocabulary_debug_print_hex().

Referenced by init_vocabulary_system(), vocabulary_create_vocabulary_direct(), vocabulary_select_runtime(), vocabulary_word_definitions(), vocabulary_word_forth(), and vocabulary_word_vocabulary().

Here is the call graph for this function:



Here is the caller graph for this function:



7.418.1.9 vocabulary_create_vocabulary_direct()

```
void vocabulary_create_vocabulary_direct (
            VM * vm,
            const char * name )
```

Direct vocabulary creation (kernel bootstrap)

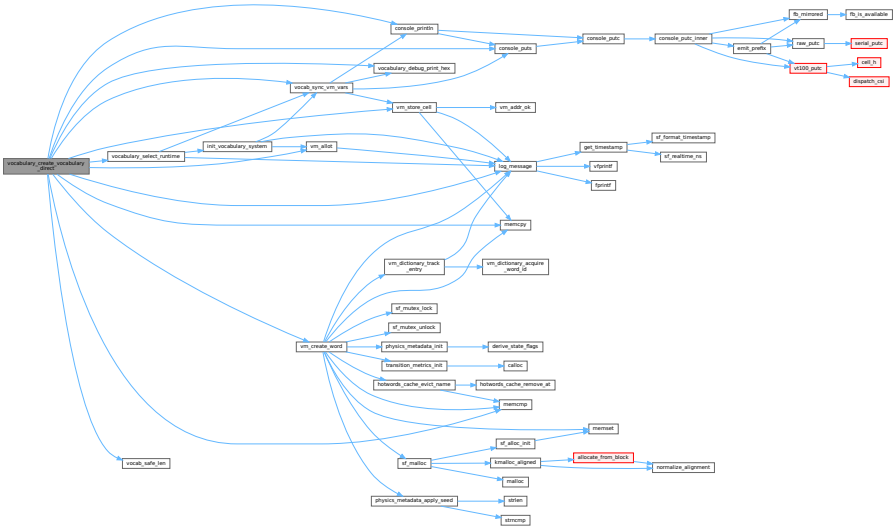
Parameters

vm	Pointer to the virtual machine instance
name	Vocabulary name

References console_println(), console_puts(), entry, VM::error, VM::here, VM::latest, LOG_DEBUG, LOG_ERROR, log_message(), memcmp(), memcpy(), VM::memory, name, vm_allot(), vm_create_word(), vm_store_cell(), vocab_safe_len(), vocab_sync_vm_vars(), vocabulary_debug_print_hex(), and vocabulary_select_runtime().

Referenced by vm_bootstrap_root_vocabulary().

Here is the call graph for this function:



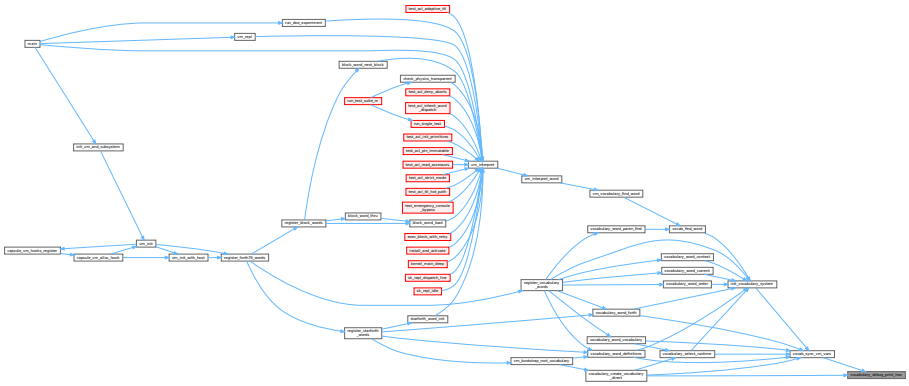
Here is the caller graph for this function:

7.418.1.10 vocabulary_debug_print_hex()

```
static void vocabulary_debug_print_hex (
            uint64_t value ) [static]
```

Referenced by vocab_sync_vm_vars(), and vocabulary_create_vocabulary_direct().

Here is the caller graph for this function:



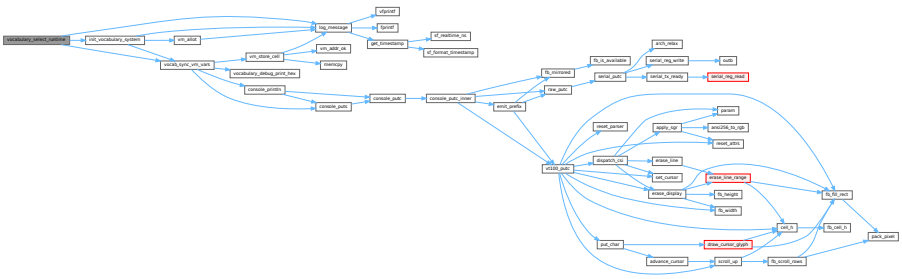
7.418.1.11 vocabulary_select_runtime()

```
static void vocabulary_select_runtime (
            VM * vm )  [static]
```

References VocabularyState::context_vocab, VM::current_executing_entry, init_vocabulary_system(), VM::latest, LOG_DEBUG, log_message(), VM::vocab, and vocab_sync_vm_vars().

Referenced by vocabulary_create_vocabulary_direct(), and vocabulary_word_vocabulary().

Here is the call graph for this function:



Here is the caller graph for this function:

7.418.1.12 vocabulary_word_context()

```
void vocabulary_word_context (
            VM * vm )
```

CONTEXT (– addr) Get CONTEXT variable address.

Returns VM address of cell containing CONTEXT vocabulary pointer

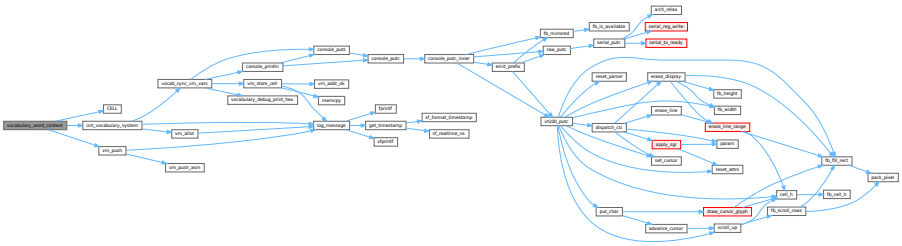
Parameters

vm	Pointer to VM instance

References CELL(), VocabularyState::context_var_addr, init_vocabulary_system(), vm_push(), and VM::vocab.

Referenced by register_vocabulary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.418.1.13 vocabulary_word_current()

```
void vocabulary_word_current (
            VM * vm )
```

CURRENT (– addr) Get CURRENT variable address.

Returns VM address of cell containing CURRENT vocabulary pointer

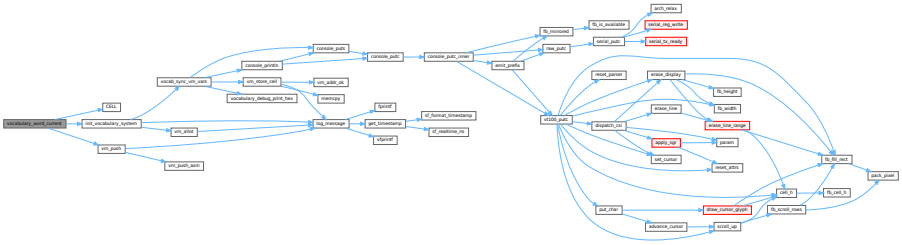
Parameters

vm	Pointer to VM instance

References CELL(), VocabularyState::current_var_addr, init_vocabulary_system(), vm_push(), and VM::vocab.

Referenced by register_vocabulary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.418.1.14 vocabulary_word_definitions()

```
void vocabulary_word_definitions (
            VM * vm )
```

DEFINITIONS (–) Set CURRENT to CONTEXT.

Makes new words be defined in the CONTEXT vocabulary

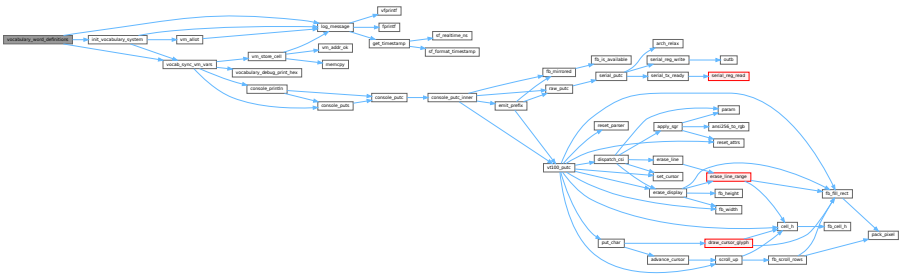
Parameters

vm	Pointer to VM instance

References VocabularyState::context_vocab, VocabularyState::current_vocab, init_vocabulary_system(), LOG_DEBUG, log_message(), VM::vocab, and vocab_sync_vm_vars().

Referenced by register_starforth_words(), register_vocabulary_words(), and vm_bootstrap_root_vocabulary().

Here is the call graph for this function:



Here is the caller graph for this function:

7.418.1.15 vocabulary_word_forth()

```
void vocabulary_word_forth (
            VM * vm )
```

FORTH (–) Select FORTH vocabulary.

Makes FORTH the CONTEXT vocabulary

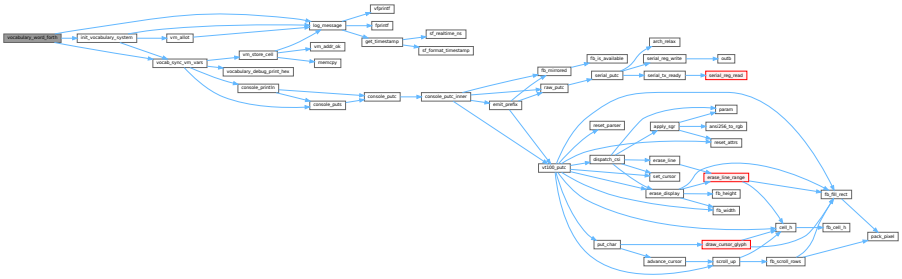
Parameters

vm	Pointer to VM instance

References VocabularyState::context_vocab, VocabularyState::forth_vocab, init_vocabulary_system(), LOG_DEBUG, log_message(), VM::vocab, and vocab_sync_vm_vars().

Referenced by register_starforth_words(), and register_vocabulary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.418.1.16 vocabulary_word_order()

```
void vocabulary_word_order (
            VM * vm )
```

ORDER (–) Display search order.

Shows CONTEXT and FORTH vocabularies plus CURRENT

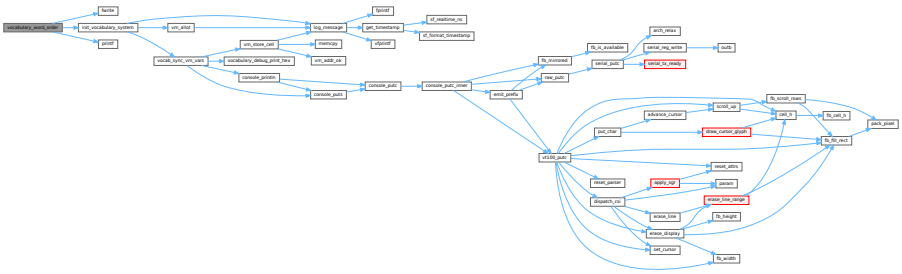
Parameters

vm	Pointer to VM instance

References VocabularyState::context_vocab, VocabularyState::current_vocab, VocabularyState::forth_vocab, fwrite(), init_vocabulary_system(), DictEntry::name, DictEntry::name_len, printf(), stdout, and VM::vocab.

Referenced by register_vocabulary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.418.1.17 vocabulary_word_paren_find()

```
void vocabulary_word_paren_find (
            VM * vm )
```

(FIND) (addr – addr flag) Find word in dictionary

Searches for counted string at addr in CONTEXT then FORTH vocabularies

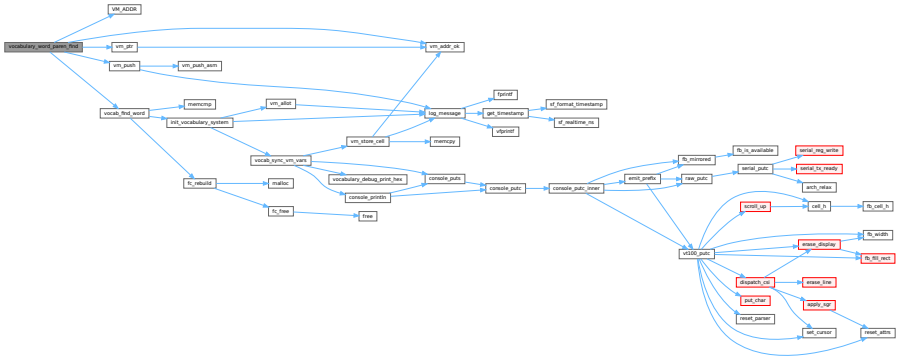
Parameters

vm	Pointer to VM instance

References VM::data_stack, VM::dsp, VM::error, DictEntry::flags, name, VM_ADDR(), vm_addr_ok(), vm_ptr(), vm_push(), vocab_find_word(), and WORD_IMMEDIATE.

Referenced by register_vocabulary_words().

Here is the call graph for this function:



Here is the caller graph for this function:

7.418.1.18 vocabulary_word_vocabulary()

```
void vocabulary_word_vocabulary (
            VM * vm )
```

VOCABULARY (–) Create a new vocabulary.

Creates a vocabulary that when executed makes itself the CONTEXT

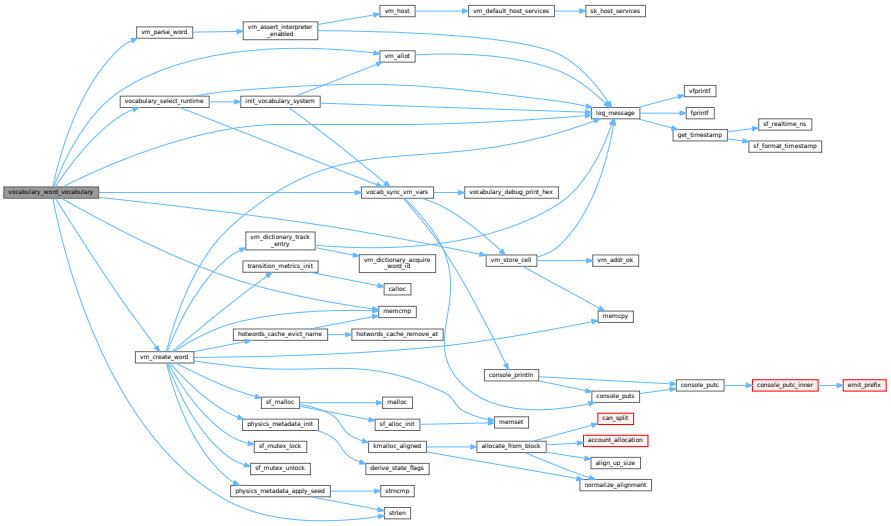
Parameters

vm	Pointer to VM instance

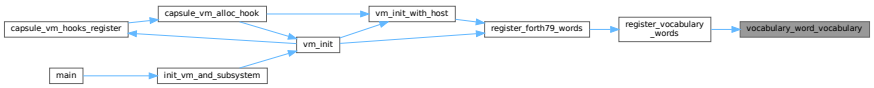
References entry, VM::error, VM::latest, LOG_DEBUG, LOG_ERROR, log_message(), memcmp(), VM::memory, name, strlen(), vm_allot(), vm_create_word(), vm_parse_word(), vm_store_cell(), vocab_sync_vm_vars(), and vocabulary_select_runtime().

Referenced by register_vocabulary_words().

Here is the call graph for this function:



Here is the caller graph for this function:



Index

_EFI_BLOCK_IO_PROTOCOL, 84
  FlushBlocks, 84
  Media, 84
  ReadBlocks, 84
  Reset, 84
  Revision, 84
  WriteBlocks, 85
_EFI_BOOT_SERVICES, 85
  AllocatePages, 86
  AllocatePool, 86
  CheckEvent, 86
  CloseEvent, 86
  CloseProtocol, 86
  ConnectController, 86
  CreateEvent, 87
  DisconnectController, 87
  Exit, 87
  ExitBootServices, 87
  FreePages, 87
  FreePool, 87
  GetMemoryMap, 87
  GetNextMonotonicCount, 87
  HandleProtocol, 87
  Hdr, 88
  InstallConfigurationTable, 88
  InstallProtocolInterface, 88
  LoadImage, 88
  LocateDevicePath, 88
  LocateHandle, 88
  LocateHandleBuffer, 88
  LocateProtocol, 88
  OpenProtocol, 88
  OpenProtocolInformation, 89
  ProtocolsPerHandle, 89
  RaiseTPL, 89
  RegisterProtocolNotify, 89
  ReinstallProtocolInterface, 89
  Reserved, 89
  RestoreTPL, 89
  SetTimer, 89
  SetWatchdogTimer, 89
  SignalEvent, 89
  Stall, 90
  StartImage, 90
  UninstallProtocolInterface, 90
  UnloadImage, 90
  WaitForEvent, 90
_EFI_FILE_PROTOCOL, 90
  Close, 91
  Delete, 91
  Flush, 91
  GetInfo, 91
  GetPosition, 91
  Open, 91
  Read, 91
  Revision, 92
  SetInfo, 92
  SetPosition, 92
  Write, 92
_EFI_GRAPHICS_OUTPUT_PROTOCOL, 92
  Blt, 93
  Mode, 93
  QueryMode, 93
  SetMode, 94
_EFI_RUNTIME_SERVICES, 94
  ConvertPointer, 95
  GetNextHighMonotonicCount, 95
  GetNextVariableName, 95
  GetTime, 95
  GetVariable, 95
  GetWakeupTime, 95
  Hdr, 95
  ResetSystem, 95
  SetTime, 95
  SetVariable, 95
  SetVirtualAddressMap, 96
  SetWakeupTime, 96
_EFI_SIMPLE_FILE_SYSTEM_PROTOCOL, 96
  OpenVolume, 96
  Revision, 96
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 97
  ClearScreen, 97
  EnableCursor, 97
  Mode, 97
  OutputString, 98
  QueryMode, 98
  Reset, 98
  SetAttribute, 98
  SetCursorPosition, 98
  SetMode, 98
  TestString, 98
_EFI_SYSTEM_TABLE, 99
  BootServices, 99
  ConfigurationTable, 99
  ConIn, 99
  ConOut, 99
  ConsoleInHandle, 100
  ConsoleOutHandle, 100

FirmwareRevision, 100
FirmwareVendor, 100
Hdr, 100
NumberOfTableEntries, 100
RuntimeServices, 100
StandardErrorHandle, 100
StdErr, 100
_POSIX_C_SOURCE
profiler.c, 2427
time.c, 2415
__attribute__, 79
access_size, 80
address, 80
address_space_id, 80
alloc_kernel.c, 2413
arch.c, 956
base_address, 80
block_subsystem.c, 2152
bus, 80
capsule.h, 431
capsule_generated.h, 456
checksum, 80
creator_id, 80
creator_revision, 80
end_bus, 80
extended_checksum, 81
flags, 81
framebuffer.c, 1313
framebuffer.h, 555
gsi, 81
gsi_base, 81
hdr, 81
interrupts.c, 979
io_apic_address, 81
io_apic_id, 81
length, 81
local_apic_address, 82
main.c, 2302
oem_id, 82
oem_revision, 82
oem_table_id, 82
register_bit_offset, 82
register_bit_width, 82
reloc_stub.c, 1079
repl.c, 2469
reserved, 82
revision, 82
rpi5_mailbox.c, 863
rsdt_address, 83
segment_group, 83
signature, 83
source, 83
start_bus, 83
type, 83
vm.h, 2012, 2075
xsdt_address, 83
__kernel_end
pmm.c, 1467
rpi5_mmu.c, 877
__kernel_start
arch.c, 966
pmm.c, 1467
rpi5_mmu.c, 877
_dummy
FILE, 190
_freestanding_errno
errno.h, 571
_pad
artemis_sig_t, 101
blk_volume_meta_t, 112
CapsuleSigEntry, 134
CDConfigEntry, 135
homeblocks_sig_t, 205
log_region_ctrl_t, 222
user_identity_seed_t, 302
zuse_cert_devblock_t, 394
zuse_genesis_marker_t, 395
zuse_genesis_marker_v2_t, 396
_reserved
zuse_genesis_marker_v2_t, 396

aarch64_current_el
apic.c, 911
arch.c, 950
interrupts.c, 968
aarch64_exception_handler
interrupts.c, 968
aarch64_install_vectors
arch.c, 951
interrupts.c, 969
aarch64_irq_handler
interrupts.c, 970
aarch64_psci_conduit_init
apic.c, 912
arch.c, 951
abort
stdlib.h, 593
abort_requested
VM, 321
access
arch.c, 960
gdt_entry, 191
access_size
__attribute__, 80
accessed
BlockBuffer, 124
account_allocation
kmalloc.c, 1444
account_free
kmalloc.c, 1445
acl_allow
blk_meta_t, 108
DictEntry, 148
vm.h, 2075
ACL_ASSERT
acl_words_test.c, 2581
acl_default

DictEntry, 148
vm.h, 2075
acl_hint
DictPhysics, 151
physics_seed_definition, 243
acl_inherit_entry
acl_words.c, 2779
acl_words.h, 2930
acl_mode
DictEntry, 148
vm.h, 2075
ACL_MODE_STRICT
vm.h, 2006
ACL_MODE_TTL
vm.h, 2006
acl_pinned
DictEntry, 148
vm.h, 2075
acl_recheck
vm.c, 2695
vm_core.c, 1651
acl_reserved
blk_meta_t, 108
acl_skip
VM, 321
acl_ttl
blk_meta_t, 108
DictEntry, 148
vm.h, 2075
ACL_TTL_OPEN
vm.h, 2006
ACL_USER_DEFAULT
vm.h, 2006
acl_words.c
acl_inherit_entry, 2779
forth_acl_allow_fetch, 2779
forth_acl_allow_store, 2780
forth_acl_heat_fetch, 2781
forth_acl_inherit, 2781
forth_acl_init_primitives, 2782
forth_acl_mode_fetch, 2783
forth_acl_mode_store, 2783
forth_acl_pin, 2785
forth_acl_pinned_query, 2786
forth_acl_ttl_fetch, 2787
forth_acl_ttl_store, 2787
forth_acl_word_id, 2789
pop_xt, 2790
register_acl_words, 2791
acl_words.h
acl_inherit_entry, 2930
register_acl_words, 2932
acl_words_test.c
ACL_ASSERT, 2581
run_acl_words_tests, 2581
test_acl_adaptive_ttl, 2582
test_acl_deny_aborts, 2584
test_acl_inherit, 2585
test_acl_inherit_word_dispatch, 2586
test_acl_init_primitives, 2588
test_acl_pin_blocks_shadow, 2589
test_acl_pin_immutable, 2590
test_acl_read_accessors, 2592
test_acl_strict_mode, 2594
test_acl_ttl_hot_path, 2596
test_emergency_console_bypass, 2597
tests_failed, 2599
tests_passed, 2599
acpi_table
BootInfo, 127
actual_lbn
blk_reloc_entry_t, 111
actual_window_size
DoeMetrics, 153
HeartbeatSnapshot, 196
HeartbeatTickSnapshot, 202
WordTransitionMetrics, 362
adaptive_check_accumulator
RollingWindowOfTruth, 256
ADAPTIVE_CHECK_FREQUENCY
starforth_config.h, 1990
adaptive_decay_slope
InferenceOutputs, 217
ADAPTIVE_GROWTH_THRESHOLD
starforth_config.h, 1990
ADAPTIVE_MIN_WINDOW_SIZE
starforth_config.h, 1990
adaptive_pending
RollingWindowOfTruth, 256
ADAPTIVE_SHRINK_RATE
starforth_config.h, 1990
adaptive_shrink_rate
DoeMetrics, 153
adaptive_window_width
InferenceOutputs, 217
add_flags
xhci_input_ctrl_ctx32_t, 379
add_memmap_entry
rpi5_native_boot.c, 880
addr
cf_item_t, 141
rpi5_range_t, 259
VirtqDesc, 313
ADDR_MASK
vmm.c, 1470
address
__attribute__, 80
address_space_id
__attribute__, 80
advance_cursor
vt100.c, 1360
align_address
dictionary_manipulation_words.c, 2868
align_up
physics_runtime.c, 2382
align_up_size

kmalloc.c, 1445
AllHandles
uefi.h, 731
alloc
VMHostServices, 342
alloc_count
sf_alloc_stats, 261
alloc_host.c
g_stats, 2400
sf_alloc_get_stats, 2398
sf_alloc_init, 2398
sf_calloc, 2398
sf_free, 2399
sf_malloc, 2399
sf_realloc, 2400
alloc_kernel.c
__attribute__, 2413
g_alloc_count, 2408
g_arena_offset, 2413
g_free_count, 2408
g_initialized, 2413
g_stats, 2413
SF_ALIGN, 2401, 2410
SF_ALIGN_UP, 2410
sf_alloc_get_stats, 2402, 2410
sf_alloc_init, 2402, 2410
SF_ARENA_SIZE, 2410
sf_calloc, 2403, 2410
sf_free, 2404, 2411
sf_malloc, 2405, 2411
sf_realloc, 2406, 2412
alloc_page_zeroed
vmm.c, 1471
allocate_from_block
kmalloc.c, 1446
AllocateAddress
uefi.h, 730
AllocateAnyPages
uefi.h, 730
allocated
blk_bam_entry_t, 104
AllocateMaxAddress
uefi.h, 730
AllocatePages
_EFI_BOOT_SERVICES, 86
AllocatePool
_EFI_BOOT_SERVICES, 86
amount
VMFleetTouchSample, 340
analytics_heap_allocate
physics_runtime.c, 2382
analytics_heap_release
physics_runtime.c, 2383
analytics_ring_free
physics_runtime.c, 2384
anova_weight_q16
CDTuning, 139
ansi256_to_rgb
vt100.c, 1361
apic.c
aarch64_current_el, 911
aarch64_psci_conduit_init, 912
apic_eoi, 913, 926, 941
apic_eoi_intid, 913
apic_id, 926
apic_init, 914, 927, 941
apic_read_iar, 915
APIC_REG_EOI, 924
APIC_REG_ID, 924
APIC_REG_ISR_BASE, 924
APIC_REG_LVT_TIMER, 924
APIC_REG_SIVR, 924
APIC_REG_TIMER_CCR, 924
APIC_REG_TIMER_DCR, 924
APIC_REG_TIMER_ICR, 924
APIC_REG_TPR, 924
apic_spi_enable, 915
apic_timer_init, 916, 928, 943
apic_timer_period_tsc, 917, 929, 944
apic_timer_ppi, 917
apic_timer_rearm, 918, 930
apic_timer_start, 918, 931, 945
apic_timer_stop, 919, 932, 946
calibrate_apic_timer, 932
gic_bases_from_dtb, 919
GICC_BASE_PA_QEMU_DEFAULT, 909
GICC_CTLR, 909
GICC_CTLR_ENABLE_GRP0, 909
GICC_EOIR, 909
GICC_IAR, 909
GICC_PMR, 909
GICD_BASE_PA_QEMU_DEFAULT, 909
GICD_CTLR, 909
GICD_CTLR_ENABLE_GRP0, 909
GICD_ICFGR, 909
GICD_IPRIORITYR, 909
GICD_ISENABLER0, 910
GICD_ISPENDR, 910
GICD_ITARGETSR, 910
IA32_APIC_BASE_GLOBAL_EN, 924
IA32_APIC_BASE_MSR, 924
IA32_APIC_BASE_X2APIC_EN, 925
lapic_base, 938
LAPIC_DEFAULT_PHYS, 925
lapic_phys_base, 938
lapic_read, 934
LAPIC_VIRT_BASE, 925
lapic_write, 935
LVT_MASKED, 925
LVT_MODE_ONESHOT, 925
LVT_MODE_PERIODIC, 925
mmio_read32, 920
mmio_write32, 920
PMR_ALLOW_ALL, 910
rdmsr64, 936
rdtime, 946

riscv64_timer_rearm, 946
s_apic_hz, 938
s_counter_hz_apic, 921
s_gicc_base, 921
s_gicd_base, 921
s_next_deadline, 949
s_sbi_time_ok, 949
s_time_hz, 949
s_timer_period_tsc, 921, 949
s_timer_ppi, 921
SBI_BASE_FID_PROBE_EXT, 940
sbi_call1, 947
SBI_EXT_BASE, 940
SBI_EXT_TIME, 940
sbi_set_timer, 948
SBI_SUCCESS, 940
SBI_TIME_FID_SET_TIMER, 940
SIE_STIE, 940
TIMER_DIV_1, 925
TIMER_DIV_128, 925
TIMER_DIV_16, 925
TIMER_DIV_2, 926
TIMER_DIV_32, 926
TIMER_DIV_4, 926
TIMER_DIV_64, 926
TIMER_DIV_8, 926
timer_initial_count, 938
timer_period_tsc_ticks, 938
TIMER_PPI_EL1, 910
TIMER_PPI_EL2, 910
TIMER_PRIORITY, 910
timer_tick_hz, 938
wrmsr64, 937
apic.h
apic_eoi, 399
apic_id, 399
apic_init, 400
APIC_SPURIOUS_VECTOR, 398
apic_timer_init, 403
apic_timer_period_tsc, 404
apic_timer_rearm, 405
apic_timer_start, 406
apic_timer_stop, 408
APIC_TIMER_VECTOR, 398
apic_eoi
apic.c, 913, 926, 941
apic.h, 399
apic_eoi_intid
apic.c, 913
interrupts.c, 971
apic_id
apic.c, 926
apic.h, 399
apic_init
apic.c, 914, 927, 941
apic.h, 400
apic_read_iar
apic.c, 915
interrupts.c, 972
APIC_REG_EOI
apic.c, 924
APIC_REG_ID
apic.c, 924
APIC_REG_ISR_BASE
apic.c, 924
APIC_REG_LVT_TIMER
apic.c, 924
APIC_REG_SIVR
apic.c, 924
APIC_REG_TIMER_CCR
apic.c, 924
APIC_REG_TIMER_DCR
apic.c, 924
APIC_REG_TIMER_ICR
apic.c, 924
APIC_REG_TPR
apic.c, 924
apic_spi_enable
apic.c, 915
APIC_SPURIOUS_VECTOR
apic.h, 398
apic_timer_init
apic.c, 916, 928, 943
apic.h, 403
apic_timer_period_tsc
apic.c, 917, 929, 944
apic.h, 404
apic_timer_ppi
apic.c, 917
interrupts.c, 972
apic_timer_rearm
apic.c, 918, 930
apic.h, 405
interrupts.c, 973
apic_timer_start
apic.c, 918, 931, 945
apic.h, 406
apic_timer_stop
apic.c, 919, 932, 946
apic.h, 408
APIC_TIMER_VECTOR
apic.h, 398
app_data
blk_meta_t, 108
append_tag
rpi5_mailbox.c, 863
apply_sgr
vt100.c, 1362
arch.c
__attribute__, 956
__kernel_start, 966
aarch64_current_el, 950
aarch64_install_vectors, 951
aarch64_psci_conduit_init, 951
access, 960
arch_cold_reset, 952, 956, 962

arch_disable_interrupts, 952, 956, 962
arch_early_init, 953, 956, 963
arch_enable_interrupts, 953, 957, 964
arch_halt, 953, 957, 964
arch_mmu_init, 953, 957, 965
arch_read_timestamp, 954, 958, 965
base, 960
base_high, 960
base_low, 960
base_mid, 960
gdt_init, 958
gdt_set_entry, 959
granularity, 960
limit, 960
limit_low, 961
print_hex64, 965
riscv64_install_vectors, 966
s_current_el, 954
s_psci_use_smc, 954
arch.h
arch_cold_reset, 410
arch_disable_interrupts, 410
arch_early_init, 411
arch_enable_interrupts, 412
arch_halt, 413
arch_interrupts_init, 413
arch_mmu_init, 415
arch_read_timestamp, 415
arch_relax, 416
ARCH_ARM64
arch_detect.h, 1709
arch_cold_reset
arch.c, 952, 956, 962
arch.h, 410
arch_detect.h
ARCH_ARM64, 1709
ARCH_NAME, 1709
ARCH_RISCV64, 1710
ARCH_X86_64, 1710
NEXT_OPT, 1710
PRIM_BRANCH_OPT, 1710
PRIM_DROP_OPT, 1710
PRIM_DUP_OPT, 1710
PRIM_FETCH_OPT, 1710
PRIM_MINUS_OPT, 1710
PRIM_PLUS_OPT, 1710
PRIM_R_FETCH_OPT, 1710
PRIM_R_FROM_OPT, 1711
PRIM_STAR_OPT, 1711
PRIM_STORE_OPT, 1711
PRIM_SWAP_OPT, 1711
PRIM_TO_R_OPT, 1711
PRIM_ZBRANCH_OPT, 1711
vm_abs_opt, 1711
vm_add_overflow_opt, 1711
vm_clz_opt, 1711
vm_ctz_opt, 1712
vm_divmod_opt, 1712
vm_get_arch_name, 1714
vm_get_build_info, 1714
vm_has_asm_opt, 1715
vm_has_direct_threading, 1715
vm_has_sse42_opt, 1716
vm_max_opt, 1712
vm_memcpy_opt, 1712
vm_memzero_opt, 1712
vm_min_opt, 1712
vm_mul_double_opt, 1712
vm_pop_opt, 1713
vm_popcnt_opt, 1713
vm_prefetch_opt, 1713
vm_prefetch_stream_opt, 1713
vm_push_opt, 1713
vm_rpop_opt, 1713
vm_rpush_opt, 1713
vm_save_registers_opt, 1713
vm_setup_registers_opt, 1714
vm_strcmp_opt, 1714
vm_sub_overflow_opt, 1714
arch_disable_interrupts
arch.c, 952, 956, 962
arch.h, 410
arch_early_init
arch.c, 953, 956, 963
arch.h, 411
arch_enable_interrupts
arch.c, 953, 957, 964
arch.h, 412
arch_halt
arch.c, 953, 957, 964
arch.h, 413
arch_interrupts_init
arch.h, 413
interrupts.c, 974, 990
arch_mmu_init
arch.c, 953, 957, 965
arch.h, 415
ARCH_NAME
arch_detect.h, 1709
arch_read_timestamp
arch.c, 954, 958, 965
arch.h, 415
arch_relax
arch.h, 416
ARCH_RISCV64
arch_detect.h, 1710
ARCH_X86_64
arch_detect.h, 1710
Architecture Detection, 19
arena_base
CapsuleDirHeader, 130
arena_size
CapsuleDirHeader, 130
args
BootInfo, 127
arithmetic_word_abs

arithmetic_words.c, 2794
arithmetic_word_max
arithmetic_words.c, 2794
arithmetic_word_min
arithmetic_words.c, 2794
arithmetic_word_minus
arithmetic_words.c, 2795
arithmetic_word_multiply
arithmetic_words.c, 2796
arithmetic_word_negate
arithmetic_words.c, 2797
arithmetic_word_one_minus
arithmetic_words.c, 2797
arithmetic_word_one_plus
arithmetic_words.c, 2797
arithmetic_word_plus
arithmetic_words.c, 2798
arithmetic_word_star_slash
arithmetic_words.c, 2799
arithmetic_word_star_slash_mod
arithmetic_words.c, 2799
arithmetic_word_suites
arithmetic_words_test.c, 2600
arithmetic_word_two_divide
arithmetic_words.c, 2800
arithmetic_word_two_minus
arithmetic_words.c, 2800
arithmetic_word_two_multiply
arithmetic_words.c, 2801
arithmetic_word_two_plus
arithmetic_words.c, 2801
arithmetic_words.c
arithmetic_word_abs, 2794
arithmetic_word_max, 2794
arithmetic_word_min, 2794
arithmetic_word_minus, 2795
arithmetic_word_multiply, 2796
arithmetic_word_negate, 2797
arithmetic_word_one_minus, 2797
arithmetic_word_one_plus, 2797
arithmetic_word_plus, 2798
arithmetic_word_star_slash, 2799
arithmetic_word_star_slash_mod, 2799
arithmetic_word_two_divide, 2800
arithmetic_word_two_minus, 2800
arithmetic_word_two_multiply, 2801
arithmetic_word_two_plus, 2801
mixed_math_word_mod, 2802
register_arithmetic_words, 2802
word_divide, 2804
word_divmod, 2805
arithmetic_words.h
register_arithmetic_words, 2934
arithmetic_words_test.c
arithmetic_word_suites, 2600
run_arithmetic_words_tests, 2600
array_ptr
inference_words.c, 3030
art
sk_v4_drive_t, 269
art_state
sk_v4_drive_t, 269
art_used
sk_v4_drive_t, 269
artemis_sig.c
artemis_sig_check, 1062
artemis_sig_compute_crc, 1062
artemis_sig_genesis_stamp, 1063
artemis_sig_start_fblock, 1064
artemis_sig.h
ARTEMIS_SIG_BAD_CRC, 420
ARTEMIS_SIG_BAD_VERSION, 420
ARTEMIS_SIG_BLANK, 420
artemis_sig_check, 420
artemis_sig_compute_crc, 421
ARTEMIS_SIG_DEVBLOCK_FROM_TOP, 419
artemis_sig_genesis_stamp, 422
ARTEMIS_SIG_GET_MAGIC, 419
ARTEMIS_SIG_GET_VERSION, 419
ARTEMIS_SIG_MAGIC, 419
ARTEMIS_SIG_OK, 420
ARTEMIS_SIG_PACK, 420
ARTEMIS_SIG_READ_ERROR, 420
artemis_sig_result_t, 420
artemis_sig_size_check, 420
ARTEMIS_SIG_VERSION_0, 420
ARTEMIS_SIG_BAD_CRC
artemis_sig.h, 420
ARTEMIS_SIG_BAD_VERSION
artemis_sig.h, 420
ARTEMIS_SIG_BLANK
artemis_sig.h, 420
artemis_sig_check
artemis_sig.c, 1062
artemis_sig.h, 420
artemis_sig_compute_crc
artemis_sig.c, 1062
artemis_sig.h, 421
ARTEMIS_SIG_DEVBLOCK_FROM_TOP
artemis_sig.h, 419
artemis_sig_genesis_stamp
artemis_sig.c, 1063
artemis_sig.h, 422
ARTEMIS_SIG_GET_MAGIC
artemis_sig.h, 419
ARTEMIS_SIG_GET_VERSION
artemis_sig.h, 419
ARTEMIS_SIG_MAGIC
artemis_sig.h, 419
ARTEMIS_SIG_OK
artemis_sig.h, 420
ARTEMIS_SIG_PACK
artemis_sig.h, 420
ARTEMIS_SIG_READ_ERROR
artemis_sig.h, 420
artemis_sig_result_t

artemis_sig.h, 420
artemis_sig_size_check
artemis_sig.h, 420
artemis_sig_start_fblock
artemis_sig.c, 1064
artemis_sig_t, 101
_pad, 101
disk_uuid, 101
genesis_time_ns, 101
hdr_crc, 101
log_region_devblocks, 101
log_region_offset, 102
magic, 102
metadata_devblocks, 102
ARTEMIS_SIG_VERSION_0
artemis_sig.h, 420
assert
assert.h, 564
assert.h
assert, 564
static_assert, 564
assert_heartbeat_safe
test_contracts.c, 2675
test_contracts.h, 2536
assert_stack_depth
test_common.c, 2653
test_common.h, 2515
assert_stack_top
test_common.c, 2654
test_common.h, 2515
assert_vm_error
test_common.c, 2654
test_common.h, 2516
atan2
math.h, 575
atoi
stdlib.h, 593
atol
stdlib.h, 593
Attribute
EFI_FILE_INFO, 165
EFI_MEMORY_DESCRIPTOR, 173
avail
VirtBlkState, 304
VirtInputState, 307
VirtRngState, 316
avail_event
VirtqUsed, 314
avail_idx
VirtBlkState, 304
VirtInputState, 307
VirtRngState, 316
avg_latency_ns
DictPhysics, 151
physics_seed_definition, 243
avg_time
WordProfile, 356
avg_word_heat
DoeMetrics, 153
HeartbeatTickSnapshot, 202

backend
blkcfg_t, 116
physics_host_snapshot, 237
backend_seq
physics_host_snapshot, 237
physics_runtime_state, 243
bam
blk_dev_slot, 105
bam_cover_devblocks
sk_v4_grow.c, 1569
bam_devblocks
blk_volume_meta_t, 112
bam_dirty
blk_dev_slot, 105
bam_flush_to_disk
block_subsystem.c, 2152
bam_pages_used
block_subsystem.c, 2153
bam_start
blk_volume_meta_t, 112
bam_sync_from_disk
block_subsystem.c, 2154
bar0_phys
xhci_dev_t, 370
base
arch.c, 960
blkio_ram_state_t, 121
FbState, 185
fdt_uart_t, 188
FramebufferInfo, 190
gdtr, 192
idtr, 214
interrupts.c, 987
physics_analytics_heap, 234
Rpi5FramebufferInfo, 259
VM, 321
base_addr
VM, 321
base_address
__attribute__, 80
base_high
arch.c, 960
gdt_entry, 191
base_low
arch.c, 960
gdt_entry, 192
base_mid
arch.c, 960
gdt_entry, 192
base_point
ed25519.c, 1196
BayesianLatencyPosterior, 102
credible_lower_95, 103
credible_lower_99, 103
credible_upper_95, 103
credible_upper_99, 103

mean_ns_q48, 103
median_ns_q48, 103
sample_count, 103
stddev_ns_q48, 104
bCBWCBLength
usb_bot_cbw_t, 298
bCBWLUN
usb_bot_cbw_t, 298
BCM2712_SOC_RANGES_OFFSET
rpi5_dtb.c, 857
bCSWStatus
usb_bot_csw_t, 300
be32
fdt.c, 1287
fdt_uart.c, 1307
be32_at
rpi5_mmu.c, 872
rpi5_native_boot.c, 880
be32_cell
plic.c, 998
be64_addr_cell_pair
plic.c, 998
benchmark
CLIConfig, 142
benchmark_iterations
CLIConfig, 142
test_runner.c, 2692
benchmark_mode
test_runner.c, 2692
benefit_score
CDConfigEntry, 135
bg
VT100State, 350
birth_capsule_id
VMRegistryEntry, 346
birth_dict_hash
VMRegistryEntry, 346
birth_timestamp_ns
VMRegistryEntry, 346
bitlen
sha512_ctx_t, 264
bitmap_clear
pmm.c, 1459
bitmap_mark_range
pmm.c, 1459
bitmap_set
pmm.c, 1459
bitmap_test
pmm.c, 1460
blk_acl_allow_get
block_subsystem.c, 2155
block_subsystem.h, 1732
blk_acl_allow_set
block_subsystem.c, 2155
block_subsystem.h, 1732
blk_acl_ttl_get
block_subsystem.c, 2156
block_subsystem.h, 1733
blk_acl_ttl_set
block_subsystem.c, 2157
block_subsystem.h, 1733
blk_allocate
block_subsystem.c, 2157
block_subsystem.h, 1734
blk_bam_entry_t, 104
allocated, 104
dirty, 104
blk_commit_format
block_subsystem.c, 2158
blk_compute_fresh_geometry
block_subsystem.c, 2159
blk_dev_slot, 105
bam, 105
bam_dirty, 105
cache, 105
dev, 106
devblock_base_4k, 106
format_state, 106
has_ever_claimed, 106
migration_scan_lbn, 106
next, 106
raw_base, 106
start_lbn, 107
total_blkio_blocks_1k, 107
user_blocks, 107
vol_meta, 107
vol_meta_dirty, 107
blk_dev_slot_t
block_subsystem.c, 2152
BLK_DEVICE_SECTOR
block_subsystem.h, 1730
BLK_DISK_START
block_subsystem.h, 1730
BLK_DISK_SYS_RESERVED
block_subsystem.c, 2151
block_subsystem.h, 1730
BLK_EDIRTY
block_subsystem.h, 1731
BLK_EINVAL
block_subsystem.h, 1731
BLK_EIO
block_subsystem.h, 1731
BLK_ENODEV
block_subsystem.h, 1731
BLK_ENOMEM
block_subsystem.h, 1731
BLK_ENOSPC
block_subsystem.h, 1731
BLK_ERANGE
block_subsystem.h, 1731
BLK_ERESERVED
block_subsystem.h, 1731
blk_firsttouch_claim
block_subsystem.c, 2160
block_subsystem.h, 1734
BLK_FIRSTTOUCH_MAX_CLAIM

block_subsystem.c, 2151
BLK_FLAG_CLAIMED
block_subsystem.h, 1730
BLK_FLAG_MIGRATING
block_subsystem.h, 1730
BLK_FLAG_STALE
block_subsystem.h, 1730
blk_flags_get
block_subsystem.c, 2160
block_subsystem.h, 1735
blk_flags_set
block_subsystem.c, 2161
block_subsystem.h, 1736
blk_flush
block_subsystem.c, 2161
block_subsystem.h, 1736
BLK_FMT_FORMATTED
block_subsystem.h, 1731
BLK_FMT_PROVISIONAL
block_subsystem.h, 1731
blk_format_or_load_disk
block_subsystem.c, 2162
BLK_FORTH_SIZE
block_subsystem.h, 1730
BLK_FORTH_SYS_RESERVED
block_subsystem.c, 2151
block_subsystem.h, 1730
blk_get_buffer
block_subsystem.c, 2163
block_subsystem.h, 1737
blk_get_device_free_blocks
block_subsystem.c, 2164
block_subsystem.h, 1738
blk_get_device_range
block_subsystem.c, 2165
block_subsystem.h, 1739
blk_get_empty_buffer
block_subsystem.c, 2166
block_subsystem.h, 1739
blk_get_first_disk_range
block_subsystem.c, 2166
block_subsystem.h, 1740
blk_get_meta
block_subsystem.c, 2167
block_subsystem.h, 1740
blk_get_timestamp
block_subsystem.c, 2167
blk_get_total_blocks
block_subsystem.c, 2168
block_subsystem.h, 1741
blk_get_volume_meta
block_subsystem.c, 2169
block_subsystem.h, 1742
blk_is_allocated
block_subsystem.c, 2169
block_subsystem.h, 1742
blk_is_valid
block_subsystem.c, 2169
block_subsystem.h, 1742
blk_lbn_device_handle
block_subsystem.c, 2170
block_subsystem.h, 1743
blk_mark_allocated
block_subsystem.c, 2171
block_subsystem.h, 1744
blk_mark_free
block_subsystem.c, 2171
block_subsystem.h, 1744
BLK_META_FENCE_INIT
block_subsystem.h, 1730
BLK_META_PER_BLOCK
block_subsystem.h, 1730
blk_meta_relocate_devblock
block_subsystem.c, 2172
block_subsystem.h, 1745
blk_meta_t, 108
acl_allow, 108
acl_reserved, 108
acl_ttl, 108
app_data, 108
chain_length, 109
checksum, 109
content_length, 109
content_type, 109
created_time, 109
encoding, 109
entropy, 109
flags, 109
hash, 109
magic, 110
modified_time, 110
next_block, 110
owner_fp, 110
padding, 110
parent_block, 110
prev_block, 110
reserved1, 110
reserved_future, 110
write_count, 111
BLK_META_TOTAL
block_subsystem.h, 1731
blk_meta_zone_read
block_subsystem.c, 2173
block_subsystem.h, 1746
blk_meta_zone_write
block_subsystem.c, 2174
block_subsystem.h, 1747
blk_migration_idle_check
block_subsystem.c, 2174
block_subsystem.h, 1747
blk_migration_idle_check_slot
block_subsystem.c, 2175
BLK_OK
block_subsystem.h, 1731
blk_owner_fp_get
block_subsystem.c, 2176

block_subsystem.h, 1748
blk_owner_fp_set
block_subsystem.c, 2177
block_subsystem.h, 1749
blk_owner_has_claim
block_subsystem.c, 2177
block_subsystem.h, 1749
BLK_PACK_RATIO
block_subsystem.h, 1731
BLK_RAM_BLOCKS
block_subsystem.h, 1731
blk_reloc_entry_t, 111
actual_lbn, 111
home_lbn, 111
BLK_RELOC_MAX_ENTRIES
block_subsystem.c, 2151
blk_set_meta
block_subsystem.c, 2178
block_subsystem.h, 1750
blk_set_volume_meta
block_subsystem.c, 2178
block_subsystem.h, 1750
blk_subsys_add_raw_device
block_subsystem.c, 2179
block_subsystem.h, 1751
blk_subsys_attach_device
block_subsystem.c, 2179
block_subsystem.h, 1751
blk_subsys_confirm_format
block_subsystem.c, 2180
block_subsystem.h, 1752
blk_subsys_detach_device
block_subsystem.c, 2181
block_subsystem.h, 1753
blk_subsys_epoch
block_subsystem.c, 2182
block_subsystem.h, 1754
blk_subsys_init
block_subsystem.c, 2182
block_subsystem.h, 1754
blk_subsys_relocate_block
block_subsystem.c, 2183
block_subsystem.h, 1755
blk_subsys_set_format_bam_for
block_subsystem.c, 2184
block_subsystem.h, 1756
blk_subsys_shutdown
block_subsystem.c, 2184
block_subsystem.h, 1756
blk_update
block_subsystem.c, 2184
block_subsystem.h, 1756
blk_vm_assign
block_words.c, 2807
blk_vm_cbuf
VM, 322
blk_vm_check_epoch
block_words.c, 2808
block_words.h, 2939
blk_vm_dirty
VM, 322
blk_vm_epoch
VM, 322
blk_vm_evict
block_words.c, 2808
blk_vm_find
block_words.c, 2809
blk_vm_flush_all
block_words.c, 2810
block_words.h, 2940
blk_vm_is_cross_device_copy
block_words.c, 2811
block_words.h, 2941
blk_vm_lbn
VM, 322
blk_vm_load
block_words.c, 2812
blk_vm_next
VM, 322
blk_vm_slot_for_addr
block_words.c, 2812
block_words.h, 2942
BLK_VM_SLOTS
vm.h, 2006
BLK_VM_WINDOW_BASE
vm.h, 2006
BLK_VM_WINDOW_SIZE
vm.h, 2006
blk_volume_compute_totals
block_subsystem.c, 2185
block_subsystem.h, 1757
blk_volume_meta_t, 111
_pad, 112
bam_devblocks, 112
bam_start, 112
created_time, 112
devblock_base, 113
first_free, 113
flags, 113
free_blocks, 113
hdr_crc, 113
label, 113
last_allocated, 113
magic, 113
meta_fence_blocks, 114
mounted_time, 114
reloc_devblocks, 114
reloc_start, 114
reserved_disk_lo, 114
reserved_ram_lo, 114
total_blocks, 114
total_devblocks, 115
total_volumes, 115
tracked_blocks, 115
version, 115
blkcfg.h

BLKCFG_BACKEND_FILE, 1717
BLKCFG_BACKEND_NONE, 1717
BLKCFG_BACKEND_RAM, 1717
blkcfg_backend_t, 1717
BLKCFG_DEFAULT_FBS, 1717
blkcfg_format_summary, 1718
blkcfg_init_from_options, 1718
BLKCFG_PATH_MAX, 1717
g_blkcfg, 1718
BLKCFG_BACKEND_FILE
blkcfg.h, 1717
BLKCFG_BACKEND_NONE
blkcfg.h, 1717
BLKCFG_BACKEND_RAM
blkcfg.h, 1717
blkcfg_backend_t
blkcfg.h, 1717
BLKCFG_DEFAULT_FBS
blkcfg.h, 1717
blkcfg_format_summary
blkcfg.h, 1718
blkcfg_init_from_options
blkcfg.h, 1718
BLKCFG_PATH_MAX
blkcfg.h, 1717
blkcfg_t, 115
backend, 116
blocks, 116
fbs, 116
file_path, 116
read_only, 116
size_bytes, 116
blkio.h
blkio_bzero, 1720
blkio_close, 1720
blkio_dev_t, 1720
BLKIO_ECLOSED, 1720
BLKIO_EINVAL, 1720
BLKIO_EIO, 1720
BLKIO_ENOSPACE, 1720
BLKIO_ENOSUP, 1720
blkio_flush, 1721
BLKIO_FORTH_BLOCK_SIZE, 1720
blkio_info, 1721
BLKIO_OK, 1720
blkio_open, 1722
blkio_read, 1722
blkio_write, 1723
blkio_bzero
blkio.h, 1720
blkio_close
blkio.h, 1720
blkio_dev, 116
forth_block_size, 117
is_open, 117
state, 117
total_blocks, 117
vt, 117
blkio_dev_t
blkio.h, 1720
BLKIO_ECLOSED
blkio.h, 1720
BLKIO_EINVAL
blkio.h, 1720
BLKIO_EIO
blkio.h, 1720
BLKIO_ENOSPACE
blkio.h, 1720
BLKIO_ENOSUP
blkio.h, 1720
blkio_factory.c
blkio_factory_open, 2125
blkio_file_init_state, 2127
blkio_file_state_size, 2128
blkio_file_vtable, 2129
blkio_ram_init_state, 2129
blkio_ram_state_size, 2130
blkio_ram_vtable, 2131
nonempty, 2131
blkio_factory.h
blkio_factory_open, 1725
blkio_factory_open
blkio_factory.c, 2125
blkio_factory.h, 1725
blkio_file.c
blkio_file_init_state, 2133
blkio_file_state_size, 2134
BLKIO_FILE_VT, 2140
blkio_file_vtable, 2134
file_close, 2135
file_flush, 2135
file_info, 2136
file_open, 2136
file_read, 2137
file_write, 2138
full_read_at, 2138
full_write_at, 2139
PATH_MAX, 2133
blkio_file_init_state
blkio_factory.c, 2127
blkio_file.c, 2133
blkio_file_state_size
blkio_factory.c, 2128
blkio_file.c, 2134
blkio_file_state_t, 118
fbs, 118
fd, 118
path, 118
read_only, 118
total_blocks, 118
BLKIO_FILE_VT
blkio_file.c, 2140
blkio_file_vtable
blkio_factory.c, 2129
blkio_file.c, 2134
blkio_flush

blkio.h, 1721
BLKIO_FORTH_BLOCK_SIZE
blkio.h, 1720
blkio_info
blkio.h, 1721
blkio_info_t, 119
forth_block_size, 119
phys_sector_size, 119
phys_size_bytes, 119
read_only, 119
total_blocks, 119
BLKIO_OK
blkio.h, 1720
blkio_open
blkio.h, 1722
blkio_params_t, 120
forth_block_size, 120
opaque, 120
total_blocks, 120
blkio_ram.c
blkio_ram_init_state, 2142
blkio_ram_state_size, 2142
BLKIO_RAM_VT, 2147
blkio_ram_vtable, 2143
in_bounds, 2143
ram_close, 2144
ram_flush, 2144
ram_info, 2145
ram_open, 2145
ram_read, 2146
ram_write, 2146
blkio_ram_init_state
blkio_factory.c, 2129
blkio_ram.c, 2142
capsule_mint.c, 1129
blkio_ram_state_size
blkio_factory.c, 2130
blkio_ram.c, 2142
capsule_mint.c, 1130
blkio_ram_state_t, 120
base, 121
fbs, 121
read_only, 121
total_blocks, 121
BLKIO_RAM_VT
blkio_ram.c, 2147
blkio_ram_vtable
blkio_factory.c, 2131
blkio_ram.c, 2143
capsule_mint.c, 1130
blkio_read
blkio.h, 1722
blkio_usb.c
blkio_usb_open_msc, 1504
count, 1508
g_usb_blk, 1508
g_usb_blk_vtable, 1508
slots, 1508
usb_blk_close, 1504
usb_blk_flush, 1505
usb_blk_info, 1505
usb_blk_open, 1505
usb_blk_read, 1505
usb_blk_state_for, 1506
usb_blk_write, 1507
xdev, 1508
blkio_usb.h
blkio_usb_open_msc, 423
blkio_usb_open_msc
blkio_usb.c, 1504
blkio_usb.h, 423
blkio_vtable_t, 121
close, 122
flush, 122
info, 122
open, 122
read, 122
write, 122
blkio_write
blkio.h, 1723
BlkioUsbState, 123
scsi_block_size, 123
scsi_blocks_per_fblock, 123
slot_id, 123
total_forth_blocks, 123
xdev, 124
block
kmalloc_prefix_t, 220
BLOCK_COUNT
io.h, 1802
BLOCK_NORMAL_NC
rpi5_mmu.c, 872
BLOCK_NORMAL_WB
rpi5_mmu.c, 872
block_number
BlockBuffer, 124
BLOCK_SIZE
block_words.h, 2939
vm.h, 2006
block_size
reloc_block, 254
reloc_stub.c, 1080
block_subsystem.c
__attribute__, 2152
bam_flush_to_disk, 2152
bam_pages_used, 2153
bam_sync_from_disk, 2154
blk_acl_allow_get, 2155
blk_acl_allow_set, 2155
blk_acl_ttl_get, 2156
blk_acl_ttl_set, 2157
blk_allocate, 2157
blk_commit_format, 2158
blk_compute_fresh_geometry, 2159
blk_dev_slot_t, 2152
BLK_DISK_SYS_RESERVED, 2151

blk_firsttouch_claim, 2160
BLK_FIRSTTOUCH_MAX_CLAIM, 2151
blk_flags_get, 2160
blk_flags_set, 2161
blk_flush, 2161
blk_format_or_load_disk, 2162
BLK_FORTH_SYS_RESERVED, 2151
blk_get_buffer, 2163
blk_get_device_free_blocks, 2164
blk_get_device_range, 2165
blk_get_empty_buffer, 2166
blk_get_first_disk_range, 2166
blk_get_meta, 2167
blk_get_timestamp, 2167
blk_get_total_blocks, 2168
blk_get_volume_meta, 2169
blk_is_allocated, 2169
blk_is_valid, 2169
blk_lbn_device_handle, 2170
blk_mark_allocated, 2171
blk_mark_free, 2171
blk_meta_relocate_devblock, 2172
blk_meta_zone_read, 2173
blk_meta_zone_write, 2174
blk_migration_idle_check, 2174
blk_migration_idle_check_slot, 2175
blk_owner_fp_get, 2176
blk_owner_fp_set, 2177
blk_owner_has_claim, 2177
BLK_RELOC_MAX_ENTRIES, 2151
blk_set_meta, 2178
blk_set_volume_meta, 2178
blk_subsys_add_raw_device, 2179
blk_subsys_attach_device, 2179
blk_subsys_confirm_format, 2180
blk_subsys_detach_device, 2181
blk_subsys_epoch, 2182
blk_subsys_init, 2182
blk_subsys_relocate_block, 2183
blk_subsys_set_format_bam_for, 2184
blk_subsys_shutdown, 2184
blk_update, 2184
blk_volume_compute_totals, 2185
cache_get_slot, 2186
cache_load_devblock, 2187
cache_writeback, 2188
chain_append, 2189
choose_B, 2190
compute_crc64, 2190
compute_totals_from_B, 2191
crc64_init, 2191
crc64_inited, 2212
crc64_table, 2212
devblock4k_to_lba1k, 2192
devblock_claimed_by_lbn, 2193
devblock_is_free, 2194
dirty_ram, 2212
DISK_CACHE_SLOTS, 2151
epoch, 2212
first_disk_slot, 2194
g, 2212
g_format_bam_for, 2212
head, 2213
initialized, 2213
lbn_to_slot, 2195
lbn_to_slot_pbn, 2196
meta_from_slice, 2197
META_PER_BLOCK, 2151
META_REGION_OFFSET, 2151
META_REGION_SIZE, 2151
meta_to_slice, 2198
MIGRATION_SCAN_BUDGET, 2151
MIGRATION_WEAR_THRESHOLD, 2151
minzu, 2199
pbam_clr, 2200
pbam_set, 2200
pbam_test, 2201
ram_base, 2213
ram_size, 2213
ram_user, 2213
read_devblock_4k, 2201
read_header_4k, 2202
reloc, 2213
reloc_count, 2213
reloc_flush_to_disk, 2203
reloc_load_from_disk, 2204
resolve_lbn, 2204
slot_by_dev, 2205
slot_pbn_pack_offset, 2206
slot_pbn_to_devblock, 2207
total_user_lbn, 2213
udiv_floor, 2208
volmeta_from_buf, 2208
volmeta_to_buf, 2209
write_devblock_4k, 2210
write_header_4k, 2211
block_subsystem.h
blk_acl_allow_get, 1732
blk_acl_allow_set, 1732
blk_acl_ttl_get, 1733
blk_acl_ttl_set, 1733
blk_allocate, 1734
BLK_DEVICE_SECTOR, 1730
BLK_DISK_START, 1730
BLK_DISK_SYS_RESERVED, 1730
BLK_EDIRTY, 1731
BLK_EINVAL, 1731
BLK_EIO, 1731
BLK_ENODEV, 1731
BLK_ENOMEM, 1731
BLK_ENOSPC, 1731
BLK_ERANGE, 1731
BLK_ERESERVED, 1731
blk_firsttouch_claim, 1734
BLK_FLAG_CLAIMED, 1730
BLK_FLAG_MIGRATING, 1730

BLK_FLAG_STALE, 1730
blk_flags_get, 1735
blk_flags_set, 1736
blk_flush, 1736
BLK_FMT_FORMATTED, 1731
BLK_FMT_PROVISIONAL, 1731
BLK_FORTH_SIZE, 1730
BLK_FORTH_SYS_RESERVED, 1730
blk_get_buffer, 1737
blk_get_device_free_blocks, 1738
blk_get_device_range, 1739
blk_get_empty_buffer, 1739
blk_get_first_disk_range, 1740
blk_get_meta, 1740
blk_get_total_blocks, 1741
blk_get_volume_meta, 1742
blk_is_allocated, 1742
blk_is_valid, 1742
blk_lbn_device_handle, 1743
blk_mark_allocated, 1744
blk_mark_free, 1744
BLK_META_FENCE_INIT, 1730
BLK_META_PER_BLOCK, 1730
blk_meta_relocate_devblock, 1745
BLK_META_TOTAL, 1731
blk_meta_zone_read, 1746
blk_meta_zone_write, 1747
blk_migration_idle_check, 1747
BLK_OK, 1731
blk_owner_fp_get, 1748
blk_owner_fp_set, 1749
blk_owner_has_claim, 1749
BLK_PACK_RATIO, 1731
BLK_RAM_BLOCKS, 1731
blk_set_meta, 1750
blk_set_volume_meta, 1750
blk_subsys_add_raw_device, 1751
blk_subsys_attach_device, 1751
blk_subsys_confirm_format, 1752
blk_subsys_detach_device, 1753
blk_subsys_epoch, 1754
blk_subsys_init, 1754
blk_subsys_relocate_block, 1755
blk_subsys_set_format_bam_for, 1756
blk_subsys_shutdown, 1756
blk_update, 1756
blk_volume_compute_totals, 1757
compute_crc64, 1758
block_word_acl_allow_fetch
block_words.c, 2813
block_word_acl_allow_store
block_words.c, 2813
block_word_acl_ttl_fetch
block_words.c, 2814
block_word_acl_ttl_store
block_words.c, 2815
block_word_blk_attach
block_words.c, 2815
block_word_block
block_words.c, 2816
block_word_buffer
block_words.c, 2817
block_word_confirm_format
block_words.c, 2818
block_word_empty_buffers
block_words.c, 2818
block_word_flush
block_words.c, 2819
block_word_list
block_words.c, 2820
block_word_load
block_words.c, 2820
block_word_next_block
block_words.c, 2821
block_word_owner_fetch
block_words.c, 2822
block_word_relocate
block_words.c, 2823
block_word_save_buffers
block_words.c, 2824
block_word_scr
block_words.c, 2824
block_word_suites
block_words_test.c, 2602
block_word_thru
block_words.c, 2825
block_word_update
block_words.c, 2826
block_words.c
blk_vm_assign, 2807
blk_vm_check_epoch, 2808
blk_vm_evict, 2808
blk_vm_find, 2809
blk_vm_flush_all, 2810
blk_vm_is_cross_device_copy, 2811
blk_vm_load, 2812
blk_vm_slot_for_addr, 2812
block_word_acl_allow_fetch, 2813
block_word_acl_allow_store, 2813
block_word_acl_ttl_fetch, 2814
block_word_acl_ttl_store, 2815
block_word_blk_attach, 2815
block_word_block, 2816
block_word_buffer, 2817
block_word_confirm_format, 2818
block_word_empty_buffers, 2818
block_word_flush, 2819
block_word_list, 2820
block_word_load, 2820
block_word_next_block, 2821
block_word_owner_fetch, 2822
block_word_relocate, 2823
block_word_save_buffers, 2824
block_word_scr, 2824
block_word_thru, 2825
block_word_update, 2826

empty_all_buffers, 2827
get_block_buffer, 2828
get_empty_buffer, 2829
init_block_system, 2830
mark_buffer_dirty, 2830
register_block_words, 2831
save_all_buffers, 2833
set_scr, 2833
block_words.h
blk_vm_check_epoch, 2939
blk_vm_flush_all, 2940
blk_vm_is_cross_device_copy, 2941
blk_vm_slot_for_addr, 2942
BLOCK_SIZE, 2939
BUFFER_COUNT, 2939
BUFFER_TYPE_HARDWARE, 2939
BUFFER_TYPE_MEMORY, 2939
BUFFER_TYPE_NETWORK, 2939
BUFFER_TYPE_STORAGE, 2939
buffer_type_t, 2939
empty_all_buffers, 2943
get_block_buffer, 2943
get_empty_buffer, 2944
init_block_system, 2945
mark_buffer_dirty, 2945
MAX_BLOCKS, 2939
register_block_words, 2946
save_all_buffers, 2948
block_words_test.c
block_word_suites, 2602
run_block_words_tests, 2601
BlockBuffer, 124
accessed, 124
block_number, 124
data, 125
dirty, 125
hw_address, 125
type, 125
blocks
blkcfg_t, 116
BlockSize
EFI_BLOCK_IO_MEDIA, 162
Blt
_EFI_GRAPHICS_OUTPUT_PROTOCOL, 93
BlueMask
EFI_PIXEL_BITMASK, 174
bmCBWFlags
usb_bot_cbw_t, 298
bmRequestType
usb_setup_packet_t, 301
bold
VT100State, 350
BOOLEAN
uefi.h, 725
boot_cpuid_phys
fdt_header_t, 187
boot_disk
BootInfo, 127
BOOT_INFO_KERNEL_STACK_BASE_OFFSET
boot_info_offsets.h, 425
BOOT_INFO_KERNEL_STACK_SIZE_OFFSET
boot_info_offsets.h, 425
boot_info_offsets.h
BOOT_INFO_KERNEL_STACK_BASE_OFFSET, 425
BOOT_INFO_KERNEL_STACK_SIZE_OFFSET, 425
BootDisk, 125
disk_guid, 126
gpt, 126
is_stick, 126
BootInfo, 126
acpi_table, 127
args, 127
boot_disk, 127
dtb, 127
framebuffer, 127
kernel_stack_base, 127
kernel_stack_size, 127
memory_map, 127
memory_map_descriptor_size, 128
memory_map_size, 128
runtime_services, 128
uefi_boot_services_exited, 128
BootServices
_EFI_SYSTEM_TABLE, 99
bootstrap_result
ParityPacket, 225
bot_cap_block_size
xhci_dev_t, 370
bot_cap_last_lba
xhci_dev_t, 370
bot_cbw
xhci_dev_t, 370
bot_cmd_kind
xhci_dev_t, 370
BOT_CMD_NONE
xhci_dev_t, 369
BOT_CMD_READ10
xhci_dev_t, 369
BOT_CMD_READ_CAPACITY10
xhci_dev_t, 369
BOT_CMD_TEST_UNIT_READY
xhci_dev_t, 369
BOT_CMD_WRITE10
xhci_dev_t, 369
bot_csw
xhci_dev_t, 370
bot_data_buf
xhci_dev_t, 370
bot_expected_data_len
xhci_dev_t, 371
bot_last_status
xhci_dev_t, 371
bot_last_tag
xhci_dev_t, 371

bot_msc_attach_pending
xhci_msc_slot_t, 382
bot_msc_attached
xhci_msc_slot_t, 382
bot_msc_detach_pending
xhci_msc_slot_t, 382
bot_next_tag
xhci_dev_t, 371
bot_read10_block_size
xhci_dev_t, 371
bot_read10_lba
xhci_dev_t, 371
bot_read10_num_blocks
xhci_dev_t, 371
bot_reset_clear_remaining
xhci_dev_t, 372
bot_stall_recoveries
xhci_dev_t, 372
BOT_STATUS_FAILED
xhci_dev_t, 369
BOT_STATUS_IDLE
xhci_dev_t, 369
BOT_STATUS_PASS
xhci_dev_t, 369
BOT_STATUS_TIMEOUT
xhci_dev_t, 369
BOT_TUR_CHAIN_NONE
xhci_dev_t, 369
BOT_TUR_CHAIN_READ10
xhci_dev_t, 369
BOT_TUR_CHAIN_READ_CAPACITY10
xhci_dev_t, 369
bot_tur_chain_target
xhci_dev_t, 372
BOT_TUR_CHAIN_WRITE10
xhci_dev_t, 369
bot_tur_retries
xhci_dev_t, 372
bot_write10_block_size
xhci_dev_t, 372
bot_write10_lba
xhci_dev_t, 372
bot_write10_num_blocks
xhci_dev_t, 372
branch_count
Profiler, 249
break_me
CLIConfig, 142
break_me_tests.c
finalize_report, 2604
init_report, 2604
report_file, 2606
run_break_me_tests, 2605
test_end_time, 2606
test_start_time, 2606
total_tests_run, 2606
bRequest
usb_setup_packet_t, 301
bucket_collapse_flag
HeartbeatState, 198
bucket_hit_percent
DoeMetrics, 154
bucket_hits
doe_metrics.c, 2269
DoeMetrics, 154
HotwordsStats, 209
bucket_hits_delta
HeartbeatTickSnapshot, 202
bucket_mode_transitions
HeartbeatState, 198
bucket_posterior
inference_words.c, 3030
bucket_reorders
HotwordsStats, 209
bucket_search_latency_ns
DoeMetrics, 154
bucket_search_samples
HotwordsStats, 210
bucket_search_stddev_ns
DoeMetrics, 154
bucket_search_total_ns_q48
HotwordsStats, 210
bucket_search_variance_sum_q48
HotwordsStats, 210
bucket_sum_heat_variance
HeartbeatState, 198
bucket_sum_K
HeartbeatState, 198
bucket_sum_K_squared
HeartbeatState, 199
bucket_sum_window_variance
HeartbeatState, 199
bucket_tick_count
HeartbeatState, 199
buf
sha512_ctx_t, 264
buf_len
sha512_ctx_t, 264
buffer
XXHash64State, 392
BUFFER_COUNT
block_words.h, 2939
buffer_size
XXHash64State, 392
buffer_to_ram
rpi5_mailbox.c, 864
BUFFER_TYPE_HARDWARE
block_words.h, 2939
BUFFER_TYPE_MEMORY
block_words.h, 2939
BUFFER_TYPE_NETWORK
block_words.h, 2939
BUFFER_TYPE_STORAGE
block_words.h, 2939
buffer_type_t
block_words.h, 2939

bufs
VirtInputState, 307
Build Information, 20
build_memory_map
rpi5_native_boot.c, 881
bulk_in_ep_addr
xhci_msc_slot_t, 382
bulk_in_max_packet
xhci_msc_slot_t, 382
bulk_in_ring
xhci_msc_slot_t, 382
bulk_in_ring_cycle
xhci_msc_slot_t, 383
bulk_in_ring_enq
xhci_msc_slot_t, 383
bulk_out_ep_addr
xhci_msc_slot_t, 383
bulk_out_max_packet
xhci_msc_slot_t, 383
bulk_out_ring
xhci_msc_slot_t, 383
bulk_out_ring_cycle
xhci_msc_slot_t, 383
bulk_out_ring_enq
xhci_msc_slot_t, 383
bus
__attribute__, 80
PciDevice, 227
bus_to_phys
rpi5_mailbox.c, 864
BX_BYTES
ed25519.c, 1205
BY_BYTES
ed25519.c, 1205
ByProtocol
uefi.h, 731
ByRegisterNotify
uefi.h, 731
bytes
physics_analytics_heap, 234
sk_v4_grow.c, 1571
sk_v4_marker.c, 1599
bytes_contain
timer.c, 1053

cache
blk_dev_slot, 105
HotwordsCache_s, 207
cache_count
HotwordsCache_s, 207
cache_demotions
DoeMetrics, 154
cache_get_slot
block_subsystem.c, 2186
cache_hit_latency_ns
DoeMetrics, 154
cache_hit_percent
DoeMetrics, 154
cache_hit_samples
HotwordsStats, 210
cache_hit_stddev_ns
DoeMetrics, 154
cache_hit_total_ns_q48
HotwordsStats, 210
cache_hit_variance_sum_q48
HotwordsStats, 211
cache_hits
doe_metrics.c, 2269
DoeMetrics, 155
HotwordsStats, 211
MemoryProfile, 224
cache_hits_delta
HeartbeatTickSnapshot, 203
cache_load_devblock
block_subsystem.c, 2187
cache_misses
MemoryProfile, 224
cache_posterior
inference_words.c, 3031
cache_promotions
DoeMetrics, 155
cache_slot_t, 128
data, 129
devblock, 129
dirty, 129
loaded, 129
meta, 129
meta_dirty, 129
valid, 129
cache_writeback
block_subsystem.c, 2188
cached_pubkey
capsule_sig.c, 1150
calculate_posterior
physics_hotwords_cache.c, 2328
calib_record
timer.c, 1050
calibrate_apic_timer
apic.c, 932
calibrate_tsc_with_pmtimer
timer.c, 1017
calibrate_window_hpet
timer.c, 1018
calibrate_window_pit
timer.c, 1019
call_count
ProfileTimer, 252
WordProfile, 356
WordStats, 358
calloc
stdlib.h, 593
can_split
kmalloc.c, 1447
cap
xhci_dev_t, 373
cap_length
xhci_cap_regs_t, 365

capacity_sectors
VirtBlkState, 305
capsule.h
__attribute__, 431
CAPSULE_BIRTH_ELIGIBLE, 428
CAPSULE_DESC_MAGIC, 428
CAPSULE_DIR_MAGIC, 428
CAPSULE_DOE_ELIGIBLE, 428
CAPSULE_ERR_BAD_HASH_ALG, 431
CAPSULE_ERR_BAD_MAGIC, 431
CAPSULE_ERR_BAD_VERSION, 431
CAPSULE_ERR_BOUNDS, 431
CAPSULE_ERR_HASH_MISMATCH, 431
CAPSULE_ERR_MODE_INVALID, 431
CAPSULE_ERR_NULL_PTR, 431
CAPSULE_ERR_REVOKED_ACTIVE, 431
capsule_find_by_id, 431
capsule_find_by_name, 432
capsule_find_mama_init, 432
CAPSULE_FLAG_ACTIVE, 428
CAPSULE_FLAG_CONTRIB, 428
CAPSULE_FLAG_DEPRECATED, 429
CAPSULE_FLAG_EXPERIMENT, 429
CAPSULE_FLAG_MAMA_INIT, 429
CAPSULE_FLAG_PINNED, 429
CAPSULE_FLAG_PRODUCTION, 429
CAPSULE_FLAG_REVOKED, 429
capsule_get_payload, 433
CAPSULE_HASH_BLAKE3, 431
CAPSULE_HASH_SHA256, 431
CAPSULE_HASH_XXHASH64, 431
CAPSULE_IS_MAMA_INIT, 429
CAPSULE_MAGIC_GET_HASHALG, 429
CAPSULE_MAGIC_GET_SIG, 430
CAPSULE_MAGIC_GET_VERSION, 430
CAPSULE_MAGIC_PACK, 430
CAPSULE_MAX_COUNT, 430
CAPSULE_MODE_VALID, 430
CAPSULE_NAME_MAX, 430
CAPSULE_VALID, 431
capsule_validate, 434
capsule_validate_result_str, 435
CAPSULE_VERSION_0, 430
CapsuleDesc, 435
CapsuleHashAlg, 430
CapsuleValidateResult, 431
capsule_birth.c
capsule_birth_baby, 1091
capsule_birth_mama, 1093
capsule_birth_set_hooks, 1094
capsule_run_experiment, 1095
capsule_vm_find_by_name, 1096
capsule_vm_find_by_name_nocase, 1097
capsule_vm_force_reap, 1098
capsule_vm_kill, 1099
capsule_vm_kill_all_nonmama, 1100
capsule_vm_registry_count, 1100
capsule_vm_registry_get, 1100
capsule_vm_registry_get_by_index, 1101
capsule_vm_registry_init, 1102
capsule_vm_registry_set_name, 1103
capsule_vm_set_pending_reap, 1103
capsule_vm_set_state, 1104
contrib_capsule_refused, 1105
dictionary_snapshot_acl_from_parent, 1105
dispatch_init_forth, 1105
vm_alloc_fn, 1109
vm_dict_hash_fn, 1109
vm_exec_fn, 1109
vm_find_entry_ptr, 1106
vm_name_copy, 1106
vm_name_eq, 1106
vm_name_eq_nocase, 1107
vm_name_prefix_eq_nocase, 1107
vm_node_t, 1091
vm_registry_alloc, 1108
vm_registry_count, 1110
vm_registry_head, 1110
vm_registry_live_count, 1108
vm_to_lower, 1109
capsule_birth.h
capsule_birth_baby, 438
capsule_birth_mama, 439
capsule_birth_set_hooks, 441
capsule_run_experiment, 442
capsule_vm_find_by_name, 443
capsule_vm_find_by_name_nocase, 444
capsule_vm_force_reap, 445
capsule_vm_hooks_register, 446
capsule_vm_kill, 447
capsule_vm_kill_all_nonmama, 448
capsule_vm_registry_count, 449
capsule_vm_registry_get, 449
capsule_vm_registry_get_by_index, 450
capsule_vm_registry_init, 450
capsule_vm_registry_set_name, 451
capsule_vm_set_pending_reap, 452
capsule_vm_set_state, 453
CapsuleDictHashFn, 437
CapsuleExecFn, 437
CapsuleVMAllocFn, 438
capsule_birth_baby
capsule_birth.c, 1091
capsule_birth.h, 438
CAPSULE_BIRTH_ELIGIBLE
capsule.h, 428
capsule_birth_mama
capsule_birth.c, 1093
capsule_birth.h, 439
capsule_birth_set_hooks
capsule_birth.c, 1094
capsule_birth.h, 441
capsule_block_header
capsule_blocks.c, 1111
capsule_blocks.h, 454
CAPSULE_BLOCK_LINES

capsule_loader.c, 1118
capsule_blocks.c
capsule_block_header, 1111
capsule_blocks.h
capsule_block_header, 454
capsule_clear_blocks
capsule_loader.c, 1119
capsule_loader.h, 458
capsule_console.c
capsule_console_birth, 1112
CONSOLE_IDENTITY_SRC, 1113
capsule_console_birth
capsule_console.c, 1112
CAPSULE_DESC_MAGIC
capsule.h, 428
capsule_dict_hash_hook
capsule_vm_hooks.c, 1154
CAPSULE_DIR_MAGIC
capsule.h, 428
CAPSULE_DOE_ELIGIBLE
capsule.h, 428
CAPSULE_ERR_BAD_HASH_ALG
capsule.h, 431
CAPSULE_ERR_BAD_MAGIC
capsule.h, 431
CAPSULE_ERR_BAD_VERSION
capsule.h, 431
CAPSULE_ERR_BOUNDS
capsule.h, 431
CAPSULE_ERR_HASH_MISMATCH
capsule.h, 431
CAPSULE_ERR_MODE_INVALID
capsule.h, 431
CAPSULE_ERR_NULL_PTR
capsule.h, 431
CAPSULE_ERR_REVOKED_ACTIVE
capsule.h, 431
capsule_exec_hook
capsule_vm_hooks.c, 1154
capsule_exec_init
capsule_loader.c, 1120
capsule_loader.h, 458
CAPSULE_EXEC_LINE_MAX
capsule_loader.c, 1118
capsule_exec_payload
capsule_loader.c, 1122
capsule_loader.h, 460
capsule_find.c
capsule_find_by_id, 1114
capsule_find_by_name, 1115
capsule_find_mama_init, 1116
capsule_get_payload, 1116
capsule_streq, 1117
capsule_find_by_id
capsule.h, 431
capsule_find.c, 1114
capsule_find_by_name
capsule.h, 432
capsule_find.c, 1115
capsule_find_mama_init
capsule.h, 432
capsule_find.c, 1116
CAPSULE_FLAG_ACTIVE
capsule.h, 428
CAPSULE_FLAG_CONTRIB
capsule.h, 428
CAPSULE_FLAG_DEPRECATED
capsule.h, 429
CAPSULE_FLAG_EXPERIMENT
capsule.h, 429
CAPSULE_FLAG_MAMA_INIT
capsule.h, 429
CAPSULE_FLAG_PINNED
capsule.h, 429
CAPSULE_FLAG_PRODUCTION
capsule.h, 429
CAPSULE_FLAG_REVOKED
capsule.h, 429
capsule_generated.h
__attribute__, 456
capsule_get_payload
capsule.h, 433
capsule_find.c, 1116
capsule_hash
CapsuleRunRecord, 132
CAPSULE_HASH_BLAKE3
capsule.h, 431
CAPSULE_HASH_SHA256
capsule.h, 431
CAPSULE_HASH_XXHASH64
capsule.h, 431
capsule_id
CapsuleRunRecord, 132
CAPSULE_IS_MAMA_INIT
capsule.h, 429
CAPSULE_LINE_COLS
capsule_loader.c, 1119
capsule_load_blocks
capsule_loader.c, 1123
capsule_loader.h, 461
capsule_loader.c
CAPSULE_BLOCK_LINES, 1118
capsule_clear_blocks, 1119
capsule_exec_init, 1120
CAPSULE_EXEC_LINE_MAX, 1118
capsule_exec_payload, 1122
CAPSULE_LINE_COLS, 1119
capsule_load_blocks, 1123
CAPSULE_MAX_DEFERRED, 1119
CAPSULE_RAM_OFFSET, 1119
exec_block_with_retry, 1124
is_block_header, 1125
LOADER_BLOCK_SIZE, 1119
popcount16, 1126
u32_to_dec, 1126
write_ramdrive_block, 1126

capsule_loader.h
capsule_clear_blocks, 458
capsule_exec_init, 458
capsule_exec_payload, 460
capsule_load_blocks, 461
CAPSULE_MAGIC_GET_HASHALG
capsule.h, 429
CAPSULE_MAGIC_GET_SIG
capsule.h, 430
CAPSULE_MAGIC_GET_VERSION
capsule.h, 430
CAPSULE_MAGIC_PACK
capsule.h, 430
CAPSULE_MAX_COUNT
capsule.h, 430
CAPSULE_MAX_DEFERRED
capsule_loader.c, 1119
CAPSULE_MAX_RUN_RECORDS
capsule_run.h, 464
capsule_mint.c
blkio_ram_init_state, 1129
blkio_ram_state_size, 1130
blkio_ram_vtable, 1130
capsule_mint_identity, 1131
capsule_mint_identity_scratch, 1132
MINT_CERT_DEVBLOCK, 1128
MINT_CERT_DEVBLOCKS, 1128
MINT_DEFAULT_PERSONALITY, 1135
MINT_IDENTITY_SRC_DEVBLOCKS, 1129
MINT_IDENTITY_SRC_OFFSET, 1129
MINT_METADATA_DEVBLOCKS, 1129
MINT_RESTRICTED_PERSONALITY, 1135
read_devblock, 1133
verify_mint, 1134
write_devblock, 1135
capsule_mint_identity
capsule_mint.c, 1131
capsule_mint_identity_scratch
capsule_mint.c, 1132
CAPSULE_MODE_VALID
capsule.h, 430
CAPSULE_NAME_MAX
capsule.h, 430
capsule_parity_log_birth
capsule_run.c, 1138
capsule_run.h, 465
capsule_parity_log_birth_failed
capsule_run.c, 1138
capsule_run.h, 466
capsule_parity_log_kill
capsule_run.c, 1139
capsule_run.h, 466
capsule_parity_log_mama_init
capsule_run.c, 1139
capsule_run.h, 467
capsule_parity_log_run
capsule_run.c, 1140
capsule_run.h, 467
capsule_parity_set_output
capsule_run.c, 1141
capsule_run.h, 468
CAPSULE_RAM_OFFSET
capsule_loader.c, 1119
capsule_run.c
capsule_parity_log_birth, 1138
capsule_parity_log_birth_failed, 1138
capsule_parity_log_kill, 1139
capsule_parity_log_mama_init, 1139
capsule_parity_log_run, 1140
capsule_parity_set_output, 1141
capsule_run_log_count, 1141
capsule_run_log_get, 1142
capsule_run_log_init, 1142
capsule_run_log_record, 1142
PARITY_EMIT, 1137
PARITY_HAVE_SINK, 1137
parity_put_hex64, 1143
parity_put_u32, 1143
parity_put_u64, 1144
parity_put_uuid, 1144
parity_putc, 1145
parity_puts, 1145
run_id_counter, 1145
run_log, 1145
run_log_count, 1145
run_log_head, 1145
capsule_run.h
CAPSULE_MAX_RUN_RECORDS, 464
capsule_parity_log_birth, 465
capsule_parity_log_birth_failed, 466
capsule_parity_log_kill, 466
capsule_parity_log_mama_init, 467
capsule_parity_log_run, 467
capsule_parity_set_output, 468
CAPSULE_RUN_ERR_EXEC_FAIL, 465
CAPSULE_RUN_ERR_FLEET_FULL, 465
CAPSULE_RUN_ERR_HASH_MISMATCH, 465
CAPSULE_RUN_ERR_INVALID, 465
CAPSULE_RUN_ERR_NOT_ELIGIBLE, 465
CAPSULE_RUN_ERR_STILLBORN, 465
capsule_run_log_count, 469
capsule_run_log_get, 469
capsule_run_log_init, 469
capsule_run_log_record, 470
CAPSULE_RUN_OK, 465
CapsuleRunResult, 464
VM_NAME_MAX, 464
VM_STATE_DEAD, 465
VM_STATE_EMBRYO, 465
VM_STATE_LIVE, 465
VM_STATE_STILLBORN, 465
VM_STATE_STOPPED, 465
VM_STATE_SWITCHED_OUT, 465
VMState, 465
CAPSULE_RUN_ERR_EXEC_FAIL
capsule_run.h, 465

CAPSULE_RUN_ERR_FLEET_FULL
capsule_run.h, 465
CAPSULE_RUN_ERR_HASH_MISMATCH
capsule_run.h, 465
CAPSULE_RUN_ERR_INVALID
capsule_run.h, 465
CAPSULE_RUN_ERR_NOT_ELIGIBLE
capsule_run.h, 465
CAPSULE_RUN_ERR_STILLBORN
capsule_run.h, 465
capsule_run_experiment
capsule_birth.c, 1095
capsule_birth.h, 442
capsule_run_log_count
capsule_run.c, 1141
capsule_run.h, 469
capsule_run_log_get
capsule_run.c, 1142
capsule_run.h, 469
capsule_run_log_init
capsule_run.c, 1142
capsule_run.h, 469
capsule_run_log_record
capsule_run.c, 1142
capsule_run.h, 470
CAPSULE_RUN_OK
capsule_run.h, 465
capsule_runcap.c
capsule_runcap_birth, 1146
capsule_runcap_birth
capsule_runcap.c, 1146
capsule_sig.c
cached_pubkey, 1150
capsule_sig_result_str, 1148
capsule_verify_signature, 1148
find_and_cache_pubkey, 1149
pubkey_state, 1150
capsule_sig.h
CAPSULE_SIG_INVALID, 472
CAPSULE_SIG_MISSING, 472
CAPSULE_SIG_NO_ROOT_KEY, 472
CAPSULE_SIG_OK, 472
capsule_sig_result_str, 473
CAPSULE_SIGNING_CERT_NAME, 472
capsule_verify_signature, 473
CapsuleSigResult, 472
CAPSULE_SIG_INVALID
capsule_sig.h, 472
CAPSULE_SIG_MISSING
capsule_sig.h, 472
CAPSULE_SIG_NO_ROOT_KEY
capsule_sig.h, 472
CAPSULE_SIG_OK
capsule_sig.h, 472
capsule_sig_result_str
capsule_sig.c, 1148
capsule_sig.h, 473
CAPSULE_SIGNING_CERT_NAME
capsule_sig.h, 472
capsule_streq
capsule_find.c, 1117
CAPSULE_VALID
capsule.h, 431
capsule_validate
capsule.h, 434
capsule_validate.c, 1151
capsule_validate.c
capsule_validate, 1151
capsule_validate_result_str, 1152
validate_result_strings, 1153
capsule_validate_result_str
capsule.h, 435
capsule_validate.c, 1152
capsule_verify_signature
capsule_sig.c, 1148
capsule_sig.h, 473
CAPSULE_VERSION_0
capsule.h, 430
capsule_vm_alloc_hook
capsule_vm_hooks.c, 1155
capsule_vm_find_by_name
capsule_birth.c, 1096
capsule_birth.h, 443
capsule_vm_find_by_name_nocase
capsule_birth.c, 1097
capsule_birth.h, 444
capsule_vm_force_reap
capsule_birth.c, 1098
capsule_birth.h, 445
capsule_vm_hooks.c
capsule_dict_hash_hook, 1154
capsule_exec_hook, 1154
capsule_vm_alloc_hook, 1155
capsule_vm_hooks_register, 1156
capsule_vm_hooks_register
capsule_birth.h, 446
capsule_vm_hooks.c, 1156
capsule_vm_kill
capsule_birth.c, 1099
capsule_birth.h, 447
capsule_vm_kill_all_nonmama
capsule_birth.c, 1100
capsule_birth.h, 448
capsule_vm_physics.c
console_put_u64, 1160
fleet_heartbeat_tick_count, 1170
fleet_last_capacity_tick, 1170
fleet_transfer_slope_q48, 1171
fleet_window, 1171
l8_regime_modulation_q16, 1160
slope_fit_quality_q48, 1171
VM_FLEET_WINDOW_DEPTH, 1159
vm_physics_conserved, 1161
VM_PHYSICS_EPSILON_Q48, 1159
vm_physics_find, 1162
vm_physics_find_root_id, 1163

vm_physics_fleet_heat_sum, 1163
vm_physics_head, 1171
vm_physics_heartbeat_tick, 1164
vm_physics_heat_of, 1165
vm_physics_init, 1166
VM_PHYSICS_MIN_INFORMATIVE_SAMPLES, 1160
vm_physics_node_t, 1160
vm_physics_retire, 1167
vm_physics_status, 1167
vm_physics_tick, 1168
vm_physics_touch, 1169
vm_physics_transfer, 1170
capsule_vm_physics.h
vm_physics_conserved, 476
vm_physics_fleet_heat_sum, 476
vm_physics_heartbeat_tick, 477
vm_physics_heat_of, 478
vm_physics_init, 479
vm_physics_retire, 480
vm_physics_status, 480
vm_physics_tick, 481
vm_physics_touch, 482
capsule_vm_registry_count
capsule_birth.c, 1100
capsule_birth.h, 449
capsule_vm_registry_get
capsule_birth.c, 1100
capsule_birth.h, 449
capsule_vm_registry_get_by_index
capsule_birth.c, 1101
capsule_birth.h, 450
capsule_vm_registry_init
capsule_birth.c, 1102
capsule_birth.h, 450
capsule_vm_registry_set_name
capsule_birth.c, 1103
capsule_birth.h, 451
capsule_vm_set_pending_reap
capsule_birth.c, 1103
capsule_birth.h, 452
capsule_vm_set_state
capsule_birth.c, 1104
capsule_birth.h, 453
capsule_wirebind.c
capsule_wirebind_attached_username, 1173
capsule_wirebind_eject, 1173
capsule_wirebind_overflow_idle_check, 1174
capsule_wirebind_reap_idle_check, 1175
capsule_wirebind_try_attach, 1176
capsule_wirebind_unclean_detach, 1177
capsule_wirebind_verify_cert, 1178
g_wirebind_attached_dev, 1182
g_wirebind_attached_username, 1182
g_wirebind_attached_valid, 1182
g_wirebind_attached_vm_id, 1183
g_wirebind_live, 1183
read_devblock, 1179
WIREBIND_CERT_MAX_DEVBLOCKS, 1173
wirebind_live_alloc, 1180
wirebind_live_find_by_dev, 1180
wirebind_live_remove, 1180
WIREBIND_MAX_LIVE, 1173
WIREBIND_OVERFLOW_CLAIM_DEVBLOCKS, 1173
WIREBIND_OVERFLOW_FREE_THRESHOLD_PCT, 1173
wirebind_resolve_attached, 1181
wirebind_teardown_console, 1181
capsule_wirebind_attached_username
capsule_wirebind.c, 1173
capsule_wirebind_eject
capsule_wirebind.c, 1173
capsule_wirebind_overflow_idle_check
capsule_wirebind.c, 1174
capsule_wirebind_reap_idle_check
capsule_wirebind.c, 1175
capsule_wirebind_try_attach
capsule_wirebind.c, 1176
capsule_wirebind_unclean_detach
capsule_wirebind.c, 1177
capsule_wirebind_verify_cert
capsule_wirebind.c, 1178
capsule_zuse_boot.c
capsule_zuse_boot_attached_dev, 1184
capsule_zuse_boot_check, 1184
capsule_zuse_boot_check_key, 1185
capsule_zuse_boot_genesis_mint, 1186
capsule_zuse_boot_load_root_pubkey, 1186
capsule_zuse_boot_logout, 1187
capsule_zuse_boot_root_pubkey, 1188
capsule_zuse_boot_try_attach, 1188
g_zuse_attached_dev, 1192
g_zuse_attached_this_device, 1192
genesis_marker_read, 1189
install_and_activate, 1190
read_devblock, 1191
capsule_zuse_boot_attached_dev
capsule_zuse_boot.c, 1184
capsule_zuse_boot_check
capsule_zuse_boot.c, 1184
capsule_zuse_boot_check_key
capsule_zuse_boot.c, 1185
capsule_zuse_boot_genesis_mint
capsule_zuse_boot.c, 1186
capsule_zuse_boot_load_root_pubkey
capsule_zuse_boot.c, 1186
capsule_zuse_boot_logout
capsule_zuse_boot.c, 1187
capsule_zuse_boot_root_pubkey
capsule_zuse_boot.c, 1188
capsule_zuse_boot_try_attach
capsule_zuse_boot.c, 1188
CapsuleDesc
capsule.h, 435
CapsuleDictHashFn

capsule_birth.h, 437
CapsuleDirHeader, 130
arena_base, 130
arena_size, 130
desc_capacity, 130
desc_count, 130
dir_hash, 131
magic, 131
name_count, 131
reserved, 131
CapsuleExecFn
capsule_birth.h, 437
CapsuleHashAlg
capsule.h, 430
CapsuleNameEntry, 131
name, 132
CapsuleRunRecord, 132
capsule_hash, 132
capsule_id, 132
ended_ns, 132
flags, 133
post_dict_hash, 133
pre_dict_hash, 133
reserved, 133
result_code, 133
run_id, 133
started_ns, 133
vm_id, 133
CapsuleRunResult
capsule_run.h, 464
CapsuleSigEntry, 134
_pad, 134
has_sig, 134
sig, 134
CapsuleSigResult
capsule_sig.h, 472
CapsuleValidateResult
capsule.h, 431
CapsuleVMAllocFn
capsule_birth.h, 438
capture_exec_result
test_contracts.c, 2676
test_contracts.h, 2537
category
ProfileScope, 251
category_timers
Profiler, 249
CBWCB
usb_bot_cbw_t, 299
cd_classify_ids
compudynamics.c, 2218
compudynamics.h, 1764
cd_config_table_force
compudynamics.c, 2219
compudynamics.h, 1765
cd_config_table_free
compudynamics.c, 2219
compudynamics.h, 1766
cd_config_table_init
compudynamics.c, 2220
compudynamics.h, 1766
cd_config_table_seed
compudynamics.c, 2221
compudynamics.h, 1767
cd_config_table_tick
compudynamics.c, 2221
compudynamics.h, 1768
cd_isqrt32
compudynamics.c, 2223
cd_isqrt64
compudynamics.c, 2224
cd_ln_approx_q16
compudynamics.c, 2224
CD_MAX_CLASSIFY_DEPTH
compudynamics.h, 1764
cd_tuning_vm
compudynamics.c, 2225
compudynamics.h, 1769
cd_tuning_word
compudynamics.c, 2225
compudynamics.h, 1769
CDConfigEntry, 134
_pad, 135
benefit_score, 135
config_bits, 135
has_prev, 135
prev_locality_q16, 135
prev_window, 135
tick_count, 135
CDConfigTable, 136
current_config, 136
current_regime, 136
entries, 137
num_configs, 137
num_regimes, 137
regime_scores, 137
regime_ticks, 137
total_regime_ticks, 137
CDRegimeMetrics, 138
diversity_q16, 138
locality_q16, 138
volatility_q16, 138
CDTuning, 138
anova_weight_q16, 139
diversity_high_threshold_q16, 139
id_window_depth, 139
joint_weight_q16, 139
locality_high_threshold_q16, 139
locality_lookback, 139
num_configs, 140
num_regimes, 140
reward_gain, 140
score_decay_factor_q16, 140
score_max, 140
score_min, 140
ucb_k, 140

volatility_high_threshold_q16, 140
ceil
math.h, 575
CELL
vm.h, 2012
CELL_BITS
format_words.c, 2911
cell_h
vt100.c, 1363
cell_prop
fdt_uart.c, 1307
cell_t
vm.h, 2010
cell_w
vt100.c, 1364
cells
fdt_uart.c, 1308
cells_at
rpi5_mmu.c, 872
cert_devblocks
homeblocks_sig_t, 205
cert_offset
homeblocks_sig_t, 205
cf
VM, 322
CF_BEGIN
vm.h, 2011
CF_CASE
vm.h, 2011
CF_DO
vm.h, 2011
CF_ELSE
vm.h, 2011
cf_epoch_sync
Control Flow Stack, 45
CF_IF
vm.h, 2011
cf_item_t, 141
addr, 141
tag, 141
cf_last_mode
ControlFlowState, 144
CF_OF
vm.h, 2011
cf_peek_item
Control Flow Stack, 46
cf_pop_item
Control Flow Stack, 46
cf_push_item
Control Flow Stack, 47
cf_sp
ControlFlowState, 144
cf_stack
ControlFlowState, 145
CF_STACK_MAX
vm.h, 2006
cf_tag_t
vm.h, 2011
CF_WHILE
vm.h, 2011
cfg_read16
pci.c, 1483
cfg_read32
pci.c, 1483
cfg_read8
pci.c, 1484
cfg_write16
pci.c, 1485
cfg_write32
pci.c, 1485
cfg_write8
pci.c, 1486
cgroup_cpu_usage_us
physics_host_snapshot, 237
cgroup_memory_current_bytes
physics_host_snapshot, 237
chain_append
block_subsystem.c, 2189
chain_length
blk_meta_t, 109
channel
physics_analytics_event_header, 228
CHAR16
uefi.h, 725
CHARGE_NEUTRAL
vm.h, 2006
check_counter
HeartbeatState, 199
check_physics_transparent
test_contracts.c, 2677
test_contracts.h, 2538
CheckEvent
_EFI_BOOT_SERVICES, 86
checksum
__attribute__, 80
blk_meta_t, 109
choose_B
block_subsystem.c, 2190
CHUNK_BITS
format_words.c, 2911
CHUNK_MASK
format_words.c, 2911
ci_lower_95
DoeMetrics, 155
ci_upper_95
DoeMetrics, 155
clamp_scalar
ed25519.c, 1196
clamp_u32
physics_metadata.c, 2349
clamped
VMFleetTouchSample, 340
cleanup_and_exit
main.c, 2302
cleanup_blkio
main.c, 2303

ClearScreen
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 97
cli.c
cli_parse, 2214
cli_print_usage, 2215
parse_u32, 2216
cli.h
cli_parse, 1761
cli_print_usage, 1762
HEARTBEAT_LOG_FULL, 1760
HEARTBEAT_LOG_OFF, 1760
HEARTBEAT_LOG_SUMMARY, 1760
HeartbeatLogMode, 1760
cli_parse
cli.c, 2214
cli.h, 1761
cli_print_usage
cli.c, 2215
cli.h, 1762
CLIConfig, 141
benchmark, 142
benchmark_iterations, 142
break_me, 142
disk_img_path, 142
doe_experiment, 142
fail_fast, 143
heartbeat_log_mode, 143
log_level, 143
log_level_explicitly_set, 143
ram_disk_mb, 143
run_tests, 143
run_tests_flag_observed, 143
script_mode, 143
Close
_EFI_FILE_PROTOCOL, 91
close
blkio_vtable_t, 122
CloseEvent
_EFI_BOOT_SERVICES, 86
CloseProtocol
_EFI_BOOT_SERVICES, 86
cmap_subtable_off
ttf_font_t, 289
cmd_ring
xhci_dev_t, 373
cmd_ring_cycle
xhci_dev_t, 373
cmd_ring_enq
xhci_dev_t, 373
cmdline.c
cmdline_parse_ascii, 1066
cmdline_parse_size, 1067
parse_log_level, 1068
sf_isspace, 1069
sf_strlcpy, 1070
sf_strncmp, 1070
cmdline.h
cmdline_parse_ascii, 485
cmdline_parse_size, 486
cmdline_parse_ascii
cmdline.c, 1066
cmdline.h, 485
cmdline_parse_size
cmdline.c, 1067
cmdline.h, 486
cmp32
scalar25519.c, 1216
cntfrq_read
timer.c, 1007
cntpct_read
timer.c, 1007
coalesce_neighbors
kmalloc.c, 1448
code
DiagEvent, 147
VirtioInputEvent, 312
codepoint
ttf_raster_cache_slot_t, 296
collect_ram_regions
rpi5_native_boot.c, 882
collect_reserved_ranges
rpi5_native_boot.c, 882
color_reset
log.c, 2299
cols
VT100State, 351
common
VirtBlkState, 305
VirtInputState, 307
VirtRngState, 316
compare_by_calls
profiler.c, 2427
compare_by_time
profiler.c, 2428
compare_dict_entry_heat_desc
dictionary_heat_optimization.c, 2227
compare_heat_ascending
dictionary_heat_optimization.c, 2227
compat_list_contains
fdt.c, 1287
compatible_has
fdt_uart.c, 1309
compile_buffer
VM, 322
COMPILE_BUFFER_SIZE
vm.h, 2006
compile_operations
Profiler, 249
compile_pos
VM, 323
compile_size
VM, 323
compiling_word
VM, 323
compudynamics.c
cd_classify_ids, 2218

cd_config_table_force, 2219
cd_config_table_free, 2219
cd_config_table_init, 2220
cd_config_table_seed, 2221
cd_config_table_tick, 2221
cd_isqrt32, 2223
cd_isqrt64, 2224
cd_ln_approx_q16, 2224
cd_tuning_vm, 2225
cd_tuning_word, 2225
compudynamics.h
cd_classify_ids, 1764
cd_config_table_force, 1765
cd_config_table_free, 1766
cd_config_table_init, 1766
cd_config_table_seed, 1767
cd_config_table_tick, 1768
CD_MAX_CLASSIFY_DEPTH, 1764
cd_tuning_vm, 1769
cd_tuning_word, 1769
compute_crc64
block_subsystem.c, 2190
block_subsystem.h, 1758
compute_fit_quality
inference_engine.c, 2273
compute_levene_statistic
inference_engine.c, 2274
compute_mean_q48
inference_engine.c, 2276
compute_median_q48
inference_engine.c, 2277
compute_totals_from_B
block_subsystem.c, 2191
compute_variance_q48
inference_engine.c, 2278
inference_engine.h, 1792
config
xhci_op_regs_t, 388
config_bits
CDConfigEntry, 135
config_descriptor
xhci_msc_slot_t, 384
config_generation
VirtioCommonCfg, 309
config_msix_vector
VirtioCommonCfg, 309
config_total_length
xhci_msc_slot_t, 384
config_word
xhci_input_ctrl_ctx32_t, 379
ConfigurationTable
_EFI_SYSTEM_TABLE, 99
ConIn
_EFI_SYSTEM_TABLE, 99
connect_state
xhci_dev_t, 373
ConnectController
_EFI_BOOT_SERVICES, 86
ConOut
_EFI_SYSTEM_TABLE, 99
console.c
console_attach_serial, 1253
console_cancel_deferred_line_start, 1254
console_describe_serial, 1254
console_ensure_line_start, 1255
console_fb_clear, 1256
console_fb_draw_cursor, 1256
console_fb_enable_ttf, 1257
console_fb_erase_cursor, 1258
console_fb_init, 1259
console_fb_mirror, 1259
console_fb_scroll_back, 1260
console_fb_scroll_fwd, 1260
console_fb_toggle_graphics, 1261
console_get_vm_name, 1262
console_getc, 1262
console_init, 1263
console_poll, 1263
console_println, 1264
console_putc, 1265
console_putc_inner, 1266
console_puts, 1268
console_save_vm_name, 1270
console_set_user_prefix_provider, 1271
console_set_vm_name, 1271
console_tx_count, 1271
CONSOLE_VM_NAME_BUF, 1252
emit_prefix, 1272
fb_mirrored, 1273
g_active_vm_name, 1282
g_active_vm_name_buf, 1282
g_console_tx_count, 1282
g_fb_mirror, 1283
g_line_start, 1283
g_pending_line_close, 1283
g_serial_base, 1283
g_serial_from_dt, 1283
g_serial_kind, 1283
g_serial_shift, 1283
g_serial_width, 1284
g_user_prefix_fn, 1284
PL011_FR, 1252
PL011_FR_RXFE, 1252
PL011_FR_TXFF, 1252
raw_putc, 1274
SERIAL_MMIO16550, 1253
SERIAL_NONE, 1253
SERIAL_PL011, 1253
SERIAL_PORT16550, 1253
serial_putc, 1276
serial_reg_read, 1278
serial_reg_write, 1279
serial_rx_ready, 1280
SERIAL_TX_MAX_SPIN, 1252
serial_tx_ready, 1281
UART16550_LSR, 1252

UART16550_LSR_DR, 1252
UART16550_LSR_THRE, 1252
console.h
console_attach_serial, 490
console_cancel_deferred_line_start, 491
console_describe_serial, 491
console_ensure_line_start, 492
console_fb_clear, 493
console_fb_draw_cursor, 493
console_fb_enable_ttf, 494
console_fb_erase_cursor, 495
console_fb_init, 496
console_fb_mirror, 496
console_fb_scroll_back, 497
console_fb_scroll_fwd, 497
console_fb_toggle_graphics, 498
console_get_vm_name, 499
console_getc, 499
console_init, 500
console_poll, 500
console_println, 501
console_putc, 502
console_puts, 503
console_save_vm_name, 505
console_set_user_prefix_provider, 506
console_set_vm_name, 506
console_tx_count, 506
console_user_prefix_fn, 490
g_sk_fault_word, 507
console_attach_serial
console.c, 1253
console.h, 490
console_cancel_deferred_line_start
console.c, 1254
console.h, 491
console_describe_serial
console.c, 1254
console.h, 491
console_ensure_line_start
console.c, 1255
console.h, 492
console_fb_clear
console.c, 1256
console.h, 493
console_fb_draw_cursor
console.c, 1256
console.h, 493
console_fb_enable_ttf
console.c, 1257
console.h, 494
console_fb_erase_cursor
console.c, 1258
console.h, 495
console_fb_init
console.c, 1259
console.h, 496
console_fb_mirror
console.c, 1259
console.h, 496
console_fb_scroll_back
console.c, 1260
console.h, 497
console_fb_scroll_fwd
console.c, 1260
console.h, 497
console_fb_toggle_graphics
console.c, 1261
console.h, 498
console_get_vm_name
console.c, 1262
console.h, 499
console_getc
console.c, 1262
console.h, 499
CONSOLE_IDENTITY_SRC
capsule_console.c, 1113
console_init
console.c, 1263
console.h, 500
console_poll
console.c, 1263
console.h, 500
console_println
console.c, 1264
console.h, 501
console_put_u64
capsule_vm_physics.c, 1160
console_putc
console.c, 1265
console.h, 502
console_putc_inner
console.c, 1266
console_puts
console.c, 1268
console.h, 503
console_save_vm_name
console.c, 1270
console.h, 505
console_set_user_prefix_provider
console.c, 1271
console.h, 506
console_set_vm_name
console.c, 1271
console.h, 506
console_tx_count
console.c, 1271
console.h, 506
console_user_prefix_fn
console.h, 490
CONSOLE_VM_NAME_BUF
console.c, 1252
ConsoleInHandle
_EFI_SYSTEM_TABLE, 100
ConsoleOutHandle
_EFI_SYSTEM_TABLE, 100
consume_seq

physics_analytics_header, 230
content_length
blk_meta_t, 109
content_type
blk_meta_t, 109
context_accuracy_percent
DoeMetrics, 155
context_correct
DoeMetrics, 155
context_predictions_total
DoeMetrics, 155
context_transitions
WordTransitionMetrics, 362
context_var_addr
VocabularyState, 348
context_vocab
VocabularyState, 348
context_window
WordTransitionMetrics, 362
context_window_pos
WordTransitionMetrics, 362
contour_count
ttf_outline_t, 293
contour_ends
ttf_outline_t, 293
contract
TestCase, 277
CONTRACT_DEFAULT_FUZZ_ROUNDS
test_contracts.h, 2535
contract_flags_t
test_contracts.h, 2536
contract_get_violation_count
test_contracts.c, 2679
test_contracts.h, 2540
CONTRACT_HEARTBEAT_SAFE
test_contracts.h, 2535
CONTRACT_NONE
test_contracts.h, 2535
CONTRACT_PHYSICS_TRANSPARENT
test_contracts.h, 2535
contract_rand
test_contracts.c, 2679
contract_reset_violation_count
test_contracts.c, 2680
test_contracts.h, 2540
contrib_capsule_refused
capsule_birth.c, 1105
control
xhci_trb_t, 391
Control Flow Stack, 45
cf_epoch_sync, 45
cf_peek_item, 46
cf_pop_item, 46
cf_push_item, 47
control_forth_0branch
Runtime Branch Helpers, 49
control_forth_again
Runtime Branch Helpers, 50
control_forth_begin
Runtime Branch Helpers, 50
control_forth_branch
Runtime Branch Helpers, 51
control_forth_case
Runtime Branch Helpers, 52
control_forth_do
Runtime Branch Helpers, 52
control_forth_else
Runtime Branch Helpers, 53
control_forth_endcase
Runtime Branch Helpers, 54
control_forth_endof
Runtime Branch Helpers, 55
control_forth_EXIT
Runtime Branch Helpers, 55
control_forth_I
Runtime Branch Helpers, 56
control_forth_if
Runtime Branch Helpers, 57
control_forth_J
Runtime Branch Helpers, 57
control_forth_leave
Runtime Branch Helpers, 58
control_forth_loop
Runtime Branch Helpers, 59
control_forth_of
Runtime Branch Helpers, 59
control_forth_plus_loop
Runtime Branch Helpers, 60
control_forth_qdo
Runtime Branch Helpers, 61
control_forth_repeat
Runtime Branch Helpers, 62
control_forth_runtime_do
Runtime Branch Helpers, 62
control_forth_runtime_leave
Runtime Branch Helpers, 63
control_forth_runtime_loop
Runtime Branch Helpers, 63
control_forth_runtime_plus_loop
Runtime Branch Helpers, 64
control_forth_runtime_qdo
Runtime Branch Helpers, 64
control_forth_then
Runtime Branch Helpers, 65
control_forth_UNLOOP
Runtime Branch Helpers, 65
control_forth_until
Runtime Branch Helpers, 66
control_forth_while
Runtime Branch Helpers, 67
control_word_suites
control_words_test.c, 2608
control_words.c
emit_cell, 2836
control_words_test.c
control_word_suites, 2608

run_control_words_tests, 2607
ControlFlowState, 144
cf_last_mode, 144
cf_sp, 144
cf_stack, 145
endof_addrs, 145
endof_mark_sp, 145
endof_mark_stack, 145
endof_sp, 145
leave_addrs, 145
leave_mark_sp, 145
leave_mark_stack, 146
leave_sp, 146
converge_tsc_bare_metal
timer.c, 1020
converged
timer_calibration_record, 284
convert_string_to_number
string_words.c, 3213
ConvertPointer
_EFI_RUNTIME_SERVICES, 95
cos
math.h, 575
count
blkio_usb.c, 1508
sk_usb_ctrl_t, 267
sk_v4_identity.c, 1593
time_window, 283
VMFleetWindow, 341
counter_freq
rpi5_mailbox.c, 865
counter_now
rpi5_mailbox.c, 865
counters
Profiler, 249
cpu_count
physics_host_snapshot, 238
cpu_freq_delta_mhz_q48
DoeMetrics, 155
cpu_has_invariant_tsc
timer.c, 1021
cpu_has_rdtscp
timer.c, 1022
cpu_idle_jiffies
physics_host_snapshot, 238
cpu_temp_delta_c_q48
DoeMetrics, 156
cpu_total_jiffies
physics_host_snapshot, 238
cpuid
timer.c, 1023
crc
user_identity_seed_t, 302
zuse_cert_devblock_t, 394
zuse_genesis_marker_t, 395
zuse_genesis_marker_v2_t, 396
CRC32
EFI_TABLE_HEADER, 175
crc64_init
block_subsystem.c, 2191
crc64_inited
block_subsystem.c, 2212
crc64_table
block_subsystem.c, 2212
crcr
xhci_op_regs_t, 388
created_time
blk_meta_t, 109
blk_volume_meta_t, 112
CreateEvent
_EFI_BOOT_SERVICES, 87
CreateTime
EFI_FILE_INFO, 165
creator_id
__attribute__, 80
creator_revision
__attribute__, 80
credible_lower_95
BayesianLatencyPosterior, 103
credible_lower_95_q48
SpeedupEstimate, 271
credible_lower_99
BayesianLatencyPosterior, 103
credible_upper_95
BayesianLatencyPosterior, 103
credible_upper_95_q48
SpeedupEstimate, 271
credible_upper_99
BayesianLatencyPosterior, 103
ctx_cached_head
VocabularyState, 348
ctx_fc
VocabularyState, 349
ctx_n
VocabularyState, 349
ctype.h
isalnum, 565
isalpha, 566
iscntrl, 566
isdigit, 566
islower, 567
isprint, 567
ispunct, 567
isspace, 568
isupper, 568
isxdigit, 568
tolower, 569
toupper, 569
current_base
format_words.c, 2912
current_config
CDConfigTable, 136
current_el
rpi5_mmu.c, 873
current_executing_entry
VM, 323

current_level
log.c, 2299
current_mode
ssm_l8_state_t, 275
current_regime
CDConfigTable, 136
current_scr
editor_words.c, 2902
current_start_time
profiler.c, 2440
current_var_addr
VocabularyState, 349
current_vocab
VocabularyState, 349
current_word
profiler.c, 2440
current_word_name
VM, 323
cv
ssm_l8_metrics_t, 274
CV_EPSILON_PPM
timer.c, 1015
cv_hpet_ppm
timer_calibration_record, 284
cv_pit_ppm
timer_calibration_record, 284
CV_STABLE_COUNT
timer.c, 1015
cx
VT100State, 351
cy
VT100State, 351

D_BYTES
ed25519.c, 1205
d_compare
double_words.c, 2889
data
BlockBuffer, 125
cache_slot_t, 129
ttf_font_t, 289
Data1
EFI_GUID, 170
Data2
EFI_GUID, 170
Data3
EFI_GUID, 170
Data4
EFI_GUID, 170
data_buf
VirtBlkState, 305
VirtRngState, 317
data_stack
VM, 323
Day
EFI_TIME, 176
Daylight
EFI_TIME, 176
db_off
xhci_cap_regs_t, 365
dbg_get_base
vm_debug.c, 2729
dcbaa
xhci_dev_t, 373
dcbaap
xhci_op_regs_t, 388
dCBWDataTransferLength
usb_bot_cbw_t, 299
dCBWSignature
usb_bot_cbw_t, 299
dCBWTag
usb_bot_cbw_t, 299
dCSWDataResidue
usb_bot_csw_t, 300
dCSWSignature
usb_bot_csw_t, 300
dCSWTag
usb_bot_csw_t, 300
debug_checkpoint
uefi_loader.c, 1083
DEBUG_CHECKPOINT_STALL_US
uefi_loader.c, 1082
DECAY_MIN_INTERVAL
starforth_config.h, 1990
decay_min_interval_ns
DoeMetrics, 156
DECAY_RATE_PER_US_Q16
starforth_config.h, 1990
decay_rate_q16
DoeMetrics, 156
decay_slope
DoeMetrics, 156
decay_slope_direction
VM, 323
decay_slope_q48
HeartbeatSnapshot, 196
PhysicsSnapshot, 244
VM, 324
decode_composite_glyph
ttf.c, 1339
decode_glyph_r
ttf.c, 1339
decode_pf_error
interrupts.c, 979
decode_simple_glyph
ttf.c, 1340
def_bg
VT100State, 351
def_fg
VT100State, 351
defer_runtime
defer_words.c, 2837
defer_words.c
defer_runtime, 2837
register_defer_words, 2838
word_defer, 2839
word_defer_fetch, 2839

word_is, 2840
defer_words.h
register_defer_words, 2950
defining_runtime_constant
defining_words.c, 2842
defining_runtime_create
defining_words.c, 2842
defining_runtime_dodoes
defining_words.c, 2843
defining_runtime_does_rt
defining_words.c, 2844
defining_runtime_lit
defining_words.c, 2845
defining_runtime_variable
defining_words.c, 2846
defining_word_bracket_compile
defining_words.c, 2847
defining_word_colon
defining_words.c, 2847
defining_word_compile
defining_words.c, 2848
defining_word_constant
defining_words.c, 2849
defining_word_create
defining_words.c, 2850
defining_word_does
defining_words.c, 2851
defining_word_immediate
defining_words.c, 2852
defining_word_left_bracket
defining_words.c, 2853
defining_word_literal
defining_words.c, 2853
defining_word_right_bracket
defining_words.c, 2854
defining_word_semicolon
defining_words.c, 2854
defining_word_state
defining_words.c, 2855
defining_word_suites
defining_words_tests.c, 2610
defining_word_variable
defining_words.c, 2855
defining_words.c
defining_runtime_constant, 2842
defining_runtime_create, 2842
defining_runtime_dodoes, 2843
defining_runtime_does_rt, 2844
defining_runtime_lit, 2845
defining_runtime_variable, 2846
defining_word_bracket_compile, 2847
defining_word_colon, 2847
defining_word_compile, 2848
defining_word_constant, 2849
defining_word_create, 2850
defining_word_does, 2851
defining_word_immediate, 2852
defining_word_left_bracket, 2853
defining_word_literal, 2853
defining_word_right_bracket, 2854
defining_word_semicolon, 2854
defining_word_state, 2855
defining_word_variable, 2855
dictionary_word_fence, 2856
dictionary_word_forget, 2857
register_defining_words, 2858
defining_words.h
register_defining_words, 2952
defining_words_tests.c
defining_word_suites, 2610
run_defining_words_tests, 2609
Delete
_EFI_FILE_PROTOCOL, 91
deltas
time_window, 283
der_next
x509_ed25519.c, 1225
der_put_ed25519_algid
x509_ed25519.c, 1225
der_put_tlv
x509_ed25519.c, 1226
der_put_uint_integer
x509_ed25519.c, 1226
der_span_t, 146
len, 146
p, 146
derive_state_flags
physics_metadata.c, 2350
derive_tsc_hz_from_cpuid
timer.c, 1024
desc
VirtBlkState, 305
VirtInputState, 307
VirtRngState, 317
desc_capacity
CapsuleDirHeader, 130
desc_count
CapsuleDirHeader, 130
dev
blk_dev_slot, 106
wirebind_live_entry_t, 354
dev_rw4k
sk_v4_grow.c, 1569
devblock
cache_slot_t, 129
devblock4k_to_lba1k
block_subsystem.c, 2192
devblock_base
blk_volume_meta_t, 113
devblock_base_4k
blk_dev_slot, 106
devblock_claimed_by_lbn
block_subsystem.c, 2193
devblock_is_free
block_subsystem.c, 2194
devblocks

log_region_ctrl_t, 222
device
PciDevice, 227
device_ctx
xhci_msc_slot_t, 384
device_descriptor
xhci_msc_slot_t, 384
device_feature
VirtioCommonCfg, 309
device_feature_select
VirtioCommonCfg, 309
device_id
PciDevice, 227
device_status
VirtioCommonCfg, 309
DeviceHandle
EFI_LOADED_IMAGE_PROTOCOL, 171
diag_push
virtio_input.c, 1619
DIAG_RING_SIZE
virtio_input.c, 1617
DiagEvent, 147
code, 147
value, 147
dict_adaptive_optimization_pass
dictionary_heat_optimization.c, 2228
dictionary_heat_optimization.h, 1771
dict_fence_here
VM, 324
dict_fence_latest
VM, 324
dict_find_word_heat_aware
dictionary_heat_optimization.c, 2230
dictionary_heat_optimization.h, 1773
dict_lock
VM, 324
dict_lookups
Profiler, 249
dict_manip_word_suites
dictionary_manipulation_words_test.c, 2612
dict_newer_of
dictionary_management.c, 2237
dict_reorganize_buckets_by_heat
dictionary_heat_optimization.c, 2232
dictionary_heat_optimization.h, 1776
dict_update_heat_percentiles
dictionary_heat_optimization.c, 2234
dictionary_heat_optimization.h, 1778
DictEntry, 147
acl_allow, 148
acl_default, 148
acl_mode, 148
acl_pinned, 148
acl_ttl, 148
execution_heat, 149
flags, 149
func, 149
link, 149
name, 149
name_len, 149
physics, 150
profiler.c, 2427
profiler.h, 1907
transition_metrics, 150
word_id, 150
DICTIONARY_BLOCKS
vm.h, 2007
dictionary_clear
dictionary_management.h, 1781
dictionary_heat_diagnostic_words.c
forth_COMPARE_LOOKUPS, 2860
forth_HEAT_PERCENTILES, 2861
forth_LOOKUP_STRATEGY_FETCH, 2862
forth_LOOKUP_STRATEGY_STORE, 2862
forth_REORG_BUCKETS, 2863
forth_SHOW_HEAT_OPTIMIZATION, 2864
register_dictionary_heat_diagnostic_words, 2865
dictionary_heat_diagnostic_words.h
register_dictionary_heat_diagnostic_words, 2956
dictionary_heat_optimization.c
compare_dict_entry_heat_desc, 2227
compare_heat_ascending, 2227
dict_adaptive_optimization_pass, 2228
dict_find_word_heat_aware, 2230
dict_reorganize_buckets_by_heat, 2232
dict_update_heat_percentiles, 2234
dictionary_heat_optimization.h
dict_adaptive_optimization_pass, 1771
dict_find_word_heat_aware, 1773
dict_reorganize_buckets_by_heat, 1776
dict_update_heat_percentiles, 1778
SF_FC_BUCKETS, 1771
sf_fc_cap, 1779
sf_fc_count, 1779
sf_fc_list, 1779
dictionary_init
dictionary_management.h, 1781
dictionary_lookup
dictionary_management.h, 1781
dictionary_lookups
DoeMetrics, 156
HeartbeatState, 199
dictionary_m_word_cfa
dictionary_manipulation_words.c, 2868
dictionary_m_word_find
dictionary_manipulation_words.c, 2868
dictionary_m_word_hidden
dictionary_manipulation_words.c, 2869
dictionary_m_word_interpret
dictionary_manipulation_words.c, 2869
dictionary_m_word_lfa
dictionary_manipulation_words.c, 2870
dictionary_m_word_link_from
dictionary_manipulation_words.c, 2870
dictionary_m_word_name_to
dictionary_manipulation_words.c, 2871

dictionary_m_word_nfa
dictionary_manipulation_words.c, 2871
dictionary_m_word_pfa
dictionary_manipulation_words.c, 2872
dictionary_m_word_smudge
dictionary_manipulation_words.c, 2872
dictionary_m_word_tick
dictionary_manipulation_words.c, 2873
dictionary_m_word_to_body
dictionary_manipulation_words.c, 2873
dictionary_m_word_to_link
dictionary_manipulation_words.c, 2874
dictionary_m_word_to_name
dictionary_manipulation_words.c, 2874
dictionary_m_word_traverse
dictionary_manipulation_words.c, 2875
dictionary_management.c
dict_newer_of, 2237
sf_cached_latest, 2261
SF_FC_BUCKETS, 2237
sf_fc_cap, 2261
sf_fc_clear_all, 2238
sf_fc_count, 2261
sf_fc_list, 2261
sf_fc_reserve, 2239
sf_rebuild_fc_index, 2240
sf_try_fast_append, 2242
sf_xrealloc, 2244
vm_create_word, 2246
vm_dict_entry_ok, 2248
vm_dict_resolve_in_bucket, 2249
vm_dictionary_acquire_word_id, 2250
vm_dictionary_find_by_func, 2251
vm_dictionary_find_latest_by_func, 2252
vm_dictionary_get_data_field, 2252
vm_dictionary_index_reset, 2253
vm_dictionary_lookup_by_word_id, 2253
vm_dictionary_track_entry, 2254
vm_dictionary_untrack_entry, 2255
vm_find_word, 2256
vm_hide_word, 2258
vm_pin_execution_heat, 2259
vm_smudge_word, 2259
vm_unpin_execution_heat, 2260
dictionary_management.h
dictionary_clear, 1781
dictionary_init, 1781
dictionary_lookup, 1781
dictionary_register, 1781
dictionary_manipulation_words.c
align_address, 2868
dictionary_m_word_cfa, 2868
dictionary_m_word_find, 2868
dictionary_m_word_hidden, 2869
dictionary_m_word_interpret, 2869
dictionary_m_word_lfa, 2870
dictionary_m_word_link_from, 2870
dictionary_m_word_name_to, 2871
dictionary_m_word_nfa, 2871
dictionary_m_word_pfa, 2872
dictionary_m_word_smudge, 2872
dictionary_m_word_tick, 2873
dictionary_m_word_to_body, 2873
dictionary_m_word_to_link, 2874
dictionary_m_word_to_name, 2874
dictionary_m_word_traverse, 2875
find_entry_from_name, 2875
get_body_address, 2876
get_name_field, 2876
register_dictionary_manipulation_words, 2877
state_variable, 2879
traverse_name_field, 2878
dictionary_manipulation_words.h
register_dictionary_manipulation_words, 2958
dictionary_manipulation_words_test.c
dict_manip_word_suites, 2612
run_dictionary_manipulation_words_tests, 2611
DICTIONARY_MEMORY_SIZE
vm.h, 2007
dictionary_register
dictionary_management.h, 1781
DICTIONARY_SIZE
vm.h, 2007
dictionary_snapshot_acl_from_parent
capsule_birth.c, 1105
dictionary_word_2comma
dictionary_words.c, 2880
dictionary_word_align
dictionary_words.c, 2880
dictionary_word_allot
dictionary_words.c, 2881
dictionary_word_c_comma
dictionary_words.c, 2882
dictionary_word_comma
dictionary_words.c, 2882
dictionary_word_fence
defining_words.c, 2856
dictionary_word_forget
defining_words.c, 2857
dictionary_word_here
dictionary_words.c, 2883
dictionary_word_latest
dictionary_words.c, 2884
dictionary_word_pad
dictionary_words.c, 2885
dictionary_word_sp_fetch
dictionary_words.c, 2885
dictionary_word_sp_store
dictionary_words.c, 2886
dictionary_word_suites
dictionary_words_test.c, 2614
dictionary_words.c
dictionary_word_2comma, 2880
dictionary_word_align, 2880
dictionary_word_allot, 2881
dictionary_word_c_comma, 2882

dictionary_word_comma, 2882
dictionary_word_here, 2883
dictionary_word_latest, 2884
dictionary_word_pad, 2885
dictionary_word_sp_fetch, 2885
dictionary_word_sp_store, 2886
register_dictionary_words, 2887
dictionary_words.h
register_dictionary_words, 2960
dictionary_words_test.c
dictionary_word_suites, 2614
run_dictionary_words_tests, 2613
DictPhysics, 150
acl_hint, 151
avg_latency_ns, 151
last_active_ns, 151
last_decay_ns, 151
last_decay_tick, 151
mass_bytes, 151
pubsub_mask, 151
state_flags, 151
temperature_q8, 151
DIE
test_common.c, 2653
diff_ppm
timer_calibration_record, 284
digit_for
format_words.c, 2912
dir_hash
CapsuleDirHeader, 131
dirty
blk_bam_entry_t, 104
BlockBuffer, 125
cache_slot_t, 129
dirty_ram
block_subsystem.c, 2212
DisconnectController
_EFI_BOOT_SERVICES, 87
disk
gpt_part_dev_t, 193
DISK_CACHE_SLOTS
block_subsystem.c, 2151
disk_guid
BootDisk, 126
disk_img_path
CLIConfig, 142
disk_uuid
artemis_sig_t, 101
dispatch_csi
vt100.c, 1365
dispatch_init_forth
capsule_birth.c, 1105
div_ud_by_base
format_words.c, 2912
diversity_high_threshold_q16
CDTuning, 139
diversity_q16
CDRegimeMetrics, 138
dn_ctrl
xhci_op_regs_t, 388
DOE_BUF_SIZE
doe_log.c, 1232
doe_experiment
CLIConfig, 142
doe_header
doe_log.c, 1235
doe_log.c
DOE_BUF_SIZE, 1232
doe_header, 1235
doe_log_heat_by_name, 1232
doe_log_tick_row, 1232
DOE_PREFIX, 1232
doe_word_hb_off, 1234
doe_word_hb_on, 1234
g_doe_log_enabled, 1235
register_doe_log_words, 1234
doe_log.h
doe_log_tick_row, 508
g_doe_log_enabled, 510
register_doe_log_words, 509
doe_log_heat_by_name
doe_log.c, 1232
doe_log_tick_row
doe_log.c, 1232
doe_log.h, 508
doe_metrics.c
bucket_hits, 2269
cache_hits, 2269
extract_cache_metrics, 2262
isqrt, 2263
metrics_from_vm, 2263
metrics_get_cpu_freq_mhz, 2264
metrics_get_cpu_temp_c, 2265
metrics_get_timestamp, 2266
metrics_print_text, 2266
metrics_write_csv_header, 2267
metrics_write_csv_row, 2267
metrics_write_james_law_csv_row, 2268
physics_global_stats, 2269
total_lookups, 2269
doe_metrics.h
metrics_from_vm, 1784
metrics_get_cpu_freq_mhz, 1785
metrics_get_cpu_temp_c, 1786
metrics_get_timestamp, 1787
metrics_print_text, 1787
metrics_write_csv_header, 1788
metrics_write_csv_row, 1789
metrics_write_james_law_csv_row, 1790
DOE_PREFIX
doe_log.c, 1232
doe_row_printing
VM, 324
doe_word_hb_off
doe_log.c, 1234
doe_word_hb_on

doe_log.c, 1234
DoeMetrics, 152
actual_window_size, 153
adaptive_shrink_rate, 153
avg_word_heat, 153
bucket_hit_percent, 154
bucket_hits, 154
bucket_search_latency_ns, 154
bucket_search_stddev_ns, 154
cache_demotions, 154
cache_hit_latency_ns, 154
cache_hit_percent, 154
cache_hit_stddev_ns, 154
cache_hits, 155
cache_promotions, 155
ci_lower_95, 155
ci_upper_95, 155
context_accuracy_percent, 155
context_correct, 155
context_predictions_total, 155
cpu_freq_delta_mhz_q48, 155
cpu_temp_delta_c_q48, 156
decay_min_interval_ns, 156
decay_rate_q16, 156
decay_slope, 156
dictionary_lookups, 156
early_exit_count, 156
enable_hotwords_cache, 156
enable_loop_1_heat_tracking, 156
enable_loop_2_rolling_window, 157
enable_loop_3_linear_decay, 157
enable_loop_4_pipelining, 157
enable_loop_5_window_inference, 157
enable_loop_6_decay_inference, 157
enable_loop_7_adaptive_heartrate, 157
enable_pipelining, 157
final_effective_window_size, 157
heat_cache_demotion_threshold, 158
hot_word_count, 158
inference_run_count, 158
memory_allocated_bytes, 158
prefetch_accuracy_percent, 158
prefetch_attempts, 158
prefetch_hits, 158
rolling_window_size, 158
rolling_window_width, 159
speedup_vs_baseline, 159
stale_word_count, 159
stale_word_ratio, 159
tick_count, 159
tick_target_ns, 159
total_executions, 159
total_heat, 159
total_lookups, 160
total_runtime_ms, 160
vm_workload_duration_ns_q48, 160
window_diversity_percent, 160
window_final_size_bytes, 160
window_tuning_checks, 160
window_variance_q48, 160
words_executed, 160
doorbell
xhci_dev_t, 373
double_word_2drop
double_words.c, 2889
double_word_2dup
double_words.c, 2890
double_word_2over
double_words.c, 2890
double_word_2r_fetch
double_words.c, 2891
double_word_2r_from
double_words.c, 2891
double_word_2rot
double_words.c, 2892
double_word_2swap
double_words.c, 2892
double_word_2to_r
double_words.c, 2893
double_word_d_equals
double_words.c, 2893
double_word_d_less
double_words.c, 2894
double_word_d_minus
double_words.c, 2894
double_word_d_plus
double_words.c, 2895
double_word_d_two_slash
double_words.c, 2895
double_word_d_two_star
double_words.c, 2896
double_word_d_zero_equals
double_words.c, 2896
double_word_d_zero_less
double_words.c, 2897
double_word_dabs
double_words.c, 2897
double_word_dmax
double_words.c, 2898
double_word_dmin
double_words.c, 2898
double_word_dnegate
double_words.c, 2899
double_word_s_to_d
double_words.c, 2899
double_word_suites
double_words_test.c, 2616
double_words.c
d_compare, 2889
double_word_2drop, 2889
double_word_2dup, 2890
double_word_2over, 2890
double_word_2r_fetch, 2891
double_word_2r_from, 2891
double_word_2rot, 2892
double_word_2swap, 2892

double_word_2to_r, 2893
double_word_d_equals, 2893
double_word_d_less, 2894
double_word_d_minus, 2894
double_word_d_plus, 2895
double_word_d_two_slash, 2895
double_word_d_two_star, 2896
double_word_d_zero_equals, 2896
double_word_d_zero_less, 2897
double_word_dabs, 2897
double_word_dmax, 2898
double_word_dmin, 2898
double_word_dnegate, 2899
double_word_s_to_d, 2899
register_double_words, 2900
double_words.h
register_double_words, 2962
double_words_test.c
double_word_suites, 2616
run_double_words_tests, 2615
draw_cursor_glyph
vt100.c, 1366
DRIFT_EPSILON_PPM
timer.c, 1015
drive_uuid
homeblocks_sig_t, 205
driver_feature
VirtioCommonCfg, 309
driver_feature_select
VirtioCommonCfg, 310
drop_flags
xhci_input_ctrl_ctx32_t, 379
dropped_events
physics_analytics_header, 230
dsp
ExecResult, 184
VM, 324
dtb
BootInfo, 127
dump_and_die
test_common.c, 2655
dump_block_preview
vm_debug.c, 2729
dword0
xhci_ep_ctx32_t, 378
xhci_slot_ctx32_t, 390
dword1
xhci_ep_ctx32_t, 378
xhci_slot_ctx32_t, 390
dword2
xhci_slot_ctx32_t, 391
dword3
xhci_slot_ctx32_t, 391
dword4
xhci_ep_ctx32_t, 378

e_ehsize
Elf64_Ehdr, 177
e_entry
Elf64_Ehdr, 177
e_flags
Elf64_Ehdr, 177
e_ident
Elf64_Ehdr, 177
e_machine
Elf64_Ehdr, 178
e_phentsize
Elf64_Ehdr, 178
e_phnum
Elf64_Ehdr, 178
e_phoff
Elf64_Ehdr, 178
e_shentsize
Elf64_Ehdr, 178
e_shnum
Elf64_Ehdr, 178
e_shoff
Elf64_Ehdr, 178
e_shstrndx
Elf64_Ehdr, 178
e_type
Elf64_Ehdr, 179
e_version
Elf64_Ehdr, 179
early_exit_count
DoeMetrics, 156
HeartbeatState, 199
early_exited
InferenceOutputs, 217
ecam_cfg_ptr
pci.c, 1486
ecam_read16
pci.c, 1487
ecam_read32
pci.c, 1487
ecam_read8
pci.c, 1488
ecam_write16
pci.c, 1489
ecam_write32
pci.c, 1489
ecam_write8
pci.c, 1490
ecw_nesting
VM, 325
ed25519.c
base_point, 1196
BX_BYTES, 1205
BY_BYTES, 1205
clamp_scalar, 1196
D_BYTES, 1205
ed25519_keygen, 1197
ed25519_sign, 1197
ed25519_verify, 1198
expand_seed, 1200
fe25519_pow, 1200
point_add, 1201

point_compress, 1202
point_decompress, 1202
point_identity, 1203
POW_2252M2_EXP, 1205
scalar_mult, 1204
SQRTM1_BYTES, 1206
ed25519.h
ed25519_keygen, 513
ed25519_sign, 513
ed25519_verify, 514
ed25519_keygen
ed25519.c, 1197
ed25519.h, 513
ed25519_sign
ed25519.c, 1197
ed25519.h, 513
ed25519_verify
ed25519.c, 1198
ed25519.h, 514
ed_point_t, 161
x, 161
y, 161
z, 162
editor_word_edit
editor_words.c, 2903
editor_word_l
editor_words.c, 2904
editor_word_s
editor_words.c, 2905
editor_word_show
editor_words.c, 2906
editor_words.c
current_scr, 2902
editor_word_edit, 2903
editor_word_l, 2904
editor_word_s, 2905
editor_word_show, 2906
line_ptr, 2906
print_line_64, 2907
register_editor_words, 2908
editor_words.h
register_editor_words, 2964
eff_bg
vt100.c, 1368
eff_fg
vt100.c, 1369
eff_window_size
PhysicsSnapshot, 244
effective_window_size
rolling_window_view_t, 255
RollingWindowOfTruth, 256
EFI_ABORTED
uefi.h, 721
EFI_ACPI_20_TABLE_GUID
uefi.h, 732
EFI_ACPI_TABLE_GUID
uefi.h, 732
EFI_ALLOCATE_PAGES
uefi.h, 725
EFI_ALLOCATE_POOL
uefi.h, 725
EFI_ALLOCATE_TYPE
uefi.h, 730
EFI_BAD_BUFFER_SIZE
uefi.h, 721
EFI_BLOCK_IO_MEDIA, 162
BlockSize, 162
IoAlign, 162
LastBlock, 162
LogicalPartition, 163
MediaId, 163
MediaPresent, 163
ReadOnly, 163
RemovableMedia, 163
WriteCaching, 163
EFI_BLOCK_IO_PROTOCOL
uefi.h, 725
EFI_BLOCK_IO_PROTOCOL_GUID
uefi.h, 732
EFI_BLOCK_READ
uefi.h, 725
EFI_BOOT_SERVICES
uefi.h, 726
EFI_BUFFER_TOO_SMALL
uefi.h, 721
EFI_CONFIGURATION_TABLE, 163
VendorGuid, 164
VendorTable, 164
EFI_DEVICE_ERROR
uefi.h, 721
EFI_DEVICE_PATH_PROTOCOL, 164
Length, 164
SubType, 164
Type, 164
EFI_DEVICE_PATH_PROTOCOL_GUID
uefi.h, 732
EFI_DEVICE_PATH_SUBTYPE_END_ENTIRE
uefi.h, 721
EFI_DEVICE_PATH_SUBTYPE_HARDDRIVE
uefi.h, 721
EFI_DEVICE_PATH_TYPE_END
uefi.h, 721
EFI_DEVICE_PATH_TYPE_MEDIA
uefi.h, 722
EFI_DTB_TABLE_GUID
uefi.h, 732
EFI_EXIT_BOOT_SERVICES
uefi.h, 726
EFI_FILE_CLOSE
uefi.h, 726
EFI_FILE_GET_INFO
uefi.h, 726
EFI_FILE_INFO, 165
Attribute, 165
CreateTime, 165
FileName, 166

FileSize, 166
LastAccessTime, 166
ModificationTime, 166
PhysicalSize, 166
Size, 166
EFI_FILE_INFO_GUID
uefi.h, 732
EFI_FILE_MODE_READ
uefi.h, 722
EFI_FILE_OPEN
uefi.h, 726
EFI_FILE_PROTOCOL
uefi.h, 726
EFI_FILE_READ
uefi.h, 726
EFI_FREE_PAGES
uefi.h, 726
EFI_FREE_POOL
uefi.h, 726
EFI_GET_MEMORY_MAP
uefi.h, 726
EFI_GET_VARIABLE
uefi.h, 727
EFI_GRAPHICS_OUTPUT_MODE_INFORMATION, 167
HorizontalResolution, 167
PixelFormat, 167
PixelInformation, 167
PixelsPerScanLine, 167
Version, 168
VerticalResolution, 168
EFI_GRAPHICS_OUTPUT_PROTOCOL
uefi.h, 727
EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID
uefi.h, 733
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE, 168
FrameBufferBase, 169
FrameBufferSize, 169
Info, 169
MaxMode, 169
Mode, 169
SizeOfInfo, 169
EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE
uefi.h, 727
EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE
uefi.h, 727
EFI_GRAPHICS_PIXEL_FORMAT
uefi.h, 731
EFI_GUID, 170
Data1, 170
Data2, 170
Data3, 170
Data4, 170
EFI_HANDLE
uefi.h, 727
EFI_HANDLE_PROTOCOL
uefi.h, 727
EFI_INVALID_PARAMETER
uefi.h, 722
EFI_LOAD_ERROR
uefi.h, 722
EFI_LOADED_IMAGE_PROTOCOL, 170
DeviceHandle, 171
FilePath, 171
ImageBase, 171
ImageCodeType, 171
ImageDataType, 171
ImageSize, 171
LoadOptions, 171
LoadOptionsSize, 172
ParentHandle, 172
Reserved, 172
Revision, 172
SystemTable, 172
Unload, 172
EFI_LOADED_IMAGE_PROTOCOL_GUID
uefi.h, 733
EFI_LOCATE_DEVICE_PATH
uefi.h, 727
EFI_LOCATE_HANDLE
uefi.h, 727
EFI_LOCATE_PROTOCOL
uefi.h, 727
EFI_LOCATE_SEARCH_TYPE
uefi.h, 731
efi_main
uefi_loader.c, 1083
EFI_MEMORY_DESCRIPTOR, 172
Attribute, 173
NumberOfPages, 173
PhysicalStart, 173
Type, 173
VirtualStart, 173
EFI_MEMORY_RP
uefi.h, 722
EFI_MEMORY_RUNTIME
uefi.h, 722
EFI_MEMORY_TYPE
uefi.h, 731
EFI_MEMORY_UC
uefi.h, 722
EFI_MEMORY_UCE
uefi.h, 722
EFI_MEMORY_WB
uefi.h, 722
EFI_MEMORY_WC
uefi.h, 722
EFI_MEMORY_WP
uefi.h, 723
EFI_MEMORY_WT
uefi.h, 723
EFI_MEMORY_XP
uefi.h, 723
EFI_NOT_FOUND
uefi.h, 723
EFI_NOT_READY

uefi.h, 723
EFI_OUT_OF_RESOURCES
uefi.h, 723
EFI_PHYSICAL_ADDRESS
uefi.h, 728
EFI_PIXEL_BITMASK, 173
BlueMask, 174
GreenMask, 174
RedMask, 174
ReservedMask, 174
EFI_RAISE_TPL
uefi.h, 728
EFI_RESET_SYSTEM
uefi.h, 728
EFI_RESET_TYPE
uefi.h, 731
EFI_RESTORE_TPL
uefi.h, 728
EFI_RUNTIME_SERVICES
uefi.h, 728
EFI_SET_VARIABLE
uefi.h, 728
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL
uefi.h, 728
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID
uefi.h, 733
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME
uefi.h, 728
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL
uefi.h, 728
EFI_STALL
uefi.h, 728
EFI_STATUS
uefi.h, 729
EFI_SUCCESS
uefi.h, 723
EFI_SYSTEM_TABLE
uefi.h, 729
EFI_TABLE_HEADER, 174
CRC32, 175
HeaderSize, 175
Reserved, 175
Revision, 175
Signature, 175
EFI_TEXT_RESET
uefi.h, 729
EFI_TEXT_STRING
uefi.h, 729
EFI_TIME, 175
Day, 176
Daylight, 176
Hour, 176
Minute, 176
Month, 176
Nanosecond, 176
Pad1, 176
Pad2, 176
Second, 176
TimeZone, 176
Year, 176
EFI_TPL
uefi.h, 729
EFI_UNSUPPORTED
uefi.h, 723
EFI_VARIABLE_BOOTSERVICE_ACCESS
uefi.h, 723
EFI_VARIABLE_NON_VOLATILE
uefi.h, 723
EFI_VARIABLE_RUNTIME_ACCESS
uefi.h, 724
EFI_WRITE_PROTECTED
uefi.h, 724
EfiACPIMemoryNVS
uefi.h, 731
EfiACPIReclaimMemory
uefi.h, 731
EFIAPI
uefi.h, 724
EfiBootServicesCode
uefi.h, 731
EfiBootServicesData
uefi.h, 731
EfiConventionalMemory
uefi.h, 731
EfiLoaderCode
uefi.h, 731
EfiLoaderData
uefi.h, 731
EfiMaxMemoryType
uefi.h, 731
EfiMemoryMappedIO
uefi.h, 731
EfiMemoryMappedIOPortSpace
uefi.h, 731
EfiPalCode
uefi.h, 731
EfiPersistentMemory
uefi.h, 731
EfiReservedMemoryType
uefi.h, 731
EfiResetCold
uefi.h, 732
EfiResetShutdown
uefi.h, 732
EfiResetWarm
uefi.h, 732
EfiRuntimeServicesCode
uefi.h, 731
EfiRuntimeServicesData
uefi.h, 731
EfiUnusableMemory
uefi.h, 731
EI_ABIVERSION
elf64.h, 518
EI_CLASS
elf64.h, 518

EI_DATA
elf64.h, 518
EI_MAG0
elf64.h, 518
EI_MAG1
elf64.h, 519
EI_MAG2
elf64.h, 519
EI_MAG3
elf64.h, 519
EI_NIDENT
elf64.h, 519
EI_OSABI
elf64.h, 519
EI_PAD
elf64.h, 519
EI_VERSION
elf64.h, 519
EINVAL
errno.h, 570
elapsed_ns
HeartbeatTickSnapshot, 203
elapsed_ticks
VMFleetTouchSample, 340
elf64.h
EI_ABIVERSION, 518
EI_CLASS, 518
EI_DATA, 518
EI_MAG0, 518
EI_MAG1, 519
EI_MAG2, 519
EI_MAG3, 519
EI_NIDENT, 519
EI_OSABI, 519
EI_PAD, 519
EI_VERSION, 519
Elf64_Addr, 525
Elf64_Half, 525
Elf64_Off, 525
ELF64_R_INFO, 519
ELF64_R_SYM, 519
ELF64_R_TYPE, 520
Elf64_Sword, 525
Elf64_Sxword, 525
Elf64_Word, 526
Elf64_Xword, 526
ELFCLASS32, 520
ELFCLASS64, 520
ELFDATA2LSB, 520
ELFDATA2MSB, 520
ELFMAG0, 520
ELFMAG1, 520
ELFMAG2, 520
ELFMAG3, 520
ELFOSABI_NONE, 520
EM_AARCH64, 521
EM_RISCV, 521
EM_X86_64, 521
ET_CORE, 521
ET_DYN, 521
ET_EXEC, 521
ET_NONE, 521
ET_REL, 521
EV_CURRENT, 521
PF_R, 521
PF_W, 522
PF_X, 522
PT_DYNAMIC, 522
PT_INTERP, 522
PT_LOAD, 522
PT_NOTE, 522
PT_NULL, 522
PT_PHDR, 522
PT_SHLIB, 522
PT_TLS, 522
R_AARCH64_ABS64, 523
R_AARCH64_NONE, 523
R_AARCH64_RELATIVE, 523
R_RISCV_64, 523
R_RISCV_NONE, 523
R_RISCV_RELATIVE, 523
R_X86_64_32, 523
R_X86_64_32S, 523
R_X86_64_64, 523
R_X86_64_NONE, 523
R_X86_64_PC32, 524
R_X86_64_PLT32, 524
R_X86_64_RELATIVE, 524
SHT_DYNAMIC, 524
SHT_DYNSYM, 524
SHT_HASH, 524
SHT_NOBITS, 524
SHT_NOTE, 524
SHT_NULL, 524
SHT_PROGBITS, 524
SHT_REL, 525
SHT_RELA, 525
SHT_SHLIB, 525
SHT_STRTAB, 525
SHT_SYMTAB, 525
Elf64_Addr
elf64.h, 525
Elf64_Ehdr, 177
e_ehsize, 177
e_entry, 177
e_flags, 177
e_ident, 177
e_machine, 178
e_phentsize, 178
e_phnum, 178
e_phoff, 178
e_shentsize, 178
e_shnum, 178
e_shoff, 178
e_shstrndx, 178
e_type, 179

e_version, 179
Elf64_Half
elf64.h, 525
Elf64_Off
elf64.h, 525
Elf64_Phdr, 179
p_align, 179
p_filesz, 179
p_flags, 179
p_memsz, 180
p_offset, 180
p_paddr, 180
p_type, 180
p_vaddr, 180
ELF64_R_INFO
elf64.h, 519
ELF64_R_SYM
elf64.h, 519
ELF64_R_TYPE
elf64.h, 520
Elf64_Rel, 180
r_info, 180
r_offset, 180
Elf64_Rela, 181
r_addend, 181
r_info, 181
r_offset, 181
Elf64_Shdr, 181
sh_addr, 182
sh_addralign, 182
sh_entsize, 182
sh_flags, 182
sh_info, 182
sh_link, 182
sh_name, 182
sh_offset, 182
sh_size, 183
sh_type, 183
Elf64_Sword
elf64.h, 525
Elf64_Sxword
elf64.h, 525
Elf64_Sym, 183
st_info, 183
st_name, 183
st_other, 183
st_shndx, 184
st_size, 184
st_value, 184
Elf64_Word
elf64.h, 526
Elf64_Xword
elf64.h, 526
elf_apply_relocations
elf_loader.c, 1072
elf_load_kernel
elf_loader.c, 1073
elf_loader.h, 527
elf_load_segments
elf_loader.c, 1075
elf_loader.c
elf_apply_relocations, 1072
elf_load_kernel, 1073
elf_load_segments, 1075
elf_memcpy, 1076
elf_memset, 1077
elf_validate_header, 1077
elf_loader.h
elf_load_kernel, 527
elf_memcpy
elf_loader.c, 1076
elf_memset
elf_loader.c, 1077
elf_validate_header
elf_loader.c, 1077
ELFCLASS32
elf64.h, 520
ELFCLASS64
elf64.h, 520
ELFDATA2LSB
elf64.h, 520
ELFDATA2MSB
elf64.h, 520
ELFMAG0
elf64.h, 520
ELFMAG1
elf64.h, 520
ELFMAG2
elf64.h, 520
ELFMAG3
elf64.h, 520
ELFOSABI_NONE
elf64.h, 520
EM_AARCH64
elf64.h, 521
EM_RISCV
elf64.h, 521
EM_X86_64
elf64.h, 521
email
user_identity_seed_t, 302
emergency_console
VM, 326
emit_cell
control_words.c, 2836
emit_prefix
console.c, 1272
emit_region_with_carveouts
rpi5_native_boot.c, 883
empty_all_buffers
block_words.c, 2827
block_words.h, 2943
enable_benchmark_mode
test_runner.c, 2687
test_runner.h, 2548
ENABLE_HOTWORDS_CACHE

starforth_config.h, 1990
enable_hotwords_cache
DoeMetrics, 156
enable_hpet
timer.c, 1025
enable_interrupt_route
virtio_input.c, 1619
enable_loop_1_heat_tracking
DoeMetrics, 156
enable_loop_2_rolling_window
DoeMetrics, 157
enable_loop_3_linear_decay
DoeMetrics, 157
enable_loop_4_pipelining
DoeMetrics, 157
enable_loop_5_window_inference
DoeMetrics, 157
enable_loop_6_decay_inference
DoeMetrics, 157
enable_loop_7_adaptive_heartrate
DoeMetrics, 157
ENABLE_PIPELINING
starforth_config.h, 1990
enable_pipelining
DoeMetrics, 157
EnableCursor
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 97
enabled
HotwordsCache_s, 207
profiler.c, 2441
encoding
blk_meta_t, 109
end_bus
__attribute__, 80
ended_ns
CapsuleRunRecord, 132
endof_addrs
ControlFlowState, 145
endof_mark_sp
ControlFlowState, 145
endof_mark_stack
ControlFlowState, 145
endof_sp
ControlFlowState, 145
ENOENT
errno.h, 570
ENOMEM
errno.h, 570
ensure_input
string_words.c, 3213
entries
CDConfigTable, 137
entropy
blk_meta_t, 109
ssm_l8_metrics_t, 274
entry
reloc_block, 254
reloc_stub.c, 1080
vm_node, 338
WordStats, 358
ep0_ring
xhci_msc_slot_t, 384
ep0_ring_cycle
xhci_msc_slot_t, 384
ep0_ring_enq
xhci_msc_slot_t, 384
EPERM
errno.h, 570
epoch
block_subsystem.c, 2212
erase_display
vt100.c, 1370
erase_line
vt100.c, 1371
erase_line_range
vt100.c, 1372
erdp
xhci_intr_regs_t, 380
erf_q48
math_portable.c, 2312
math_portable.h, 1814
errno
errno.h, 570
errno.h
_freestanding_errno, 571
EINVAL, 570
ENOENT, 570
ENOMEM, 570
EPERM, 570
errno, 570
error
ExecResult, 184
sbiret_t, 260
VM, 326
erstba
xhci_intr_regs_t, 380
erstsz
xhci_intr_regs_t, 380
estimated_jitter_ns
HeartbeatTickSnapshot, 203
ET_CORE
elf64.h, 521
ET_DYN
elf64.h, 521
ET_EXEC
elf64.h, 521
ET_NONE
elf64.h, 521
ET_REL
elf64.h, 521
EV_CURRENT
elf64.h, 521
EVENTQ_SIZE
virtio_input.c, 1617
evict_next
ttf_raster_cache_t, 297

evictions
HotwordsStats, 211
evt_ring
xhci_dev_t, 374
evt_ring_cycle
xhci_dev_t, 374
evt_ring_deq
xhci_dev_t, 374
evt_ring_seg_table
xhci_dev_t, 374
exec_block_with_retry
capsule_loader.c, 1124
exec_results_equal
test_contracts.c, 2680
test_contracts.h, 2541
ExecResult, 184
dsp, 184
error, 184
rsp, 185
stack, 185
executable
vmm_page_info_t, 344
execute_colon_word
vm.c, 2696
vm.h, 2012
vm_core.c, 1652
vm_internal.h, 2748
execution_heat
DictEntry, 149
vm.h, 2075
execution_heat_q48
VMPhysics, 345
execution_history
RollingWindowOfTruth, 256
Exit
_EFI_BOOT_SERVICES, 87
exit_colon
VM, 327
ExitBootServices
_EFI_BOOT_SERVICES, 87
exp
math.h, 575
exp_q48
math_portable.c, 2314
expand_seed
ed25519.c, 1200
expected
TestCase, 277
expected_delta
time_trust_state, 282
extended_checksum
__attribute__, 81
extract_cache_metrics
doe_metrics.c, 2262
extract_heat_trajectory
inference_engine.c, 2279
extract_name
lifecycle_words_hosted.c, 3063
fabs
math.h, 575
fadt_find_pm_tmr_port
timer.c, 1025
fail_and_exit
test_common.c, 2656
fail_fast
CLIConfig, 143
test_common.c, 2673
test_common.h, 2533
failed
sk_v4_mint.c, 1601
FALSE
uefi.h, 724
Fast Primitive Word Implementations, 37
PRIM_2DROP_ARM64, 38
PRIM_2DUP_ARM64, 38
PRIM_AND_ARM64, 39
PRIM_BRANCH_ARM64, 39
PRIM_C_FETCH_ARM64, 39
PRIM_C_STORE_ARM64, 39
PRIM_DROP_ARM64, 39
PRIM_DUP_ARM64, 40
PRIM_EQUALS_ARM64, 40
PRIM_EXECUTE_ARM64, 40
PRIM_FETCH_ARM64, 40
PRIM_GREATER_ARM64, 40
PRIM_INVERT_ARM64, 41
PRIM_LESS_ARM64, 41
PRIM_MINUS_ARM64, 41
PRIM_MOD_ARM64, 41
PRIM_NEGATE_ARM64, 41
PRIM_OR_ARM64, 42
PRIM_OVER_ARM64, 42
PRIM_PLUS_ARM64, 42
PRIM_R_FETCH_ARM64, 42
PRIM_R_FROM_ARM64, 42
PRIM_ROT_ARM64, 43
PRIM_SLASH_ARM64, 43
PRIM_STAR_ARM64, 43
PRIM_STORE_ARM64, 43
PRIM_SWAP_ARM64, 43
PRIM_TO_R_ARM64, 44
PRIM_XOR_ARM64, 44
PRIM_ZBRANCH_ARM64, 44
PRIM_ZERO_EQUALS_ARM64, 44
PRIM_ZERO_LESS_ARM64, 44
FB_B
framebuffer.h, 554
fb_cell_h
framebuffer.c, 1313
framebuffer.h, 555
fb_cell_w
framebuffer.c, 1313
framebuffer.h, 556
fb_draw_glyph
framebuffer.c, 1314
framebuffer.h, 556

fb_draw_orientation_test
framebuffer.c, 1315
framebuffer.h, 557
fb_fill_rect
framebuffer.c, 1315
framebuffer.h, 558
FB_G
framebuffer.h, 554
fb_height
framebuffer.c, 1316
framebuffer.h, 558
fb_init
framebuffer.c, 1317
framebuffer.h, 559
fb_is_available
framebuffer.c, 1317
framebuffer.h, 559
fb_mirrored
console.c, 1273
FB_PIXEL_BGRX32
framebuffer.h, 555
FB_PIXEL_RGBX32
framebuffer.h, 555
fb_put_pixel
framebuffer.c, 1318
framebuffer.h, 560
FB_R
framebuffer.h, 554
FB_RGB
framebuffer.h, 555
fb_scroll_rect
framebuffer.c, 1319
framebuffer.h, 561
fb_scroll_rows
framebuffer.c, 1319
framebuffer.h, 561
fb_width
framebuffer.c, 1321
framebuffer.h, 563
fblocks
gpt_part_dev_t, 193
FbPixelFormat
framebuffer.h, 555
fbs
blkcfg_t, 116
blkio_file_state_t, 118
blkio_ram_state_t, 121
FbState, 185
base, 185
fmt, 185
height, 185
ready, 186
scale, 186
stride, 186
width, 186
fbw_height
framebuffer_words.c, 2927
fbw_plot
framebuffer_words.c, 2928
fbw_width
framebuffer_words.c, 2928
fc_free
vocabulary_words.c, 3250
fc_rebuild
vocabulary_words.c, 3251
fclose
stdio.h, 581
fd
blkio_file_state_t, 118
fdt.c
be32, 1287
compat_list_contains, 1287
FDT_BEGIN_NODE, 1286
fdt_complete, 1288
FDT_END, 1286
FDT_END_NODE, 1286
fdt_find_child, 1289
fdt_find_node_by_compatible, 1290
fdt_find_node_by_device_type, 1291
fdt_find_node_by_name, 1292
fdt_find_prop, 1293
fdt_find_prop_in_node, 1294
FDT_MAGIC, 1286
fdt_next_child_node, 1295
FDT_NOP, 1286
FDT_PATH_MAX, 1286
FDT_PROP, 1286
fdt_prop_u32, 1297
fdt_reg_address, 1298
fdt_root_node, 1299
fdt_valid, 1300
node_cells, 1301
node_enabled, 1302
node_name_eq, 1303
path_to, 1303
read_cells_at, 1304
skip_node_body, 1304
str_eq, 1305
fdt.h
fdt_complete, 529
fdt_find_child, 530
fdt_find_node_by_compatible, 531
fdt_find_node_by_device_type, 532
fdt_find_node_by_name, 533
fdt_find_prop, 534
fdt_find_prop_in_node, 535
fdt_next_child_node, 536
fdt_prop_u32, 537
fdt_reg_address, 538
fdt_root_node, 539
fdt_valid, 540
FDT_BEGIN_NODE
fdt.c, 1286
fdt_complete
fdt.c, 1288
fdt.h, 529

fdt_console_uart
fdt_uart.c, 1309
fdt_uart.h, 543
FDT_END
fdt.c, 1286
FDT_END_NODE
fdt.c, 1286
fdt_find_child
fdt.c, 1289
fdt.h, 530
fdt_find_node_by_compatible
fdt.c, 1290
fdt.h, 531
fdt_find_node_by_device_type
fdt.c, 1291
fdt.h, 532
fdt_find_node_by_name
fdt.c, 1292
fdt.h, 533
fdt_find_prop
fdt.c, 1293
fdt.h, 534
fdt_find_prop_in_node
fdt.c, 1294
fdt.h, 535
fdt_header_t, 186
boot_cpuid_phys, 187
last_comp_version, 187
magic, 187
off_dt_strings, 187
off_dt_struct, 187
off_mem_rsvmap, 187
size_dt_strings, 187
size_dt_struct, 187
totalsize, 187
version, 188
FDT_MAGIC
fdt.c, 1286
fdt_next_child_node
fdt.c, 1295
fdt.h, 536
FDT_NOP
fdt.c, 1286
FDT_PATH_MAX
fdt.c, 1286
FDT_PROP
fdt.c, 1286
fdt_prop_u32
fdt.c, 1297
fdt.h, 537
fdt_reg_address
fdt.c, 1298
fdt.h, 538
fdt_root_node
fdt.c, 1299
fdt.h, 539
fdt_uart.c
be32, 1307
cell_prop, 1307
cells, 1308
compatible_has, 1309
fdt_console_uart, 1309
PATH_DEPTH, 1307
fdt_uart.h
fdt_console_uart, 543
FDT_UART_16550, 543
FDT_UART_NONE, 543
FDT_UART_PL011, 543
FDT_UART_16550
fdt_uart.h, 543
FDT_UART_NONE
fdt_uart.h, 543
FDT_UART_PL011
fdt_uart.h, 543
fdt_uart_t, 188
base, 188
io_width, 188
kind, 188
reg_shift, 189
fdt_valid
fdt.c, 1300
fdt.h, 540
fe25519, 189
v, 189
fe25519.c
fe25519_0, 1208
fe25519_1, 1208
fe25519_add, 1209
fe25519_carry, 1209
fe25519_copy, 1209
fe25519_eq, 1210
fe25519_invert, 1210
fe25519_mul, 1211
fe25519_mul_small, 1211
fe25519_neg, 1212
fe25519_pack, 1212
fe25519_parity, 1213
fe25519_sq, 1213
fe25519_sub, 1214
fe25519_unpack, 1215
MASK51, 1207
u128, 1207
WBITS, 1207
fe25519.h
fe25519_0, 545
fe25519_1, 546
fe25519_add, 546
fe25519_copy, 547
fe25519_eq, 547
fe25519_invert, 548
fe25519_mul, 548
fe25519_mul_small, 549
fe25519_neg, 549
fe25519_pack, 550
fe25519_parity, 551
fe25519_sq, 551

fe25519_sub, 552
fe25519_unpack, 552
fe25519_0
fe25519.c, 1208
fe25519.h, 545
fe25519_1
fe25519.c, 1208
fe25519.h, 546
fe25519_add
fe25519.c, 1209
fe25519.h, 546
fe25519_carry
fe25519.c, 1209
fe25519_copy
fe25519.c, 1209
fe25519.h, 547
fe25519_eq
fe25519.c, 1210
fe25519.h, 547
fe25519_invert
fe25519.c, 1210
fe25519.h, 548
fe25519_mul
fe25519.c, 1211
fe25519.h, 548
fe25519_mul_small
fe25519.c, 1211
fe25519.h, 549
fe25519_neg
fe25519.c, 1212
fe25519.h, 549
fe25519_pack
fe25519.c, 1212
fe25519.h, 550
fe25519_parity
fe25519.c, 1213
fe25519.h, 551
fe25519_pow
ed25519.c, 1200
fe25519_sq
fe25519.c, 1213
fe25519.h, 551
fe25519_sub
fe25519.c, 1214
fe25519.h, 552
fe25519_unpack
fe25519.c, 1215
fe25519.h, 552
fflush
stdio.h, 581
fg
VT100State, 351
fgetc
stdio.h, 582
fgets
stdio.h, 582
FILE, 189
_dummy, 190
file_close
blkio_file.c, 2135
file_flush
blkio_file.c, 2135
file_info
blkio_file.c, 2136
file_open
blkio_file.c, 2136
file_path
blkcfg_t, 116
file_read
blkio_file.c, 2137
file_write
blkio_file.c, 2138
FileName
EFI_FILE_INFO, 166
FilePath
EFI_LOADED_IMAGE_PROTOCOL, 171
FileSize
EFI_FILE_INFO, 166
final_effective_window_size
DoeMetrics, 157
finalize_report
break_me_tests.c, 2604
find_and_cache_pubkey
capsule_sig.c, 1149
find_contiguous_free
pmm.c, 1460
find_entry_from_name
dictionary_manipulation_words.c, 2875
find_table
ttf.c, 1341
find_variance_inflection
inference_engine.c, 2280
inference_engine.h, 1793
FirmwareRevision
_EFI_SYSTEM_TABLE, 100
FirmwareVendor
_EFI_SYSTEM_TABLE, 100
first_disk_slot
block_subsystem.c, 2194
first_fblock
gpt_part_dev_t, 194
first_free
blk_volume_meta_t, 113
first_lba
gpt_part_t, 194
fits
ttf.c, 1342
flags
__attribute__, 81
blk_meta_t, 109
blk_volume_meta_t, 113
CapsuleRunRecord, 133
DictEntry, 149
physics_analytics_header, 231
physics_host_snapshot, 238
VirtqAvail, 312

VirtqDesc, 313
VirtqUsed, 314
vm.h, 2076
VMRegistryEntry, 346
WordContract, 355
flat_push
ttf.c, 1343
flatten_contour
ttf.c, 1343
flatten_quad
ttf.c, 1344
fleet_heartbeat_tick_count
capsule_vm_physics.c, 1170
fleet_last_capacity_tick
capsule_vm_physics.c, 1170
fleet_transfer_slope_q48
capsule_vm_physics.c, 1171
fleet_window
capsule_vm_physics.c, 1171
floor
math.h, 576
Flush
_EFI_FILE_PROTOCOL, 91
flush
blkio_vtable_t, 122
FlushBlocks
_EFI_BLOCK_IO_PROTOCOL, 84
fmt
FbState, 185
fnv1a_64
parity.c, 1635
parity.h, 770
FNV1A_64_OFFSET_BASIS
parity.h, 769
FNV1A_64_PRIME
parity.h, 769
fnv1a_64_u32
parity.c, 1636
fnv1a_64_u8
parity.c, 1637
font
ttf_raster_cache_slot_t, 296
font_8x16.c
font_8x16_data, 1311
font_8x16_glyph, 1311
font_8x16_data
font_8x16.c, 1311
font_8x16_glyph
font_8x16.c, 1311
framebuffer.c, 1322
fopen
stdio.h, 583
format_state
blk_dev_slot, 106
format_timestamp
sf_time_backend_t, 262
format_word_base
format_words.c, 2913
format_word_begin_conversion
format_words.c, 2913
format_word_d_dot
format_words.c, 2914
format_word_d_dot_r
format_words.c, 2914
format_word_decimal
format_words.c, 2915
format_word_dot
format_words.c, 2915
format_word_dot_r
format_words.c, 2916
format_word_dot_s
format_words.c, 2917
format_word_dump
format_words.c, 2917
format_word_end_conversion
format_words.c, 2918
format_word_hash
format_words.c, 2918
format_word_hash_s
format_words.c, 2919
format_word_hex
format_words.c, 2920
format_word_hold
format_words.c, 2920
format_word_octal
format_words.c, 2921
format_word_question
format_words.c, 2921
format_word_sign
format_words.c, 2922
format_word_suites
format_words_test.c, 2618
format_word_u_dot
format_words.c, 2923
format_word_u_dot_r
format_words.c, 2923
format_words.c
CELL_BITS, 2911
CHUNK_BITS, 2911
CHUNK_MASK, 2911
current_base, 2912
digit_for, 2912
div_ud_by_base, 2912
format_word_base, 2913
format_word_begin_conversion, 2913
format_word_d_dot, 2914
format_word_d_dot_r, 2914
format_word_decimal, 2915
format_word_dot, 2915
format_word_dot_r, 2916
format_word_dot_s, 2917
format_word_dump, 2917
format_word_end_conversion, 2918
format_word_hash, 2918
format_word_hash_s, 2919
format_word_hex, 2920

format_word_hold, 2920
format_word_octal, 2921
format_word_question, 2921
format_word_sign, 2922
format_word_u_dot, 2923
format_word_u_dot_r, 2923
PN_BUF, 2911
PN_CAP, 2911
print_number_formatted, 2924
register_format_words, 2925
format_words.h
register_format_words, 2966
format_words_test.c
format_word_suites, 2618
run_format_words_tests, 2617
FORTH-79 Control Flow Words, 70
FORTH-79 Defining Words, 70
FORTH-79 Dictionary & Compilation Words, 72
FORTH-79 Dictionary Manipulation Words, 70
FORTH-79 Line Editor Words, 72
FORTH-79 Logical Operations, 75
FORTH-79 Memory Operations, 75
FORTH-79 Return Stack Operations, 76
FORTH-79 System & Environment Words, 76
FORTH-79 Vocabulary System Words, 77
forth_79_standard
system_words.c, 3247
forth_acl_allow_fetch
acl_words.c, 2779
forth_acl_allow_store
acl_words.c, 2780
forth_acl_heat_fetch
acl_words.c, 2781
forth_acl_inherit
acl_words.c, 2781
forth_acl_init_primitives
acl_words.c, 2782
forth_acl_mode_fetch
acl_words.c, 2783
forth_acl_mode_store
acl_words.c, 2783
forth_acl_pin
acl_words.c, 2785
forth_acl_pinned_query
acl_words.c, 2786
forth_acl_ttl_fetch
acl_words.c, 2787
forth_acl_ttl_store
acl_words.c, 2787
forth_acl_word_id
acl_words.c, 2789
forth_ALL_HEATS
physics_freeze_words.c, 3136
forth_backslash
system_words.c, 3234
forth_BENCH_DICT_LOOKUP
physics_benchmark_words.c, 3124
forth_block_size
blkio_dev, 117
blkio_info_t, 119
blkio_params_t, 120
forth_cached_head
VocabularyState, 349
forth_COMPARE_LOOKUPS
dictionary_heat_diagnostic_words.c, 2860
forth_DECAY_RATE_FETCH
physics_freeze_words.c, 3137
FORTH_FALSE
logical_words.c, 3087
forth_fc
VocabularyState, 349
forth_FREEZE_CRITICAL
physics_freeze_words.c, 3138
forth_FREEZE_WORD
physics_freeze_words.c, 3139
forth_FROZEN_QUERY
physics_freeze_words.c, 3139
forth_HEAT_FETCH
physics_freeze_words.c, 3140
forth_HEAT_PERCENTILES
dictionary_heat_diagnostic_words.c, 2861
forth_HEAT_STORE
physics_freeze_words.c, 3141
forth_LOOKUP_STRATEGY_FETCH
dictionary_heat_diagnostic_words.c, 2862
forth_LOOKUP_STRATEGY_STORE
dictionary_heat_diagnostic_words.c, 2862
forth_n
VocabularyState, 349
forth_paren
system_words.c, 3234
forth_PHYSICS_BAYESIAN_REPORT
physics_benchmark_words.c, 3125
forth_PHYSICS_BUILD_INFO
physics_benchmark_words.c, 3126
forth_PHYSICS_BURN
physics_diagnostic_words.c, 3131
physics_diagnostic_words.h, 2991
forth_PHYSICS_CACHE_STATS
physics_benchmark_words.c, 3127
forth_PHYSICS_CALC_KNOBS
physics_diagnostic_words.c, 3132
physics_diagnostic_words.h, 2992
forth_PHYSICS_RESET_STATS
physics_benchmark_words.c, 3127
forth_PHYSICS_SHOW_FEEDBACK
physics_diagnostic_words.c, 3133
physics_diagnostic_words.h, 2993
forth_PHYSICS_TOGGLE_CACHE
physics_benchmark_words.c, 3128
forth_PHYSICS_WORD_METRICS
physics_diagnostic_words.c, 3134
physics_diagnostic_words.h, 2994
forth_PIPELINING_ANALYZE_WORD
physics_pipelining_diagnostic_words.c, 3146
forth_PIPELINING_ENABLE

physics_pipelining_diagnostic_words.c, 3146
forth_PIPELINING_RESET_ALL
physics_pipelining_diagnostic_words.c, 3147
forth_PIPELINING_SHOW_STATS
physics_pipelining_diagnostic_words.c, 3147
forth_PIPELINING_SHOW_TOP_TRANSITIONS
physics_pipelining_diagnostic_words.c, 3148
forth_PIPELINING_STATS
physics_pipelining_diagnostic_words.c, 3149
forth_REORG_BUCKETS
dictionary_heat_diagnostic_words.c, 2863
forth_SHOW_HEAT
physics_freeze_words.c, 3142
forth_SHOW_HEAT_OPTIMIZATION
dictionary_heat_diagnostic_words.c, 2864
FORTH_TRUE
logical_words.c, 3087
forth_UNFREEZE_WORD
physics_freeze_words.c, 3143
forth_vocab
VocabularyState, 349
fprintf
stdio.h, 583
fputc
stdio.h, 584
fputs
stdio.h, 585
framebuffer
BootInfo, 127
Framebuffer Words, 72
framebuffer.c
__attribute__, 1313
fb_cell_h, 1313
fb_cell_w, 1313
fb_draw_glyph, 1314
fb_draw_orientation_test, 1315
fb_fill_rect, 1315
fb_height, 1316
fb_init, 1317
fb_is_available, 1317
fb_put_pixel, 1318
fb_scroll_rect, 1319
fb_scroll_rows, 1319
fb_width, 1321
font_8x16_glyph, 1322
g_fb, 1323
pack_pixel, 1322
framebuffer.h
__attribute__, 555
FB_B, 554
fb_cell_h, 555
fb_cell_w, 556
fb_draw_glyph, 556
fb_draw_orientation_test, 557
fb_fill_rect, 558
FB_G, 554
fb_height, 558
fb_init, 559
fb_is_available, 559
FB_PIXEL_BGRX32, 555
FB_PIXEL_RGBX32, 555
fb_put_pixel, 560
FB_R, 554
FB_RGB, 555
fb_scroll_rect, 561
fb_scroll_rows, 561
fb_width, 563
FbPixelFormat, 555
framebuffer_words.c
fbw_height, 2927
fbw_plot, 2928
fbw_width, 2928
register_framebuffer_words, 2929
framebuffer_words.h
register_framebuffer_words, 2969
FrameBufferBase
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE, 169
FramebufferInfo, 190
base, 190
height, 190
pixel_format, 190
pixels_per_scanline, 190
size, 191
width, 191
FrameBufferSize
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE, 169
fread
stdio.h, 585
free
heap_block, 195
stdlib.h, 594
VMHostServices, 342
free_blocks
blk_volume_meta_t, 113
free_bytes
kmalloc_stats_t, 221
pmm_stats_t, 247
free_count
sf_alloc_stats, 261
free_pages
pmm_stats_t, 247
FreePages
_EFI_BOOT_SERVICES, 87
FreePool
_EFI_BOOT_SERVICES, 87
fscanf
stdio.h, 585
fseek
stdio.h, 586
ftell
stdio.h, 586
full_name
user_identity_seed_t, 302
full_read_at

blkio_file.c, 2138
full_write_at
blkio_file.c, 2139
func
DictEntry, 149
vm.h, 2076
function
PciDevice, 227
fuzz_rounds
WordContract, 355
fwrite
stdio.h, 586

g
block_subsystem.c, 2212
g_active_vm_name
console.c, 1282
g_active_vm_name_buf
console.c, 1282
g_adaptive_period_ns
heartbeat.c, 1410
g_alloc_count
alloc_kernel.c, 2408
g_arena_offset
alloc_kernel.c, 2413
g_artemis_usb_dev
repl.c, 2466
g_attached_blk_dev
repl.c, 2466
g_blkcfg
blkcfg.h, 1718
g_blkio_opened
main.c, 2311
g_boot_disk
sk_v4_identity.c, 1593
g_boot_info
uefi_loader.c, 1088
g_boot_line_stride
vt100.c, 1392
g_boot_ring
vt100.c, 1392
g_boot_ring_cap
vt100.c, 1393
g_boot_ring_count
vt100.c, 1393
g_boot_ring_head
vt100.c, 1393
g_boot_rows
vt100.c, 1393
g_boot_shadow
vt100.c, 1393
g_buf
sk_v4_marker.c, 1599
g_bus_first
pci.c, 1499
g_bus_last
pci.c, 1499
g_console_pending_key
repl.c, 2466
g_console_tx_count
console.c, 1282
g_count
usb_kbd.c, 1514
g_cursor_blink_tick
repl.c, 2466
g_cursor_visible
repl.c, 2466
g_diag_head
virtio_input.c, 1624
g_diag_ring
virtio_input.c, 1624
g_diag_tail
virtio_input.c, 1624
g_doe_log_enabled
doe_log.c, 1235
doe_log.h, 510
g_ecam_base
pci.c, 1499
g_events
usb_kbd.c, 1514
g_fb
framebuffer.c, 1323
g_fb_mirror
console.c, 1283
g_format_bam_for
block_subsystem.c, 2212
g_free_count
alloc_kernel.c, 2408
g_glyph_mode
vt100.c, 1393
g_gpt_alternate_lba
gpt.c, 1248
g_gpt_disk_guid
gpt.c, 1248
g_gpt_entries
gpt.c, 1248
g_gpt_hdr_block
gpt.c, 1249
g_gpt_new_entries
gpt.c, 1249
g_gpt_parts
sk_v4_identity.c, 1594
g_gpt_rmw
gpt.c, 1249
g_hdr
sk_v4_grow.c, 1571
g_head
usb_kbd.c, 1514
g_heartbeat
heartbeat.c, 1410
g_homeblocks_dev
repl.c, 2467
g_homeblocks_sig
repl.c, 2467
g_homeblocks_sig_valid
repl.c, 2467
g_hotwords_cache

physics_hotwords_cache.c, 2348
g_i8042_isr_count
i8042.c, 898
i8042.h, 627
keyboard_words.c, 3061
g_i8042_present
i8042.c, 898
g_idle_pump_active
repl.c, 2467
g_initialized
alloc_kernel.c, 2413
g_ioapic_base
ioapic.c, 906
g_ioapic_gsi_base
ioapic.c, 906
g_ioapic_ready
ioapic.c, 906
g_kbd_alt_down
kbd_input.c, 1331
g_kbd_caps_lock
kbd_input.c, 1331
g_kbd_num_lock
kbd_input.c, 1331
g_kbd_poll_tick
kbd_input.c, 1331
g_kbd_shift_down
kbd_input.c, 1331
g_kh_blk_attach_buf
repl.c, 2467
g_kh_console_cmd_buf
repl.c, 2467
g_last_beat
sk_v4_identity.c, 1594
g_last_beat_tick
repl.c, 2467
g_leds_want
usb_kbd.c, 1514
g_line_start
console.c, 1283
g_line_stride
vt100.c, 1393
g_log_attrib_vm
vm_core.c, 1693
g_mama_interpreting
repl.c, 2467
g_marker
sk_v4_identity.c, 1594
g_mint_waiting
sk_v4_identity.c, 1594
g_move
sk_v4_grow.c, 1572
g_offered
sk_v4_identity.c, 1594
g_override_count
ioapic.c, 906
g_overrides
ioapic.c, 906
g_part_vt
gpt.c, 1249
g_pending_counter
heartbeat.c, 1410
g_pending_line_close
console.c, 1283
g_pending_real
sk_v4_identity.c, 1594
g_pending_valid
heartbeat.c, 1410
g_plic_claim_count
interrupts.c, 995
g_prng_state
starforth_words.c, 3211
g_profiler
profiler.c, 2441
profiler.h, 1917
g_proxy_fallback
sk_v4_identity.c, 1594
g_ram_base
main.c, 2311
g_ram_bytes
main.c, 2311
g_ram_is_mmap
main.c, 2311
g_real_drive
sk_v4_identity.c, 1595
g_redir_low
ioapic.c, 906
g_redir_valid
ioapic.c, 907
g_repl_active_vm
repl.c, 2468
g_ring
vt100.c, 1394
g_ring_cap
vt100.c, 1394
g_ring_count
vt100.c, 1394
g_ring_head
vt100.c, 1394
g_rng_backend
rng.c, 1502
g_rpi5_boot_info
rpi5_native_boot.c, 890
g_rpi5_dtb_ptr
rpi5_native_entry.h, 682
g_rpi5_l1
rpi5_mmu.c, 877
g_rpi5_l2
rpi5_mmu.c, 877
g_rpi5_memmap
rpi5_native_boot.c, 890
g_runtime
physics_runtime.c, 2397
g_scroll_offset
vt100.c, 1394
g_serial_base
console.c, 1283

g_serial_from_dt
console.c, 1283
g_serial_kind
console.c, 1283
g_serial_shift
console.c, 1283
g_serial_width
console.c, 1284
g_shadow
vt100.c, 1394
g_sk_fault_word
console.h, 507
interrupts.c, 977, 987, 995
g_sk_runtime_services
kernel_main.c, 1426
g_spare
sk_v4_identity.c, 1595
g_spurious_count
interrupts.c, 987
keyboard_words.c, 3061
g_stats
alloc_host.c, 2400
alloc_kernel.c, 2413
g_stick_in
sk_v4_identity.c, 1595
g_terminal_visible
vt100.c, 1394
g_ttf_cache
vt100.c, 1395
g_ttf_font
vt100.c, 1395
g_ttf_ready
vt100.c, 1395
g_usb
sk_v4_identity.c, 1595
g_usb_blk
blkio_usb.c, 1508
g_usb_blk_vtable
blkio_usb.c, 1508
g_usb_ctrl
repl.c, 2468
g_user_prefix_fn
console.c, 1284
g_vblk
virtio_blk.c, 1615
g_vblk_dead
virtio_blk.c, 1615
g_vblk_ready
virtio_blk.c, 1615
g_vblk_vtable
virtio_blk.c, 1615
g_vinput
virtio_input.c, 1624
g_vinput_ready
virtio_input.c, 1624
g_virtio_input_gic_intid
interrupts.c, 977
virtio_input.c, 1624
g_virtio_input_isr_count
keyboard_words.c, 3062
virtio_input.c, 1625
g_virtio_input_plic_source
interrupts.c, 995
virtio_input.c, 1625
g_vm
vm_debug.c, 2734
g_vm_interpret_depth
vm.c, 2712
vm_core.c, 1693
g_vrng
virtio_rng.c, 1633
g_vrng_ready
virtio_rng.c, 1633
g_vt
vt100.c, 1395
g_wirebind_attached_dev
capsule_wirebind.c, 1182
g_wirebind_attached_username
capsule_wirebind.c, 1182
g_wirebind_attached_valid
capsule_wirebind.c, 1182
g_wirebind_attached_vm_id
capsule_wirebind.c, 1183
g_wirebind_live
capsule_wirebind.c, 1183
g_xhci_dev_count
xhci.c, 1560
g_xhci_devs
xhci.c, 1560
g_xhci_keyboards
xhci.c, 1560
g_zuse_attached_dev
capsule_zuse_boot.c, 1192
g_zuse_attached_this_device
capsule_zuse_boot.c, 1192
g_zuse_cert_installed
sk_v4_identity.c, 1595
g_zuse_dev
sk_v4_identity.c, 1595
g_zuse_pubkey
sk_v4_identity.c, 1595
g_zuse_root_known
sk_v4_identity.c, 1596
g_zuse_root_pubkey
sk_v4_identity.c, 1596
g_zuse_seed
sk_v4_identity.c, 1596
g_zuse_session
sk_v4_identity.c, 1596
gdt_entry, 191
access, 191
base_high, 191
base_low, 192
base_mid, 192
granularity, 192
limit_low, 192

gdt_init
arch.c, 958
gdt_set_entry
arch.c, 959
gdtr, 192
base, 192
limit, 192
genesis_marker_read
capsule_zuse_boot.c, 1189
genesis_time_ns
artemis_sig_t, 101
get_block_buffer
block_words.c, 2828
block_words.h, 2943
get_body_address
dictionary_manipulation_words.c, 2876
get_empty_buffer
block_words.c, 2829
block_words.h, 2944
get_monotonic_ns
sf_time_backend_t, 263
get_name_field
dictionary_manipulation_words.c, 2876
get_or_alloc_table
vmm.c, 1471
get_realtime_ns
sf_time_backend_t, 263
get_table
vmm.c, 1472
get_time_ns
profiler.c, 2428
test_runner.c, 2687
get_timestamp
log.c, 2292
get_word_stats
profiler.c, 2429
getbit
scalar25519.c, 1216
getchar
stdio.h, 587
GetInfo
_EFI_FILE_PROTOCOL, 91
GetMemoryMap
_EFI_BOOT_SERVICES, 87
GetNextHighMonotonicCount
_EFI_RUNTIME_SERVICES, 95
GetNextMonotonicCount
_EFI_BOOT_SERVICES, 87
GetNextVariableName
_EFI_RUNTIME_SERVICES, 95
GetPosition
_EFI_FILE_PROTOCOL, 91
GetTime
_EFI_RUNTIME_SERVICES, 95
gettimeofday
time.h, 606
GetVariable
_EFI_RUNTIME_SERVICES, 95
GetWakeupTime
_EFI_RUNTIME_SERVICES, 95
GIB
rpi5_mmu.c, 872
gic_bases_from_dtb
apic.c, 919
GIC_INTID_SPURIOUS
interrupts.c, 968
GICC_BASE_PA_QEMU_DEFAULT
apic.c, 909
GICC_CTLR
apic.c, 909
GICC_CTLR_ENABLE_GRP0
apic.c, 909
GICC_EOIR
apic.c, 909
GICC_IAR
apic.c, 909
GICC_PMR
apic.c, 909
GICD_BASE_PA_QEMU_DEFAULT
apic.c, 909
GICD_CTLR
apic.c, 909
GICD_CTLR_ENABLE_GRP0
apic.c, 909
GICD_ICFGR
apic.c, 909
GICD_IPRIORITYR
apic.c, 909
GICD_ISENABLER0
apic.c, 910
GICD_ISPENDR
apic.c, 910
GICD_ITARGETSR
apic.c, 910
gid_t
types.h, 608
global_contract_violations
test_contracts.c, 2685
test_contracts.h, 2545
global_test_stats
test_runner.c, 2693
test_runner.h, 2579
glyf_len
ttf_font_t, 289
glyf_length
ttf_glyph_header_t, 292
glyf_off
ttf_font_t, 289
glyf_offset
ttf_glyph_header_t, 292
gpt
BootDisk, 126
gpt.c
g_gpt_alternate_lba, 1248
g_gpt_disk_guid, 1248
g_gpt_entries, 1248

g_gpt_hdr_block, 1249
g_gpt_new_entries, 1249
g_gpt_rmw, 1249
g_part_vt, 1249
gpt_alternate_lba, 1239
gpt_crc32, 1239
gpt_disk_guid, 1239
GPT_ENT_SECTORS, 1238
GPT_ENTRIES_MAX, 1238
GPT_ESIZE, 1239
gpt_find, 1240
gpt_grow_last, 1241
GPT_NENT, 1239
gpt_new_guid, 1241
gpt_part_open, 1242
gpt_read, 1243
GPT_SECTOR, 1239
gpt_type_starship_artemis, 1249
gpt_type_starship_identity, 1249
gpt_write_one, 1243
le32, 1244
le64, 1244
part_close, 1245
part_flush, 1245
part_info, 1245
part_open, 1245
part_read, 1246
part_write, 1246
put_le32, 1246
put_le64, 1247
put_sectors, 1247
gpt.h
gpt_alternate_lba, 611
GPT_BAD, 610
gpt_crc32, 611
gpt_disk_guid, 611
GPT_EIO, 610
gpt_find, 612
gpt_grow_last, 612
GPT_NONE, 610
GPT_OK, 610
gpt_part_open, 613
gpt_read, 614
gpt_type_starship_artemis, 615
gpt_type_starship_identity, 615
gpt_write_one, 615
gpt_alternate_lba
gpt.c, 1239
gpt.h, 611
GPT_BAD
gpt.h, 610
gpt_crc32
gpt.c, 1239
gpt.h, 611
gpt_disk_guid
gpt.c, 1239
gpt.h, 611
GPT_EIO
gpt.h, 610
GPT_ENT_SECTORS
gpt.c, 1238
GPT_ENTRIES_MAX
gpt.c, 1238
GPT_ESIZE
gpt.c, 1239
gpt_find
gpt.c, 1240
gpt.h, 612
gpt_grow_last
gpt.c, 1241
gpt.h, 612
GPT_NENT
gpt.c, 1239
gpt_new_guid
gpt.c, 1241
GPT_NONE
gpt.h, 610
GPT_OK
gpt.h, 610
gpt_part_dev_t, 193
disk, 193
fblocks, 193
first_fblock, 194
gpt_part_open
gpt.c, 1242
gpt.h, 613
gpt_part_t, 194
first_lba, 194
last_lba, 194
type, 194
gpt_read
gpt.c, 1243
gpt.h, 614
GPT_SECTOR
gpt.c, 1239
gpt_type_starship_artemis
gpt.c, 1249
gpt.h, 615
gpt_type_starship_identity
gpt.c, 1249
gpt.h, 615
gpt_write_one
gpt.c, 1243
gpt.h, 615
granularity
arch.c, 960
gdt_entry, 192
GreenMask
EFI_PIXEL_BITMASK, 174
gsi
__attribute__, 81
gsi_base
__attribute__, 81
guid_equals
uefi_loader.c, 1085

hal_mem_alloc

hal_memory.h, 617
memory.c, 1333
hal_mem_alloc_aligned
hal_memory.h, 617
memory.c, 1333
hal_mem_free
hal_memory.h, 617
memory.c, 1333
hal_memory.h
hal_mem_alloc, 617
hal_mem_alloc_aligned, 617
hal_mem_free, 617
halted
VM, 327
handle
sf_mutex, 262
HandleProtocol
_EFI_BOOT_SERVICES, 87
has_cmap
ttf_font_t, 289
has_ever_claimed
blk_dev_slot, 106
has_prev
CDConfigEntry, 135
has_real
sk_v4_marker_t, 270
zuse_genesis_marker_v2_t, 396
has_rtc
sf_time_backend_t, 263
has_sig
CapsuleSigEntry, 134
has_variance_stabilized
inference_engine.c, 2281
hash
blk_meta_t, 109
hash_colon_body
parity.c, 1637
hcc_params1
xhci_cap_regs_t, 365
hcc_params2
xhci_cap_regs_t, 366
hci_version
xhci_cap_regs_t, 366
hcs_params1
xhci_cap_regs_t, 366
hcs_params2
xhci_cap_regs_t, 366
hcs_params3
xhci_cap_regs_t, 366
Hdr
_EFI_BOOT_SERVICES, 88
_EFI_RUNTIME_SERVICES, 95
_EFI_SYSTEM_TABLE, 100
hdr
__attribute__, 81
hdr_crc
artemis_sig_t, 101
blk_volume_meta_t, 113
homeblocks_sig_t, 206
log_region_ctrl_t, 222
head
block_subsystem.c, 2213
VMFleetWindow, 341
head_off
ttf_font_t, 290
head_slot
log_region_ctrl_t, 222
header
physics_analytics_heap, 234
header_hash64
ParityPacket, 225
HeaderSize
EFI_TABLE_HEADER, 175
heap
physics_runtime_state, 243
heap_base
kmalloc.c, 1456
heap_block, 195
free, 195
next, 195
prev, 195
requested, 195
size, 195
heap_block_t
kmalloc.c, 1444
heap_bytes
physics_analytics_header, 231
heap_head
kmalloc.c, 1456
heap_initialized
kmalloc.c, 1456
heap_ready
physics_runtime_state, 243
heap_size
KernelArgs, 219
kmalloc.c, 1457
heap_stats
kmalloc.c, 1457
heartbeat
VM, 327
heartbeat.c
g_adaptive_period_ns, 1410
g_heartbeat, 1410
g_pending_counter, 1410
g_pending_valid, 1410
HEARTBEAT_BASE_PERIOD_NS, 1404
HEARTBEAT_HAS_VARIANCE, 1404
heartbeat_init, 1405
heartbeat_next_period_ns, 1405
heartbeat_service, 1405
heartbeat_set_adaptive_period_ns, 1406
heartbeat_state, 1407
heartbeat_tick, 1407
heartbeat_ticks, 1408
heartbeat_trust, 1409
HEARTBEAT_BASE_PERIOD_NS

heartbeat.c, 1404
heartbeat_capture_tick_snapshot
heartbeat_export.c, 2270
vm.h, 2014
HEARTBEAT_CHECK_FREQUENCY
starforth_config.h, 1991
HEARTBEAT_DECAY_BATCH
vm_internal.h, 2735, 2747
heartbeat_decay_cursor_id
VM, 328
heartbeat_emit_tick_row
heartbeat_export.c, 2271
vm.h, 2015
heartbeat_enabled
HeartbeatState, 199
heartbeat_export.c
heartbeat_capture_tick_snapshot, 2270
heartbeat_emit_tick_row, 2271
HEARTBEAT_HAS_THREADS
vm_internal.h, 2735, 2747
HEARTBEAT_HAS_VARIANCE
heartbeat.c, 1404
HEARTBEAT_INFERENCE_FREQUENCY
starforth_config.h, 1991
heartbeat_init
heartbeat.c, 1405
timer.h, 693
HEARTBEAT_LOG_FULL
cli.h, 1760
heartbeat_log_mode
CLIConfig, 143
HEARTBEAT_LOG_OFF
cli.h, 1760
HEARTBEAT_LOG_SUMMARY
cli.h, 1760
heartbeat_next_period_ns
heartbeat.c, 1405
timer.h, 693
heartbeat_publish_snapshot
vm_internal.h, 2736
vm_runtime.c, 1695
heartbeat_read_counter
timer.c, 1008, 1026, 1054
timer.h, 694
heartbeat_service
heartbeat.c, 1405
timer.h, 695
heartbeat_set_adaptive_period_ns
heartbeat.c, 1406
timer.h, 696
HEARTBEAT_SLOPE_VALIDATION_FREQUENCY
starforth_config.h, 1991
heartbeat_snapshot_index_load
vm_internal.h, 2737, 2750
vm_runtime.c, 1695
heartbeat_snapshot_index_store
vm_internal.h, 2737, 2751
vm_runtime.c, 1696
heartbeat_state
heartbeat.c, 1407
timer.h, 697
HEARTBEAT_THREAD_ENABLED
starforth_config.h, 1991
heartbeat_thread_main
vm_runtime.c, 1697
heartbeat_tick
heartbeat.c, 1407
timer.h, 697
HEARTBEAT_TICK_BUFFER_SIZE
vm.h, 2007
HEARTBEAT_TICK_NS
starforth_config.h, 1991
heartbeat_ticks
heartbeat.c, 1408
timer.h, 698
heartbeat_trust
heartbeat.c, 1409
timer.h, 699
HEARTBEAT_WINDOW_TUNING_FREQUENCY
starforth_config.h, 1991
HeartbeatLogMode
cli.h, 1760
HeartbeatSnapshot, 196
actual_window_size, 196
decay_slope_q48, 196
hot_word_count, 196
published_ns, 196
published_tick, 197
stale_word_count, 197
total_heat, 197
window_width, 197
HeartbeatState, 197
bucket_collapse_flag, 198
bucket_mode_transitions, 198
bucket_sum_heat_variance, 198
bucket_sum_K, 198
bucket_sum_K_squared, 199
bucket_sum_window_variance, 199
bucket_tick_count, 199
check_counter, 199
dictionary_lookups, 199
early_exit_count, 199
heartbeat_enabled, 199
inference_run_count, 199
last_inference_tick, 200
m5_time_trust, 200
m5_variance, 200
run_start_ns, 200
snapshot_index, 200
snapshots, 200
tick_buffer, 200
tick_buffer_size, 200
tick_buffer_write_index, 201
tick_count, 201
tick_count_total, 201
tick_number_offset, 201

tick_target_ns, 201
words_executed, 201
worker, 201
HeartbeatTickSnapshot, 202
actual_window_size, 202
avg_word_heat, 202
bucket_hits_delta, 202
cache_hits_delta, 203
elapsed_ns, 203
estimated_jitter_ns, 203
hot_word_count, 203
l8_mode, 203
predicted_label_hits, 203
tick_interval_ns, 203
tick_number, 203
window_width, 204
word_executions_delta, 204
HeartbeatWorker, 204
running, 204
stop_requested, 204
tick_ns, 204
vm.h, 2010
vm_internal.h, 2736, 2748
heat_25th
PhysicsSnapshot, 245
heat_50th
PhysicsSnapshot, 245
heat_75th
PhysicsSnapshot, 245
HEAT_CACHE_DEMOTION_THRESHOLD
vm.h, 2007
heat_cache_demotion_threshold
DoeMetrics, 158
heat_threshold_25th
VM, 328
heat_threshold_50th
VM, 328
heat_threshold_75th
VM, 328
height
FbState, 185
FramebufferInfo, 190
Rpi5FramebufferInfo, 259
ttf_bitmap_t, 287
ttf_raster_cache_slot_t, 296
held
usb_kbd.c, 1510
here
VM, 328
here_offset
ParityPacket, 225
hid_buf
xhci_msc_slot_t, 385
hid_ep_addr
xhci_msc_slot_t, 385
hid_iface
xhci_msc_slot_t, 385
hid_interval
xhci_msc_slot_t, 385
hid_kbd
xhci_msc_slot_t, 385
hid_leds_sent
xhci_msc_slot_t, 385
hid_max_packet
xhci_msc_slot_t, 385
hid_prev
xhci_msc_slot_t, 386
hid_ring
xhci_msc_slot_t, 386
hid_ring_cycle
xhci_msc_slot_t, 386
hid_ring_enq
xhci_msc_slot_t, 386
hid_running
xhci_msc_slot_t, 386
history
rolling_window_view_t, 255
hits
ttf_raster_cache_slot_t, 296
hmtx_off
ttf_font_t, 290
HOLA_PROTOCOL_MAJOR
physics_runtime.h, 1876
HOLA_PROTOCOL_MINOR
physics_runtime.h, 1876
HOLA_SHARED_MAGIC
physics_runtime.h, 1876
hold_addr
VM, 329
hold_pos
VM, 329
home_lbn
blk_reloc_entry_t, 111
homeblocks_sig.c
homeblocks_sig_check, 1411
homeblocks_sig_compute_crc, 1411
homeblocks_sig.h
HOMEBLOCKS_SIG_BAD_CRC, 621
HOMEBLOCKS_SIG_BAD_VERSION, 621
HOMEBLOCKS_SIG_BLANK, 621
homeblocks_sig_check, 621
homeblocks_sig_compute_crc, 621
HOMEBLOCKS_SIG_GET_MAGIC, 619
HOMEBLOCKS_SIG_GET_VERSION, 619
HOMEBLOCKS_SIG_MAGIC, 619
HOMEBLOCKS_SIG_OK, 621
HOMEBLOCKS_SIG_PACK, 620
HOMEBLOCKS_SIG_READ_ERROR, 621
homeblocks_sig_result_t, 620
homeblocks_sig_size_check, 620
HOMEBLOCKS_SIG_START_FBLOCK, 620
HOMEBLOCKS_SIG_VERSION_0, 620
HOMEBLOCKS_SIG_BAD_CRC
homeblocks_sig.h, 621
HOMEBLOCKS_SIG_BAD_VERSION
homeblocks_sig.h, 621

HOMEBLOCKS_SIG_BLANK
homeblocks_sig.h, 621
homeblocks_sig_check
homeblocks_sig.c, 1411
homeblocks_sig.h, 621
homeblocks_sig_compute_crc
homeblocks_sig.c, 1411
homeblocks_sig.h, 621
HOMEBLOCKS_SIG_GET_MAGIC
homeblocks_sig.h, 619
HOMEBLOCKS_SIG_GET_VERSION
homeblocks_sig.h, 619
HOMEBLOCKS_SIG_MAGIC
homeblocks_sig.h, 619
HOMEBLOCKS_SIG_OK
homeblocks_sig.h, 621
HOMEBLOCKS_SIG_PACK
homeblocks_sig.h, 620
HOMEBLOCKS_SIG_READ_ERROR
homeblocks_sig.h, 621
homeblocks_sig_result_t
homeblocks_sig.h, 620
homeblocks_sig_size_check
homeblocks_sig.h, 620
HOMEBLOCKS_SIG_START_FBLOCK
homeblocks_sig.h, 620
homeblocks_sig_t, 205
_pad, 205
cert_devblocks, 205
cert_offset, 205
drive_uuid, 205
hdr_crc, 206
identity_src_devblocks, 206
identity_src_offset, 206
magic, 206
metadata_devblocks, 206
minted_time_ns, 206
HOMEBLOCKS_SIG_VERSION_0
homeblocks_sig.h, 620
HorizontalResolution
EFI_GRAPHICS_OUTPUT_MODE_INFORMATION, 167
host
VM, 329
hosted_alloc
vm_core.c, 1654
hosted_fake_ns
vm_core.c, 1693
hosted_free
vm_core.c, 1655
hosted_lifecycle_birth
lifecycle_words_hosted.c, 3064
hosted_lifecycle_kill
lifecycle_words_hosted.c, 3064
hosted_lifecycle_pause
lifecycle_words_hosted.c, 3065
hosted_lifecycle_resume
lifecycle_words_hosted.c, 3066
hosted_lifecycle_use
lifecycle_words_hosted.c, 3066
hosted_mutex_destroy
vm_core.c, 1655
hosted_mutex_init
vm_core.c, 1655
hosted_mutex_lock
vm_core.c, 1656
hosted_mutex_unlock
vm_core.c, 1656
hosted_panic
vm_core.c, 1656
hosted_putc
vm_core.c, 1657
hosted_puts
vm_core.c, 1657
hosted_services
vm_core.c, 1693
hosted_time_ns
vm_core.c, 1658
hosted_xt_entry_owned
vm_core.c, 1658
hosted_xt_is_executable
vm_core.c, 1658
hot_word_count
DoeMetrics, 158
HeartbeatSnapshot, 196
HeartbeatTickSnapshot, 203
InferenceInputs, 215
hot_word_count_at_check
VM, 329
hotwords_bayesian_report
physics_hotwords_cache.c, 2329
physics_hotwords_cache.h, 1829
hotwords_bucket_reorder
physics_hotwords_cache.c, 2330
physics_hotwords_cache.h, 1830
hotwords_cache
VM, 329
hotwords_cache_cleanup
physics_hotwords_cache.c, 2331
physics_hotwords_cache.h, 1831
hotwords_cache_evict_entry
physics_hotwords_cache.c, 2332
physics_hotwords_cache.h, 1831
hotwords_cache_evict_name
physics_hotwords_cache.c, 2332
physics_hotwords_cache.h, 1832
hotwords_cache_init
physics_hotwords_cache.c, 2334
physics_hotwords_cache.h, 1833
hotwords_cache_lookup
physics_hotwords_cache.c, 2335
physics_hotwords_cache.h, 1835
hotwords_cache_promote
physics_hotwords_cache.c, 2338
physics_hotwords_cache.h, 1837
hotwords_cache_remove_at

physics_hotwords_cache.c, 2340
hotwords_cache_set_enabled
physics_hotwords_cache.c, 2341
physics_hotwords_cache.h, 1839
HOTWORDS_CACHE_SIZE
physics_hotwords_cache.h, 1828
HOTWORDS_EXECUTION_HEAT_DELTA_THRESHOLD
physics_hotwords_cache.h, 1828
HOTWORDS_EXECUTION_HEAT_THRESHOLD
physics_hotwords_cache.h, 1828
hotwords_posterior_bucket_searches
physics_hotwords_cache.c, 2342
physics_hotwords_cache.h, 1840
hotwords_posterior_cache_hits
physics_hotwords_cache.c, 2343
physics_hotwords_cache.h, 1841
hotwords_speedup_estimate
physics_hotwords_cache.c, 2344
physics_hotwords_cache.h, 1842
hotwords_stats_print
physics_hotwords_cache.c, 2345
physics_hotwords_cache.h, 1843
hotwords_stats_reset
physics_hotwords_cache.c, 2346
physics_hotwords_cache.h, 1844
HotwordsCache
physics_hotwords_cache.h, 1829
vm.h, 2010
HotwordsCache_s, 207
cache, 207
cache_count, 207
enabled, 207
lru_index, 208
stats, 208
HotwordsStats, 208
bucket_hits, 209
bucket_reorders, 209
bucket_search_samples, 210
bucket_search_total_ns_q48, 210
bucket_search_variance_sum_q48, 210
cache_hit_samples, 210
cache_hit_total_ns_q48, 210
cache_hit_variance_sum_q48, 211
cache_hits, 211
evictions, 211
max_bucket_search_ns, 211
max_cache_hit_ns, 211
min_bucket_search_ns, 212
min_cache_hit_ns, 212
misses, 212
promotions, 212
total_lookups, 212
Hour
EFI_TIME, 176
hpet_freq_hz
timer.c, 1050
HPET_GEN_CAP_ID
timer.c, 1015
HPET_GEN_CONFIG
timer.c, 1015
hpet_hz
timer_calibration_record, 284
HPET_MAIN_COUNTER
timer.c, 1015
HPET_PHYS_BASE
timer.c, 1015
hpet_phys_base
vmm.c, 1481
hpet_read
timer.c, 1027
hpet_regs
timer.c, 1050
HPET_VIRT_BASE
timer.c, 1016
hpet_write
timer.c, 1028
HUGE_VAL
math.h, 575
hw_address
BlockBuffer, 125
hysteresis_counter
ssm_l8_state_t, 275

i8042.c
g_i8042_isr_count, 898
g_i8042_present, 898
I8042_CMD_PORT, 892
I8042_CMD_READ_CONFIG, 892
I8042_CMD_WRITE_CONFIG, 892
I8042_CONFIG_IRQ1_ENABLE, 892
I8042_DATA_PORT, 892
I8042_DEV_ACK, 892
I8042_DEV_CMD_ENABLE_SCANNING, 892
i8042_drain, 893
i8042_drain_stale, 893
i8042_handle_irq, 894
i8042_init, 895
I8042_INIT_WAIT_ITERS, 893
i8042_pop_scancode, 896
I8042_STATUS_INPUT_FULL, 893
I8042_STATUS_OUTPUT_FULL, 893
I8042_STATUS_PORT, 893
inb, 897
outb, 897
ring, 898
ring_head, 898
RING_SIZE, 893
ring_tail, 899
i8042.h
g_i8042_isr_count, 627
i8042_drain_stale, 624
i8042_handle_irq, 624
i8042_init, 625
I8042_KEYBOARD_VECTOR, 623
i8042_pop_scancode, 626
I8042_CMD_PORT
i8042.c, 892

I8042_CMD_READ_CONFIG
i8042.c, 892
I8042_CMD_WRITE_CONFIG
i8042.c, 892
I8042_CONFIG_IRQ1_ENABLE
i8042.c, 892
I8042_DATA_PORT
i8042.c, 892
I8042_DEV_ACK
i8042.c, 892
I8042_DEV_CMD_ENABLE_SCANNING
i8042.c, 892
i8042_drain
i8042.c, 893
i8042_drain_stale
i8042.c, 893
i8042.h, 624
i8042_handle_irq
i8042.c, 894
i8042.h, 624
i8042_init
i8042.c, 895
i8042.h, 625
I8042_INIT_WAIT_ITERS
i8042.c, 893
I8042_KEYBOARD_VECTOR
i8042.h, 623
i8042_pop_scancode
i8042.c, 896
i8042.h, 626
I8042_STATUS_INPUT_FULL
i8042.c, 893
I8042_STATUS_OUTPUT_FULL
i8042.c, 893
I8042_STATUS_PORT
i8042.c, 893
IA32_APIC_BASE_GLOBAL_EN
apic.c, 924
IA32_APIC_BASE_MSR
apic.c, 924
IA32_APIC_BASE_X2APIC_EN
apic.c, 925
id
VirtqUsedElem, 315
id_window_depth
CDTuning, 139
identity_src_devblocks
homeblocks_sig_t, 206
identity_src_offset
homeblocks_sig_t, 206
idt
interrupts.c, 987
IDT_ENTRIES
interrupts.c, 978
idt_entry, 213
ist, 213
offset_high, 213
offset_low, 213
offset_mid, 213
selector, 214
type_attr, 214
zero, 214
idtr, 214
base, 214
limit, 214
idx
VirtqAvail, 312
VirtqUsed, 314
ImageBase
EFI_LOADED_IMAGE_PROTOCOL, 171
ImageCodeType
EFI_LOADED_IMAGE_PROTOCOL, 171
ImageDataType
EFI_LOADED_IMAGE_PROTOCOL, 171
ImageSize
EFI_LOADED_IMAGE_PROTOCOL, 171
iman
xhci_intr_regs_t, 380
imod
xhci_intr_regs_t, 380
implemented
TestCase, 277
in_bounds
blkio_ram.c, 2143
ttf.c, 1345
in_use
wirebind_live_entry_t, 354
xhci_msc_slot_t, 386
in_var
VM, 330
inb
i8042.c, 897
timer.c, 1028
include/, 1
index_to_loc_format
ttf_font_t, 290
infer_decay_slope_q48
inference_engine.c, 2282
inference_engine.h, 1794
infer_word_bayes_bucket_lower
inference_words.c, 3031
infer_word_bayes_bucket_mean
inference_words.c, 3032
infer_word_bayes_bucket_upper
inference_words.c, 3032
infer_word_bayes_cache_lower
inference_words.c, 3033
infer_word_bayes_cache_mean
inference_words.c, 3033
infer_word_bayes_cache_upper
inference_words.c, 3034
infer_word_decay_fetch
inference_words.c, 3034
infer_word_decay_slope
inference_words.c, 3035
infer_word_early_exit_fetch

inference_words.c, 3036
infer_word_fit_fetch
inference_words.c, 3036
infer_word_l8_apply
inference_words.c, 3036
infer_word_l8_mode
inference_words.c, 3037
infer_word_l8_table_force
inference_words.c, 3037
infer_word_l8_update
inference_words.c, 3038
infer_word_q_variance
inference_words.c, 3039
infer_word_run
inference_words.c, 3039
infer_word_variance_fetch
inference_words.c, 3040
infer_word_window_diversity
inference_words.c, 3041
infer_word_window_fetch
inference_words.c, 3041
infer_word_window_width
inference_words.c, 3042
inference_early_exited
ssm_l8_metrics_t, 274
inference_engine.c
compute_fit_quality, 2273
compute_levene_statistic, 2274
compute_mean_q48, 2276
compute_median_q48, 2277
compute_variance_q48, 2278
extract_heat_trajectory, 2279
find_variance_inflection, 2280
has_variance_stabilized, 2281
infer_decay_slope_q48, 2282
inference_engine_run, 2284
inference_outputs_to_string, 2286
inference_outputs_validate, 2287
inference_engine.h
compute_variance_q48, 1792
find_variance_inflection, 1793
infer_decay_slope_q48, 1794
inference_engine_run, 1796
inference_outputs_to_string, 1798
inference_outputs_validate, 1799
inference_engine_run
inference_engine.c, 2284
inference_engine.h, 1796
inference_outputs_to_string
inference_engine.c, 2286
inference_engine.h, 1798
inference_outputs_validate
inference_engine.c, 2287
inference_engine.h, 1799
inference_ran_this_tick
ssm_l8_metrics_t, 274
inference_run_count
DoeMetrics, 158
HeartbeatState, 199
inference_word_suites
inference_words_test.c, 2620
inference_words.c
array_ptr, 3030
bucket_posterior, 3030
cache_posterior, 3031
infer_word_bayes_bucket_lower, 3031
infer_word_bayes_bucket_mean, 3032
infer_word_bayes_bucket_upper, 3032
infer_word_bayes_cache_lower, 3033
infer_word_bayes_cache_mean, 3033
infer_word_bayes_cache_upper, 3034
infer_word_decay_fetch, 3034
infer_word_decay_slope, 3035
infer_word_early_exit_fetch, 3036
infer_word_fit_fetch, 3036
infer_word_l8_apply, 3036
infer_word_l8_mode, 3037
infer_word_l8_table_force, 3037
infer_word_l8_update, 3038
infer_word_q_variance, 3039
infer_word_run, 3039
infer_word_variance_fetch, 3040
infer_word_window_diversity, 3041
infer_word_window_fetch, 3041
infer_word_window_width, 3042
q48_pop_inf, 3042
q48_push_inf, 3043
q48_to_dbl, 3044
register_inference_words, 3044
inference_words.h
register_inference_words, 2971
inference_words_test.c
inference_word_suites, 2620
run_inference_words_tests, 2619
InferenceInputs, 215
hot_word_count, 215
last_stale_count, 215
last_total_heat, 216
prefetch_attempts, 216
prefetch_hits, 216
stale_word_count, 216
total_heat, 216
trajectory_length, 216
vm, 216
window, 216
word_count, 217
InferenceOutputs, 217
adaptive_decay_slope, 217
adaptive_window_width, 217
early_exited, 217
last_check_tick, 218
slope_fit_quality_q48, 218
vm.h, 2011
window_variance_q48, 218
INFINITY
math.h, 575

Info
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE, 169
info
blkio_vtable_t, 122
init_blkio
main.c, 2303
init_block_system
block_words.c, 2830
block_words.h, 2945
init_report
break_me_tests.c, 2604
init_timer_vm_mode
timer.c, 1030
init_vm_and_subsystem
main.c, 2304
init_vocabulary_system
vocabulary_words.c, 3252
INITIAL_DECAY_SLOPE_Q48
starforth_config.h, 1991
initialized
block_subsystem.c, 2213
VocabularyState, 350
VT100State, 351
inl
timer.c, 1031
input
TestCase, 277
input_buffer
VM, 330
INPUT_BUFFER_SIZE
vm.h, 2007
input_ctx
xhci_msc_slot_t, 386
input_length
VM, 330
input_pos
VM, 330
install_and_activate
capsule_zuse_boot.c, 1190
InstallConfigurationTable
_EFI_BOOT_SERVICES, 88
InstallProtocolInterface
_EFI_BOOT_SERVICES, 88
instruction_count
Profiler, 249
INT16
uefi.h, 729
INT32
uefi.h, 729
INT64
uefi.h, 729
INT8
uefi.h, 729
integration_suites
integration_tests.c, 2622
integration_tests.c
integration_suites, 2622
run_integration_tests, 2621
interpreter_enabled
VM, 330
INTERRUPT_GATE
interrupts.c, 978
interrupts.c
__attribute__, 979
aarch64_current_el, 968
aarch64_exception_handler, 968
aarch64_install_vectors, 969
aarch64_irq_handler, 970
apic_eoi_intid, 971
apic_read_iar, 972
apic_timer_ppi, 972
apic_timer_rearm, 973
arch_interrupts_init, 974, 990
base, 987
decode_pf_error, 979
g_plic_claim_count, 995
g_sk_fault_word, 977, 987, 995
g_spurious_count, 987
g_virtio_input_gic_intid, 977
g_virtio_input_plic_source, 995
GIC_INTID_SPURIOUS, 968
idt, 987
IDT_ENTRIES, 978
INTERRUPT_GATE, 978
isr_common_handler, 980
isr_stub0, 981
ist, 987
lidt, 982
limit, 987
offset_high, 988
offset_low, 988
offset_mid, 988
outb, 982
pic_disable, 984
print_dec_u64, 984
print_hex, 975, 991
print_hex64, 985
riscv64_exception_handler, 992
riscv64_install_vectors, 993
riscv64_interrupt_handler, 993
riscv64_timer_rearm, 994
SCAUSE_INTERRUPT_BIT, 990
SCAUSE_S_EXTERNAL, 990
SCAUSE_S_TIMER, 990
selector, 988
set_idt_entry, 986
type_attr, 988
virtio_input_isr, 976, 995
zero, 988
INTN
uefi.h, 729
intr0
xhci_dev_t, 374
inttypes.h
PRId16, 572

PRId32, 572
PRId64, 572
PRId8, 572
PRIi16, 572
PRIi32, 572
PRIi64, 572
PRIi8, 572
PRIu16, 573
PRIu32, 573
PRIu64, 573
PRIu8, 573
PRIX16, 573
PRIx16, 573
PRIX32, 573
PRIx32, 573
PRIX64, 573
PRIx64, 573
PRIX8, 574
PRIx8, 574
invlpg
vmm.c, 1473
io.c
io_init, 2289
io_write_block, 2290
sf_terminal_ready, 2291
io.h
BLOCK_COUNT, 1802
io_init, 1802
io_read_block, 1802
io_write_block, 1803
MEMORY_SIZE, 1802
io_apic_address
__attribute__, 81
io_apic_id
__attribute__, 81
io_init
io.c, 2289
io.h, 1802
io_read_block
io.h, 1802
io_runtime_do_string
io_words.c, 3046
io_width
fdt_uart_t, 188
io_word_cr
io_words.c, 3047
io_word_dot_quote
io_words.c, 3048
io_word_emit
io_words.c, 3048
io_word_key
io_words.c, 3049
io_word_question_terminal
io_words.c, 3050
io_word_space
io_words.c, 3051
io_word_spaces
io_words.c, 3051
io_word_suites
io_words_test.c, 2624
io_word_type
io_words.c, 3053
io_words.c
io_runtime_do_string, 3046
io_word_cr, 3047
io_word_dot_quote, 3048
io_word_emit, 3048
io_word_key, 3049
io_word_question_terminal, 3050
io_word_space, 3051
io_word_spaces, 3051
io_word_type, 3053
register_io_words, 3054
io_words.h
IO_WORDS_H, 2974
register_io_words, 2974
IO_WORDS_H
io_words.h, 2974
io_words_test.c
io_word_suites, 2624
run_io_words_tests, 2623
io_write_block
io.c, 2290
io.h, 1803
IoAlign
EFI_BLOCK_IO_MEDIA, 162
ioapic.c
g_ioapic_base, 906
g_ioapic_gsi_base, 906
g_ioapic_ready, 906
g_override_count, 906
g_overrides, 906
g_redir_low, 906
g_redir_valid, 907
ioapic_init, 901
ioapic_route_legacy_irq, 902
ioapic_unmask_legacy_irq, 903
ioapic_write, 904
ISO_POLARITY_HIGH, 900
ISO_POLARITY_LOW, 900
ISO_POLARITY_MASK, 900
ISO_TRIGGER_EDGE, 900
ISO_TRIGGER_LEVEL, 901
ISO_TRIGGER_MASK, 901
MADT_TYPE_INT_SRC, 901
MADT_TYPE_IOAPIC, 901
MAX_OVERRIDES, 901
MAX_REDIR_ENTRIES, 901
parse_madt, 904
sig4_eq, 905
sig8_eq, 905
ioapic.h
ioapic_init, 628
ioapic_route_legacy_irq, 629
ioapic_unmask_legacy_irq, 630
ioapic_init

ioapic.c, 901
ioapic.h, 628
ioapic_route_legacy_irq
ioapic.c, 902
ioapic.h, 629
ioapic_unmask_legacy_irq
ioapic.c, 903
ioapic.h, 630
ioapic_write
ioapic.c, 904
ip
VM, 330
is_artemis
sk_blk_attach_pending_t, 266
is_block_header
capsule_loader.c, 1125
is_connect
xhci_dev_t, 374
is_executable_ptr
VMHostServices, 342
is_live
VMPhysics, 345
is_open
blkio_dev, 117
is_ram_type
kernel_main.c, 1413
pmm.c, 1461
vmm.c, 1473
is_stick
BootDisk, 126
is_usable_memory
pmm.c, 1461
is_warm
rolling_window_view_t, 255
RollingWindowOfTruth, 256
VMFleetWindow, 341
isalnum
ctype.h, 565
isalpha
ctype.h, 566
iscntrl
ctype.h, 566
isdigit
ctype.h, 566
islower
ctype.h, 567
ISO_POLARITY_HIGH
ioapic.c, 900
ISO_POLARITY_LOW
ioapic.c, 900
ISO_POLARITY_MASK
ioapic.c, 900
ISO_TRIGGER_EDGE
ioapic.c, 900
ISO_TRIGGER_LEVEL
ioapic.c, 901
ISO_TRIGGER_MASK
ioapic.c, 901
isprint
ctype.h, 567
ispunct
ctype.h, 567
isqrt
doe_metrics.c, 2263
isr_common_handler
interrupts.c, 980
isr_status
VirtInputState, 307
isr_stub0
interrupts.c, 981
isspace
ctype.h, 568
ist
idt_entry, 213
interrupts.c, 987
isupper
ctype.h, 568
isxdigit
ctype.h, 568
itoa_simple
kernel_main.c, 1414

joint_weight_q16
CDTuning, 139

K
sha512.c, 1224
KARGS_DEFAULT_HEAP_SIZE
kernel_args.h, 635
KARGS_DEFAULT_LOG_LEVEL
kernel_args.h, 635
KARGS_DEFAULT_RUN_DOE
kernel_args.h, 635
KARGS_DEFAULT_STACK_SIZE
kernel_args.h, 635
KARGS_LOG_DEBUG
kernel_args.h, 635
KARGS_LOG_ERROR
kernel_args.h, 635
KARGS_LOG_INFO
kernel_args.h, 635
KARGS_LOG_WARN
kernel_args.h, 635
KARGS_SYSTEM_DEFAULT
kernel_args.h, 635
KARGS_SYSTEM_V3
kernel_args.h, 636
KARGS_SYSTEM_V4
kernel_args.h, 636
KBD_EVENTS
usb_kbd.c, 1510
kbd_input.c
g_kbd_alt_down, 1331
g_kbd_caps_lock, 1331
g_kbd_num_lock, 1331
g_kbd_poll_tick, 1331
g_kbd_shift_down, 1331

sk_input_drop, 1326
sk_input_getc, 1327
sk_kbd_getc, 1329
sk_kbd_keypad, 1329
sk_kbd_shifted, 1331
sk_kbd_show_locks, 1330
SK_KBD_TABLE_SIZE, 1324
sk_kbd_unshifted, 1331
SK_KEY_102ND, 1324
SK_KEY_BACKSPACE, 1325
SK_KEY_CAPSLOCK, 1325
SK_KEY_ENTER, 1325
SK_KEY_KPENTER, 1325
SK_KEY_KPSLASH, 1325
SK_KEY_LEFTALT, 1325
SK_KEY_LEFTSHIFT, 1325
SK_KEY_NUMLOCK, 1325
SK_KEY_RIGHTALT, 1325
SK_KEY_RIGHTSHIFT, 1325
SK_KEY_TAB, 1326
kbd_input.h
sk_input_drop, 631
sk_input_getc, 632
kbw_alt_tab
keyboard_words.c, 3056
kbw_debug
keyboard_words.c, 3057
kbw_key_event
keyboard_words.c, 3057
kbw_scan
keyboard_words.c, 3058
kbw_vdebug
keyboard_words.c, 3058
kbw_vevent
keyboard_words.c, 3059
kernel/include/starkernel/apic.h, 397
kernel/include/starkernel/arch.h, 409
kernel/include/starkernel/artemis_sig.h, 417
kernel/include/starkernel/blkio_usb.h, 423
kernel/include/starkernel/boot_info_offsets.h, 425
kernel/include/starkernel/capsule.h, 425
kernel/include/starkernel/capsule_birth.h, 435
kernel/include/starkernel/capsule_blocks.h, 453
kernel/include/starkernel/capsule_console.h, 455
kernel/include/starkernel/capsule_generated.h, 455
kernel/include/starkernel/capsule_loader.h, 456
kernel/include/starkernel/capsule_mint.h, 462
kernel/include/starkernel/capsule_run.h, 463
kernel/include/starkernel/capsule_runcap.h, 471
kernel/include/starkernel/capsule_sig.h, 471
kernel/include/starkernel/capsule_vm_physics.h, 474
kernel/include/starkernel/capsule_vm_switch_signal.h, 483
kernel/include/starkernel/capsule_wirebind.h, 483
kernel/include/starkernel/capsule_zuse_boot.h, 484
kernel/include/starkernel/cmdline.h, 484
kernel/include/starkernel/console.h, 487
kernel/include/starkernel/doe_log.h, 507
kernel/include/starkernel/ed25519.h, 512
kernel/include/starkernel/elf64.h, 516
kernel/include/starkernel/elf_loader.h, 526
kernel/include/starkernel/fdt.h, 528
kernel/include/starkernel/fdt_uart.h, 542
kernel/include/starkernel/fe25519.h, 544
kernel/include/starkernel/framebuffer.h, 553
kernel/include/starkernel/freestanding/assert.h, 564
kernel/include/starkernel/freestanding/ctype.h, 565
kernel/include/starkernel/freestanding/errno.h, 570
kernel/include/starkernel/freestanding/inttypes.h, 571
kernel/include/starkernel/freestanding/math.h, 574
kernel/include/starkernel/freestanding/README.md, 1943
kernel/include/starkernel/freestanding/sched.h, 576
kernel/include/starkernel/freestanding/signal.h, 577
kernel/include/starkernel/freestanding/stdio.h, 579
kernel/include/starkernel/freestanding/stdlib.h, 592
kernel/include/starkernel/freestanding/string.h, 598
kernel/include/starkernel/freestanding/sys/README.md, 1943
kernel/include/starkernel/freestanding/sys/time.h, 605
kernel/include/starkernel/freestanding/sys/types.h, 607
kernel/include/starkernel/freestanding/time.h, 606
kernel/include/starkernel/gpt.h, 609
kernel/include/starkernel/hal/hal.h, 616
kernel/include/starkernel/hal/README.md, 1943
kernel/include/starkernel/hal_memory.h, 616
kernel/include/starkernel/homeblocks_sig.h, 618
kernel/include/starkernel/i8042.h, 622
kernel/include/starkernel/ioapic.h, 627
kernel/include/starkernel/kbd_input.h, 631
kernel/include/starkernel/kernel_args.h, 634
kernel/include/starkernel/kmalloc.h, 636
kernel/include/starkernel/log.h, 1804
kernel/include/starkernel/log_attrib.h, 642
kernel/include/starkernel/log_region.h, 642
kernel/include/starkernel/pci.h, 647
kernel/include/starkernel/pie_hidden.h, 659
kernel/include/starkernel/plic.h, 659
kernel/include/starkernel/pmm.h, 664
kernel/include/starkernel/q48_16.h, 1918
kernel/include/starkernel/README.md, 1943
kernel/include/starkernel/repl.h, 1943
kernel/include/starkernel/rng.h, 669
kernel/include/starkernel/rpi5_dtb.h, 672
kernel/include/starkernel/rpi5_mailbox.h, 674
kernel/include/starkernel/rpi5_mmu.h, 676
kernel/include/starkernel/rpi5_native_boot.h, 679
kernel/include/starkernel/rpi5_native_entry.h, 681
kernel/include/starkernel/scalar25519.h, 682
kernel/include/starkernel/session.h, 686
kernel/include/starkernel/sha512.h, 686
kernel/include/starkernel/timer.h, 689
kernel/include/starkernel/ttf.h, 706
kernel/include/starkernel/uefi.h, 717
kernel/include/starkernel/usb_kbd.h, 734
kernel/include/starkernel/user_identity_seed.h, 738

kernel/include/starkernel/v4/sk_v4.h, 740
kernel/include/starkernel/v4/sk_v4_grow.h, 742
kernel/include/starkernel/v4/sk_v4_identity.h, 744
kernel/include/starkernel/v4/sk_v4_marker.h, 754
kernel/include/starkernel/v4/sk_v4_mint.h, 756
kernel/include/starkernel/virtio_blk.h, 758
kernel/include/starkernel/virtio_input.h, 761
kernel/include/starkernel/virtio_rng.h, 764
kernel/include/starkernel/vm/arena.h, 767
kernel/include/starkernel/vm/bootstrap/README.md, 1943
kernel/include/starkernel/vm/bootstrap/sk_vm_bootstrap.h, 767
kernel/include/starkernel/vm/kernel_hermes.h, 768
kernel/include/starkernel/vm/parity.h, 768
kernel/include/starkernel/vm/README.md, 1943
kernel/include/starkernel/vm/stadium.h, 775
kernel/include/starkernel/vm/stadium_blocks.h, 775
kernel/include/starkernel/vm/stadium_words.h, 776
kernel/include/starkernel/vm/switch.h, 776
kernel/include/starkernel/vm_identity.h, 777
kernel/include/starkernel/vm_log_buffer.h, 777
kernel/include/starkernel/vm_uuid.h, 777
kernel/include/starkernel/vmm.h, 777
kernel/include/starkernel/vt100.h, 784
kernel/include/starkernel/x509_ed25519.h, 796
kernel/include/starkernel/xhci.h, 801
kernel/include/starkernel/xhci_driver.h, 820
kernel/include/starkernel/xxhash64.h, 848
kernel/include/starkernel/zuse_cert_devblock.h, 853
kernel/include/starkernel/zuse_genesis_marker.h, 855
kernel/src/arch/aarch64/apic.c, 907
kernel/src/arch/aarch64/arch.c, 949
kernel/src/arch/aarch64/interrupts.c, 967
kernel/src/arch/aarch64/README.md, 1943
kernel/src/arch/aarch64/rpi5_dtb.c, 856
kernel/src/arch/aarch64/rpi5_mailbox.c, 860
kernel/src/arch/aarch64/rpi5_mmu.c, 871
kernel/src/arch/aarch64/rpi5_native_boot.c, 878
kernel/src/arch/aarch64/timer.c, 1006
kernel/src/arch/amd64/apic.c, 922
kernel/src/arch/amd64/arch.c, 955
kernel/src/arch/amd64/i8042.c, 891
kernel/src/arch/amd64/interrupts.c, 977
kernel/src/arch/amd64/ioapic.c, 899
kernel/src/arch/amd64/README.md, 1943
kernel/src/arch/amd64/timer.c, 1012
kernel/src/arch/riscv64/apic.c, 939
kernel/src/arch/riscv64/arch.c, 961
kernel/src/arch/riscv64/interrupts.c, 989
kernel/src/arch/riscv64/plic.c, 996
kernel/src/arch/riscv64/README.md, 1943
kernel/src/arch/riscv64/timer.c, 1052
kernel/src/artemis_sig.c, 1061
kernel/src/boot/cmdline.c, 1065
kernel/src/boot/elf_loader.c, 1071
kernel/src/boot/README.md, 1943
kernel/src/boot/reloc_stub.c, 1078
kernel/src/boot/uefi_loader.c, 1081
kernel/src/capsule/capsule_birth.c, 1089
kernel/src/capsule/capsule_blocks.c, 1111
kernel/src/capsule/capsule_console.c, 1111
kernel/src/capsule/capsule_find.c, 1114
kernel/src/capsule/capsule_loader.c, 1117
kernel/src/capsule/capsule_mint.c, 1127
kernel/src/capsule/capsule_run.c, 1136
kernel/src/capsule/capsule_runcap.c, 1146
kernel/src/capsule/capsule_sig.c, 1147
kernel/src/capsule/capsule_validate.c, 1151
kernel/src/capsule/capsule_vm_hooks.c, 1153
kernel/src/capsule/capsule_vm_physics.c, 1158
kernel/src/capsule/capsule_vm_switch_signal.c, 1171
kernel/src/capsule/capsule_wirebind.c, 1171
kernel/src/capsule/capsule_zuse_boot.c, 1183
kernel/src/capsule/mama_forth_words.c, 1192
kernel/src/capsule/README.md, 1943
kernel/src/capsule/vm_identity.c, 1193
kernel/src/capsule/vm_uuid.c, 1195
kernel/src/crypto/ed25519.c, 1195
kernel/src/crypto/fe25519.c, 1206
kernel/src/crypto/scalar25519.c, 1215
kernel/src/crypto/sha512.c, 1220
kernel/src/crypto/x509_ed25519.c, 1224
kernel/src/doe_log.c, 1231
kernel/src/gpt.c, 1237
kernel/src/hal/console.c, 1250
kernel/src/hal/fdt.c, 1284
kernel/src/hal/fdt_uart.c, 1306
kernel/src/hal/font_8x16.c, 1310
kernel/src/hal/framebuffer.c, 1312
kernel/src/hal/hal.c, 1323
kernel/src/hal/host_services.c, 1323
kernel/src/hal/kbd_input.c, 1323
kernel/src/hal/memory.c, 1332
kernel/src/hal/README.md, 1943
kernel/src/hal/ttf.c, 1334
kernel/src/hal/vt100.c, 1356
kernel/src/hash/README.md, 1943
kernel/src/hash/xxhash64.c, 1395
kernel/src/heartbeat.c, 1403
kernel/src/homeblocks_sig.c, 1410
kernel/src/kernel_main.c, 1412
kernel/src/log_region.c, 1427
kernel/src/math/q48_16.c, 1433
kernel/src/math/README.md, 1943
kernel/src/memory/kmalloc.c, 1441
kernel/src/memory/pmm.c, 1457
kernel/src/memory/README.md, 1943
kernel/src/memory/vmm.c, 1468
kernel/src/pci/pci.c, 1482
kernel/src/pci/README.md, 1943
kernel/src/README.md, 1943
kernel/src/repl.c, 2442
kernel/src/rng/rng.c, 1500
kernel/src/usb/blkio_usb.c, 1503
kernel/src/usb/usb_kbd.c, 1509

kernel/src/usb/xhci.c, 1515
kernel/src/v4/sk_v4.c, 1560
kernel/src/v4/sk_v4_grow.c, 1568
kernel/src/v4/sk_v4_identity.c, 1572
kernel/src/v4/sk_v4_marker.c, 1597
kernel/src/v4/sk_v4_mint.c, 1600
kernel/src/virtio/README.md, 1943
kernel/src/virtio/virtio_blk.c, 1604
kernel/src/virtio/virtio_input.c, 1615
kernel/src/virtio/virtio_rng.c, 1625
kernel/src/vm/alloc_kernel.c, 2400
kernel/src/vm/arena.c, 1633
kernel/src/vm/bootstrap/README.md, 1943
kernel/src/vm/bootstrap/sk_vm_bootstrap.c, 1633
kernel/src/vm/host/README.md, 1943
kernel/src/vm/host/shim.c, 1633
kernel/src/vm/kernel_hermes.c, 1633
kernel/src/vm/parity.c, 1633
kernel/src/vm/q48_stubs.c, 1646
kernel/src/vm/README.md, 1943
kernel/src/vm/session.c, 1648
kernel/src/vm/stadium.c, 1648
kernel/src/vm/stadium_blocks.c, 1648
kernel/src/vm/stadium_words.c, 1648
kernel/src/vm/switch.c, 1649
kernel/src/vm/vm_bootstrap.c, 2719
kernel/src/vm/vm_core.c, 1649
kernel/src/vm/vm_internal.h, 2734
kernel/src/vm/vm_log_buffer.c, 1694
kernel/src/vm/vm_runtime.c, 1694
kernel_args.h
  KARGS_DEFAULT_HEAP_SIZE, 635
  KARGS_DEFAULT_LOG_LEVEL, 635
  KARGS_DEFAULT_RUN_DOE, 635
  KARGS_DEFAULT_STACK_SIZE, 635
  KARGS_LOG_DEBUG, 635
  KARGS_LOG_ERROR, 635
  KARGS_LOG_INFO, 635
  KARGS_LOG_WARN, 635
  KARGS_SYSTEM_DEFAULT, 635
  KARGS_SYSTEM_V3, 636
  KARGS_SYSTEM_V4, 636
  KERNEL_ARGS_CMDLINE_MAX, 636
  REBOOT_MAX_TRIES, 636
KERNEL_ARGS_CMDLINE_MAX
  kernel_args.h, 636
kernel_elf_buffer
  uefi_loader.c, 1088
kernel_elf_size
  uefi_loader.c, 1089
kernel_main
  kernel_main.c, 1415
  rpi5_native_boot.c, 884
kernel_main.c
  g_sk_runtime_services, 1426
  is_ram_type, 1413
  itoa_simple, 1414
  kernel_main, 1415
  kernel_main_deep, 1418
  print_banner, 1421
  print_boot_info, 1422
  print_heap_stats, 1423
  print_pmm_stats, 1424
  print_uint, 1425
kernel_main_deep
  kernel_main.c, 1418
KERNEL_MAX_SIZE
  uefi_loader.c, 1082
kernel_stack_base
  BootInfo, 127
kernel_stack_size
  BootInfo, 127
KernelArgs, 218
  heap_size, 219
  log_level, 219
  raw, 219
  run_doe, 219
  screen_boot, 219
  stack_size, 219
  system, 219
KernelEntry
  uefi_loader.c, 1082
key_for
  usb_kbd.c, 1510
Keyboard Words, 73
keyboard_words.c
  g_i8042_isr_count, 3061
  g_spurious_count, 3061
  g_virtio_input_isr_count, 3062
  kbw_alt_tab, 3056
  kbw_debug, 3057
  kbw_key_event, 3057
  kbw_scan, 3058
  kbw_vdebug, 3058
  kbw_vevent, 3059
  register_keyboard_words, 3059
  sk_key_event_poll, 3060
keyboard_words.h
  register_keyboard_words, 2976
  sk_key_event_poll, 2977
keycode
  usb_kbd.c, 1514
kfree
  kmalloc.c, 1448
  kmalloc.h, 637
kind
  fdt_uart_t, 188
kmalloc
  kmalloc.c, 1449
  kmalloc.h, 638
kmalloc.c
  account_allocation, 1444
  account_free, 1445
  align_up_size, 1445
  allocate_from_block, 1446
  can_split, 1447

coalesce_neighbors, 1448
heap_base, 1456
heap_block_t, 1444
heap_head, 1456
heap_initialized, 1456
heap_size, 1457
heap_stats, 1457
kfree, 1448
kmalloc, 1449
kmalloc_aligned, 1450
kmalloc_debug_census, 1451
kmalloc_debug_census_bytes, 1451
KMALLOC_DEFAULT_HEAP_SIZE, 1443
kmalloc_get_stats, 1451
kmalloc_heap_base_addr, 1452
kmalloc_heap_end_addr, 1452
kmalloc_init, 1452
kmalloc_is_initialized, 1453
KMALLOC_MAGIC, 1443
KMALLOC_MIN_ALIGN, 1443
KMALLOC_MIN_PAGES, 1443
min_splittable_payload, 1453
normalize_alignment, 1454
pointer_within_heap, 1455
update_free_bytes, 1455
kmalloc.h
kfree, 637
kmalloc, 638
kmalloc_aligned, 638
kmalloc_debug_census, 639
kmalloc_debug_census_bytes, 640
kmalloc_get_stats, 640
kmalloc_heap_base_addr, 640
kmalloc_heap_end_addr, 640
kmalloc_init, 641
kmalloc_is_initialized, 641
kmalloc_aligned
kmalloc.c, 1450
kmalloc.h, 638
kmalloc_debug_census
kmalloc.c, 1451
kmalloc.h, 639
kmalloc_debug_census_bytes
kmalloc.c, 1451
kmalloc.h, 640
KMALLOC_DEFAULT_HEAP_SIZE
kmalloc.c, 1443
kmalloc_get_stats
kmalloc.c, 1451
kmalloc.h, 640
kmalloc_heap_base_addr
kmalloc.c, 1452
kmalloc.h, 640
kmalloc_heap_end_addr
kmalloc.c, 1452
kmalloc.h, 640
kmalloc_init
kmalloc.c, 1452
kmalloc.h, 641
kmalloc_is_initialized
kmalloc.c, 1453
kmalloc.h, 641
KMALLOC_MAGIC
kmalloc.c, 1443
KMALLOC_MIN_ALIGN
kmalloc.c, 1443
KMALLOC_MIN_PAGES
kmalloc.c, 1443
kmalloc_prefix_t, 220
block, 220
magic, 220
kmalloc_stats_t, 220
free_bytes, 221
peak_bytes, 221
total_bytes, 221
used_bytes, 221

L1_heat_tracking
ssm_config_t, 272
l2_gib
rpi5_mmu.c, 873
L2_rolling_window
ssm_config_t, 272
L3_linear_decay
ssm_config_t, 273
L4_pipelining
ssm_config_t, 273
L5_window_inference
ssm_config_t, 273
L6_decay_inference
ssm_config_t, 273
L7_adaptive_heartrate
ssm_config_t, 273
l8_mode
HeartbeatTickSnapshot, 203
l8_regime_modulation_q16
capsule_vm_physics.c, 1160
L_BYTES
scalar25519.c, 1220
label
blk_volume_meta_t, 113
lapic_base
apic.c, 938
LAPIC_DEFAULT_PHYS
apic.c, 925
lapic_phys_base
apic.c, 938
vmm.c, 1481
lapic_read
apic.c, 934
LAPIC_VIRT_BASE
apic.c, 925
lapic_write
apic.c, 935
last_active_ns
DictPhysics, 151
last_active_tick

VMPhysics, 345
last_allocated
blk_volume_meta_t, 113
last_bucket_reorg_ns
VM, 331
last_check_tick
InferenceOutputs, 218
last_checked_accuracy
PipelineGlobalMetrics, 246
last_checked_window_size
PipelineGlobalMetrics, 246
last_comp_version
fdt_header_t, 187
last_decay_check_ns
VM, 331
last_decay_ns
DictPhysics, 151
last_decay_tick
DictPhysics, 151
last_inference_outputs
VM, 331
last_inference_tick
HeartbeatState, 200
last_lba
gpt_part_t, 194
last_pattern_diversity
RollingWindowOfTruth, 257
last_stale_count
InferenceInputs, 215
last_total_heat
InferenceInputs, 216
last_tsc
time_trust_state, 282
last_used_idx
VirtBlkState, 305
VirtInputState, 308
VirtRngState, 317
LastAccessTime
EFI_FILE_INFO, 166
LastBlock
EFI_BLOCK_IO_MEDIA, 162
latest
VM, 331
latest_word_id
ParityPacket, 226
lbn_to_slot
block_subsystem.c, 2195
lbn_to_slot_pbn
block_subsystem.c, 2196
le32
gpt.c, 1244
le64
gpt.c, 1244
leave_addrs
ControlFlowState, 145
leave_mark_sp
ControlFlowState, 145
leave_mark_stack
ControlFlowState, 146
leave_sp
ControlFlowState, 146
len
der_span_t, 146
VirtqDesc, 313
VirtqUsedElem, 315
Length
EFI_DEVICE_PATH_PROTOCOL, 164
length
__attribute__, 81
level
log_slot_t, 223
Profiler, 249
profiler.c, 2441
level_colors
log.c, 2300
level_names
log.c, 2300
lidt
interrupts.c, 982
lifecycle_words_hosted.c
extract_name, 3063
hosted_lifecycle_birth, 3064
hosted_lifecycle_kill, 3064
hosted_lifecycle_pause, 3065
hosted_lifecycle_resume, 3066
hosted_lifecycle_use, 3066
NAME_MAX_LEN, 3063
register_lifecycle_words, 3067
LIKELY
vm.h, 2007
limit
arch.c, 960
gdtr, 192
idtr, 214
interrupts.c, 987
limit_low
arch.c, 961
gdt_entry, 192
line_ptr
editor_words.c, 2906
link
DictEntry, 149
vm.h, 2076
LITHOS_VERSION
version.h, 2000
lithos_version
repl.c, 2468
LITHOS_VERSION_STR
version.h, 2000
load_avg_milli
physics_host_snapshot, 238
load_cfg_from_esp
uefi_loader.c, 1086
load_cr3
vmm.c, 1473
load_kernel_from_esp

uefi_loader.c, 1087
loaded
cache_slot_t, 129
LOADER_BLOCK_SIZE
capsule_loader.c, 1119
LoadImage
_EFI_BOOT_SERVICES, 88
LoadOptions
EFI_LOADED_IMAGE_PROTOCOL, 171
LoadOptionsSize
EFI_LOADED_IMAGE_PROTOCOL, 172
loca_len
ttf_font_t, 290
loca_off
ttf_font_t, 290
local_apic_address
__attribute__, 82
locality_high_threshold_q16
CDTuning, 139
locality_lookback
CDTuning, 139
locality_q16
CDRegimeMetrics, 138
LocateDevicePath
_EFI_BOOT_SERVICES, 88
LocateHandle
_EFI_BOOT_SERVICES, 88
LocateHandleBuffer
_EFI_BOOT_SERVICES, 88
LocateProtocol
_EFI_BOOT_SERVICES, 88
log
math.h, 576
Log Words, 74
log.c
color_reset, 2299
current_level, 2299
get_timestamp, 2292
level_colors, 2300
level_names, 2300
log_get_level, 2294
log_message, 2295
log_set_level, 2297
log_test_result, 2297
test_fail_color, 2300
test_pass_color, 2300
test_skip_color, 2300
log.h
LOG_DEBUG, 1804, 1807
LOG_ERROR, 1804, 1807
log_get_level, 1805, 1808
LOG_INFO, 1804, 1807
log_message, 1805, 1808
LOG_MSG_LINE_MAX, 1804, 1807
LOG_NONE, 1804, 1807
log_set_level, 1805, 1810
log_set_vm, 1806, 1811
LOG_TEST, 1804, 1807
log_test_result, 1806, 1811
LOG_WARN, 1804, 1807
LogLevel, 1804, 1807
TEST_ERROR, 1805, 1808
TEST_FAIL, 1805, 1808
TEST_PASS, 1805, 1808
TEST_SKIP, 1805, 1808
TestResult, 1805, 1807
log_attrib.h
vm_log_attributed_vm, 642
LOG_BLOCKS
vm.h, 2007
LOG_BLOCKS_END
vm.h, 2007
LOG_BLOCKS_START
vm.h, 2007
LOG_DEBUG
log.h, 1804, 1807
log_do_debug
log_words.c, 3069
log_do_emit
log_words.c, 3070
log_do_error
log_words.c, 3070
log_do_info
log_words.c, 3071
log_do_test
log_words.c, 3071
log_do_warn
log_words.c, 3072
log_emit_string
log_words.c, 3072
LOG_ERROR
log.h, 1804, 1807
log_get_level
log.c, 2294
log.h, 1805, 1808
LOG_INFO
log.h, 1804, 1807
LOG_LAYER1_MAX_LINES
vm.h, 2008
log_level
CLIConfig, 143
KernelArgs, 219
log_level_explicitly_set
CLIConfig, 143
LOG_LINE_MAX
vm.h, 2008
LOG_LINES_PER_BLOCK
vm.h, 2008
log_message
log.c, 2295
log.h, 1805, 1808
LOG_MSG_LINE_MAX
log.h, 1804, 1807
LOG_NONE
log.h, 1804, 1807
log_region.c

log_region_append, 1427
log_region_ctrl_compute_crc, 1428
log_region_ctrl_init_fresh, 1429
log_region_ctrl_read, 1429
log_region_ctrl_write, 1430
log_region_read_recent, 1431
log_region_read_slot, 1431
log_region_slot_devblock_from_top, 1432
log_region_write_slot, 1432
log_region.h
log_region_append, 646
log_region_ctrl_size_check, 646
LOG_REGION_DEVBLOCK_FROM_TOP_BASE, 644
LOG_REGION_GET_MAGIC, 644
LOG_REGION_GET_VERSION, 644
LOG_REGION_GROWTH_INCREMENT, 644
LOG_REGION_INITIAL_DEVBLOCKS, 644
LOG_REGION_MAGIC, 645
LOG_REGION_MAX_DEVBLOCKS, 645
LOG_REGION_PACK, 645
LOG_REGION_PROTECTED_MAX_LEVEL, 645
log_region_read_recent, 646
LOG_REGION_VERSION_0, 645
LOG_SLOT_MSG_MAX, 645
LOG_SLOT_SIZE, 645
log_slot_size_check, 646
LOG_SLOT_SOURCE_MAX, 645
LOG_SLOTS_PER_DEVBLOCK, 645
log_region_append
log_region.c, 1427
log_region.h, 646
log_region_ctrl_compute_crc
log_region.c, 1428
log_region_ctrl_init_fresh
log_region.c, 1429
log_region_ctrl_read
log_region.c, 1429
log_region_ctrl_size_check
log_region.h, 646
log_region_ctrl_t, 221
_pad, 222
devblocks, 222
hdr_crc, 222
head_slot, 222
magic, 222
record_count, 222
tail_slot, 222
log_region_ctrl_write
log_region.c, 1430
LOG_REGION_DEVBLOCK_FROM_TOP_BASE
log_region.h, 644
log_region_devblocks
artemis_sig_t, 101
LOG_REGION_GET_MAGIC
log_region.h, 644
LOG_REGION_GET_VERSION
log_region.h, 644
LOG_REGION_GROWTH_INCREMENT
log_region.h, 644
LOG_REGION_INITIAL_DEVBLOCKS
log_region.h, 644
LOG_REGION_MAGIC
log_region.h, 645
LOG_REGION_MAX_DEVBLOCKS
log_region.h, 645
log_region_offset
artemis_sig_t, 102
LOG_REGION_PACK
log_region.h, 645
LOG_REGION_PROTECTED_MAX_LEVEL
log_region.h, 645
log_region_read_recent
log_region.c, 1431
log_region.h, 646
log_region_read_slot
log_region.c, 1431
log_region_slot_devblock_from_top
log_region.c, 1432
LOG_REGION_VERSION_0
log_region.h, 645
log_region_write_slot
log_region.c, 1432
log_set_level
log.c, 2297
log.h, 1805, 1810
log_set_vm
log.h, 1806, 1811
LOG_SLOT_MSG_MAX
log_region.h, 645
LOG_SLOT_SIZE
log_region.h, 645
log_slot_size_check
log_region.h, 646
LOG_SLOT_SOURCE_MAX
log_region.h, 645
log_slot_t, 223
level, 223
msg, 223
msg_len, 223
source, 223
timestamp, 223
LOG_SLOTS_PER_DEVBLOCK
log_region.h, 645
log_str_emit
log_words.c, 3073
LOG_TEST
log.h, 1804, 1807
log_test_result
log.c, 2297
log.h, 1806, 1811
LOG_WARN
log.h, 1804, 1807
log_word_debug
log_words.c, 3074
log_word_debug_quote

log_words.c, 3074
log_word_debug_str
log_words.c, 3075
log_word_error
log_words.c, 3076
log_word_error_quote
log_words.c, 3076
log_word_error_str
log_words.c, 3077
log_word_get_level
log_words.c, 3077
log_word_info
log_words.c, 3078
log_word_info_quote
log_words.c, 3079
log_word_info_str
log_words.c, 3079
log_word_set_level
log_words.c, 3080
log_word_test
log_words.c, 3081
log_word_test_quote
log_words.c, 3081
log_word_test_str
log_words.c, 3082
log_word_warn
log_words.c, 3082
log_word_warn_quote
log_words.c, 3083
log_word_warn_str
log_words.c, 3084
log_words.c
log_do_debug, 3069
log_do_emit, 3070
log_do_error, 3070
log_do_info, 3071
log_do_test, 3071
log_do_warn, 3072
log_emit_string, 3072
log_str_emit, 3073
log_word_debug, 3074
log_word_debug_quote, 3074
log_word_debug_str, 3075
log_word_error, 3076
log_word_error_quote, 3076
log_word_error_str, 3077
log_word_get_level, 3077
log_word_info, 3078
log_word_info_quote, 3079
log_word_info_str, 3079
log_word_set_level, 3080
log_word_test, 3081
log_word_test_quote, 3081
log_word_test_str, 3082
log_word_warn, 3082
log_word_warn_quote, 3083
log_word_warn_str, 3084
register_log_words, 3084
log_words.h
register_log_words, 2979
logical_word_and
logical_words.c, 3087
logical_word_equals
logical_words.c, 3088
logical_word_false
logical_words.c, 3088
logical_word_greater_equal
logical_words.c, 3089
logical_word_greater_than
logical_words.c, 3089
logical_word_invert
logical_words.c, 3090
logical_word_less_equal
logical_words.c, 3090
logical_word_less_than
logical_words.c, 3091
logical_word_lshift
logical_words.c, 3091
logical_word_not
logical_words.c, 3092
logical_word_not_equals
logical_words.c, 3092
logical_word_or
logical_words.c, 3093
logical_word_rshift
logical_words.c, 3093
logical_word_suites
logical_words_test.c, 2626
logical_word_true
logical_words.c, 3094
logical_word_u_greater_than
logical_words.c, 3094
logical_word_u_less_than
logical_words.c, 3095
logical_word_within
logical_words.c, 3095
logical_word_xor
logical_words.c, 3096
logical_word_zero_equals
logical_words.c, 3096
logical_word_zero_greater
logical_words.c, 3097
logical_word_zero_less
logical_words.c, 3097
logical_word_zero_not_equal
logical_words.c, 3098
logical_words.c
FORTH_FALSE, 3087
FORTH_TRUE, 3087
logical_word_and, 3087
logical_word_equals, 3088
logical_word_false, 3088
logical_word_greater_equal, 3089
logical_word_greater_than, 3089
logical_word_invert, 3090
logical_word_less_equal, 3090

logical_word_less_than, 3091
logical_word_lshift, 3091
logical_word_not, 3092
logical_word_not_equals, 3092
logical_word_or, 3093
logical_word_rshift, 3093
logical_word_true, 3094
logical_word_u_greater_than, 3094
logical_word_u_less_than, 3095
logical_word_within, 3095
logical_word_xor, 3096
logical_word_zero_equals, 3096
logical_word_zero_greater, 3097
logical_word_zero_less, 3097
logical_word_zero_not_equal, 3098
register_logical_words, 3098
logical_words.h
register_logical_words, 2981
logical_words_test.c
logical_word_suites, 2626
run_logical_words_tests, 2625
LogicalPartition
EFI_BLOCK_IO_MEDIA, 163
LogLevel
log.h, 1804, 1807
lookup_strategy
VM, 331
loop_iterations
Profiler, 250
lru_index
HotwordsCache_s, 208
LVT_MASKED
apic.c, 925
LVT_MODE_ONESHOT
apic.c, 925
LVT_MODE_PERIODIC
apic.c, 925

m5_time_trust
HeartbeatState, 200
m5_variance
HeartbeatState, 200
MADT_TYPE_INT_SRC
ioapic.c, 901
MADT_TYPE_IOAPIC
ioapic.c, 901
magic
artemis_sig_t, 102
blk_meta_t, 110
blk_volume_meta_t, 113
CapsuleDirHeader, 131
fdt_header_t, 187
homeblocks_sig_t, 206
kmalloc_prefix_t, 220
log_region_ctrl_t, 222
physics_analytics_header, 231
user_identity_seed_t, 302
zuse_cert_devblock_t, 394
zuse_genesis_marker_t, 395
zuse_genesis_marker_v2_t, 396
main
main.c, 2306
main.c
__attribute__, 2302
cleanup_and_exit, 2302
cleanup_blkio, 2303
g_blkio_opened, 2311
g_ram_base, 2311
g_ram_bytes, 2311
g_ram_is_mmap, 2311
init_blkio, 2303
init_vm_and_subsystem, 2304
main, 2306
print_usage, 2307
run_benchmark_mode, 2308
run_doe_experiment, 2308
run_optional_tests, 2309
signal_handler, 2310
STARFORTH_STATE_BYTES, 2302
make_pte
vmm.c, 1474
malloc
stdlib.h, 595
mama_forth_words.h
register_mama_forth_words, 2984
mama_forth_words_test.c
MAMA_IMPLEMENTED, 2628
mama_word_suites, 2629
run_mama_forth_words_tests, 2628
MAMA_IMPLEMENTED
mama_forth_words_test.c, 2628
mama_word_suites
mama_forth_words_test.c, 2629
mark_buffer_dirty
block_words.c, 2830
block_words.h, 2945
MASK51
fe25519.c, 1207
mass_bytes
DictPhysics, 151
math.h
atan2, 575
ceil, 575
cos, 575
exp, 575
fabs, 575
floor, 576
HUGE_VAL, 575
INFINITY, 575
log, 576
NAN, 575
pow, 576
sin, 576
sqrt, 576
math_portable.c
erf_q48, 2312
exp_q48, 2314

sqrt_q48, 2314
math_portable.h
erf_q48, 1814
Q48_SCALE, 1813
sqrt_q48, 1815
MAX_BLOCKS
block_words.h, 2939
vm.h, 2008
max_bucket_search_ns
HotwordsStats, 211
max_cache_hit_ns
HotwordsStats, 211
max_contours
ttf_outline_t, 294
MAX_DICT_ENTRIES
parity.c, 1634
max_intrs
xhci_dev_t, 374
MAX_OVERRIDES
ioapic.c, 901
max_points
ttf_outline_t, 294
max_ports
xhci_dev_t, 375
MAX_REDIR_ENTRIES
ioapic.c, 901
max_scratchpad_bufs
xhci_dev_t, 375
max_slots
xhci_dev_t, 375
MAX_TEST_INPUT
test_common.h, 2514
MAX_TEST_OUTPUT
test_common.h, 2514
MAX_TESTS_PER_WORD
test_common.h, 2514
max_time
ProfileTimer, 252
WordProfile, 356
max_time_ns
WordStats, 358
max_transition_probability_q48
WordTransitionMetrics, 363
MaxAllocateType
uefi.h, 730
MaxMode
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE, 169
maxp_off
ttf_font_t, 290
mb_be32
rpi5_mailbox.c, 865
mb_cell_prop
rpi5_mailbox.c, 866
mb_cells
rpi5_mailbox.c, 866
mb_compatible
rpi5_mailbox.c, 867
mb_find_path
rpi5_mailbox.c, 868
MBOX_CH_PROPERTY
rpi5_mailbox.c, 861
MBOX_CODE_REQUEST
rpi5_mailbox.c, 861
MBOX_CODE_RESPONSE_OK
rpi5_mailbox.c, 861
MBOX_EMPTY
rpi5_mailbox.c, 861
MBOX_FULL
rpi5_mailbox.c, 861
MBOX_OFF_READ
rpi5_mailbox.c, 861
MBOX_OFF_STATUS0
rpi5_mailbox.c, 862
MBOX_OFF_STATUS1
rpi5_mailbox.c, 862
MBOX_OFF_WRITE
rpi5_mailbox.c, 862
mean_ns_q48
BayesianLatencyPosterior, 103
Media
_EFI_BLOCK_IO_PROTOCOL, 84
MediaId
EFI_BLOCK_IO_MEDIA, 163
median_ns_q48
BayesianLatencyPosterior, 103
MediaPresent
EFI_BLOCK_IO_MEDIA, 163
memchr
string.h, 599
memcmp
string.h, 599
memcpy
string.h, 600
memmove
string.h, 601
memory
VM, 332
memory.c
hal_mem_alloc, 1333
hal_mem_alloc_aligned, 1333
hal_mem_free, 1333
memory_allocated_bytes
DoeMetrics, 158
memory_allocations
Profiler, 250
memory_management.c
vm_addr_to_block, 2316
vm_align, 2317
vm_allot, 2318
vm_get_block_addr, 2320
memory_management.h
vm_align, 1817
vm_allot, 1818
memory_map
BootInfo, 127

memory_map_descriptor_size
BootInfo, 128
memory_map_size
BootInfo, 128
MEMORY_SIZE
io.h, 1802
memory_stats
Profiler, 250
memory_word_2fetch
memory_words.c, 3102
memory_word_2store
memory_words.c, 3102
memory_word_cells
memory_words.c, 3103
memory_word_cfetch
memory_words.c, 3104
memory_word_cstore
memory_words.c, 3105
memory_word_erase
memory_words.c, 3105
memory_word_fetch
memory_words.c, 3107
memory_word_fill
memory_words.c, 3108
memory_word_minus_store
memory_words.c, 3109
memory_word_move
memory_words.c, 3109
memory_word_plus_store
memory_words.c, 3111
memory_word_store
memory_words.c, 3112
memory_word_suites
memory_words_test.c, 2630
memory_words.c
memory_word_2fetch, 3102
memory_word_2store, 3102
memory_word_cells, 3103
memory_word_cfetch, 3104
memory_word_cstore, 3105
memory_word_erase, 3105
memory_word_fetch, 3107
memory_word_fill, 3108
memory_word_minus_store, 3109
memory_word_move, 3109
memory_word_plus_store, 3111
memory_word_store, 3112
register_memory_words, 3113
memory_words.h
register_memory_words, 2985
memory_words_test.c
memory_word_suites, 2630
run_memory_words_tests, 2630
MemoryProfile, 224
cache_hits, 224
cache_misses, 224
reads, 224
total_bytes_read, 224
total_bytes_written, 224
writes, 224
memset
string.h, 601
meta
cache_slot_t, 129
sk_v4_grow.c, 1572
meta_dirty
cache_slot_t, 129
meta_fence_blocks
blk_volume_meta_t, 114
meta_from_slice
block_subsystem.c, 2197
META_PER_BLOCK
block_subsystem.c, 2151
META_REGION_OFFSET
block_subsystem.c, 2151
META_REGION_SIZE
block_subsystem.c, 2151
meta_to_slice
block_subsystem.c, 2198
metadata_devblocks
artemis_sig_t, 102
homeblocks_sig_t, 206
metrics_from_vm
doe_metrics.c, 2263
doe_metrics.h, 1784
metrics_get_cpu_freq_mhz
doe_metrics.c, 2264
doe_metrics.h, 1785
metrics_get_cpu_temp_c
doe_metrics.c, 2265
doe_metrics.h, 1786
metrics_get_timestamp
doe_metrics.c, 2266
doe_metrics.h, 1787
metrics_print_text
doe_metrics.c, 2266
doe_metrics.h, 1787
metrics_write_csv_header
doe_metrics.c, 2267
doe_metrics.h, 1788
metrics_write_csv_row
doe_metrics.c, 2267
doe_metrics.h, 1789
metrics_write_james_law_csv_row
doe_metrics.c, 2268
doe_metrics.h, 1790
mf_index
xhci_runtime_regs_t, 390
MIB2
rpi5_mmu.c, 872
MIGRATION_SCAN_BUDGET
block_subsystem.c, 2151
migration_scan_lbn
blk_dev_slot, 106
MIGRATION_WEAR_THRESHOLD
block_subsystem.c, 2151

min_bucket_search_ns
HotwordsStats, 212
min_cache_hit_ns
HotwordsStats, 212
MIN_SAMPLES_FOR_SPECULATION
starforth_config.h, 1991
min_splittable_payload
kmalloc.c, 1453
min_time
ProfileTimer, 252
WordProfile, 356
min_time_ns
WordStats, 358
MINIMUM_PREFETCH_ROI
starforth_config.h, 1991
MINT_CERT_DEVBLOCK
capsule_mint.c, 1128
MINT_CERT_DEVBLOCKS
capsule_mint.c, 1128
MINT_DEFAULT_PERSONALITY
capsule_mint.c, 1135
MINT_IDENT_SECTORS
sk_v4_mint.c, 1600
MINT_IDENTITY_SRC_DEVBLOCKS
capsule_mint.c, 1129
MINT_IDENTITY_SRC_OFFSET
capsule_mint.c, 1129
MINT_METADATA_DEVBLOCKS
capsule_mint.c, 1129
MINT_MIN_BYTES
sk_v4_mint.c, 1600
MINT_RESTRICTED_PERSONALITY
capsule_mint.c, 1135
MINT_ZERO_FBLOCKS
sk_v4_mint.c, 1600
minted_time_ns
homeblocks_sig_t, 206
Minute
EFI_TIME, 176
minzu
block_subsystem.c, 2199
MISPREDICTION_COST_Q48
starforth_config.h, 1991
misprediction_cost_q48
WordTransitionMetrics, 363
misses
HotwordsStats, 212
mixed_arithmetic_word_suites
mixed_arithmetic_words_test.c, 2632
mixed_arithmetic_words.c
mixed_math_word_m_minus, 3115
mixed_math_word_m_plus, 3116
mixed_math_word_m_slash_mod, 3117
mixed_math_word_m_star, 3118
mixed_math_word_mod, 3119
mixed_math_word_slash_mod, 3120
mixed_math_word_star_slash, 3120
mixed_math_word_star_slash_mod, 3121
register_mixed_arithmetic_words, 3122
mixed_arithmetic_words.h
register_mixed_arithmetic_words, 2987
mixed_arithmetic_words_test.c
mixed_arithmetic_word_suites, 2632
run_mixed_arithmetic_words_tests, 2631
mixed_math_word_m_minus
mixed_arithmetic_words.c, 3115
mixed_math_word_m_plus
mixed_arithmetic_words.c, 3116
mixed_math_word_m_slash_mod
mixed_arithmetic_words.c, 3117
mixed_math_word_m_star
mixed_arithmetic_words.c, 3118
mixed_math_word_mod
arithmetic_words.c, 2802
mixed_arithmetic_words.c, 3119
mixed_math_word_slash_mod
mixed_arithmetic_words.c, 3120
mixed_math_word_star_slash
mixed_arithmetic_words.c, 3120
mixed_math_word_star_slash_mod
mixed_arithmetic_words.c, 3121
mmio_read32
apic.c, 920
rpi5_mailbox.c, 868
mmio_write32
apic.c, 920
rpi5_mailbox.c, 869
Mode
_EFI_GRAPHICS_OUTPUT_PROTOCOL, 93
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 97
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE, 169
mode
VM, 332
MODE_COMPILE
vm.h, 2012
MODE_INTERPRET
vm.h, 2012
mode_str
vm_debug.c, 2730
mode_t
types.h, 608
ModificationTime
EFI_FILE_INFO, 166
modified_time
blk_meta_t, 110
modifier_key
usb_kbd.c, 1514
module_name
TestModule, 279
monotonic_ns
VMHostServices, 342
monotonic_time_ns
physics_host_snapshot, 238
Month
EFI_TIME, 176

most_likely_next_word_id
WordTransitionMetrics, 363
mounted_time
blk_volume_meta_t, 114
msc_slots
xhci_dev_t, 375
msg
log_slot_t, 223
msg_len
log_slot_t, 223
mul256
scalar25519.c, 1216
muldiv64
timer.c, 1032
mutex_destroy
VMHostServices, 342
mutex_init
VMHostServices, 343
mutex_lock
VMHostServices, 343
mutex_unlock
VMHostServices, 343

name
CapsuleNameEntry, 132
DictEntry, 149
physics_seed_definition, 244
ProfileTimer, 252
TestCase, 278
vm.h, 2076
VMRegistryEntry, 346
name_count
CapsuleDirHeader, 131
name_len
DictEntry, 149
vm.h, 2076
NAME_MAX_LEN
lifecycle_words_hosted.c, 3063
NAN
math.h, 575
Nanosecond
EFI_TIME, 176
nanosleep
time.h, 607
next
blk_dev_slot, 106
heap_block, 195
VirtqDesc, 313
vm_node, 338
vm_physics_node, 339
next_action
xhci_msc_slot_t, 386
next_action_config_value
xhci_msc_slot_t, 387
next_action_length
xhci_msc_slot_t, 387
NEXT_ARM64
vm_inner_interp_arm64.h, 2101
NEXT_ASM
vm_inner_interp_asm.h, 2104
next_block
blk_meta_t, 110
NEXT_OPT
arch_detect.h, 1710
NEXT_RV64
vm_inner_interp_riscv64.h, 2113
next_word_id
VM, 332
node_cells
fdt.c, 1301
node_cells_or_fallback
rpi5_native_boot.c, 887
node_enabled
fdt.c, 1302
node_name_eq
fdt.c, 1303
nonempty
blkio_factory.c, 2131
normalize_alignment
kmalloc.c, 1454
notify
VirtBlkState, 305
VirtInputState, 308
VirtRngState, 317
notify_off_mult
VirtBlkState, 305
VirtInputState, 308
VirtRngState, 317
ns_to_q48
physics_pipelining_metrics.c, 2364
num_configs
CDConfigTable, 137
CDTuning, 140
num_contours
ttf_glyph_header_t, 292
num_glyphs
ttf_font_t, 290
num_h_metrics
ttf_font_t, 290
num_queues
VirtioCommonCfg, 310
num_regimes
CDConfigTable, 137
CDTuning, 140
NumberOfPages
EFI_MEMORY_DESCRIPTOR, 173
NumberOfTableEntries
_EFI_SYSTEM_TABLE, 100

oem_id
__attribute__, 82
oem_revision
__attribute__, 82
oem_table_id
__attribute__, 82
off_dt_strings
fdt_header_t, 187
off_dt_struct

fdt_header_t, 187
off_mem_rsvmap
fdt_header_t, 187
off_t
types.h, 608
offset_high
idt_entry, 213
interrupts.c, 988
offset_low
idt_entry, 213
interrupts.c, 988
offset_mid
idt_entry, 213
interrupts.c, 988
on_curve
ttf_point_t, 295
op
xhci_dev_t, 375
opaque
blkio_params_t, 120
Open
_EFI_FILE_PROTOCOL, 91
open
blkio_vtable_t, 122
OpenProtocol
_EFI_BOOT_SERVICES, 88
OpenProtocolInformation
_EFI_BOOT_SERVICES, 89
OpenVolume
_EFI_SIMPLE_FILE_SYSTEM_PROTOCOL, 96
outb
i8042.c, 897
interrupts.c, 982
timer.c, 1033
OutputString
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 98
owner_fp
blk_meta_t, 110
owns_xt_entry
VMHostServices, 343

p
der_span_t, 146
p_align
Elf64_Phdr, 179
p_filesz
Elf64_Phdr, 179
p_flags
Elf64_Phdr, 179
p_memsz
Elf64_Phdr, 180
p_offset
Elf64_Phdr, 180
p_paddr
Elf64_Phdr, 180
p_type
Elf64_Phdr, 180
p_vaddr
Elf64_Phdr, 180
pack_pixel
framebuffer.c, 1322
Pad1
EFI_TIME, 176
Pad2
EFI_TIME, 176
padding
blk_meta_t, 110
paddr_to_virt
vmm.c, 1474
page_rva
reloc_block, 254
reloc_stub.c, 1081
page_size
xhci_op_regs_t, 388
panic
VMHostServices, 343
param
vt100.c, 1373
param_count
VT100State, 352
param_has_value
VT100State, 352
parameter
xhci_trb_t, 391
params
VT100State, 352
parent_block
blk_meta_t, 110
parent_vm_id
VMRegistryEntry, 346
ParentHandle
EFI_LOADED_IMAGE_PROTOCOL, 172
parity.c
fnv1a_64, 1635
fnv1a_64_u32, 1636
fnv1a_64_u8, 1637
hash_colon_body, 1637
MAX_DICT_ENTRIES, 1634
print_hex64, 1638
print_nl, 1639
print_str, 1640
print_u32, 1641
sk_dict_canonical_hash, 1642
sk_dict_word_count, 1643
sk_parity_collect, 1644
SK_PARITY_DEBUG, 1634
sk_parity_debug_check_ptr, 1634
sk_parity_debug_log_msg, 1635
sk_parity_print, 1645
parity.h
fnv1a_64, 770
FNV1A_64_OFFSET_BASIS, 769
FNV1A_64_PRIME, 769
SK_BOOTSTRAP_ARENA_FAIL, 770
SK_BOOTSTRAP_DICT_FAIL, 770
SK_BOOTSTRAP_INIT_FAIL, 770
SK_BOOTSTRAP_OK, 770

sk_dict_canonical_hash, 771
sk_dict_word_count, 772
sk_parity_collect, 773
sk_parity_print, 774
PARITY_EMIT
capsule_run.c, 1137
PARITY_HAVE_SINK
capsule_run.c, 1137
PARITY_MODE
vm_internal.h, 2735
parity_mode
VMHostServices, 343
parity_put_hex64
capsule_run.c, 1143
parity_put_u32
capsule_run.c, 1143
parity_put_u64
capsule_run.c, 1144
parity_put_uuid
capsule_run.c, 1144
parity_putc
capsule_run.c, 1145
parity_puts
capsule_run.c, 1145
ParityPacket, 225
bootstrap_result, 225
header_hash64, 225
here_offset, 225
latest_word_id, 226
tests_errors, 226
tests_failed, 226
tests_passed, 226
tests_skipped, 226
tests_total, 226
window_hash64, 226
word_count, 226
parse_avg_milli
physics_runtime.c, 2385
parse_log_level
cmdline.c, 1068
parse_madt
ioapic.c, 904
parse_mcfg
pci.c, 1490
parse_u32
cli.c, 2216
ParseState
vt100.c, 1359
part
sk_v4_drive_t, 269
part_close
gpt.c, 1245
part_flush
gpt.c, 1245
part_info
gpt.c, 1245
part_open
gpt.c, 1245
part_read
gpt.c, 1246
part_state
sk_v4_drive_t, 269
part_write
gpt.c, 1246
path
blkio_file_state_t, 118
PATH_DEPTH
fdt_uart.c, 1307
PATH_MAX
blkio_file.c, 2133
physics_runtime.c, 2381
path_to
fdt.c, 1303
pattern_diversity_check_count
RollingWindowOfTruth, 257
payload_bytes
physics_analytics_event_header, 228
pbam_clr
block_subsystem.c, 2200
pbam_set
block_subsystem.c, 2200
pbam_test
block_subsystem.c, 2201
pci
xhci_dev_t, 375
pci.c
cfg_read16, 1483
cfg_read32, 1483
cfg_read8, 1484
cfg_write16, 1485
cfg_write32, 1485
cfg_write8, 1486
ecam_cfg_ptr, 1486
ecam_read16, 1487
ecam_read32, 1487
ecam_read8, 1488
ecam_write16, 1489
ecam_write32, 1489
ecam_write8, 1490
g_bus_first, 1499
g_bus_last, 1499
g_ecam_base, 1499
parse_mcfg, 1490
pci_bar, 1491
pci_bar_size, 1492
pci_enable, 1492
pci_find_class, 1493
pci_find_first, 1493
pci_init, 1494
pci_map_bar, 1495
pci_read16, 1495
pci_read32, 1496
pci_read8, 1496
pci_write16, 1497
pci_write32, 1497
pci_write8, 1498

sig4_eq, 1498
sig8_eq, 1499
pci.h
pci_bar, 651
PCI_BAR_IO, 648
PCI_BAR_MEM_64, 648
pci_bar_size, 651
PCI_BAR_TYPE_MASK, 649
PCI_CFG_BAR0, 649
PCI_CFG_BAR1, 649
PCI_CFG_BAR2, 649
PCI_CFG_BAR3, 649
PCI_CFG_BAR4, 649
PCI_CFG_BAR5, 649
PCI_CFG_CAP_PTR, 649
PCI_CFG_CLASS_REV, 649
PCI_CFG_COMMAND, 649
PCI_CFG_DEVICE_ID, 650
PCI_CFG_HEADER_TYPE, 650
PCI_CFG_INT_LINE, 650
PCI_CFG_INT_PIN, 650
PCI_CFG_STATUS, 650
PCI_CFG_VENDOR_ID, 650
PCI_CMD_BUS_MASTER, 650
PCI_CMD_INTX_DISABLE, 650
PCI_CMD_IO_SPACE, 650
PCI_CMD_MEM_SPACE, 651
pci_enable, 652
pci_find_class, 653
pci_find_first, 653
pci_init, 654
pci_map_bar, 655
pci_read16, 655
pci_read32, 656
pci_read8, 656
pci_write16, 657
pci_write32, 657
pci_write8, 658
pci_bar
pci.c, 1491
pci.h, 651
PCI_BAR_IO
pci.h, 648
PCI_BAR_MEM_64
pci.h, 648
pci_bar_size
pci.c, 1492
pci.h, 651
PCI_BAR_TYPE_MASK
pci.h, 649
PCI_CFG_BAR0
pci.h, 649
PCI_CFG_BAR1
pci.h, 649
PCI_CFG_BAR2
pci.h, 649
PCI_CFG_BAR3
pci.h, 649
PCI_CFG_BAR4
pci.h, 649
PCI_CFG_BAR5
pci.h, 649
PCI_CFG_CAP_PTR
pci.h, 649
PCI_CFG_CLASS_REV
pci.h, 649
PCI_CFG_COMMAND
pci.h, 649
PCI_CFG_DEVICE_ID
pci.h, 650
PCI_CFG_HEADER_TYPE
pci.h, 650
PCI_CFG_INT_LINE
pci.h, 650
PCI_CFG_INT_PIN
pci.h, 650
PCI_CFG_STATUS
pci.h, 650
PCI_CFG_VENDOR_ID
pci.h, 650
PCI_CMD_BUS_MASTER
pci.h, 650
PCI_CMD_INTX_DISABLE
pci.h, 650
PCI_CMD_IO_SPACE
pci.h, 650
PCI_CMD_MEM_SPACE
pci.h, 651
pci_enable
pci.c, 1492
pci.h, 652
pci_find_class
pci.c, 1493
pci.h, 653
pci_find_first
pci.c, 1493
pci.h, 653
pci_init
pci.c, 1494
pci.h, 654
pci_map_bar
pci.c, 1495
pci.h, 655
pci_read16
pci.c, 1495
pci.h, 655
pci_read32
pci.c, 1496
pci.h, 656
pci_read8
pci.c, 1496
pci.h, 656
pci_write16
pci.c, 1497
pci.h, 657
pci_write32

pci.c, 1497
pci.h, 657
pci_write8
pci.c, 1498
pci.h, 658
PciDevice, 227
bus, 227
device, 227
device_id, 227
function, 227
vendor_id, 228
pd_index
vmm.c, 1474
pdpt_index
vmm.c, 1475
peak_bytes
kmalloc_stats_t, 221
sf_alloc_stats, 261
pending
sk_blk_attach_pending_t, 266
sk_usb_ctrl_t, 267
pending_connect_port_id
xhci_dev_t, 375
pending_connect_slot_id
xhci_dev_t, 376
pending_connect_speed
xhci_dev_t, 376
pending_disable_slot_id
xhci_dev_t, 376
pending_events
xhci_dev_t, 376
pending_mode
ssm_l8_state_t, 276
pending_reap
VMRegistryEntry, 347
percentage
WordProfile, 356
perturb_physics
test_contracts.c, 2681
test_contracts.h, 2542
PF_R
elf64.h, 521
PF_W
elf64.h, 522
PF_X
elf64.h, 522
phone
user_identity_seed_t, 302
phys_sector_size
blkio_info_t, 119
phys_size_bytes
blkio_info_t, 119
PhysicalSize
EFI_FILE_INFO, 166
PhysicalStart
EFI_MEMORY_DESCRIPTOR, 173
physics
DictEntry, 150
vm.h, 2076
vm_physics_node, 339
PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT
physics_runtime.h, 1876
PHYSICS_ANALYTICS_DEFAULT_HEAP_BYTES
physics_runtime.h, 1876
physics_analytics_event_header, 228
channel, 228
payload_bytes, 228
reserved, 229
timestamp_ns, 229
physics_analytics_event_header_t
physics_runtime.h, 1877
physics_analytics_header, 229
consume_seq, 230
dropped_events, 230
flags, 231
heap_bytes, 231
magic, 231
produce_seq, 231
read_offset, 231
ring_bytes, 231
ring_offset, 232
scratch_bytes, 232
scratch_offset, 232
summary_bytes, 232
summary_offset, 232
version_major, 232
version_minor, 233
write_offset, 233
physics_analytics_header_t
physics_runtime.h, 1877
physics_analytics_heap, 233
base, 234
bytes, 234
header, 234
ring, 234
ring_bytes, 235
scratch, 235
scratch_bytes, 235
summary, 235
summary_bytes, 235
physics_analytics_heap_info
physics_runtime.c, 2386
physics_runtime.h, 1878
physics_analytics_heap_t
physics_runtime.h, 1877
physics_analytics_publish_event
physics_runtime.c, 2386
physics_runtime.h, 1878
physics_benchmark_words.c
forth_BENCH_DICT_LOOKUP, 3124
forth_PHYSICS_BAYESIAN_REPORT, 3125
forth_PHYSICS_BUILD_INFO, 3126
forth_PHYSICS_CACHE_STATS, 3127
forth_PHYSICS_RESET_STATS, 3127
forth_PHYSICS_TOGGLE_CACHE, 3128
register_physics_benchmark_words, 3129

sf_fc_count, 3130
sf_fc_list, 3130
physics_benchmark_words.h
register_physics_benchmark_words, 2989
physics_decay_slope_load
physics_metadata.h, 1846
physics_diagnostic_words.c
forth_PHYSICS_BURN, 3131
forth_PHYSICS_CALC_KNOBS, 3132
forth_PHYSICS_SHOW_FEEDBACK, 3133
forth_PHYSICS_WORD_METRICS, 3134
register_physics_diagnostic_words, 3135
physics_diagnostic_words.h
forth_PHYSICS_BURN, 2991
forth_PHYSICS_CALC_KNOBS, 2992
forth_PHYSICS_SHOW_FEEDBACK, 2993
forth_PHYSICS_WORD_METRICS, 2994
register_physics_diagnostic_words, 2994
physics_execution_heat_increment
physics_metadata.h, 1846
physics_execution_heat_load
physics_metadata.h, 1847
physics_execution_hooks.c
physics_on_lookup, 2322
physics_post_execute, 2323
physics_pre_execute, 2325
physics_execution_hooks.h
physics_on_lookup, 1821
physics_post_execute, 1823
physics_pre_execute, 1825
physics_freeze_word_suites
physics_freeze_words_test.c, 2634
physics_freeze_words.c
forth_ALL_HEATS, 3136
forth_DECAY_RATE_FETCH, 3137
forth_FREEZE_CRITICAL, 3138
forth_FREEZE_WORD, 3139
forth_FROZEN_QUERY, 3139
forth_HEAT_FETCH, 3140
forth_HEAT_STORE, 3141
forth_SHOW_HEAT, 3142
forth_UNFREEZE_WORD, 3143
register_physics_freeze_words, 3143
physics_freeze_words.h
register_physics_freeze_words, 2996
physics_freeze_words_test.c
physics_freeze_word_suites, 2634
run_physics_freeze_words_tests, 2633
physics_global_stats
doe_metrics.c, 2269
physics_host_backend
physics_runtime.h, 1877
PHYSICS_HOST_BACKEND_FAKE
physics_runtime.h, 1878
PHYSICS_HOST_BACKEND_L4RE
physics_runtime.h, 1878
PHYSICS_HOST_BACKEND_POSIX
physics_runtime.h, 1878
physics_host_backend_t
physics_runtime.h, 1877
PHYSICS_HOST_FLAG_CGROUP
physics_runtime.h, 1876
PHYSICS_HOST_FLAG_CPU_STAT
physics_runtime.h, 1876
PHYSICS_HOST_FLAG_PSI
physics_runtime.h, 1877
physics_host_snapshot, 236
backend, 237
backend_seq, 237
cgroup_cpu_usage_us, 237
cgroup_memory_current_bytes, 237
cpu_count, 238
cpu_idle_jiffies, 238
cpu_total_jiffies, 238
flags, 238
load_avg_milli, 238
monotonic_time_ns, 238
physics_runtime.c, 2387
physics_runtime.h, 1879
psi_cpu_full_avg10_milli, 239
psi_cpu_full_avg300_milli, 239
psi_cpu_full_avg60_milli, 239
psi_cpu_some_avg10_milli, 239
psi_cpu_some_avg300_milli, 239
psi_cpu_some_avg60_milli, 239
psi_io_full_avg10_milli, 240
psi_io_full_avg300_milli, 240
psi_io_full_avg60_milli, 240
psi_io_some_avg10_milli, 240
psi_io_some_avg300_milli, 240
psi_io_some_avg60_milli, 240
psi_mem_some_avg10_milli, 241
psi_mem_some_avg300_milli, 241
psi_mem_some_avg60_milli, 241
realtime_ns, 241
runnable_threads, 241
scheduler_policy, 241
scheduler_priority, 242
scheduler_quantum_ns, 242
physics_host_snapshot_t
physics_runtime.h, 1877
physics_hotwords_cache.c
calculate_posterior, 2328
g_hotwords_cache, 2348
hotwords_bayesian_report, 2329
hotwords_bucket_reorder, 2330
hotwords_cache_cleanup, 2331
hotwords_cache_evict_entry, 2332
hotwords_cache_evict_name, 2332
hotwords_cache_init, 2334
hotwords_cache_lookup, 2335
hotwords_cache_promote, 2338
hotwords_cache_remove_at, 2340
hotwords_cache_set_enabled, 2341
hotwords_posterior_bucket_searches, 2342
hotwords_posterior_cache_hits, 2343

hotwords_speedup_estimate, 2344
hotwords_stats_print, 2345
hotwords_stats_reset, 2346
q48_to_ns_double, 2347
physics_hotwords_cache.h
hotwords_bayesian_report, 1829
hotwords_bucket_reorder, 1830
hotwords_cache_cleanup, 1831
hotwords_cache_evict_entry, 1831
hotwords_cache_evict_name, 1832
hotwords_cache_init, 1833
hotwords_cache_lookup, 1835
hotwords_cache_promote, 1837
hotwords_cache_set_enabled, 1839
HOTWORDS_CACHE_SIZE, 1828
HOTWORDS_EXECUTION_HEAT_DELTA_THRESHOLD, 1828
HOTWORDS_EXECUTION_HEAT_THRESHOLD, 1828
hotwords_posterior_bucket_searches, 1840
hotwords_posterior_cache_hits, 1841
hotwords_speedup_estimate, 1842
hotwords_stats_print, 1843
hotwords_stats_reset, 1844
HotwordsCache, 1829
physics_metadata.c
clamp_u32, 2349
derive_state_flags, 2350
physics_metadata_apply_linear_decay, 2351
physics_metadata_apply_seed, 2352
physics_metadata_init, 2354
physics_metadata_record_latency, 2356
physics_metadata_refresh_state, 2358
physics_metadata_set_mass, 2359
physics_metadata_touch, 2360
physics_seed_definition_t, 2349
physics_seed_table, 2362
temperature_from_execution_heat, 2361
physics_metadata.h
physics_decay_slope_load, 1846
physics_execution_heat_increment, 1846
physics_execution_heat_load, 1847
physics_metadata_apply_linear_decay, 1847
physics_metadata_apply_seed, 1849
physics_metadata_init, 1850
physics_metadata_record_latency, 1852
physics_metadata_refresh_state, 1854
physics_metadata_set_mass, 1855
physics_metadata_touch, 1856
physics_metadata_apply_linear_decay
physics_metadata.c, 2351
physics_metadata.h, 1847
physics_metadata_apply_seed
physics_metadata.c, 2352
physics_metadata.h, 1849
physics_metadata_init
physics_metadata.c, 2354
physics_metadata.h, 1850
physics_metadata_record_latency
physics_metadata.c, 2356
physics_metadata.h, 1852
physics_metadata_refresh_state
physics_metadata.c, 2358
physics_metadata.h, 1854
physics_metadata_set_mass
physics_metadata.c, 2359
physics_metadata.h, 1855
physics_metadata_touch
physics_metadata.c, 2360
physics_metadata.h, 1856
physics_on_lookup
physics_execution_hooks.c, 2322
physics_execution_hooks.h, 1821
physics_pipelining_diagnostic_words.c
forth_PIPELINING_ANALYZE_WORD, 3146
forth_PIPELINING_ENABLE, 3146
forth_PIPELINING_RESET_ALL, 3147
forth_PIPELINING_SHOW_STATS, 3147
forth_PIPELINING_SHOW_TOP_TRANSITIONS, 3148
forth_PIPELINING_STATS, 3149
register_physics_pipelining_diagnostic_words, 3149
physics_pipelining_diagnostic_words.h
register_physics_pipelining_diagnostic_words, 2998
physics_pipelining_metrics.c
ns_to_q48, 2364
q48_div_u64, 2364
q48_to_ns, 2365
transition_metrics_binary_chop_suggest_window, 2365
transition_metrics_cleanup, 2366
transition_metrics_context_accuracy_string, 2367
transition_metrics_get_probability_q48, 2368
transition_metrics_init, 2369
transition_metrics_record, 2371
transition_metrics_record_context, 2372
transition_metrics_record_prefetch_hit, 2373
transition_metrics_record_prefetch_miss, 2374
transition_metrics_reset, 2375
transition_metrics_should_speculate, 2376
transition_metrics_stats_string, 2377
transition_metrics_update_cache, 2378
transition_metrics_update_context_window, 2379
physics_pipelining_metrics.h
transition_metrics_binary_chop_suggest_window, 1859
transition_metrics_cleanup, 1860
transition_metrics_context_accuracy_string, 1861
transition_metrics_get_probability_q48, 1862
transition_metrics_init, 1863
transition_metrics_record, 1865
transition_metrics_record_context, 1866
transition_metrics_record_prefetch_hit, 1867
transition_metrics_record_prefetch_miss, 1868

transition_metrics_reset, 1869
transition_metrics_should_speculate, 1870
transition_metrics_stats_string, 1871
transition_metrics_update_cache, 1872
transition_metrics_update_context_window, 1873
WordTransitionMetrics, 1859
physics_post_execute
physics_execution_hooks.c, 2323
physics_execution_hooks.h, 1823
physics_pre_execute
physics_execution_hooks.c, 2325
physics_execution_hooks.h, 1825
physics_runtime.c
align_up, 2382
analytics_heap_allocate, 2382
analytics_heap_release, 2383
analytics_ring_free, 2384
g_runtime, 2397
parse_avg_milli, 2385
PATH_MAX, 2381
physics_analytics_heap_info, 2386
physics_analytics_publish_event, 2386
physics_host_snapshot, 2387
physics_runtime_init, 2388
physics_runtime_shutdown, 2389
physics_runtime_state_t, 2382
read_cgroup_cpu_usage_us, 2390
read_cgroup_memory_current, 2391
read_proc_stat_totals, 2392
read_psi_file, 2393
resolve_cgroup_path, 2394
snapshot_posix, 2395
physics_runtime.h
HOLA_PROTOCOL_MAJOR, 1876
HOLA_PROTOCOL_MINOR, 1876
HOLA_SHARED_MAGIC, 1876
PHYSICS_ANALYTICS_CHANNEL_HOST_SNAPSHOT, 1876
PHYSICS_ANALYTICS_DEFAULT_HEAP_BYTES, 1876
physics_analytics_event_header_t, 1877
physics_analytics_header_t, 1877
physics_analytics_heap_info, 1878
physics_analytics_heap_t, 1877
physics_analytics_publish_event, 1878
physics_host_backend, 1877
PHYSICS_HOST_BACKEND_FAKE, 1878
PHYSICS_HOST_BACKEND_L4RE, 1878
PHYSICS_HOST_BACKEND_POSIX, 1878
physics_host_backend_t, 1877
PHYSICS_HOST_FLAG_CGROUP, 1876
PHYSICS_HOST_FLAG_CPU_STAT, 1876
PHYSICS_HOST_FLAG_PSI, 1877
physics_host_snapshot, 1879
physics_host_snapshot_t, 1877
physics_runtime_init, 1880
physics_runtime_shutdown, 1881
physics_runtime_init
physics_runtime.c, 2388
physics_runtime.h, 1880
physics_runtime_shutdown
physics_runtime.c, 2389
physics_runtime.h, 1881
physics_runtime_state, 242
backend_seq, 243
heap, 243
heap_ready, 243
physics_runtime_state_t
physics_runtime.c, 2382
physics_seed_definition, 243
acl_hint, 243
avg_latency_ns, 243
name, 244
pubsub_mask, 244
temperature_q8, 244
physics_seed_definition_t
physics_metadata.c, 2349
physics_seed_table
physics_metadata.c, 2362
PHYSICS_STATE_COMPILED
vm.h, 2008
PHYSICS_STATE_HIDDEN
vm.h, 2008
PHYSICS_STATE_IMMEDIATE
vm.h, 2008
PHYSICS_STATE_PINNED
vm.h, 2008
PhysicsSnapshot, 244
decay_slope_q48, 244
eff_window_size, 244
heat_25th, 245
heat_50th, 245
heat_75th, 245
prefetch_attempts, 245
tick_target_ns, 245
total_executions, 245
pic_disable
interrupts.c, 984
pid_t
types.h, 608
pipeline_metrics
VM, 332
PipelineGlobalMetrics, 246
last_checked_accuracy, 246
last_checked_window_size, 246
prefetch_attempts, 246
prefetch_hits, 246
suggested_next_size, 246
window_tuning_checks, 246
pit_available
timer.c, 1050
PIT_CHANNEL0_DATA
timer.c, 1016
PIT_COMMAND
timer.c, 1016
PIT_FREQ_HZ

timer.c, 1016
pit_hz_mean
timer_calibration_record, 284
pit_probe_running
timer.c, 1033
pit_program_reload
timer.c, 1034
pit_read_counter16
timer.c, 1035
pit_tsc_hz_mean
timer.c, 1051
pixel_format
FramebufferInfo, 190
Rpi5FramebufferInfo, 260
PIXEL_ORDER_RGB
rpi5_mailbox.c, 862
PixelBitMask
uefi.h, 731
PixelBltOnly
uefi.h, 731
PixelBlueGreenRedReserved8BitPerColor
uefi.h, 731
PixelFormat
EFI_GRAPHICS_OUTPUT_MODE_INFORMATION, 167
PixelFormatMax
uefi.h, 731
PixelInformation
EFI_GRAPHICS_OUTPUT_MODE_INFORMATION, 167
PixelRedGreenBlueReserved8BitPerColor
uefi.h, 731
pixels
ttf_bitmap_t, 287
ttf_raster_cache_slot_t, 296
pixels_per_scanline
FramebufferInfo, 190
Rpi5FramebufferInfo, 260
PixelsPerScanLine
EFI_GRAPHICS_OUTPUT_MODE_INFORMATION, 167
PL011_FR
console.c, 1252
PL011_FR_RXFE
console.c, 1252
PL011_FR_TXFF
console.c, 1252
platform_alloc.h
sf_alloc_get_stats, 1883
sf_alloc_init, 1884
sf_alloc_stats_t, 1883
sf_calloc, 1885
sf_free, 1886
sf_malloc, 1887
sf_realloc, 1888
platform_init.c
sf_time_backend, 2419
sf_time_init, 2419
platform_io.h
sf_terminal_ready, 1891
platform_lock.h
sf_mutex_destroy, 1893
sf_mutex_init, 1893
sf_mutex_lock, 1894
sf_mutex_t, 1892
sf_mutex_unlock, 1895
platform_time.h
sf_format_timestamp, 1898
sf_has_rtc, 1899
sf_monotonic_ns, 1900
sf_ns_to_ms, 1901
sf_ns_to_seconds, 1901
sf_ns_to_us, 1901
sf_realtime_ns, 1901
sf_seconds_to_ns, 1902
sf_set_realtime_ns, 1902
sf_time_backend, 1903
sf_time_backend_l4re, 1903
sf_time_backend_posix, 1903
sf_time_init, 1902
sf_time_ns_t, 1898
SF_TIME_STAMP_SIZE, 1898
plic.c
be32_cell, 998
be64_addr_cell_pair, 998
plic_base_from_dtb, 999
PLIC_BASE_QEMU_DEFAULT, 997
PLIC_CLAIM, 997
plic_claim, 1000
plic_complete, 1001
PLIC_CONTEXT_S, 997
plic_disable, 1001
PLIC_ENABLE, 998
plic_enable, 1002
plic_init, 1002
PLIC_PRIORITY, 998
plic_set_priority, 1004
PLIC_THRESHOLD, 998
reg32, 1005
s_plic_base, 1005
plic.h
plic_claim, 660
plic_complete, 660
plic_disable, 661
plic_enable, 661
plic_init, 662
plic_set_priority, 663
plic_base_from_dtb
plic.c, 999
PLIC_BASE_QEMU_DEFAULT
plic.c, 997
PLIC_CLAIM
plic.c, 997
plic_claim
plic.c, 1000
plic.h, 660

plic_complete
plic.c, 1001
plic.h, 660
PLIC_CONTEXT_S
plic.c, 997
plic_disable
plic.c, 1001
plic.h, 661
PLIC_ENABLE
plic.c, 998
plic_enable
plic.c, 1002
plic.h, 661
plic_init
plic.c, 1002
plic.h, 662
PLIC_PRIORITY
plic.c, 998
plic_set_priority
plic.c, 1004
plic.h, 663
PLIC_THRESHOLD
plic.c, 998
pml4_index
vmm.c, 1475
pmm.c
__kernel_end, 1467
__kernel_start, 1467
bitmap_clear, 1459
bitmap_mark_range, 1459
bitmap_set, 1459
bitmap_test, 1460
find_contiguous_free, 1460
is_ram_type, 1461
is_usable_memory, 1461
pmm_alloc_contiguous, 1462
pmm_alloc_floor_page, 1467
pmm_alloc_page, 1462
pmm_bitmap, 1467
PMM_BITMAP_BITS, 1458
pmm_bytes_to_pages, 1463
pmm_free_contiguous, 1463
pmm_free_page, 1464
pmm_free_pages, 1468
pmm_get_stats, 1464
pmm_init, 1465
pmm_initialized, 1468
pmm_is_initialized, 1466
PMM_MAX_BITMAP_BYTES, 1458
pmm_memset, 1466
pmm_total_pages, 1468
pmm_tracked_pages, 1468
recompute_free_pages, 1466
pmm.h
pmm_alloc_contiguous, 665
pmm_alloc_page, 665
pmm_free_contiguous, 666
pmm_free_page, 667
pmm_get_stats, 667
pmm_init, 667
pmm_is_initialized, 668
PMM_PAGE_SIZE, 665
pmm_alloc_contiguous
pmm.c, 1462
pmm.h, 665
pmm_alloc_floor_page
pmm.c, 1467
pmm_alloc_page
pmm.c, 1462
pmm.h, 665
pmm_bitmap
pmm.c, 1467
PMM_BITMAP_BITS
pmm.c, 1458
pmm_bytes_to_pages
pmm.c, 1463
pmm_free_contiguous
pmm.c, 1463
pmm.h, 666
pmm_free_page
pmm.c, 1464
pmm.h, 667
pmm_free_pages
pmm.c, 1468
pmm_get_stats
pmm.c, 1464
pmm.h, 667
pmm_init
pmm.c, 1465
pmm.h, 667
pmm_initialized
pmm.c, 1468
pmm_is_initialized
pmm.c, 1466
pmm.h, 668
PMM_MAX_BITMAP_BYTES
pmm.c, 1458
pmm_memset
pmm.c, 1466
PMM_PAGE_SIZE
pmm.h, 665
pmm_stats_t, 247
free_bytes, 247
free_pages, 247
total_bytes, 247
total_pages, 247
used_bytes, 248
used_pages, 248
pmm_total_pages
pmm.c, 1468
pmm_tracked_pages
pmm.c, 1468
PMR_ALLOW_ALL
apic.c, 910
pmtimer_delta
timer.c, 1036

PMTIMER_FREQ_HZ
timer.c, 1016
PMTIMER_IO_PORT_DEFAULT
timer.c, 1016
PMTIMER_MASK
timer.c, 1016
pmtimer_port
timer.c, 1051
pmtimer_read
timer.c, 1037
PMTIMER_WIDTH_BITS
timer.c, 1016
PN_BUF
format_words.c, 2911
PN_CAP
format_words.c, 2911
point_add
ed25519.c, 1201
point_compress
ed25519.c, 1202
point_count
ttf_outline_t, 294
point_decompress
ed25519.c, 1202
point_identity
ed25519.c, 1203
pointer_within_heap
kmalloc.c, 1455
points
ttf_outline_t, 294
pop_xt
acl_words.c, 2790
popcount16
capsule_loader.c, 1126
port_id
xhci_dev_t, 376
port_slot_id
xhci_dev_t, 376
porthlpmc
xhci_port_regs_t, 389
portli
xhci_port_regs_t, 389
portpmsc
xhci_port_regs_t, 389
portsc
xhci_dev_t, 376
xhci_port_regs_t, 389
pos
time_window, 283
posix_format_timestamp
time.c, 2416
posix_get_monotonic_ns
time.c, 2416
posix_get_realtime_ns
time.c, 2416
posix_has_rtc
time.c, 2417
posix_set_realtime_ns
time.c, 2417
post_dict_hash
CapsuleRunRecord, 133
pow
math.h, 576
POW_2252M2_EXP
ed25519.c, 1205
pre_dict_hash
CapsuleRunRecord, 133
predicted_label_hits
HeartbeatTickSnapshot, 203
prefetch_accuracy_percent
DoeMetrics, 158
prefetch_attempts
DoeMetrics, 158
InferenceInputs, 216
PhysicsSnapshot, 245
PipelineGlobalMetrics, 246
WordTransitionMetrics, 363
prefetch_hits
DoeMetrics, 158
InferenceInputs, 216
PipelineGlobalMetrics, 246
WordTransitionMetrics, 363
prefetch_latency_saved_q48
WordTransitionMetrics, 364
prefetch_misses
WordTransitionMetrics, 364
present
sk_v4_marker_t, 270
vmm_page_info_t, 344
pressed
usb_kbd.c, 1514
prev
heap_block, 195
prev_block
blk_meta_t, 110
prev_locality_q16
CDConfigEntry, 135
prev_window
CDConfigEntry, 135
PRId16
inttypes.h, 572
PRId32
inttypes.h, 572
PRId64
inttypes.h, 572
PRId8
inttypes.h, 572
PRIi16
inttypes.h, 572
PRIi32
inttypes.h, 572
PRIi64
inttypes.h, 572
PRIi8
inttypes.h, 572
PRIM_2DROP

vm_inner_interp_asm.h, 2104
PRIM_2DROP_ARM64
Fast Primitive Word Implementations, 38
PRIM_2DROP_RV64
vm_inner_interp_riscv64.h, 2113
PRIM_2DUP
vm_inner_interp_asm.h, 2104
PRIM_2DUP_ARM64
Fast Primitive Word Implementations, 38
PRIM_2DUP_RV64
vm_inner_interp_riscv64.h, 2113
PRIM_AND
vm_inner_interp_asm.h, 2105
PRIM_AND_ARM64
Fast Primitive Word Implementations, 39
PRIM_AND_RV64
vm_inner_interp_riscv64.h, 2113
PRIM_BRANCH
vm_inner_interp_asm.h, 2105
PRIM_BRANCH_ARM64
Fast Primitive Word Implementations, 39
PRIM_BRANCH_OPT
arch_detect.h, 1710
PRIM_BRANCH_RV64
vm_inner_interp_riscv64.h, 2114
PRIM_C_FETCH
vm_inner_interp_asm.h, 2105
PRIM_C_FETCH_ARM64
Fast Primitive Word Implementations, 39
PRIM_C_FETCH_RV64
vm_inner_interp_riscv64.h, 2114
PRIM_C_STORE
vm_inner_interp_asm.h, 2105
PRIM_C_STORE_ARM64
Fast Primitive Word Implementations, 39
PRIM_C_STORE_RV64
vm_inner_interp_riscv64.h, 2114
PRIM_DROP
vm_inner_interp_asm.h, 2105
PRIM_DROP_ARM64
Fast Primitive Word Implementations, 39
PRIM_DROP_OPT
arch_detect.h, 1710
PRIM_DROP_RV64
vm_inner_interp_riscv64.h, 2114
PRIM_DUP
vm_inner_interp_asm.h, 2106
PRIM_DUP_ARM64
Fast Primitive Word Implementations, 40
PRIM_DUP_OPT
arch_detect.h, 1710
PRIM_DUP_RV64
vm_inner_interp_riscv64.h, 2114
PRIM_EQUALS
vm_inner_interp_asm.h, 2106
PRIM_EQUALS_ARM64
Fast Primitive Word Implementations, 40
PRIM_EQUALS_RV64
vm_inner_interp_riscv64.h, 2115
PRIM_EXECUTE
vm_inner_interp_asm.h, 2106
PRIM_EXECUTE_ARM64
Fast Primitive Word Implementations, 40
PRIM_EXECUTE_RV64
vm_inner_interp_riscv64.h, 2115
PRIM_FETCH
vm_inner_interp_asm.h, 2106
PRIM_FETCH_ARM64
Fast Primitive Word Implementations, 40
PRIM_FETCH_OPT
arch_detect.h, 1710
PRIM_FETCH_RV64
vm_inner_interp_riscv64.h, 2115
PRIM_GREATER
vm_inner_interp_asm.h, 2106
PRIM_GREATER_ARM64
Fast Primitive Word Implementations, 40
PRIM_GREATER_RV64
vm_inner_interp_riscv64.h, 2115
PRIM_INVERT
vm_inner_interp_asm.h, 2107
PRIM_INVERT_ARM64
Fast Primitive Word Implementations, 41
PRIM_INVERT_RV64
vm_inner_interp_riscv64.h, 2115
PRIM_LESS
vm_inner_interp_asm.h, 2107
PRIM_LESS_ARM64
Fast Primitive Word Implementations, 41
PRIM_LESS_RV64
vm_inner_interp_riscv64.h, 2116
PRIM_MINUS
vm_inner_interp_asm.h, 2107
PRIM_MINUS_ARM64
Fast Primitive Word Implementations, 41
PRIM_MINUS_OPT
arch_detect.h, 1710
PRIM_MINUS_RV64
vm_inner_interp_riscv64.h, 2116
PRIM_MOD
vm_inner_interp_asm.h, 2107
PRIM_MOD_ARM64
Fast Primitive Word Implementations, 41
PRIM_MOD_RV64
vm_inner_interp_riscv64.h, 2116
PRIM_NEGATE
vm_inner_interp_asm.h, 2107
PRIM_NEGATE_ARM64
Fast Primitive Word Implementations, 41
PRIM_NEGATE_RV64
vm_inner_interp_riscv64.h, 2116
PRIM_OR
vm_inner_interp_asm.h, 2108
PRIM_OR_ARM64
Fast Primitive Word Implementations, 42
PRIM_OR_RV64

vm_inner_interp_riscv64.h, 2116
PRIM_OVER
vm_inner_interp_asm.h, 2108
PRIM_OVER_ARM64
Fast Primitive Word Implementations, 42
PRIM_OVER_RV64
vm_inner_interp_riscv64.h, 2117
PRIM_PLUS
vm_inner_interp_asm.h, 2108
PRIM_PLUS_ARM64
Fast Primitive Word Implementations, 42
PRIM_PLUS_OPT
arch_detect.h, 1710
PRIM_PLUS_RV64
vm_inner_interp_riscv64.h, 2117
PRIM_R_FETCH
vm_inner_interp_asm.h, 2108
PRIM_R_FETCH_ARM64
Fast Primitive Word Implementations, 42
PRIM_R_FETCH_OPT
arch_detect.h, 1710
PRIM_R_FETCH_RV64
vm_inner_interp_riscv64.h, 2117
PRIM_R_FROM
vm_inner_interp_asm.h, 2108
PRIM_R_FROM_ARM64
Fast Primitive Word Implementations, 42
PRIM_R_FROM_OPT
arch_detect.h, 1711
PRIM_R_FROM_RV64
vm_inner_interp_riscv64.h, 2117
PRIM_ROT
vm_inner_interp_asm.h, 2109
PRIM_ROT_ARM64
Fast Primitive Word Implementations, 43
PRIM_ROT_RV64
vm_inner_interp_riscv64.h, 2117
PRIM_SLASH
vm_inner_interp_asm.h, 2109
PRIM_SLASH_ARM64
Fast Primitive Word Implementations, 43
PRIM_SLASH_RV64
vm_inner_interp_riscv64.h, 2118
PRIM_STAR
vm_inner_interp_asm.h, 2109
PRIM_STAR_ARM64
Fast Primitive Word Implementations, 43
PRIM_STAR_OPT
arch_detect.h, 1711
PRIM_STAR_RV64
vm_inner_interp_riscv64.h, 2118
PRIM_STORE
vm_inner_interp_asm.h, 2109
PRIM_STORE_ARM64
Fast Primitive Word Implementations, 43
PRIM_STORE_OPT
arch_detect.h, 1711
PRIM_STORE_RV64
vm_inner_interp_riscv64.h, 2118
PRIM_SWAP
vm_inner_interp_asm.h, 2109
PRIM_SWAP_ARM64
Fast Primitive Word Implementations, 43
PRIM_SWAP_OPT
arch_detect.h, 1711
PRIM_SWAP_RV64
vm_inner_interp_riscv64.h, 2118
PRIM_TO_R
vm_inner_interp_asm.h, 2110
PRIM_TO_R_ARM64
Fast Primitive Word Implementations, 44
PRIM_TO_R_OPT
arch_detect.h, 1711
PRIM_TO_R_RV64
vm_inner_interp_riscv64.h, 2118
PRIM_XOR
vm_inner_interp_asm.h, 2110
PRIM_XOR_ARM64
Fast Primitive Word Implementations, 44
PRIM_XOR_RV64
vm_inner_interp_riscv64.h, 2119
PRIM_ZBRANCH
vm_inner_interp_asm.h, 2110
PRIM_ZBRANCH_ARM64
Fast Primitive Word Implementations, 44
PRIM_ZBRANCH_OPT
arch_detect.h, 1711
PRIM_ZBRANCH_RV64
vm_inner_interp_riscv64.h, 2119
PRIM_ZERO_EQUALS
vm_inner_interp_asm.h, 2110
PRIM_ZERO_EQUALS_ARM64
Fast Primitive Word Implementations, 44
PRIM_ZERO_EQUALS_RV64
vm_inner_interp_riscv64.h, 2119
PRIM_ZERO_LESS
vm_inner_interp_asm.h, 2110
PRIM_ZERO_LESS_ARM64
Fast Primitive Word Implementations, 44
PRIM_ZERO_LESS_RV64
vm_inner_interp_riscv64.h, 2119
print_banner
kernel_main.c, 1421
print_boot_info
kernel_main.c, 1422
print_dec
timer.c, 1037, 1055
print_dec_u64
interrupts.c, 984
print_heap_stats
kernel_main.c, 1423
print_hex
interrupts.c, 975, 991
print_hex64
arch.c, 965
interrupts.c, 985

parity.c, 1638
print_line_64
editor_words.c, 2907
print_module_summary
test_common.c, 2657
test_common.h, 2516
print_nl
parity.c, 1639
print_number_formatted
format_words.c, 2924
print_pmm_stats
kernel_main.c, 1424
print_str
parity.c, 1640
print_u32
parity.c, 1641
print_uint
kernel_main.c, 1425
print_usage
main.c, 2307
print_word_name
system_words.c, 3234
print_word_name_safe
profiler.c, 2430
printf
stdio.h, 587
PRIu16
inttypes.h, 573
PRIu32
inttypes.h, 573
PRIu64
inttypes.h, 573
PRIu8
inttypes.h, 573
PRIX16
inttypes.h, 573
PRIx16
inttypes.h, 573
PRIX32
inttypes.h, 573
PRIx32
inttypes.h, 573
PRIX64
inttypes.h, 573
PRIx64
inttypes.h, 573
PRIX8
inttypes.h, 574
PRIx8
inttypes.h, 574
prng_next
starforth_words.c, 3197
probability_gt_10pct_q48
SpeedupEstimate, 271
probability_gt_double_q48
SpeedupEstimate, 272
produce_seq
physics_analytics_header, 231
PROF_CATEGORY_COUNT
profiler.h, 1907
PROF_CONTROL_FLOW
profiler.h, 1907
PROF_DICT_ADD
profiler.h, 1907
PROF_DICT_LOOKUP
profiler.h, 1907
PROF_IO_OPS
profiler.h, 1907
PROF_MEMORY_ACCESS
profiler.h, 1907
PROF_STACK_OPS
profiler.h, 1907
PROF_VM_COMPILE
profiler.h, 1907
PROF_VM_EXECUTE
profiler.h, 1907
PROF_VM_INIT
profiler.h, 1907
PROF_WORD_EXEC
profiler.h, 1907
PROFILE_BASIC
profiler.h, 1908
PROFILE_DETAILED
profiler.h, 1908
PROFILE_DISABLED
profiler.h, 1908
PROFILE_ENABLED
profiler.h, 1906
PROFILE_END
profiler.h, 1906
PROFILE_FULL
profiler.h, 1908
PROFILE_INC_CYCLES
profiler.h, 1906
PROFILE_INC_DICT_LOOKUP
profiler.h, 1906
PROFILE_INC_STACK_OP
profiler.h, 1906
PROFILE_MEMORY_READ
profiler.h, 1906
PROFILE_MEMORY_WRITE
profiler.h, 1906
PROFILE_START
profiler.h, 1906
PROFILE_VERBOSE
profiler.h, 1908
PROFILE_WORD_ENTER
profiler.h, 1906
PROFILE_WORD_EXIT
profiler.h, 1907
ProfileCategory
profiler.h, 1907
ProfileLevel
profiler.h, 1907
Profiler, 248
branch_count, 249

category_timers, 249
compile_operations, 249
counters, 249
dict_lookups, 249
instruction_count, 249
level, 249
loop_iterations, 250
memory_allocations, 250
memory_stats, 250
profiling_overhead, 250
stack_operations, 250
total_execution_time, 250
vm_cycles, 250
word_profile_capacity, 250
word_profile_count, 250
word_profiles, 250
profiler.c
_POSIX_C_SOURCE, 2427
compare_by_calls, 2427
compare_by_time, 2428
current_start_time, 2440
current_word, 2440
DictEntry, 2427
enabled, 2441
g_profiler, 2441
get_time_ns, 2428
get_word_stats, 2429
level, 2441
print_word_name_safe, 2430
profiler_analyze_performance, 2431
profiler_end_timer, 2431
profiler_generate_report, 2431
profiler_get_time_ns, 2432
profiler_inc_branch, 2432
profiler_inc_dict_lookup, 2433
profiler_inc_loop_iter, 2433
profiler_inc_stack_op, 2433
profiler_inc_vm_cycles, 2433
profiler_init, 2433
profiler_memory_alloc, 2434
profiler_memory_read, 2434
profiler_memory_write, 2434
profiler_print_hotspots, 2434
profiler_print_memory_analysis, 2435
profiler_print_summary, 2435
profiler_print_word_analysis, 2436
profiler_reset, 2436
profiler_set_level, 2436
profiler_shutdown, 2437
profiler_start_timer, 2437
profiler_state, 2441
profiler_suggest_optimizations, 2437
profiler_word_count, 2437
profiler_word_enter, 2438
profiler_word_exit, 2439
total_dict_lookups, 2441
total_memory_reads, 2441
total_memory_writes, 2441
total_stack_ops, 2441
word_capacity, 2441
word_count, 2442
word_stats, 2442
profiler.h
DictEntry, 1907
g_profiler, 1917
PROF_CATEGORY_COUNT, 1907
PROF_CONTROL_FLOW, 1907
PROF_DICT_ADD, 1907
PROF_DICT_LOOKUP, 1907
PROF_IO_OPS, 1907
PROF_MEMORY_ACCESS, 1907
PROF_STACK_OPS, 1907
PROF_VM_COMPILE, 1907
PROF_VM_EXECUTE, 1907
PROF_VM_INIT, 1907
PROF_WORD_EXEC, 1907
PROFILE_BASIC, 1908
PROFILE_DETAILED, 1908
PROFILE_DISABLED, 1908
PROFILE_ENABLED, 1906
PROFILE_END, 1906
PROFILE_FULL, 1908
PROFILE_INC_CYCLES, 1906
PROFILE_INC_DICT_LOOKUP, 1906
PROFILE_INC_STACK_OP, 1906
PROFILE_MEMORY_READ, 1906
PROFILE_MEMORY_WRITE, 1906
PROFILE_START, 1906
PROFILE_VERBOSE, 1908
PROFILE_WORD_ENTER, 1906
PROFILE_WORD_EXIT, 1907
ProfileCategory, 1907
ProfileLevel, 1907
profiler_analyze_performance, 1908
profiler_end_timer, 1908
profiler_generate_report, 1908
profiler_get_time_ns, 1909
profiler_inc_branch, 1909
profiler_inc_dict_lookup, 1910
profiler_inc_loop_iter, 1910
profiler_inc_stack_op, 1910
profiler_inc_vm_cycles, 1910
profiler_init, 1910
profiler_memory_alloc, 1911
profiler_memory_read, 1911
profiler_memory_write, 1911
profiler_print_hotspots, 1911
profiler_print_memory_analysis, 1912
profiler_print_summary, 1912
profiler_print_word_analysis, 1913
profiler_reset, 1913
profiler_set_level, 1913
profiler_shutdown, 1914
profiler_start_timer, 1914
profiler_suggest_optimizations, 1914
profiler_word_count, 1914

profiler_word_enter, 1915
profiler_word_exit, 1916
VM, 1907
profiler_analyze_performance
profiler.c, 2431
profiler.h, 1908
profiler_end_timer
profiler.c, 2431
profiler.h, 1908
profiler_generate_report
profiler.c, 2431
profiler.h, 1908
profiler_get_time_ns
profiler.c, 2432
profiler.h, 1909
profiler_inc_branch
profiler.c, 2432
profiler.h, 1909
profiler_inc_dict_lookup
profiler.c, 2433
profiler.h, 1910
profiler_inc_loop_iter
profiler.c, 2433
profiler.h, 1910
profiler_inc_stack_op
profiler.c, 2433
profiler.h, 1910
profiler_inc_vm_cycles
profiler.c, 2433
profiler.h, 1910
profiler_init
profiler.c, 2433
profiler.h, 1910
profiler_memory_alloc
profiler.c, 2434
profiler.h, 1911
profiler_memory_read
profiler.c, 2434
profiler.h, 1911
profiler_memory_write
profiler.c, 2434
profiler.h, 1911
profiler_print_hotspots
profiler.c, 2434
profiler.h, 1911
profiler_print_memory_analysis
profiler.c, 2435
profiler.h, 1912
profiler_print_summary
profiler.c, 2435
profiler.h, 1912
profiler_print_word_analysis
profiler.c, 2436
profiler.h, 1913
profiler_reset
profiler.c, 2436
profiler.h, 1913
profiler_set_level
profiler.c, 2436
profiler.h, 1913
profiler_shutdown
profiler.c, 2437
profiler.h, 1914
profiler_start_timer
profiler.c, 2437
profiler.h, 1914
profiler_state
profiler.c, 2441
profiler_suggest_optimizations
profiler.c, 2437
profiler.h, 1914
profiler_word_count
profiler.c, 2437
profiler.h, 1914
profiler_word_enter
profiler.c, 2438
profiler.h, 1915
profiler_word_exit
profiler.c, 2439
profiler.h, 1916
ProfileScope, 251
category, 251
word_name, 251
ProfileTimer, 251
call_count, 252
max_time, 252
min_time, 252
name, 252
start_time, 252
total_time, 253
profiling_overhead
Profiler, 250
promotions
HotwordsStats, 212
ProtocolsPerHandle
_EFI_BOOT_SERVICES, 89
proxy
sk_v4_marker_t, 270
proxy_pubkey
zuse_genesis_marker_v2_t, 396
psi_cpu_full_avg10_milli
physics_host_snapshot, 239
psi_cpu_full_avg300_milli
physics_host_snapshot, 239
psi_cpu_full_avg60_milli
physics_host_snapshot, 239
psi_cpu_some_avg10_milli
physics_host_snapshot, 239
psi_cpu_some_avg300_milli
physics_host_snapshot, 239
psi_cpu_some_avg60_milli
physics_host_snapshot, 239
psi_io_full_avg10_milli
physics_host_snapshot, 240
psi_io_full_avg300_milli
physics_host_snapshot, 240

psi_io_full_avg60_milli
physics_host_snapshot, 240
psi_io_some_avg10_milli
physics_host_snapshot, 240
psi_io_some_avg300_milli
physics_host_snapshot, 240
psi_io_some_avg60_milli
physics_host_snapshot, 240
psi_mem_some_avg10_milli
physics_host_snapshot, 241
psi_mem_some_avg300_milli
physics_host_snapshot, 241
psi_mem_some_avg60_milli
physics_host_snapshot, 241
PT_DYNAMIC
elf64.h, 522
pt_index
vmm.c, 1475
PT_INTERP
elf64.h, 522
PT_LOAD
elf64.h, 522
PT_NOTE
elf64.h, 522
PT_NULL
elf64.h, 522
PT_PHDR
elf64.h, 522
PT_SHLIB
elf64.h, 522
PT_TLS
elf64.h, 522
PTE_ACCESSED
vmm.c, 1470
PTE_DIRTY
vmm.c, 1470
PTE_GLOBAL
vmm.c, 1470
PTE_NX
vmm.c, 1470
PTE_PCD
vmm.c, 1470
PTE_PRESENT
vmm.c, 1470
PTE_PS
vmm.c, 1471
PTE_PWT
vmm.c, 1471
PTE_USER
vmm.c, 1471
PTE_WRITABLE
vmm.c, 1471
pubkey
user_identity_seed_t, 302
zuse_cert_devblock_t, 394
pubkey_state
capsule_sig.c, 1150
published_ns
HeartbeatSnapshot, 196
published_tick
HeartbeatSnapshot, 197
pubsub_mask
DictPhysics, 151
physics_seed_definition, 244
push
usb_kbd.c, 1510
put_char
vt100.c, 1374
put_le32
gpt.c, 1246
put_le64
gpt.c, 1247
put_sectors
gpt.c, 1247
putc
VMHostServices, 343
putchar
stdio.h, 588
puts
stdio.h, 589
VMHostServices, 343
px_of
vt100.c, 1375
py_of
vt100.c, 1376

q48_16.c
q48_cos_approx, 1434
q48_div, 1435
q48_exp_approx, 1436
q48_log_approx, 1437
q48_mul, 1438
q48_reduce_angle, 1439
q48_sin_approx, 1439
q48_sqrt_approx, 1440
q48_16.h
q48_16_t, 1919, 1928
q48_abs, 1919, 1928
q48_add, 1919, 1928
q48_cos_approx, 1920, 1929
q48_div, 1920, 1930
q48_exp_approx, 1921, 1931
q48_from_double, 1932
q48_from_u64, 1922, 1933
q48_is_valid, 1933
q48_log_approx, 1922, 1934
q48_mul, 1923, 1935
Q48_ONE, 1919
q48_sin_approx, 1924, 1936
q48_sqrt_approx, 1925, 1937
q48_sub, 1926, 1938
q48_to_double, 1939
q48_to_string, 1939
q48_to_u64, 1926, 1940
q48_16_t
q48_16.h, 1919, 1928
q48_16_words.c

q48_cos_approx, 3152
q48_div, 3152
q48_exp_approx, 3153
q48_from_double, 3153
q48_is_valid, 3154
q48_log_approx, 3154
q48_mul, 3155
q48_reduce_angle, 3155
q48_sin_approx, 3157
q48_sqrt_approx, 3157
q48_to_double, 3158
q48_to_string, 3158
q48_abs
q48_16.h, 1919, 1928
q48_add
q48_16.h, 1919, 1928
q48_cos_approx
q48_16.c, 1434
q48_16.h, 1920, 1929
q48_16_words.c, 3152
q48_div
q48_16.c, 1435
q48_16.h, 1920, 1930
q48_16_words.c, 3152
q48_div_u64
physics_pipelining_metrics.c, 2364
q48_exp_approx
q48_16.c, 1436
q48_16.h, 1921, 1931
q48_16_words.c, 3153
q48_from_double
q48_16.h, 1932
q48_16_words.c, 3153
q48_stubs.c, 1646
q48_from_i32
ttf.c, 1346
q48_from_u64
q48_16.h, 1922, 1933
q48_is_valid
q48_16.h, 1933
q48_16_words.c, 3154
q48_log_approx
q48_16.c, 1437
q48_16.h, 1922, 1934
q48_16_words.c, 3154
q48_mid
ttf.c, 1346
q48_mul
q48_16.c, 1438
q48_16.h, 1923, 1935
q48_16_words.c, 3155
Q48_ONE
q48_16.h, 1919
q48_pop
q48_words.c, 3160
q48_pop_inf
inference_words.c, 3042
q48_push
q48_words.c, 3161
q48_push_inf
inference_words.c, 3043
q48_reduce_angle
q48_16.c, 1439
q48_16_words.c, 3155
Q48_SCALE
math_portable.h, 1813
q48_sin_approx
q48_16.c, 1439
q48_16.h, 1924, 1936
q48_16_words.c, 3157
q48_smul
ttf.c, 1346
q48_sqrt_approx
q48_16.c, 1440
q48_16.h, 1925, 1937
q48_16_words.c, 3157
q48_stubs.c
q48_from_double, 1646
q48_to_double, 1647
q48_sub
q48_16.h, 1926, 1938
q48_to_dbl
inference_words.c, 3044
q48_to_double
q48_16.h, 1939
q48_16_words.c, 3158
q48_stubs.c, 1647
q48_to_ns
physics_pipelining_metrics.c, 2365
q48_to_ns_double
physics_hotwords_cache.c, 2347
q48_to_string
q48_16.h, 1939
q48_16_words.c, 3158
q48_to_u64
q48_16.h, 1926, 1940
q48_word_abs
q48_words.c, 3161
q48_word_add
q48_words.c, 3162
q48_word_cos
q48_words.c, 3163
q48_word_div
q48_words.c, 3164
q48_word_eq
q48_words.c, 3164
q48_word_exp
q48_words.c, 3165
q48_word_from_int
q48_words.c, 3166
q48_word_gt
q48_words.c, 3166
q48_word_log
q48_words.c, 3167
q48_word_lt
q48_words.c, 3168

q48_word_max
q48_words.c, 3168
q48_word_min
q48_words.c, 3169
q48_word_mul
q48_words.c, 3169
q48_word_neg
q48_words.c, 3170
q48_word_one
q48_words.c, 3171
q48_word_print
q48_words.c, 3171
q48_word_scale
q48_words.c, 3172
q48_word_sin
q48_words.c, 3172
q48_word_sqrt
q48_words.c, 3173
q48_word_sub
q48_words.c, 3174
q48_word_suites
q48_words_test.c, 2636
q48_word_to_int
q48_words.c, 3175
q48_word_zero
q48_words.c, 3175
q48_word_zero_eq
q48_words.c, 3176
q48_words.c
q48_pop, 3160
q48_push, 3161
q48_word_abs, 3161
q48_word_add, 3162
q48_word_cos, 3163
q48_word_div, 3164
q48_word_eq, 3164
q48_word_exp, 3165
q48_word_from_int, 3166
q48_word_gt, 3166
q48_word_log, 3167
q48_word_lt, 3168
q48_word_max, 3168
q48_word_min, 3169
q48_word_mul, 3169
q48_word_neg, 3170
q48_word_one, 3171
q48_word_print, 3171
q48_word_scale, 3172
q48_word_sin, 3172
q48_word_sqrt, 3173
q48_word_sub, 3174
q48_word_to_int, 3175
q48_word_zero, 3175
q48_word_zero_eq, 3176
register_q48_words, 3176
q48_words.h
register_q48_words, 3000
q48_words_test.c
q48_word_suites, 2636
run_q48_words_tests, 2635
qsort
stdlib.h, 596
QueryMode
_EFI_GRAPHICS_OUTPUT_PROTOCOL, 93
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 98
queue_desc
VirtioCommonCfg, 310
queue_device
VirtioCommonCfg, 310
queue_driver
VirtioCommonCfg, 310
queue_enable
VirtioCommonCfg, 310
queue_msix_vector
VirtioCommonCfg, 310
queue_notify_data
VirtioCommonCfg, 310
queue_notify_off
VirtBlkState, 305
VirtInputState, 308
VirtioCommonCfg, 311
VirtRngState, 317
queue_reset
VirtioCommonCfg, 311
queue_select
VirtioCommonCfg, 311
queue_size
VirtioCommonCfg, 311

R_AARCH64_ABS64
elf64.h, 523
R_AARCH64_NONE
elf64.h, 523
R_AARCH64_RELATIVE
elf64.h, 523
r_addend
Elf64_Rela, 181
r_info
Elf64_Rel, 180
Elf64_Rela, 181
r_offset
Elf64_Rel, 180
Elf64_Rela, 181
R_RISCV_64
elf64.h, 523
R_RISCV_NONE
elf64.h, 523
R_RISCV_RELATIVE
elf64.h, 523
R_X86_64_32
elf64.h, 523
R_X86_64_32S
elf64.h, 523
R_X86_64_64
elf64.h, 523
R_X86_64_NONE
elf64.h, 523

R_X86_64_PC32
elf64.h, 524
R_X86_64_PLT32
elf64.h, 524
R_X86_64_RELATIVE
elf64.h, 524
RaiseTPL
_EFI_BOOT_SERVICES, 89
ram_base
block_subsystem.c, 2213
ram_close
blkio_ram.c, 2144
ram_disk_mb
CLIConfig, 143
ram_flush
blkio_ram.c, 2144
ram_info
blkio_ram.c, 2145
ram_open
blkio_ram.c, 2145
ram_read
blkio_ram.c, 2146
ram_size
block_subsystem.c, 2213
ram_user
block_subsystem.c, 2213
ram_write
blkio_ram.c, 2146
raw
KernelArgs, 219
sk_v4_drive_t, 269
raw_base
blk_dev_slot, 106
RAW_LOG
uefi_loader.c, 1082
raw_midpoint
ttf.c, 1347
raw_putc
console.c, 1274
rd_i16
ttf.c, 1347
rd_i8
ttf.c, 1348
rd_u16
ttf.c, 1348
rd_u32
ttf.c, 1348
rd_u8
ttf.c, 1349
rdmsr64
apic.c, 936
rdtime
apic.c, 946
timer.c, 1055
rdtsc
timer.c, 1038
rdtscp_aux
timer.c, 1039
Read
_EFI_FILE_PROTOCOL, 91
read
blkio_vtable_t, 122
read_answer
sk_v4_mint.c, 1601
read_cells
rpi5_native_boot.c, 887
read_cells_at
fdt.c, 1304
read_cgroup_cpu_usage_us
physics_runtime.c, 2390
read_cgroup_memory_current
physics_runtime.c, 2391
read_devblock
capsule_mint.c, 1133
capsule_wirebind.c, 1179
capsule_zuse_boot.c, 1191
read_devblock_4k
block_subsystem.c, 2201
read_header
sk_v4_grow.c, 1569
read_header_4k
block_subsystem.c, 2202
read_offset
physics_analytics_header, 231
read_only
blkcfg_t, 116
blkio_file_state_t, 118
blkio_info_t, 119
blkio_ram_state_t, 121
read_proc_stat_totals
physics_runtime.c, 2392
read_psi_file
physics_runtime.c, 2393
ReadBlocks
_EFI_BLOCK_IO_PROTOCOL, 84
README.md, 1943
ReadOnly
EFI_BLOCK_IO_MEDIA, 163
reads
MemoryProfile, 224
ready
FbState, 186
real
sk_v4_marker_t, 270
real_pubkey
zuse_genesis_marker_v2_t, 396
realloc
stdlib.h, 596
realtime_ns
physics_host_snapshot, 241
REBOOT_MAX_TRIES
kernel_args.h, 636
recompute_free_pages
pmm.c, 1466
record_count
log_region_ctrl_t, 222

recycled_word_id_count
VM, 333
recycled_word_ids
VM, 333
RedMask
EFI_PIXEL_BITMASK, 174
reg32
plic.c, 1005
reg_first_cell_be32
rpi5_dtb.c, 857
reg_shift
fdt_uart_t, 189
regime_scores
CDConfigTable, 137
regime_ticks
CDConfigTable, 137
register_acl_words
acl_words.c, 2791
acl_words.h, 2932
register_arithmetic_words
arithmetic_words.c, 2802
arithmetic_words.h, 2934
register_bit_offset
__attribute__, 82
register_bit_width
__attribute__, 82
register_block_words
block_words.c, 2831
block_words.h, 2946
register_control_words
Runtime Branch Helpers, 67
register_defer_words
defer_words.c, 2838
defer_words.h, 2950
register_defining_words
defining_words.c, 2858
defining_words.h, 2952
register_dictionary_heat_diagnostic_words
dictionary_heat_diagnostic_words.c, 2865
dictionary_heat_diagnostic_words.h, 2956
register_dictionary_manipulation_words
dictionary_manipulation_words.c, 2877
dictionary_manipulation_words.h, 2958
register_dictionary_words
dictionary_words.c, 2887
dictionary_words.h, 2960
register_doe_log_words
doe_log.c, 1234
doe_log.h, 509
register_double_words
double_words.c, 2900
double_words.h, 2962
register_editor_words
editor_words.c, 2908
editor_words.h, 2964
register_format_words
format_words.c, 2925
format_words.h, 2966
register_forth79_words
word_registry.c, 2774
word_registry.h, 2121
register_framebuffer_words
framebuffer_words.c, 2929
framebuffer_words.h, 2969
register_inference_words
inference_words.c, 3044
inference_words.h, 2971
register_io_words
io_words.c, 3054
io_words.h, 2974
register_keyboard_words
keyboard_words.c, 3059
keyboard_words.h, 2976
register_lifecycle_words
lifecycle_words_hosted.c, 3067
register_log_words
log_words.c, 3084
log_words.h, 2979
register_logical_words
logical_words.c, 3098
logical_words.h, 2981
register_mama_forth_words
mama_forth_words.h, 2984
register_memory_words
memory_words.c, 3113
memory_words.h, 2985
register_mixed_arithmetic_words
mixed_arithmetic_words.c, 3122
mixed_arithmetic_words.h, 2987
register_physics_benchmark_words
physics_benchmark_words.c, 3129
physics_benchmark_words.h, 2989
register_physics_diagnostic_words
physics_diagnostic_words.c, 3135
physics_diagnostic_words.h, 2994
register_physics_freeze_words
physics_freeze_words.c, 3143
physics_freeze_words.h, 2996
register_physics_pipelining_diagnostic_words
physics_pipelining_diagnostic_words.c, 3149
physics_pipelining_diagnostic_words.h, 2998
register_q48_words
q48_words.c, 3176
q48_words.h, 3000
register_return_stack_words
return_stack_words.c, 3180
return_stack_words.h, 3003
register_scroll_words
scroll_words.c, 3184
scroll_words.h, 3004
register_stack_words
stack_words.c, 3187
stack_words.h, 3006
register_starforth_words
starforth_words.c, 3197
starforth_words.h, 3009

register_string_words
string_words.c, 3214
string_words.h, 3018
register_system_words
system_words.c, 3235
system_words.h, 3021
register_ttf_words
ttf_words.c, 3248
ttf_words.h, 3023
register_vocabulary_words
vocabulary_words.c, 3253
vocabulary_words.h, 3025
register_word
word_registry.c, 2776
word_registry.h, 2123
RegisterProtocolNotify
_EFI_BOOT_SERVICES, 89
ReinstallProtocolInterface
_EFI_BOOT_SERVICES, 89
reloc
block_subsystem.c, 2213
reloc_block, 253
block_size, 254
entry, 254
page_rva, 254
reloc_count
block_subsystem.c, 2213
reloc_devblocks
blk_volume_meta_t, 114
reloc_flush_to_disk
block_subsystem.c, 2203
reloc_load_from_disk
block_subsystem.c, 2204
reloc_start
blk_volume_meta_t, 114
reloc_stub.c
__attribute__, 1079
block_size, 1080
entry, 1080
page_rva, 1081
RemovableMedia
EFI_BLOCK_IO_MEDIA, 163
REPL Scrollback Words, 76
repl.c
__attribute__, 2469
g_artemis_usb_dev, 2466
g_attached_blk_dev, 2466
g_console_pending_key, 2466
g_cursor_blink_tick, 2466
g_cursor_visible, 2466
g_homeblocks_dev, 2467
g_homeblocks_sig, 2467
g_homeblocks_sig_valid, 2467
g_idle_pump_active, 2467
g_kh_blk_attach_buf, 2467
g_kh_console_cmd_buf, 2467
g_last_beat_tick, 2467
g_mama_interpreting, 2467
g_repl_active_vm, 2468
g_usb_ctrl, 2468
lithos_version, 2468
sk_console_getkey, 2445
sk_console_identity_present, 2446
sk_console_key_available, 2447
sk_console_readline, 2448
sk_console_user_prefix, 2450
SK_CURSOR_BLINK_INTERVAL, 2444
sk_cursor_show, 2451
SK_IDLE_BEAT_INTERVAL, 2444
SK_KH_BLK_ATTACH_BUF_SIZE, 2444
sk_print_prompt, 2452
SK_PROMPT_TEXT, 2444
sk_repl, 2452
sk_repl_dispatch_line, 2454
sk_repl_get_active_vm, 2455
sk_repl_get_artemis_usb_dev, 2455
sk_repl_get_attached_blk_dev, 2455
sk_repl_get_homeblocks_dev, 2455
sk_repl_get_homeblocks_sig, 2456
sk_repl_headless_wait, 2456
sk_repl_idle, 2457
sk_repl_line_calls_use, 2460
sk_repl_register_words, 2460
sk_repl_run, 2461
sk_repl_set_active_vm, 2463
sk_repl_step, 2463
sk_word_blk_attach_ack, 2464
sk_word_kh_blk_attach_send, 2465
vm_repl, 2469
repl.h
sk_console_getkey, 1945
sk_console_key_available, 1946
sk_console_readline, 1947
sk_repl, 1949
sk_repl_get_active_vm, 1951
sk_repl_get_attached_blk_dev, 1951
sk_repl_get_homeblocks_dev, 1951
sk_repl_get_homeblocks_sig, 1951
sk_repl_headless_wait, 1952
sk_repl_register_words, 1953
sk_repl_run, 1955
sk_repl_set_active_vm, 1956
sk_repl_step, 1957
vm_repl, 1959
replace_entry
rpi5_mmu.c, 874
report_file
break_me_tests.c, 2606
req_buf
VirtBlkState, 306
requested
heap_block, 195
Reserved
_EFI_BOOT_SERVICES, 89
EFI_LOADED_IMAGE_PROTOCOL, 172
EFI_TABLE_HEADER, 175

reserved
__attribute__, 82
CapsuleDirHeader, 131
CapsuleRunRecord, 133
physics_analytics_event_header, 229
TestModule, 279
timer_calibration_record, 284
VirtBlkReq, 303
xhci_ep_ctx32_t, 378
xhci_input_ctrl_ctx32_t, 379
xhci_slot_ctx32_t, 391
reserved0
xhci_cap_regs_t, 366
xhci_intr_regs_t, 380
xhci_op_regs_t, 388
xhci_runtime_regs_t, 390
reserved1
blk_meta_t, 110
xhci_op_regs_t, 388
reserved_disk_lo
blk_volume_meta_t, 114
reserved_future
blk_meta_t, 110
reserved_int
TestModule, 279
reserved_ram_lo
blk_volume_meta_t, 114
ReservedMask
EFI_PIXEL_BITMASK, 174
Reset
_EFI_BLOCK_IO_PROTOCOL, 84
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 98
reset_attrs
vt100.c, 1377
reset_parser
vt100.c, 1377
reset_vm_state
system_words.c, 3237
ResetSystem
_EFI_RUNTIME_SERVICES, 95
resolve_cgroup_path
physics_runtime.c, 2394
resolve_contract
test_common.c, 2659
resolve_lbn
block_subsystem.c, 2204
resolve_pt
vmm.c, 1476
restore_dict_state
test_common.c, 2660
test_common.h, 2518
restore_physics
test_contracts.c, 2683
test_contracts.h, 2543
restore_vm_state
test_common.c, 2661
test_common.h, 2521
RestoreTPL
_EFI_BOOT_SERVICES, 89
result_code
CapsuleRunRecord, 133
return_stack
VM, 333
return_stack_word_r_fetch
return_stack_words.c, 3180
return_stack_word_r_from
return_stack_words.c, 3182
return_stack_word_suites
return_stack_words_test.c, 2638
return_stack_word_to_r
return_stack_words.c, 3183
return_stack_words.c
register_return_stack_words, 3180
return_stack_word_r_fetch, 3180
return_stack_word_r_from, 3182
return_stack_word_to_r, 3183
return_stack_words.h
register_return_stack_words, 3003
return_stack_words_test.c
return_stack_word_suites, 2638
run_return_stack_words_tests, 2637
reverse
VT100State, 352
Revision
_EFI_BLOCK_IO_PROTOCOL, 84
_EFI_FILE_PROTOCOL, 92
_EFI_SIMPLE_FILE_SYSTEM_PROTOCOL, 96
EFI_LOADED_IMAGE_PROTOCOL, 172
EFI_TABLE_HEADER, 175
revision
__attribute__, 82
reward_gain
CDTuning, 140
rewind
stdio.h, 589
ring
i8042.c, 898
physics_analytics_heap, 234
VirtqAvail, 312
VirtqUsed, 315
ring_bytes
physics_analytics_header, 231
physics_analytics_heap, 235
ring_head
i8042.c, 898
ring_offset
physics_analytics_header, 232
RING_SIZE
i8042.c, 893
ring_tail
i8042.c, 899
riscv64_exception_handler
interrupts.c, 992
riscv64_install_vectors
arch.c, 966
interrupts.c, 993

riscv64_interrupt_handler
interrupts.c, 993
riscv64_timer_rearm
apic.c, 946
interrupts.c, 994
RISCV_TIMEBASE_HZ_FALLBACK
timer.c, 1053
rmb
virtio_blk.c, 1609
virtio_rng.c, 1628
rng.c
g_rng_backend, 1502
RNG_BACKEND_NONE, 1501
RNG_BACKEND_VIRTIO, 1501
rng_get_bytes, 1501
rng_init, 1501
rng_ready, 1502
rng.h
RNG_ERR_NO_BACKEND, 670
rng_get_bytes, 670
rng_init, 670
rng_ready, 671
RNG_BACKEND_NONE
rng.c, 1501
RNG_BACKEND_VIRTIO
rng.c, 1501
RNG_ERR_NO_BACKEND
rng.h, 670
rng_get_bytes
rng.c, 1501
rng.h, 670
rng_init
rng.c, 1501
rng.h, 670
rng_ready
rng.c, 1502
rng.h, 671
rng_seed
test_contracts.c, 2684
rng_state
test_contracts.c, 2685
rolling_window
VM, 333
rolling_window_check_adaptive_shrink
rolling_window_of_truth.c, 2473
rolling_window_of_truth.h, 1963
rolling_window_cleanup
rolling_window_of_truth.c, 2473
rolling_window_of_truth.h, 1963
rolling_window_count_transition
rolling_window_of_truth.c, 2474
rolling_window_of_truth.h, 1964
rolling_window_export_execution_history
rolling_window_of_truth.c, 2475
rolling_window_of_truth.h, 1965
rolling_window_find_hottest_word
rolling_window_of_truth.c, 2476
rolling_window_of_truth.h, 1966
rolling_window_get_recent_sequence
rolling_window_of_truth.c, 2477
rolling_window_of_truth.h, 1967
rolling_window_init
rolling_window_of_truth.c, 2478
rolling_window_of_truth.h, 1968
rolling_window_is_warm
rolling_window_of_truth.c, 2480
rolling_window_of_truth.h, 1970
rolling_window_measure_diversity
rolling_window_of_truth.c, 2480
rolling_window_of_truth.h, 1970
rolling_window_measure_diversity_view
rolling_window_of_truth.c, 2481
rolling_window_of_truth.c
rolling_window_check_adaptive_shrink, 2473
rolling_window_cleanup, 2473
rolling_window_count_transition, 2474
rolling_window_export_execution_history, 2475
rolling_window_find_hottest_word, 2476
rolling_window_get_recent_sequence, 2477
rolling_window_init, 2478
rolling_window_is_warm, 2480
rolling_window_measure_diversity, 2480
rolling_window_measure_diversity_view, 2481
rolling_window_pattern_capture_rate, 2482
rolling_window_publish_snapshot, 2483
rolling_window_publish_snapshot_if_needed, 2484
rolling_window_record_execution, 2485
rolling_window_reset, 2486
rolling_window_run_adaptive_pass, 2487
rolling_window_seed_hotwords_cache, 2488
rolling_window_seed_pipelining_context, 2489
rolling_window_service, 2490
rolling_window_service_internal, 2491
rolling_window_snapshot_view, 2492
rolling_window_stats_string, 2493
rolling_window_of_truth.h
rolling_window_check_adaptive_shrink, 1963
rolling_window_cleanup, 1963
rolling_window_count_transition, 1964
rolling_window_export_execution_history, 1965
rolling_window_find_hottest_word, 1966
rolling_window_get_recent_sequence, 1967
rolling_window_init, 1968
rolling_window_is_warm, 1970
rolling_window_measure_diversity, 1970
rolling_window_pattern_capture_rate, 1971
rolling_window_record_execution, 1972
rolling_window_reset, 1973
rolling_window_service, 1975
rolling_window_stats_string, 1976
rolling_window_pattern_capture_rate
rolling_window_of_truth.c, 2482
rolling_window_of_truth.h, 1971
rolling_window_publish_snapshot
rolling_window_of_truth.c, 2483

rolling_window_publish_snapshot_if_needed
rolling_window_of_truth.c, 2484
rolling_window_record_execution
rolling_window_of_truth.c, 2485
rolling_window_of_truth.h, 1972
rolling_window_reset
rolling_window_of_truth.c, 2486
rolling_window_of_truth.h, 1973
rolling_window_run_adaptive_pass
rolling_window_of_truth.c, 2487
rolling_window_seed_hotwords_cache
rolling_window_of_truth.c, 2488
rolling_window_seed_pipelining_context
rolling_window_of_truth.c, 2489
rolling_window_service
rolling_window_of_truth.c, 2490
rolling_window_of_truth.h, 1975
rolling_window_service_internal
rolling_window_of_truth.c, 2491
ROLLING_WINDOW_SIZE
starforth_config.h, 1992
rolling_window_size
DoeMetrics, 158
rolling_window_snapshot_view
rolling_window_of_truth.c, 2492
rolling_window_stats_string
rolling_window_of_truth.c, 2493
rolling_window_of_truth.h, 1976
rolling_window_view_t, 254
effective_window_size, 255
history, 255
is_warm, 255
total_executions, 255
window_pos, 255
rolling_window_width
DoeMetrics, 159
RollingWindowOfTruth, 255
adaptive_check_accumulator, 256
adaptive_pending, 256
effective_window_size, 256
execution_history, 256
is_warm, 256
last_pattern_diversity, 257
pattern_diversity_check_count, 257
snapshot_buffers, 257
snapshot_effective_window_size, 257
snapshot_index, 257
snapshot_is_warm, 257
snapshot_pending, 257
snapshot_total_executions, 258
snapshot_window_pos, 258
total_executions, 258
window_pos, 258
rotr64
sha512.c, 1221
rows
VT100State, 352
rpi5_dtb.c
BCM2712_SOC_RANGES_OFFSET, 857
reg_first_cell_be32, 857
rpi5_mailbox_base, 857
rpi5_peripheral_base, 858
rpi5_uart_base, 859
rpi5_dtb.h
rpi5_mailbox_base, 672
rpi5_uart_base, 673
RPI5_FB_DEFAULT_BPP
rpi5_native_boot.c, 879
RPI5_FB_DEFAULT_HEIGHT
rpi5_native_boot.c, 879
RPI5_FB_DEFAULT_WIDTH
rpi5_native_boot.c, 879
rpi5_mailbox.c
__attribute__, 863
append_tag, 863
buffer_to_ram, 864
bus_to_phys, 864
counter_freq, 865
counter_now, 865
mb_be32, 865
mb_cell_prop, 866
mb_cells, 866
mb_compatible, 867
mb_find_path, 868
MBOX_CH_PROPERTY, 861
MBOX_CODE_REQUEST, 861
MBOX_CODE_RESPONSE_OK, 861
MBOX_EMPTY, 861
MBOX_FULL, 861
MBOX_OFF_READ, 861
MBOX_OFF_STATUS0, 862
MBOX_OFF_STATUS1, 862
MBOX_OFF_WRITE, 862
mmio_read32, 868
mmio_write32, 869
PIXEL_ORDER_RGB, 862
rpi5_mailbox_get_framebuffer, 869
TAG_ALLOCATE_BUFFER, 862
TAG_END, 863
TAG_GET_PITCH, 863
TAG_SET_DEPTH, 863
TAG_SET_PHYS_SIZE, 863
TAG_SET_PIXEL_ORDER, 863
TAG_SET_VIRT_SIZE, 863
TAG_SET_VIRTUAL_OFFS, 863
rpi5_mailbox.h
rpi5_mailbox_get_framebuffer, 675
rpi5_mailbox_base
rpi5_dtb.c, 857
rpi5_dtb.h, 672
rpi5_mailbox_get_framebuffer
rpi5_mailbox.c, 869
rpi5_mailbox.h, 675
RPI5_MAX_MEMMAP_ENTRIES
rpi5_native_boot.c, 880
RPI5_MAX_RAM_REGIONS

rpi5_native_boot.c, 880
RPI5_MAX_RESERVED_RANGES
rpi5_native_boot.c, 880
rpi5_mmu.c
__kernel_end, 877
__kernel_start, 877
be32_at, 872
BLOCK_NORMAL_NC, 872
BLOCK_NORMAL_WB, 872
cells_at, 872
current_el, 873
g_rpi5_l1, 877
g_rpi5_l2, 877
GIB, 872
l2_gib, 873
MIB2, 872
replace_entry, 874
rpi5_mmu_map_ram, 874
rpi5_mmu_uncached, 875
tlb_flush_all, 876
rpi5_mmu.h
rpi5_mmu_map_ram, 677
rpi5_mmu_uncached, 678
rpi5_mmu_map_ram
rpi5_mmu.c, 874
rpi5_mmu.h, 677
rpi5_mmu_uncached
rpi5_mmu.c, 875
rpi5_mmu.h, 678
rpi5_native_boot
rpi5_native_boot.c, 888
rpi5_native_boot.h, 680
rpi5_native_boot.c
add_memmap_entry, 880
be32_at, 880
build_memory_map, 881
collect_ram_regions, 882
collect_reserved_ranges, 882
emit_region_with_carveouts, 883
g_rpi5_boot_info, 890
g_rpi5_memmap, 890
kernel_main, 884
node_cells_or_fallback, 887
read_cells, 887
RPI5_FB_DEFAULT_BPP, 879
RPI5_FB_DEFAULT_HEIGHT, 879
RPI5_FB_DEFAULT_WIDTH, 879
RPI5_MAX_MEMMAP_ENTRIES, 880
RPI5_MAX_RAM_REGIONS, 880
RPI5_MAX_RESERVED_RANGES, 880
rpi5_native_boot, 888
rpi5_native_halt, 889
sort_ranges_by_addr, 890
rpi5_native_boot.h
rpi5_native_boot, 680
rpi5_native_entry.h
g_rpi5_dtb_ptr, 682
rpi5_native_halt
rpi5_native_boot.c, 889
rpi5_peripheral_base
rpi5_dtb.c, 858
rpi5_range_t, 258
addr, 259
size, 259
rpi5_uart_base
rpi5_dtb.c, 859
rpi5_dtb.h, 673
Rpi5FramebufferInfo, 259
base, 259
height, 259
pixel_format, 260
pixels_per_scanline, 260
size, 260
width, 260
rsdt_address
__attribute__, 83
rsp
ExecResult, 185
VM, 333
rts_off
xhci_cap_regs_t, 366
run_acl_words_tests
acl_words_test.c, 2581
test_runner.h, 2548
run_all_tests
test_runner.c, 2688
test_runner.h, 2550
run_arithmetic_words_tests
arithmetic_words_test.c, 2600
test_runner.h, 2551
run_benchmark_mode
main.c, 2308
run_block_words_tests
block_words_test.c, 2601
test_runner.h, 2552
run_break_me_tests
break_me_tests.c, 2605
test_runner.h, 2553
run_control_words_tests
control_words_test.c, 2607
test_runner.h, 2554
run_defining_words_tests
defining_words_tests.c, 2609
test_runner.h, 2555
run_dictionary_manipulation_words_tests
dictionary_manipulation_words_test.c, 2611
test_runner.h, 2556
run_dictionary_words_tests
dictionary_words_test.c, 2613
test_runner.h, 2557
run_doe
KernelArgs, 219
run_doe_experiment
main.c, 2308
run_double_words_tests
double_words_test.c, 2615

test_runner.h, 2558
run_format_words_tests
format_words_test.c, 2617
test_runner.h, 2559
run_id
CapsuleRunRecord, 133
run_id_counter
capsule_run.c, 1145
run_inference_words_tests
inference_words_test.c, 2619
test_runner.h, 2560
run_integration_tests
integration_tests.c, 2621
test_runner.h, 2561
run_io_words_tests
io_words_test.c, 2623
test_runner.h, 2562
run_log
capsule_run.c, 1145
run_log_count
capsule_run.c, 1145
run_log_head
capsule_run.c, 1145
run_logical_words_tests
logical_words_test.c, 2625
test_runner.h, 2563
run_mama_forth_words_tests
mama_forth_words_test.c, 2628
test_runner.h, 2564
run_memory_words_tests
memory_words_test.c, 2630
test_runner.h, 2565
run_mixed_arithmetic_words_tests
mixed_arithmetic_words_test.c, 2631
test_runner.h, 2566
run_module_tests
test_runner.c, 2689
test_runner.h, 2567
TestModule, 279
run_optional_tests
main.c, 2309
run_physics_freeze_words_tests
physics_freeze_words_test.c, 2633
test_runner.h, 2568
run_q48_words_tests
q48_words_test.c, 2635
test_runner.h, 2569
run_return_stack_words_tests
return_stack_words_test.c, 2637
test_runner.h, 2570
run_single_test
test_common.c, 2663
test_common.h, 2523
run_stack_words_tests
stack_words_test.c, 2639
test_runner.h, 2571
run_starforth_words_tests
starforth_words_test.c, 2641
test_runner.h, 2572
run_start_ns
HeartbeatState, 200
run_stress_tests
stress_tests.c, 2644
test_runner.h, 2573
run_string_words_tests
string_words_test.c, 2646
test_runner.h, 2574
run_system_words_tests
system_words_test.c, 2648
test_runner.h, 2575
run_test_suite
test_common.c, 2666
test_common.h, 2526
run_test_suite_m
test_common.c, 2668
test_common.h, 2528
run_tests
CLIConfig, 143
run_tests_flag_observed
CLIConfig, 143
run_vocabulary_words_tests
test_runner.h, 2576
vocabulary_words_test.c, 2650
run_word_tests
test_runner.c, 2690
test_runner.h, 2577
runnable_threads
physics_host_snapshot, 241
running
HeartbeatWorker, 204
running_under_hypervisor
timer.c, 1008, 1040, 1056
runtime
xhci_dev_t, 376
Runtime Architecture Detection, 19
Runtime Branch Helpers, 48
control_forth_0branch, 49
control_forth_again, 50
control_forth_begin, 50
control_forth_branch, 51
control_forth_case, 52
control_forth_do, 52
control_forth_else, 53
control_forth_endcase, 54
control_forth_endof, 55
control_forth_EXIT, 55
control_forth_I, 56
control_forth_if, 57
control_forth_J, 57
control_forth_leave, 58
control_forth_loop, 59
control_forth_of, 59
control_forth_plus_loop, 60
control_forth_qdo, 61
control_forth_repeat, 62
control_forth_runtime_do, 62

control_forth_runtime_leave, 63
control_forth_runtime_loop, 63
control_forth_runtime_plus_loop, 64
control_forth_runtime_qdo, 64
control_forth_then, 65
control_forth_UNLOOP, 65
control_forth_until, 66
control_forth_while, 67
register_control_words, 67
runtime_abortq
system_words.c, 3237
runtime_services
BootInfo, 128
RuntimeServices
_EFI_SYSTEM_TABLE, 100

s_apic_hz
apic.c, 938
s_base_count
timer.c, 1011, 1060
s_base_ns
timer.c, 1011, 1060
s_cal
timer.c, 1011, 1061
s_counter_hz
timer.c, 1011, 1061
s_counter_hz_apic
apic.c, 921
s_current_el
arch.c, 954
s_gicc_base
apic.c, 921
s_gicd_base
apic.c, 921
s_next_deadline
apic.c, 949
s_ns_per_tick
timer.c, 1011, 1061
s_plic_base
plic.c, 1005
s_psci_use_smc
arch.c, 954
s_sbi_time_ok
apic.c, 949
s_time_hz
apic.c, 949
s_timer_period_tsc
apic.c, 921, 949
s_timer_ppi
apic.c, 921
sa_flags
sigaction, 265
sa_handler
sigaction, 265
SA_RESETHAND
signal.h, 578
sample_count
BayesianLatencyPosterior, 103
SAMPLE_WINDOWS
timer.c, 1016
sanity_check_hpet_running_quick
timer.c, 1040
save_all_buffers
block_words.c, 2833
block_words.h, 2948
save_dict_state
test_common.c, 2670
test_common.h, 2530
save_physics
test_contracts.c, 2684
test_contracts.h, 2544
save_vm_state
test_common.c, 2671
test_common.h, 2531
saved_cx
VT100State, 352
saved_cy
VT100State, 352
say_size
sk_v4_mint.c, 1602
SBI_BASE_FID_PROBE_EXT
apic.c, 940
sbi_call1
apic.c, 947
SBI_EXT_BASE
apic.c, 940
SBI_EXT_TIME
apic.c, 940
sbi_set_timer
apic.c, 948
SBI_SUCCESS
apic.c, 940
SBI_TIME_FID_SET_TIMER
apic.c, 940
sbiret_t, 260
error, 260
value, 260
scalar25519.c
cmp32, 1216
getbit, 1216
L_BYTES, 1220
mul256, 1216
scalar_lt_L, 1217
scalar_muladd, 1217
scalar_reduce512, 1218
sub_L, 1219
scalar25519.h
scalar_lt_L, 683
scalar_muladd, 684
scalar_reduce512, 684
scalar_lt_L
scalar25519.c, 1217
scalar25519.h, 683
scalar_muladd
scalar25519.c, 1217
scalar25519.h, 684
scalar_mult

ed25519.c, 1204
scalar_reduce512
scalar25519.c, 1218
scalar25519.h, 684
scale
FbState, 186
SCAUSE_INTERRUPT_BIT
interrupts.c, 990
SCAUSE_S_EXTERNAL
interrupts.c, 990
SCAUSE_S_TIMER
interrupts.c, 990
sched.h
sched_yield, 577
sched_yield
sched.h, 577
scheduler_policy
physics_host_snapshot, 241
scheduler_priority
physics_host_snapshot, 242
scheduler_quantum_ns
physics_host_snapshot, 242
score_decay_factor_q16
CDTuning, 140
score_max
CDTuning, 140
score_min
CDTuning, 140
scr_addr
VM, 334
scratch
physics_analytics_heap, 235
scratch_bytes
physics_analytics_header, 232
physics_analytics_heap, 235
scratch_offset
physics_analytics_header, 232
scratchpad_arr
xhci_dev_t, 377
screen_boot
KernelArgs, 219
script_mode
CLIConfig, 143
scroll_up
vt100.c, 1378
scroll_words.c
register_scroll_words, 3184
scroll_words.h
register_scroll_words, 3004
scrollback_line_at
vt100.c, 1379
scrollback_redraw
vt100.c, 1380
scsi_block_size
BlkioUsbState, 123
scsi_blocks_per_fblock
BlkioUsbState, 123
SCSI_CDB_LEN_READ10
xhci.h, 805
SCSI_CDB_LEN_READ_CAPACITY10
xhci.h, 805
SCSI_CDB_LEN_TEST_UNIT_READY
xhci.h, 805
SCSI_CDB_LEN_WRITE10
xhci.h, 805
SCSI_CMD_READ10
xhci.h, 805
SCSI_CMD_READ_CAPACITY10
xhci.h, 805
SCSI_CMD_TEST_UNIT_READY
xhci.h, 805
SCSI_CMD_WRITE10
xhci.h, 805
SCSI_READ_CAPACITY10_DATA_LEN
xhci.h, 805
Second
EFI_TIME, 176
sector
VirtBlkReq, 303
seed
user_identity_seed_t, 302
XXHash64State, 393
zuse_cert_devblock_t, 394
SEEK_CUR
stdio.h, 580
SEEK_END
stdio.h, 580
SEEK_SET
stdio.h, 580
segment_group
__attribute__, 83
selector
idt_entry, 214
interrupts.c, 988
SERIAL_MMIO16550
console.c, 1253
SERIAL_NONE
console.c, 1253
SERIAL_PL011
console.c, 1253
SERIAL_PORT16550
console.c, 1253
serial_putc
console.c, 1276
serial_reg_read
console.c, 1278
serial_reg_write
console.c, 1279
serial_rx_ready
console.c, 1280
SERIAL_TX_MAX_SPIN
console.c, 1252
serial_tx_ready
console.c, 1281
set_cursor
vt100.c, 1381

set_forth_79_compliance
system_words.c, 3237
set_idt_entry
interrupts.c, 986
set_realtime_ns
sf_time_backend_t, 263
set_scr
block_words.c, 2833
SetAttribute
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 98
SetCursorPosition
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 98
SetInfo
_EFI_FILE_PROTOCOL, 92
SetMode
_EFI_GRAPHICS_OUTPUT_PROTOCOL, 94
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 98
SetPosition
_EFI_FILE_PROTOCOL, 92
SetTime
_EFI_RUNTIME_SERVICES, 95
SetTimer
_EFI_BOOT_SERVICES, 89
SetVariable
_EFI_RUNTIME_SERVICES, 95
SetVirtualAddressMap
_EFI_RUNTIME_SERVICES, 96
SetWakeupTime
_EFI_RUNTIME_SERVICES, 96
SetWatchdogTimer
_EFI_BOOT_SERVICES, 89
SF_ALIGN
alloc_kernel.c, 2401, 2410
SF_ALIGN_UP
alloc_kernel.c, 2410
sf_alloc_get_stats
alloc_host.c, 2398
alloc_kernel.c, 2402, 2410
platform_alloc.h, 1883
sf_alloc_init
alloc_host.c, 2398
alloc_kernel.c, 2402, 2410
platform_alloc.h, 1884
sf_alloc_stats, 261
alloc_count, 261
free_count, 261
peak_bytes, 261
total_bytes, 261
used_bytes, 261
sf_alloc_stats_t
platform_alloc.h, 1883
SF_ARENA_SIZE
alloc_kernel.c, 2410
sf_cached_latest
dictionary_management.c, 2261
sf_calloc
alloc_host.c, 2398
alloc_kernel.c, 2403, 2410
platform_alloc.h, 1885
SF_FC_BUCKETS
dictionary_heat_optimization.h, 1771
dictionary_management.c, 2237
sf_fc_cap
dictionary_heat_optimization.h, 1779
dictionary_management.c, 2261
sf_fc_clear_all
dictionary_management.c, 2238
sf_fc_count
dictionary_heat_optimization.h, 1779
dictionary_management.c, 2261
physics_benchmark_words.c, 3130
sf_fc_list
dictionary_heat_optimization.h, 1779
dictionary_management.c, 2261
physics_benchmark_words.c, 3130
sf_fc_reserve
dictionary_management.c, 2239
sf_format_timestamp
platform_time.h, 1898
sf_free
alloc_host.c, 2399
alloc_kernel.c, 2404, 2411
platform_alloc.h, 1886
sf_has_rtc
platform_time.h, 1899
sf_isspace
cmdline.c, 1069
sf_malloc
alloc_host.c, 2399
alloc_kernel.c, 2405, 2411
platform_alloc.h, 1887
sf_monotonic_ns
platform_time.h, 1900
sf_mutex, 262
handle, 262
sf_mutex_destroy
platform_lock.h, 1893
threading.c, 2420
sf_mutex_init
platform_lock.h, 1893
threading.c, 2421
sf_mutex_lock
platform_lock.h, 1894
threading.c, 2421
sf_mutex_t
platform_lock.h, 1892
sf_mutex_unlock
platform_lock.h, 1895
threading.c, 2422
sf_ns_to_ms
platform_time.h, 1901
sf_ns_to_seconds
platform_time.h, 1901
sf_ns_to_us
platform_time.h, 1901
sf_realloc

alloc_host.c, 2400
alloc_kernel.c, 2406, 2412
platform_alloc.h, 1888
sf_realtime_ns
platform_time.h, 1901
sf_rebuild_fc_index
dictionary_management.c, 2240
sf_seconds_to_ns
platform_time.h, 1902
sf_set_realtime_ns
platform_time.h, 1902
sf_strlcpy
cmdline.c, 1070
sf_strncmp
cmdline.c, 1070
sf_terminal_ready
io.c, 2291
platform_io.h, 1891
sf_time_backend
platform_init.c, 2419
platform_time.h, 1903
sf_time_backend_l4re
platform_time.h, 1903
time.c, 2414
sf_time_backend_posix
platform_time.h, 1903
time.c, 2418
sf_time_backend_t, 262
format_timestamp, 262
get_monotonic_ns, 263
get_realtime_ns, 263
has_rtc, 263
set_realtime_ns, 263
sf_time_init
platform_init.c, 2419
platform_time.h, 1902
sf_time_init_l4re
time.c, 2414
sf_time_ns_t
platform_time.h, 1898
SF_TIME_STAMP_SIZE
platform_time.h, 1898
sf_try_fast_append
dictionary_management.c, 2242
SF_VAR_BOOT_ARGS
uefi.h, 724
SF_VAR_REBOOT_TRIES
uefi.h, 724
SF_VAR_ZUSE_CERT
uefi.h, 724
sf_xrealloc
dictionary_management.c, 2244
sh_addr
Elf64_Shdr, 182
sh_addralign
Elf64_Shdr, 182
sh_entsize
Elf64_Shdr, 182
sh_flags
Elf64_Shdr, 182
sh_info
Elf64_Shdr, 182
sh_link
Elf64_Shdr, 182
sh_name
Elf64_Shdr, 182
sh_offset
Elf64_Shdr, 182
sh_size
Elf64_Shdr, 183
sh_type
Elf64_Shdr, 183
sha512
sha512.c, 1221
sha512.h, 687
sha512.c
K, 1224
rotr64, 1221
sha512, 1221
sha512_block, 1221
sha512_final, 1222
sha512_init, 1223
sha512_update, 1223
sha512.h
sha512, 687
sha512_final, 688
sha512_init, 688
sha512_update, 688
sha512_block
sha512.c, 1221
sha512_ctx_t, 264
bitlen, 264
buf, 264
buf_len, 264
state, 264
sha512_final
sha512.c, 1222
sha512.h, 688
sha512_init
sha512.c, 1223
sha512.h, 688
sha512_update
sha512.c, 1223
sha512.h, 688
should_error
TestCase, 278
SHT_DYNAMIC
elf64.h, 524
SHT_DYNSYM
elf64.h, 524
SHT_HASH
elf64.h, 524
SHT_NOBITS
elf64.h, 524
SHT_NOTE
elf64.h, 524

SHT_NULL
elf64.h, 524
SHT_PROGBITS
elf64.h, 524
SHT_REL
elf64.h, 525
SHT_RELA
elf64.h, 525
SHT_SHLIB
elf64.h, 525
SHT_STRTAB
elf64.h, 525
SHT_SYMTAB
elf64.h, 525
SIE_STIE
apic.c, 940
sig
CapsuleSigEntry, 134
sk_blk_attach_pending_t, 266
sig4_eq
ioapic.c, 905
pci.c, 1498
sig8_eq
ioapic.c, 905
pci.c, 1499
SIG_DFL
signal.h, 578
sig_handler
vm_debug.c, 2731
SIG_IGN
signal.h, 578
sig_rc
sk_blk_attach_pending_t, 266
SIGABRT
signal.h, 578
sigaction, 265
sa_flags, 265
sa_handler, 265
signal.h, 579
sighandler_t
signal.h, 578
SIGINT
signal.h, 578
signal
signal.h, 579
signal.h
SA_RESETHAND, 578
SIG_DFL, 578
SIG_IGN, 578
SIGABRT, 578
sigaction, 579
sighandler_t, 578
SIGINT, 578
signal, 579
SIGSEGV, 578
SIGTERM, 578
signal_handler
main.c, 2310
SignalEvent
_EFI_BOOT_SERVICES, 89
Signature
EFI_TABLE_HEADER, 175
signature
__attribute__, 83
SIGSEGV
signal.h, 578
SIGTERM
signal.h, 578
sin
math.h, 576
Size
EFI_FILE_INFO, 166
size
FramebufferInfo, 191
heap_block, 195
rpi5_range_t, 259
Rpi5FramebufferInfo, 260
ttf_font_t, 291
size_bytes
blkcfg_t, 116
size_dt_strings
fdt_header_t, 187
size_dt_struct
fdt_header_t, 187
size_px
ttf_raster_cache_slot_t, 296
SizeOfInfo
EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE, 169
SK_ALLOC
vm_host.h, 2097
SK_ALLOC_ALIGNED
vm_host.h, 2097
sk_alloc_fn
vm_host.h, 2098
sk_blk_attach_pending_t, 266
is_artemis, 266
pending, 266
sig, 266
sig_rc, 266
SK_BOOTSTRAP_ARENA_FAIL
parity.h, 770
SK_BOOTSTRAP_DICT_FAIL
parity.h, 770
SK_BOOTSTRAP_INIT_FAIL
parity.h, 770
SK_BOOTSTRAP_OK
parity.h, 770
sk_console_getkey
repl.c, 2445
repl.h, 1945
sk_console_identity_present
repl.c, 2446
sk_console_key_available
repl.c, 2447
repl.h, 1946

sk_console_readline
repl.c, 2448
repl.h, 1947
sk_console_user_prefix
repl.c, 2450
SK_CURSOR_BLINK_INTERVAL
repl.c, 2444
sk_cursor_show
repl.c, 2451
sk_dict_canonical_hash
parity.c, 1642
parity.h, 771
sk_dict_word_count
parity.c, 1643
parity.h, 772
SK_FREE
vm_host.h, 2097
sk_free_fn
vm_host.h, 2098
sk_host_services
vm_host.h, 2100
SK_IDLE_BEAT_INTERVAL
repl.c, 2444
sk_input_drop
kbd_input.c, 1326
kbd_input.h, 631
sk_input_getc
kbd_input.c, 1327
kbd_input.h, 632
sk_kbd_getc
kbd_input.c, 1329
sk_kbd_keypad
kbd_input.c, 1329
sk_kbd_shifted
kbd_input.c, 1331
sk_kbd_show_locks
kbd_input.c, 1330
SK_KBD_TABLE_SIZE
kbd_input.c, 1324
sk_kbd_unshifted
kbd_input.c, 1331
SK_KEY_102ND
kbd_input.c, 1324
SK_KEY_BACKSPACE
kbd_input.c, 1325
SK_KEY_CAPSLOCK
kbd_input.c, 1325
SK_KEY_ENTER
kbd_input.c, 1325
sk_key_event_poll
keyboard_words.c, 3060
keyboard_words.h, 2977
SK_KEY_KPENTER
kbd_input.c, 1325
SK_KEY_KPSLASH
kbd_input.c, 1325
SK_KEY_LEFTALT
kbd_input.c, 1325
SK_KEY_LEFTSHIFT
kbd_input.c, 1325
SK_KEY_NUMLOCK
kbd_input.c, 1325
SK_KEY_RIGHTALT
kbd_input.c, 1325
SK_KEY_RIGHTSHIFT
kbd_input.c, 1325
SK_KEY_TAB
kbd_input.c, 1326
SK_KH_BLK_ATTACH_BUF_SIZE
repl.c, 2444
SK_MUTEX_DESTROY
vm_host.h, 2097
sk_mutex_destroy_fn
vm_host.h, 2098
SK_MUTEX_INIT
vm_host.h, 2097
sk_mutex_init_fn
vm_host.h, 2098
SK_MUTEX_LOCK
vm_host.h, 2097
sk_mutex_lock_fn
vm_host.h, 2098
SK_MUTEX_UNLOCK
vm_host.h, 2097
sk_mutex_unlock_fn
vm_host.h, 2098
sk_panic_fn
vm_host.h, 2099
sk_parity_collect
parity.c, 1644
parity.h, 773
SK_PARITY_DEBUG
parity.c, 1634
vm_internal.h, 2735
sk_parity_debug_check_ptr
parity.c, 1634
sk_parity_debug_log_msg
parity.c, 1635
sk_parity_print
parity.c, 1645
parity.h, 774
sk_print_prompt
repl.c, 2452
SK_PROMPT_TEXT
repl.c, 2444
SK_PUTC
vm_host.h, 2097
sk_putc_fn
vm_host.h, 2099
SK_PUTS
vm_host.h, 2098
sk_puts_fn
vm_host.h, 2099
sk_repl
repl.c, 2452
repl.h, 1949

sk_repl_dispatch_line
repl.c, 2454
sk_repl_get_active_vm
repl.c, 2455
repl.h, 1951
sk_repl_get_artemis_usb_dev
repl.c, 2455
sk_repl_get_attached_blk_dev
repl.c, 2455
repl.h, 1951
sk_repl_get_homeblocks_dev
repl.c, 2455
repl.h, 1951
sk_repl_get_homeblocks_sig
repl.c, 2456
repl.h, 1951
sk_repl_headless_wait
repl.c, 2456
repl.h, 1952
sk_repl_idle
repl.c, 2457
sk_repl_line_calls_use
repl.c, 2460
sk_repl_register_words
repl.c, 2460
repl.h, 1953
sk_repl_run
repl.c, 2461
repl.h, 1955
sk_repl_set_active_vm
repl.c, 2463
repl.h, 1956
sk_repl_step
repl.c, 2463
repl.h, 1957
sk_time_fn
vm_host.h, 2099
SK_TIME_NS
vm_host.h, 2098
sk_usb_ctrl_t, 267
count, 267
pending, 267
slots, 268
sk_v4.c
SK_V4_CURSOR_BLINK, 1561
sk_v4_es, 1567
sk_v4_heat, 1567
sk_v4_image, 1567
sk_v4_key, 1562
SK_V4_LINE, 1561
sk_v4_node, 1567
sk_v4_out, 1563
sk_v4_post_blk_ram, 1567
SK_V4_PROMPT, 1561
sk_v4_readline, 1563
sk_v4_run, 1564
sk_v4_serve, 1566
sk_v4_text, 1567
sk_v4_words, 1567
sk_v4.h
sk_v4_run, 741
SK_V4_CAP_SECTORS
sk_v4_identity.c, 1574
SK_V4_CURSOR_BLINK
sk_v4.c, 1561
sk_v4_discard_input
sk_v4_identity.c, 1574
sk_v4_drive_arrived
sk_v4_identity.c, 1575
sk_v4_drive_left
sk_v4_identity.c, 1577
sk_v4_drive_t, 268
art, 269
art_state, 269
art_used, 269
part, 269
part_state, 269
raw, 269
used, 269
sk_v4_es
sk_v4.c, 1567
sk_v4_fixed_screen
sk_v4_identity.c, 1577
sk_v4_genesis
sk_v4_identity.c, 1578
SK_V4_GPT_PARTS
sk_v4_identity.c, 1574
sk_v4_grow.c
bam_cover_devblocks, 1569
bytes, 1571
dev_rw4k, 1569
g_hdr, 1571
g_move, 1572
meta, 1572
read_header, 1569
sk_v4_grow_artemis, 1570
sk_v4_grow_cap_sectors, 1571
sk_v4_grow.h
sk_v4_grow_artemis, 743
sk_v4_grow_cap_sectors, 743
sk_v4_grow_artemis
sk_v4_grow.c, 1570
sk_v4_grow.h, 743
sk_v4_grow_cap_sectors
sk_v4_grow.c, 1571
sk_v4_grow.h, 743
SK_V4_GROW_MIN_BYTES
sk_v4_identity.c, 1574
sk_v4_grow_stick
sk_v4_identity.c, 1579
sk_v4_heat
sk_v4.c, 1567
sk_v4_identity.c
count, 1593
g_boot_disk, 1593
g_gpt_parts, 1594

g_last_beat, 1594
g_marker, 1594
g_mint_waiting, 1594
g_offered, 1594
g_pending_real, 1594
g_proxy_fallback, 1594
g_real_drive, 1595
g_spare, 1595
g_stick_in, 1595
g_usb, 1595
g_zuse_cert_installed, 1595
g_zuse_dev, 1595
g_zuse_pubkey, 1595
g_zuse_root_known, 1596
g_zuse_root_pubkey, 1596
g_zuse_seed, 1596
g_zuse_session, 1596
SK_V4_CAP_SECTORS, 1574
sk_v4_discard_input, 1574
sk_v4_drive_arrived, 1575
sk_v4_drive_left, 1577
sk_v4_fixed_screen, 1577
sk_v4_genesis, 1578
SK_V4_GPT_PARTS, 1574
SK_V4_GROW_MIN_BYTES, 1574
sk_v4_grow_stick, 1579
sk_v4_identity_beat, 1580
sk_v4_identity_boot, 1581
sk_v4_identity_check_now, 1582
sk_v4_identity_marker, 1583
sk_v4_identity_offer_wait, 1583
sk_v4_identity_offered_present, 1584
sk_v4_identity_present, 1585
sk_v4_identity_recheck_offered, 1585
sk_v4_identity_release_offered, 1586
sk_v4_identity_set_marker, 1587
sk_v4_identity_stick_in, 1587
sk_v4_identity_wait, 1587
sk_v4_idle, 1588
SK_V4_IDLE_BEAT, 1574
sk_v4_is_this_stick, 1589
sk_v4_login, 1590
sk_v4_real_zuse_arrived, 1591
sk_v4_session_end, 1592
sk_v4_user_prefix, 1593
slots, 1596
sk_v4_identity.h
sk_v4_identity_beat, 745
sk_v4_identity_boot, 746
sk_v4_identity_check_now, 747
sk_v4_identity_marker, 748
sk_v4_identity_offer_wait, 748
sk_v4_identity_offered_present, 749
sk_v4_identity_present, 750
sk_v4_identity_recheck_offered, 750
sk_v4_identity_release_offered, 751
sk_v4_identity_set_marker, 752
sk_v4_identity_stick_in, 752
sk_v4_identity_wait, 752
sk_v4_identity_beat
sk_v4_identity.c, 1580
sk_v4_identity.h, 745
sk_v4_identity_boot
sk_v4_identity.c, 1581
sk_v4_identity.h, 746
sk_v4_identity_check_now
sk_v4_identity.c, 1582
sk_v4_identity.h, 747
sk_v4_identity_marker
sk_v4_identity.c, 1583
sk_v4_identity.h, 748
sk_v4_identity_offer_wait
sk_v4_identity.c, 1583
sk_v4_identity.h, 748
sk_v4_identity_offered_present
sk_v4_identity.c, 1584
sk_v4_identity.h, 749
sk_v4_identity_present
sk_v4_identity.c, 1585
sk_v4_identity.h, 750
sk_v4_identity_recheck_offered
sk_v4_identity.c, 1585
sk_v4_identity.h, 750
sk_v4_identity_release_offered
sk_v4_identity.c, 1586
sk_v4_identity.h, 751
sk_v4_identity_set_marker
sk_v4_identity.c, 1587
sk_v4_identity.h, 752
sk_v4_identity_stick_in
sk_v4_identity.c, 1587
sk_v4_identity.h, 752
sk_v4_identity_wait
sk_v4_identity.c, 1587
sk_v4_identity.h, 752
sk_v4_idle
sk_v4_identity.c, 1588
SK_V4_IDLE_BEAT
sk_v4_identity.c, 1574
sk_v4_image
sk_v4.c, 1567
sk_v4_is_this_stick
sk_v4_identity.c, 1589
sk_v4_key
sk_v4.c, 1562
SK_V4_LINE
sk_v4.c, 1561
sk_v4_login
sk_v4_identity.c, 1590
sk_v4_marker.c
bytes, 1599
g_buf, 1599
sk_v4_marker_read, 1597
sk_v4_marker_write, 1598
v1, 1599
v2, 1599

sk_v4_marker.h
sk_v4_marker_read, 755
sk_v4_marker_write, 755
sk_v4_marker_read
sk_v4_marker.c, 1597
sk_v4_marker.h, 755
sk_v4_marker_t, 270
has_real, 270
present, 270
proxy, 270
real, 270
sk_v4_marker_write
sk_v4_marker.c, 1598
sk_v4_marker.h, 755
sk_v4_mint.c
failed, 1601
MINT_IDENT_SECTORS, 1600
MINT_MIN_BYTES, 1600
MINT_ZERO_FBLOCKS, 1600
read_answer, 1601
say_size, 1602
sk_v4_mint_zuse, 1603
sk_v4_mint.h
sk_v4_mint_zuse, 757
sk_v4_mint_zuse
sk_v4_mint.c, 1603
sk_v4_mint.h, 757
sk_v4_node
sk_v4.c, 1567
sk_v4_out
sk_v4.c, 1563
sk_v4_post_blk_ram
sk_v4.c, 1567
SK_V4_PROMPT
sk_v4.c, 1561
sk_v4_readline
sk_v4.c, 1563
sk_v4_real_zuse_arrived
sk_v4_identity.c, 1591
sk_v4_run
sk_v4.c, 1564
sk_v4.h, 741
sk_v4_serve
sk_v4.c, 1566
sk_v4_session_end
sk_v4_identity.c, 1592
sk_v4_text
sk_v4.c, 1567
sk_v4_user_prefix
sk_v4_identity.c, 1593
sk_v4_words
sk_v4.c, 1567
sk_vm_at_outermost_interpret
vm_core.c, 1658
sk_word_blk_attach_ack
repl.c, 2464
sk_word_kh_blk_attach_send
repl.c, 2465
sk_xt_entry_owns_fn
vm_host.h, 2099
sk_xt_validate_fn
vm_host.h, 2099
skip_node_body
fdt.c, 1304
slope_fit_quality_q48
capsule_vm_physics.c, 1171
InferenceOutputs, 218
slot_by_dev
block_subsystem.c, 2205
slot_count
ttf_raster_cache_t, 297
slot_id
BlkioUsbState, 123
xhci_dev_t, 377
slot_pbn_pack_offset
block_subsystem.c, 2206
slot_pbn_to_devblock
block_subsystem.c, 2207
slots
blkio_usb.c, 1508
sk_usb_ctrl_t, 268
sk_v4_identity.c, 1596
ttf_raster_cache_t, 298
snapshot_buffers
RollingWindowOfTruth, 257
snapshot_effective_window_size
RollingWindowOfTruth, 257
snapshot_index
HeartbeatState, 200
RollingWindowOfTruth, 257
snapshot_is_warm
RollingWindowOfTruth, 257
snapshot_pending
RollingWindowOfTruth, 257
snapshot_posix
physics_runtime.c, 2395
snapshot_total_executions
RollingWindowOfTruth, 258
snapshot_window_pos
RollingWindowOfTruth, 258
snapshots
HeartbeatState, 200
snprintf
stdio.h, 590
sort_ranges_by_addr
rpi5_native_boot.c, 890
source
__attribute__, 83
log_slot_t, 223
span_var
VM, 334
SPECULATION_DEPTH
starforth_config.h, 1992
SPECULATION_THRESHOLD_Q48
starforth_config.h, 1992
speedup_factor_q48

SpeedupEstimate, 272
speedup_vs_baseline
DoeMetrics, 159
SpeedupEstimate, 271
credible_lower_95_q48, 271
credible_upper_95_q48, 271
probability_gt_10pct_q48, 271
probability_gt_double_q48, 272
speedup_factor_q48, 272
SPIN_IDLE
vm.h, 2008
sqrt
math.h, 576
sqrt_q48
math_portable.c, 2314
math_portable.h, 1815
SQRTM1_BYTES
ed25519.c, 1206
sscanf
stdio.h, 590
ssize_t
types.h, 608
ssm_apply_mode
ssm_jacquard.c, 2496
ssm_jacquard.h, 1980
ssm_apply_mode_from_table
ssm_jacquard.c, 2496
ssm_jacquard.h, 1980
SSM_CFG_L1
ssm_jacquard.h, 1978
SSM_CFG_L2
ssm_jacquard.h, 1978
SSM_CFG_L3
ssm_jacquard.h, 1979
SSM_CFG_L4
ssm_jacquard.h, 1979
SSM_CFG_L5
ssm_jacquard.h, 1979
SSM_CFG_L6
ssm_jacquard.h, 1979
SSM_CFG_L7
ssm_jacquard.h, 1979
ssm_config
VM, 334
ssm_config_t, 272
L1_heat_tracking, 272
L2_rolling_window, 272
L3_linear_decay, 273
L4_pipelining, 273
L5_window_inference, 273
L6_decay_inference, 273
L7_adaptive_heartrate, 273
SSM_CV_HIGH_THRESHOLD
starforth_config.h, 1992
SSM_ENTROPY_HIGH_THRESHOLD
starforth_config.h, 1992
SSM_HYSTERESIS_TICKS
starforth_config.h, 1992
ssm_jacquard.c
ssm_apply_mode, 2496
ssm_apply_mode_from_table, 2496
ssm_l8_force_config, 2497
ssm_l8_free_table, 2498
ssm_l8_init, 2499
ssm_l8_init_table, 2500
ssm_l8_mode_name, 2501
ssm_l8_seed_from_doe, 2502
ssm_l8_tick_score_and_select, 2503
ssm_l8_update, 2504
ssm_l8_update_table, 2505
ssm_jacquard.h
ssm_apply_mode, 1980
ssm_apply_mode_from_table, 1980
SSM_CFG_L1, 1978
SSM_CFG_L2, 1978
SSM_CFG_L3, 1979
SSM_CFG_L4, 1979
SSM_CFG_L5, 1979
SSM_CFG_L6, 1979
SSM_CFG_L7, 1979
ssm_l8_force_config, 1981
ssm_l8_free_table, 1982
ssm_l8_init, 1983
ssm_l8_init_table, 1984
ssm_l8_mode_name, 1985
ssm_l8_mode_t, 1979
ssm_l8_update, 1986
ssm_l8_update_table, 1987
SSM_MODE_C0, 1979
SSM_MODE_C1, 1979
SSM_MODE_C10, 1979
SSM_MODE_C11, 1979
SSM_MODE_C12, 1979
SSM_MODE_C13, 1979
SSM_MODE_C14, 1979
SSM_MODE_C15, 1979
SSM_MODE_C2, 1979
SSM_MODE_C3, 1979
SSM_MODE_C4, 1979
SSM_MODE_C5, 1979
SSM_MODE_C6, 1979
SSM_MODE_C7, 1979
SSM_MODE_C8, 1979
SSM_MODE_C9, 1979
ssm_l8_force_config
ssm_jacquard.c, 2497
ssm_jacquard.h, 1981
ssm_l8_free_table
ssm_jacquard.c, 2498
ssm_jacquard.h, 1982
ssm_l8_init
ssm_jacquard.c, 2499
ssm_jacquard.h, 1983
ssm_l8_init_table
ssm_jacquard.c, 2500
ssm_jacquard.h, 1984

ssm_l8_metrics_t, 274
cv, 274
entropy, 274
inference_early_exited, 274
inference_ran_this_tick, 274
stability_score, 274
temporal_decay, 274
ssm_l8_mode_name
ssm_jacquard.c, 2501
ssm_jacquard.h, 1985
ssm_l8_mode_t
ssm_jacquard.h, 1979
ssm_l8_seed_from_doe
ssm_jacquard.c, 2502
ssm_l8_state
VM, 334
ssm_l8_state_t, 275
current_mode, 275
hysteresis_counter, 275
pending_mode, 276
table, 276
ssm_l8_tick_score_and_select
ssm_jacquard.c, 2503
ssm_l8_update
ssm_jacquard.c, 2504
ssm_jacquard.h, 1986
ssm_l8_update_table
ssm_jacquard.c, 2505
ssm_jacquard.h, 1987
SSM_MODE_C0
ssm_jacquard.h, 1979
SSM_MODE_C1
ssm_jacquard.h, 1979
SSM_MODE_C10
ssm_jacquard.h, 1979
SSM_MODE_C11
ssm_jacquard.h, 1979
SSM_MODE_C12
ssm_jacquard.h, 1979
SSM_MODE_C13
ssm_jacquard.h, 1979
SSM_MODE_C14
ssm_jacquard.h, 1979
SSM_MODE_C15
ssm_jacquard.h, 1979
SSM_MODE_C2
ssm_jacquard.h, 1979
SSM_MODE_C3
ssm_jacquard.h, 1979
SSM_MODE_C4
ssm_jacquard.h, 1979
SSM_MODE_C5
ssm_jacquard.h, 1979
SSM_MODE_C6
ssm_jacquard.h, 1979
SSM_MODE_C7
ssm_jacquard.h, 1979
SSM_MODE_C8
ssm_jacquard.h, 1979
SSM_MODE_C9
ssm_jacquard.h, 1979
SSM_TEMPORAL_DECAY_LOW_THRESHOLD
starforth_config.h, 1992
SSM_TEMPORAL_DECAY_THRESHOLD
starforth_config.h, 1992
ST_CSI
vt100.c, 1360
ST_ESC
vt100.c, 1360
ST_ESC_CHAR
vt100.c, 1360
ST_ESC_HASH
vt100.c, 1360
st_info
Elf64_Sym, 183
st_name
Elf64_Sym, 183
ST_NORMAL
vt100.c, 1360
st_other
Elf64_Sym, 183
st_shndx
Elf64_Sym, 184
st_size
Elf64_Sym, 184
st_value
Elf64_Sym, 184
stability_score
ssm_l8_metrics_t, 274
stack
ExecResult, 185
Stack Operations, 25
vm_add_check_overflow, 27
vm_cpuid, 27
vm_divmod, 28
vm_has_sse42, 28
vm_max_asm, 29
vm_memcpy_asm, 29
vm_memzero_asm, 31
vm_min_asm, 31
vm_mul_double, 31
vm_pop_asm, 32
vm_push_asm, 33
vm_rpop_asm, 35
vm_rpush_asm, 36
vm_strcmp_asm, 37
vm_sub_check_overflow, 37
stack_management.c
vm_pop, 2507
vm_push, 2509
vm_rpop, 2510
vm_rpush, 2511
stack_operations
Profiler, 250
STACK_SIZE
vm.h, 2008

stack_size
KernelArgs, 219
stack_word_depth
stack_words.c, 3187
stack_word_drop
stack_words.c, 3188
stack_word_dup
stack_words.c, 3189
stack_word_minus_rot
stack_words.c, 3190
stack_word_over
stack_words.c, 3190
stack_word_pick
stack_words.c, 3191
stack_word_question_dup
stack_words.c, 3192
stack_word_roll
stack_words.c, 3193
stack_word_rot
stack_words.c, 3193
stack_word_suites
stack_words_test.c, 2640
stack_word_swap
stack_words.c, 3194
stack_words.c
register_stack_words, 3187
stack_word_depth, 3187
stack_word_drop, 3188
stack_word_dup, 3189
stack_word_minus_rot, 3190
stack_word_over, 3190
stack_word_pick, 3191
stack_word_question_dup, 3192
stack_word_roll, 3193
stack_word_rot, 3193
stack_word_swap, 3194
stack_words.h
register_stack_words, 3006
stack_words_test.c
run_stack_words_tests, 2639
stack_word_suites, 2640
STADIUM_BLOCK_COOL_RATE_Q48
starforth_config.h, 1992
STADIUM_BLOCK_HEAT_QUANTUM
starforth_config.h, 1992
STADIUM_BLOCK_TRACK_CAP_MULT
starforth_config.h, 1993
STADIUM_CAPACITY_TICK
starforth_config.h, 1993
STADIUM_CONTAINS_DEPTH_MAX
starforth_config.h, 1993
STADIUM_MEMORY_PERCENT
starforth_config.h, 1993
stadium_patron_cell
VMRegistryEntry, 347
STADIUM_VM_MEMORY_PERCENT
starforth_config.h, 1993
STADIUM_WORD_COOL_RATE_Q48
starforth_config.h, 1993
STADIUM_WORD_HEAT_QUANTUM
starforth_config.h, 1993
stale_word_count
DoeMetrics, 159
HeartbeatSnapshot, 197
InferenceInputs, 216
stale_word_count_at_check
VM, 334
stale_word_ratio
DoeMetrics, 159
Stall
_EFI_BOOT_SERVICES, 90
stall_dci
xhci_dev_t, 377
stall_ep_addr
xhci_dev_t, 377
stall_retry_action
xhci_dev_t, 377
StandardErrorHandle
_EFI_SYSTEM_TABLE, 100
STARFORTH_ARCH
version.h, 2000
STARFORTH_CHECK_ARENA
starforth_words.c, 3196
starforth_config.h
ADAPTIVE_CHECK_FREQUENCY, 1990
ADAPTIVE_GROWTH_THRESHOLD, 1990
ADAPTIVE_MIN_WINDOW_SIZE, 1990
ADAPTIVE_SHRINK_RATE, 1990
DECAY_MIN_INTERVAL, 1990
DECAY_RATE_PER_US_Q16, 1990
ENABLE_HOTWORDS_CACHE, 1990
ENABLE_PIPELINING, 1990
HEARTBEAT_CHECK_FREQUENCY, 1991
HEARTBEAT_INFERENCE_FREQUENCY, 1991
HEARTBEAT_SLOPE_VALIDATION_FREQUENCY, 1991
HEARTBEAT_THREAD_ENABLED, 1991
HEARTBEAT_TICK_NS, 1991
HEARTBEAT_WINDOW_TUNING_FREQUENCY, 1991
INITIAL_DECAY_SLOPE_Q48, 1991
MIN_SAMPLES_FOR_SPECULATION, 1991
MINIMUM_PREFETCH_ROI, 1991
MISPREDICTION_COST_Q48, 1991
ROLLING_WINDOW_SIZE, 1992
SPECULATION_DEPTH, 1992
SPECULATION_THRESHOLD_Q48, 1992
SSM_CV_HIGH_THRESHOLD, 1992
SSM_ENTROPY_HIGH_THRESHOLD, 1992
SSM_HYSTERESIS_TICKS, 1992
SSM_TEMPORAL_DECAY_LOW_THRESHOLD, 1992
SSM_TEMPORAL_DECAY_THRESHOLD, 1992
STADIUM_BLOCK_COOL_RATE_Q48, 1992
STADIUM_BLOCK_HEAT_QUANTUM, 1992
STADIUM_BLOCK_TRACK_CAP_MULT, 1993

STADIUM_CAPACITY_TICK, 1993
STADIUM_CONTAINS_DEPTH_MAX, 1993
STADIUM_MEMORY_PERCENT, 1993
STADIUM_VM_MEMORY_PERCENT, 1993
STADIUM_WORD_COOL_RATE_Q48, 1993
STADIUM_WORD_HEAT_QUANTUM, 1993
STARFORTH_CONFIG_ADAPTIVE_CHECK_FREQUENCY_DEFAULT, 1993
STARFORTH_CONFIG_ADAPTIVE_GROWTH_THRESHOLD_DEFAULT, 1993
STARFORTH_CONFIG_ADAPTIVE_MIN_WINDOW_SIZE_DEFAULT, 1993
STARFORTH_CONFIG_ADAPTIVE_SHRINK_RATE_DEFAULT, 1994
STARFORTH_CONFIG_DECAY_MIN_INTERVAL_DEFAULT, 1994
STARFORTH_CONFIG_DECAY_RATE_PER_US_Q16_DEFAULT, 1994
STARFORTH_CONFIG_ENABLE_HOTWORDS_CACHE_DEFAULT, 1994
STARFORTH_CONFIG_ENABLE_PIPELINING_DEFAULT, 1994
STARFORTH_CONFIG_HEARTBEAT_CHECK_FREQUENCY_DEFAULT, 1994
STARFORTH_CONFIG_HEARTBEAT_INFERENCE_FREQUENCY_DEFAULT, 1994
STARFORTH_CONFIG_HEARTBEAT_SLOPE_VALIDATION_FREQUENCY_DEFAULT, 1994
STARFORTH_CONFIG_HEARTBEAT_THREAD_ENABLED_DEFAULT, 1994
STARFORTH_CONFIG_HEARTBEAT_TICK_NS_DEFAULT, 1994
STARFORTH_CONFIG_HEARTBEAT_WINDOW_TUNING_FREQUENCY_DEFAULT, 1995
STARFORTH_CONFIG_INITIAL_DECAY_SLOPE_Q48_DEFAULT, 1995
STARFORTH_CONFIG_MIN_SAMPLES_FOR_SPECULATION_DEFAULT, 1995
STARFORTH_CONFIG_MINIMUM_PREFETCH_ROI_DEFAULT, 1995
STARFORTH_CONFIG_MISPREDICTION_COST_Q48_DEFAULT, 1995
STARFORTH_CONFIG_ROLLING_WINDOW_SIZE_DEFAULT, 1995
STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT, 1995
STARFORTH_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT, 1995
STARFORTH_CONFIG_SSM_CV_HIGH_THRESHOLD_DEFAULT, 1995
STARFORTH_CONFIG_SSM_ENTROPY_HIGH_THRESHOLD_DEFAULT, 1995
STARFORTH_CONFIG_SSM_HYSTERESIS_TICKS_DEFAULT, 1996
STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_LOW_THRESHOLD_DEFAULT, 1996
STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_THRESHOLD_DEFAULT, 1996
STARFORTH_CONFIG_STADIUM_BLOCK_COOL_RATE_Q48_DEF 1996
STARFORTH_CONFIG_STADIUM_BLOCK_HEAT_QUANTUM_DEF 1996
STARFORTH_CONFIG_STADIUM_BLOCK_TRACK_CAP_MULT_DE 1996
STARFORTH_CONFIG_STADIUM_CAPACITY_TICK_DEFAULT, 1996
STARFORTH_CONFIG_STADIUM_CONTAINS_DEPTH_MAX_DEFA 1996
STARFORTH_CONFIG_STADIUM_MEMORY_PERCENT_DEFAULT 1996
STARFORTH_CONFIG_STADIUM_VM_MEMORY_PERCENT_DEFA 1996
STARFORTH_CONFIG_STADIUM_WORD_COOL_RATE_Q48_DEF 1997
STARFORTH_CONFIG_STADIUM_WORD_HEAT_QUANTUM_DEFA 1997
STARFORTH_CONFIG_STRICT_PTR_DEFAULT, 1997
STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT, 1997
STRICT_PTR_DEFAULT, 1997
TRANSITION_WINDOW_SIZE, 1997
VM_STRICT_DEFAULT, 1997
starforth_config.h, 1993
starforth_config.h, 1994
starforth_config.h, 1995

starforth_config.h, 1995
STARFORTH_CONFIG_MISPREDICTION_COST_Q48_DEFAULT
starforth_config.h, 1995
STARFORTH_CONFIG_ROLLING_WINDOW_SIZE_DEFAULT
starforth_config.h, 1995
STARFORTH_CONFIG_SPECULATION_DEPTH_DEFAULT
starforth_config.h, 1995
STARFORTH_CONFIG_SPECULATION_THRESHOLD_Q48_DEFAULT
starforth_config.h, 1995
STARFORTH_CONFIG_SSM_CV_HIGH_THRESHOLD_DEFAULT
starforth_config.h, 1995
STARFORTH_CONFIG_SSM_ENTROPY_HIGH_THRESHOLD_DEFAULT
starforth_config.h, 1995
STARFORTH_CONFIG_SSM_HYSTERESIS_TICKS_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_LOW_THRESHOLD_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_SSM_TEMPORAL_DECAY_THRESHOLD_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_STADIUM_BLOCK_COOL_RATE_Q48_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_STADIUM_BLOCK_HEAT_QUANTUM_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_STADIUM_BLOCK_TRACK_CAP_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_STADIUM_CAPACITY_TICK_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_STADIUM_CONTAINS_DEPTH_MAX_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_STADIUM_MEMORY_PERCENT_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_STADIUM_VM_MEMORY_PERCENT_DEFAULT
starforth_config.h, 1996
STARFORTH_CONFIG_STADIUM_WORD_COOL_RATE_Q48_DEFAULT
starforth_config.h, 1997
STARFORTH_CONFIG_STADIUM_WORD_HEAT_QUANTUM_DEFAULT
starforth_config.h, 1997
STARFORTH_CONFIG_STRICT_PTR_DEFAULT
starforth_config.h, 1997
STARFORTH_CONFIG_TRANSITION_WINDOW_SIZE_DEFAULT
starforth_config.h, 1997
STARFORTH_STATE_BYTES
main.c, 2302
STARFORTH_TARGET
version.h, 2000
STARFORTH_TIMESTAMP
version.h, 2000
STARFORTH_VENDOR_GUID
uefi.h, 724
STARFORTH_VERSION
version.h, 2001
STARFORTH_VERSION_FULL
version.h, 2001
starforth_word_entropy_fetch
starforth_words.h, 3011
starforth_word_entropy_store
starforth_words.h, 3011
starforth_word_execution_heat_fetch
starforth_words.c, 3199
starforth_word_execution_heat_store
starforth_words.c, 3199
starforth_word_heartbeat_ticks
starforth_words.c, 3200
starforth_word_init
starforth_words.c, 3201
starforth_word_paren_dash
starforth_words.c, 3202
starforth_word_random
starforth_words.c, 3203
starforth_word_reset_entropy
starforth_words.h, 3013
starforth_word_reset_execution_heat
starforth_words.c, 3204
starforth_word_seed
starforth_words.c, 3204
starforth_word_suites
starforth_words_test.c, 2642
starforth_word_top_words
starforth_words.c, 3206
starforth_word_version
starforth_words.c, 3207
starforth_word_wait
starforth_words.c, 3208
starforth_word_zuse_authenticate
starforth_words.c, 3209
starforth_word_zuse_cert_installed_query
starforth_words.c, 3209
starforth_word_zuse_pubkey_fetch
starforth_words.c, 3210
starforth_word_zuse_session
starforth_words.c, 3210
starforth_words.c
g_prng_state, 3211
prng_next, 3197
register_starforth_words, 3197
STARFORTH_CHECK_ARENA, 3196
starforth_word_execution_heat_fetch, 3199
starforth_word_execution_heat_store, 3199
starforth_word_heartbeat_ticks, 3200
starforth_word_init, 3201
starforth_word_paren_dash, 3202
starforth_word_random, 3203
starforth_word_reset_execution_heat, 3204
starforth_word_seed, 3204
starforth_word_top_words, 3206
starforth_word_version, 3207

starforth_word_wait, 3208
starforth_word_word_execution_heat, 3208
starforth_word_zuse_authenticate, 3209
starforth_word_zuse_cert_installed_query, 3209
starforth_word_zuse_pubkey_fetch, 3210
starforth_word_zuse_session, 3210
starforth_words.h
register_starforth_words, 3009
starforth_word_entropy_fetch, 3011
starforth_word_entropy_store, 3011
starforth_word_init, 3011
starforth_word_paren_dash, 3012
starforth_word_random, 3013
starforth_word_reset_entropy, 3014
starforth_word_seed, 3014
starforth_word_top_words, 3015
starforth_word_wait, 3016
starforth_word_word_entropy, 3016
starforth_words_test.c
run_starforth_words_tests, 2641
starforth_word_suites, 2642
start_bus
__attribute__, 83
start_lbn
blk_dev_slot, 107
start_time
ProfileTimer, 252
started_ns
CapsuleRunRecord, 133
StartImage
_EFI_BOOT_SERVICES, 90
startup.h
startup_blkio_cleanup, 1998
startup_blkio_init, 1998
startup_block_subsys_init, 1999
startup_vm_init, 1999
startup_blkio_cleanup
startup.h, 1998
startup_blkio_init
startup.h, 1998
startup_block_subsys_init
startup.h, 1999
startup_vm_init
startup.h, 1999
state
blkio_dev, 117
sha512_ctx_t, 264
VMRegistryEntry, 347
VT100State, 353
state_addr
VM, 335
state_flags
DictPhysics, 151
state_var
VM, 335
state_variable
dictionary_manipulation_words.c, 2879
static_assert
assert.h, 564
stats
HotwordsCache_s, 208
status
xhci_trb_t, 392
status_buf
VirtBlkState, 306
stddev_ns_q48
BayesianLatencyPosterior, 104
StdErr
_EFI_SYSTEM_TABLE, 100
stderr
stdio.h, 581
stdin
stdio.h, 581
stdio.h
fclose, 581
fflush, 581
fgetc, 582
fgets, 582
fopen, 583
fprintf, 583
fputc, 584
fputs, 585
fread, 585
fscanf, 585
fseek, 586
ftell, 586
fwrite, 586
getchar, 587
printf, 587
putchar, 588
puts, 589
rewind, 589
SEEK_CUR, 580
SEEK_END, 580
SEEK_SET, 580
snprintf, 590
sscanf, 590
stderr, 581
stdin, 581
stdout, 581
vfprintf, 590
vsnprintf, 591
stdlib.h
abort, 593
atoi, 593
atol, 593
calloc, 593
free, 594
malloc, 595
qsort, 596
realloc, 596
strtol, 597
strtoll, 598
strtoul, 598
stdout
stdio.h, 581

stop_requested
HeartbeatWorker, 204
str_eq
fdt.c, 1305
strchr
string.h, 601
strcmp
string.h, 602
strcpy
string.h, 602
strerror
string.h, 602
stress_suites
stress_tests.c, 2645
stress_tests.c
run_stress_tests, 2644
stress_suites, 2645
STRICT_PTR
starforth_config.h, 1997
stride
FbState, 186
string.h
memchr, 599
memcmp, 599
memcpy, 600
memmove, 601
memset, 601
strchr, 601
strcmp, 602
strcpy, 602
strerror, 602
strlen, 602
strncmp, 603
strncpy, 604
strrchr, 604
string_runtime_s_quote
string_words.c, 3217
string_word_bl
string_words.c, 3217
string_word_blank
string_words.c, 3217
string_word_bracket_literal
string_words.c, 3218
string_word_bracket_tick
string_words.c, 3219
string_word_cmove
string_words.c, 3219
string_word_cmove_greater
string_words.c, 3220
string_word_compare
string_words.c, 3221
string_word_convert
string_words.c, 3221
string_word_count
string_words.c, 3222
string_word_enclose
string_words.c, 3223
string_word_expect
string_words.c, 3224
string_word_literal
string_words.c, 3224
string_word_minus_trailing
string_words.c, 3224
string_word_number
string_words.c, 3225
string_word_query
string_words.c, 3226
string_word_s_quote
string_words.c, 3226
string_word_scan
string_words.c, 3227
string_word_search
string_words.c, 3228
string_word_skip
string_words.c, 3228
string_word_source
string_words.c, 3229
string_word_span
string_words.c, 3229
string_word_suites
string_words_test.c, 2647
string_word_tib
string_words.c, 3230
string_word_to_in
string_words.c, 3230
string_word_word
string_words.c, 3231
string_words.c
convert_string_to_number, 3213
ensure_input, 3213
register_string_words, 3214
string_runtime_s_quote, 3217
string_word_bl, 3217
string_word_blank, 3217
string_word_bracket_literal, 3218
string_word_bracket_tick, 3219
string_word_cmove, 3219
string_word_cmove_greater, 3220
string_word_compare, 3221
string_word_convert, 3221
string_word_count, 3222
string_word_enclose, 3223
string_word_expect, 3224
string_word_literal, 3224
string_word_minus_trailing, 3224
string_word_number, 3225
string_word_query, 3226
string_word_s_quote, 3226
string_word_scan, 3227
string_word_search, 3228
string_word_skip, 3228
string_word_source, 3229
string_word_span, 3229
string_word_tib, 3230
string_word_to_in, 3230
string_word_word, 3231

vaddr_from_ptr, 3232
word_scratch_addr, 3232
WORD_SCRATCH_CAP, 3213
string_words.h
register_string_words, 3018
string_words_test.c
run_string_words_tests, 2646
string_word_suites, 2647
strlen
string.h, 602
strncmp
string.h, 603
strncpy
string.h, 604
strrchr
string.h, 604
strtol
stdlib.h, 597
strtoll
stdlib.h, 598
strtoul
stdlib.h, 598
sub_L
scalar25519.c, 1219
SubType
EFI_DEVICE_PATH_PROTOCOL, 164
suggested_next_size
PipelineGlobalMetrics, 246
suite_contract
WordTestSuite, 359
summary
physics_analytics_heap, 235
summary_bytes
physics_analytics_header, 232
physics_analytics_heap, 235
summary_offset
physics_analytics_header, 232
suseconds_t
time.h, 606
switch_slot
wirebind_live_entry_t, 354
system
KernelArgs, 219
system_is_running
system_words.c, 3238
system_running
system_words.c, 3247
system_word_79_standard
system_words.c, 3238
system_word_abort
system_words.c, 3238
system_word_abort_quote
system_words.c, 3239
system_word_bye
system_words.c, 3240
system_word_cold
system_words.c, 3240
system_word_execute
system_words.c, 3241
system_word_nop
system_words.c, 3241
system_word_page
system_words.c, 3242
system_word_quit
system_words.c, 3242
system_word_reboot
system_words.c, 3243
system_word_save_system
system_words.c, 3243
system_word_see
system_words.c, 3244
system_word_suites
system_words_test.c, 2649
system_word_vlist
system_words.c, 3245
system_word_warm
system_words.c, 3246
system_word_words
system_words.c, 3247
system_words.c
forth_79_standard, 3247
forth_backslash, 3234
forth_paren, 3234
print_word_name, 3234
register_system_words, 3235
reset_vm_state, 3237
runtime_abortq, 3237
set_forth_79_compliance, 3237
system_is_running, 3238
system_running, 3247
system_word_79_standard, 3238
system_word_abort, 3238
system_word_abort_quote, 3239
system_word_bye, 3240
system_word_cold, 3240
system_word_execute, 3241
system_word_nop, 3241
system_word_page, 3242
system_word_quit, 3242
system_word_reboot, 3243
system_word_save_system, 3243
system_word_see, 3244
system_word_vlist, 3245
system_word_warm, 3246
system_word_words, 3247
system_words.h
register_system_words, 3021
system_words_test.c
run_system_words_tests, 2648
system_word_suites, 2649
SystemTable
EFI_LOADED_IMAGE_PROTOCOL, 172

table
ssm_l8_state_t, 276
TAG
ttf.c, 1336

tag
cf_item_t, 141
TAG_ALLOCATE_BUFFER
rpi5_mailbox.c, 862
TAG_END
rpi5_mailbox.c, 863
TAG_GET_PITCH
rpi5_mailbox.c, 863
TAG_SET_DEPTH
rpi5_mailbox.c, 863
TAG_SET_PHYS_SIZE
rpi5_mailbox.c, 863
TAG_SET_PIXEL_ORDER
rpi5_mailbox.c, 863
TAG_SET_VIRT_SIZE
rpi5_mailbox.c, 863
TAG_SET_VIRTUAL_OFFS
rpi5_mailbox.c, 863
tail_slot
log_region_ctrl_t, 222
temperature_from_execution_heat
physics_metadata.c, 2361
temperature_q8
DictPhysics, 151
physics_seed_definition, 244
temporal_decay
ssm_l8_metrics_t, 274
test_acl_adaptive_ttl
acl_words_test.c, 2582
test_acl_deny_aborts
acl_words_test.c, 2584
test_acl_inherit
acl_words_test.c, 2585
test_acl_inherit_word_dispatch
acl_words_test.c, 2586
test_acl_init_primitives
acl_words_test.c, 2588
test_acl_pin_blocks_shadow
acl_words_test.c, 2589
test_acl_pin_immutable
acl_words_test.c, 2590
test_acl_read_accessors
acl_words_test.c, 2592
test_acl_strict_mode
acl_words_test.c, 2594
test_acl_ttl_hot_path
acl_words_test.c, 2596
test_common.c
assert_stack_depth, 2653
assert_stack_top, 2654
assert_vm_error, 2654
DIE, 2653
dump_and_die, 2655
fail_and_exit, 2656
fail_fast, 2673
print_module_summary, 2657
resolve_contract, 2659
restore_dict_state, 2660
restore_vm_state, 2661
run_single_test, 2663
run_test_suite, 2666
run_test_suite_m, 2668
save_dict_state, 2670
save_vm_state, 2671
test_fatal_exit, 2653
test_common.h
assert_stack_depth, 2515
assert_stack_top, 2515
assert_vm_error, 2516
fail_fast, 2533
MAX_TEST_INPUT, 2514
MAX_TEST_OUTPUT, 2514
MAX_TESTS_PER_WORD, 2514
print_module_summary, 2516
restore_dict_state, 2518
restore_vm_state, 2521
run_single_test, 2523
run_test_suite, 2526
run_test_suite_m, 2528
save_dict_state, 2530
save_vm_state, 2531
TEST_EDGE_CASE, 2514
TEST_ERROR_CASE, 2514
TEST_NORMAL, 2514
TestType, 2514
test_contracts.c
assert_heartbeat_safe, 2675
capture_exec_result, 2676
check_physics_transparent, 2677
contract_get_violation_count, 2679
contract_rand, 2679
contract_reset_violation_count, 2680
exec_results_equal, 2680
global_contract_violations, 2685
perturb_physics, 2681
restore_physics, 2683
rng_seed, 2684
rng_state, 2685
save_physics, 2684
test_contracts.h
assert_heartbeat_safe, 2536
capture_exec_result, 2537
check_physics_transparent, 2538
CONTRACT_DEFAULT_FUZZ_ROUNDS, 2535
contract_flags_t, 2536
contract_get_violation_count, 2540
CONTRACT_HEARTBEAT_SAFE, 2535
CONTRACT_NONE, 2535
CONTRACT_PHYSICS_TRANSPARENT, 2535
contract_reset_violation_count, 2540
exec_results_equal, 2541
global_contract_violations, 2545
perturb_physics, 2542
restore_physics, 2543
save_physics, 2544
test_count

WordTestSuite, 359
TEST_EDGE_CASE
test_common.h, 2514
test_emergency_console_bypass
acl_words_test.c, 2597
test_end_time
break_me_tests.c, 2606
TEST_ERROR
log.h, 1805, 1808
TEST_ERROR_CASE
test_common.h, 2514
TEST_FAIL
log.h, 1805, 1808
test_fail_color
log.c, 2300
test_fatal_exit
test_common.c, 2653
test_modules
test_runner.c, 2693
TEST_NORMAL
test_common.h, 2514
TEST_PASS
log.h, 1805, 1808
test_pass_color
log.c, 2300
test_runner.c
benchmark_iterations, 2692
benchmark_mode, 2692
enable_benchmark_mode, 2687
get_time_ns, 2687
global_test_stats, 2693
run_all_tests, 2688
run_module_tests, 2689
run_word_tests, 2690
test_modules, 2693
test_stats_accumulate, 2691
test_runner.h
enable_benchmark_mode, 2548
global_test_stats, 2579
run_acl_words_tests, 2548
run_all_tests, 2550
run_arithmetic_words_tests, 2551
run_block_words_tests, 2552
run_break_me_tests, 2553
run_control_words_tests, 2554
run_defining_words_tests, 2555
run_dictionary_manipulation_words_tests, 2556
run_dictionary_words_tests, 2557
run_double_words_tests, 2558
run_format_words_tests, 2559
run_inference_words_tests, 2560
run_integration_tests, 2561
run_io_words_tests, 2562
run_logical_words_tests, 2563
run_mama_forth_words_tests, 2564
run_memory_words_tests, 2565
run_mixed_arithmetic_words_tests, 2566
run_module_tests, 2567
run_physics_freeze_words_tests, 2568
run_q48_words_tests, 2569
run_return_stack_words_tests, 2570
run_stack_words_tests, 2571
run_starforth_words_tests, 2572
run_stress_tests, 2573
run_string_words_tests, 2574
run_system_words_tests, 2575
run_vocabulary_words_tests, 2576
run_word_tests, 2577
test_stats_accumulate, 2578
TEST_SKIP
log.h, 1805, 1808
test_skip_color
log.c, 2300
test_start_time
break_me_tests.c, 2606
test_stats_accumulate
test_runner.c, 2691
test_runner.h, 2578
TestCase, 276
contract, 277
expected, 277
implemented, 277
input, 277
name, 278
should_error, 278
type, 278
TestModule, 278
module_name, 279
reserved, 279
reserved_int, 279
run_module_tests, 279
TestResult
log.h, 1805, 1807
tests
WordTestSuite, 360
tests_errors
ParityPacket, 226
tests_failed
acl_words_test.c, 2599
ParityPacket, 226
tests_passed
acl_words_test.c, 2599
ParityPacket, 226
tests_skipped
ParityPacket, 226
tests_total
ParityPacket, 226
TestStats, 280
total_error, 280
total_fail, 280
total_pass, 280
total_skip, 280
total_tests, 281
TestString
_EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 98
TestType

test_common.h, 2514
threading.c
sf_mutex_destroy, 2420
sf_mutex_init, 2421
sf_mutex_lock, 2421
sf_mutex_unlock, 2422
tib_buf
VM, 335
tib_cap
VM, 335
tick_buffer
HeartbeatState, 200
tick_buffer_size
HeartbeatState, 200
tick_buffer_write_index
HeartbeatState, 201
tick_count
CDConfigEntry, 135
DoeMetrics, 159
HeartbeatState, 201
tick_count_total
HeartbeatState, 201
tick_interval_ns
HeartbeatTickSnapshot, 203
tick_ns
HeartbeatWorker, 204
tick_number
HeartbeatTickSnapshot, 203
tick_number_offset
HeartbeatState, 201
tick_target_ns
DoeMetrics, 159
HeartbeatState, 201
PhysicsSnapshot, 245
ticks
time_trust_state, 282
time.c
_POSIX_C_SOURCE, 2415
posix_format_timestamp, 2416
posix_get_monotonic_ns, 2416
posix_get_realtime_ns, 2416
posix_has_rtc, 2417
posix_set_realtime_ns, 2417
sf_time_backend_l4re, 2414
sf_time_backend_posix, 2418
sf_time_init_l4re, 2414
time.h
gettimeofday, 606
nanosleep, 607
suseconds_t, 606
time_t, 606
time_t
time.h, 606
TIME_TRUST_HIGH
timer.h, 691
TIME_TRUST_LOW
timer.h, 691
time_trust_state, 281
expected_delta, 282
last_tsc, 282
ticks, 282
total_samples, 282
trust, 282
variance, 282
window, 282
time_trust_t
timer.h, 692
time_window, 283
count, 283
deltas, 283
pos, 283
TIME_WINDOW_SIZE
timer.h, 691
timer.c
bytes_contain, 1053
calib_record, 1050
calibrate_tsc_with_pmtimer, 1017
calibrate_window_hpet, 1018
calibrate_window_pit, 1019
cntfrq_read, 1007
cntpct_read, 1007
converge_tsc_bare_metal, 1020
cpu_has_invariant_tsc, 1021
cpu_has_rdtscp, 1022
cpuid, 1023
CV_EPSILON_PPM, 1015
CV_STABLE_COUNT, 1015
derive_tsc_hz_from_cpuid, 1024
DRIFT_EPSILON_PPM, 1015
enable_hpet, 1025
fadt_find_pm_tmr_port, 1025
heartbeat_read_counter, 1008, 1026, 1054
hpet_freq_hz, 1050
HPET_GEN_CAP_ID, 1015
HPET_GEN_CONFIG, 1015
HPET_MAIN_COUNTER, 1015
HPET_PHYS_BASE, 1015
hpet_read, 1027
hpet_regs, 1050
HPET_VIRT_BASE, 1016
hpet_write, 1028
inb, 1028
init_timer_vm_mode, 1030
inl, 1031
muldiv64, 1032
outb, 1033
pit_available, 1050
PIT_CHANNEL0_DATA, 1016
PIT_COMMAND, 1016
PIT_FREQ_HZ, 1016
pit_probe_running, 1033
pit_program_reload, 1034
pit_read_counter16, 1035
pit_tsc_hz_mean, 1051
pmtimer_delta, 1036
PMTIMER_FREQ_HZ, 1016

PMTIMER_IO_PORT_DEFAULT, 1016
PMTIMER_MASK, 1016
pmtimer_port, 1051
pmtimer_read, 1037
PMTIMER_WIDTH_BITS, 1016
print_dec, 1037, 1055
rdtime, 1055
rdtsc, 1038
rdtscp_aux, 1039
RISCV_TIMEBASE_HZ_FALLBACK, 1053
running_under_hypervisor, 1008, 1040, 1056
s_base_count, 1011, 1060
s_base_ns, 1011, 1060
s_cal, 1011, 1061
s_counter_hz, 1011, 1061
s_ns_per_tick, 1011, 1061
SAMPLE_WINDOWS, 1016
sanity_check_hpet_running_quick, 1040
timer_calibration_record, 1009, 1041, 1057
timer_check_drift_now, 1009, 1041, 1057
timer_drift_check_bare_metal, 1042
timer_fatal, 1043
timer_init, 1010, 1044, 1057
timer_now_ns, 1010, 1045, 1059
timer_now_ns_vm_relative, 1046
TIMER_REQUIRE_PIT, 1016
timer_sig4_eq, 1047
timer_sig8_eq, 1048
timer_tsc_hz, 1010, 1048, 1060
TIMER_VM_STRICT_INVARIANT_TSC, 1017
tsc_hz_locked, 1051
update_stats, 1048
vm_ns_base, 1051
vm_pm_start, 1051
vm_timebase_init_from_pmtimer, 1049
WARMUP_WINDOWS, 1017
WINDOW_TICKS_HPET, 1017
WINDOW_TICKS_PIT, 1017
windows_used, 1051
timer.h
heartbeat_init, 693
heartbeat_next_period_ns, 693
heartbeat_read_counter, 694
heartbeat_service, 695
heartbeat_set_adaptive_period_ns, 696
heartbeat_state, 697
heartbeat_tick, 697
heartbeat_ticks, 698
heartbeat_trust, 699
TIME_TRUST_HIGH, 691
TIME_TRUST_LOW, 691
time_trust_t, 692
TIME_WINDOW_SIZE, 691
timer_calibration_record, 699
timer_calibration_record_t, 692
timer_check_drift_now, 700
timer_init, 701
timer_now_ns, 704
TIMER_TRUST_ABSOLUTE, 693
timer_trust_level, 692
timer_trust_level_t, 692
TIMER_TRUST_NONE, 693
TIMER_TRUST_RELATIVE, 693
timer_tsc_hz, 705
TimeTrustState, 692
TimeWindow, 692
timer_calibration_record, 283
converged, 284
cv_hpet_ppm, 284
cv_pit_ppm, 284
diff_ppm, 284
hpet_hz, 284
pit_hz_mean, 284
reserved, 284
timer.c, 1009, 1041, 1057
timer.h, 699
trust, 284
tsc_hz_mean, 284
vm_mode, 285
windows_used, 285
timer_calibration_record_t
timer.h, 692
timer_check_drift_now
timer.c, 1009, 1041, 1057
timer.h, 700
TIMER_DIV_1
apic.c, 925
TIMER_DIV_128
apic.c, 925
TIMER_DIV_16
apic.c, 925
TIMER_DIV_2
apic.c, 926
TIMER_DIV_32
apic.c, 926
TIMER_DIV_4
apic.c, 926
TIMER_DIV_64
apic.c, 926
TIMER_DIV_8
apic.c, 926
timer_drift_check_bare_metal
timer.c, 1042
timer_fatal
timer.c, 1043
timer_init
timer.c, 1010, 1044, 1057
timer.h, 701
timer_initial_count
apic.c, 938
timer_now_ns
timer.c, 1010, 1045, 1059
timer.h, 704
timer_now_ns_vm_relative
timer.c, 1046
timer_period_tsc_ticks

apic.c, 938
TIMER_PPI_EL1
apic.c, 910
TIMER_PPI_EL2
apic.c, 910
TIMER_PRIORITY
apic.c, 910
TIMER_REQUIRE_PIT
timer.c, 1016
timer_sig4_eq
timer.c, 1047
timer_sig8_eq
timer.c, 1048
timer_tick_hz
apic.c, 938
TIMER_TRUST_ABSOLUTE
timer.h, 693
timer_trust_level
timer.h, 692
timer_trust_level_t
timer.h, 692
TIMER_TRUST_NONE
timer.h, 693
TIMER_TRUST_RELATIVE
timer.h, 693
timer_tsc_hz
timer.c, 1010, 1048, 1060
timer.h, 705
TIMER_VM_STRICT_INVARIANT_TSC
timer.c, 1017
timespec, 285
tv_nsec, 285
tv_sec, 285
timestamp
log_slot_t, 223
timestamp_ns
physics_analytics_event_header, 229
TimeTrustState
timer.h, 692
timeval, 286
tv_sec, 286
tv_usec, 286
TimeWindow
timer.h, 692
TimeZone
EFI_TIME, 176
timezone, 286
tz_dsttime, 287
tz_minuteswest, 287
tlb_flush_all
rpi5_mmu.c, 876
to_pixel_space
ttf.c, 1349
tolower
ctype.h, 569
total_blkio_blocks_1k
blk_dev_slot, 107
total_blocks
blk_volume_meta_t, 114
blkio_dev, 117
blkio_file_state_t, 118
blkio_info_t, 119
blkio_params_t, 120
blkio_ram_state_t, 121
total_bytes
kmalloc_stats_t, 221
pmm_stats_t, 247
sf_alloc_stats, 261
total_bytes_read
MemoryProfile, 224
total_bytes_written
MemoryProfile, 224
total_context_transitions
WordTransitionMetrics, 364
total_devblocks
blk_volume_meta_t, 115
total_dict_lookups
profiler.c, 2441
total_error
TestStats, 280
total_execution_time
Profiler, 250
total_executions
DoeMetrics, 159
PhysicsSnapshot, 245
rolling_window_view_t, 255
RollingWindowOfTruth, 258
total_fail
TestStats, 280
total_forth_blocks
BlkioUsbState, 123
VirtBlkState, 306
total_heat
DoeMetrics, 159
HeartbeatSnapshot, 197
InferenceInputs, 216
total_heat_at_last_check
VM, 335
total_len
XXHash64State, 393
total_lookups
doe_metrics.c, 2269
DoeMetrics, 160
HotwordsStats, 212
total_memory_reads
profiler.c, 2441
total_memory_writes
profiler.c, 2441
total_pages
pmm_stats_t, 247
total_pass
TestStats, 280
total_regime_ticks
CDConfigTable, 137
total_runtime_ms
DoeMetrics, 160

total_samples
time_trust_state, 282
total_skip
TestStats, 280
total_stack_ops
profiler.c, 2441
total_tests
TestStats, 281
total_tests_run
break_me_tests.c, 2606
total_time
ProfileTimer, 253
WordProfile, 356
total_time_ns
WordStats, 358
total_transitions
WordTransitionMetrics, 364
total_user_lbn
block_subsystem.c, 2213
total_volumes
blk_volume_meta_t, 115
totalsize
fdt_header_t, 187
touch_samples
VMFleetWindow, 341
touched
VMPhysics, 345
toupper
ctype.h, 569
TPL_APPLICATION
uefi.h, 724
TPL_CALLBACK
uefi.h, 725
TPL_HIGH_LEVEL
uefi.h, 725
TPL_NOTIFY
uefi.h, 725
tr_dequeue_ptr
xhci_ep_ctx32_t, 378
tracked_blocks
blk_volume_meta_t, 115
trajectory_length
InferenceInputs, 216
transfer_purpose
xhci_msc_slot_t, 387
transition_heat
WordTransitionMetrics, 364
transition_metrics
DictEntry, 150
vm.h, 2077
transition_metrics_binary_chop_suggest_window
physics_pipelining_metrics.c, 2365
physics_pipelining_metrics.h, 1859
transition_metrics_cleanup
physics_pipelining_metrics.c, 2366
physics_pipelining_metrics.h, 1860
transition_metrics_context_accuracy_string
physics_pipelining_metrics.c, 2367
physics_pipelining_metrics.h, 1861
transition_metrics_get_probability_q48
physics_pipelining_metrics.c, 2368
physics_pipelining_metrics.h, 1862
transition_metrics_init
physics_pipelining_metrics.c, 2369
physics_pipelining_metrics.h, 1863
transition_metrics_record
physics_pipelining_metrics.c, 2371
physics_pipelining_metrics.h, 1865
transition_metrics_record_context
physics_pipelining_metrics.c, 2372
physics_pipelining_metrics.h, 1866
transition_metrics_record_prefetch_hit
physics_pipelining_metrics.c, 2373
physics_pipelining_metrics.h, 1867
transition_metrics_record_prefetch_miss
physics_pipelining_metrics.c, 2374
physics_pipelining_metrics.h, 1868
transition_metrics_reset
physics_pipelining_metrics.c, 2375
physics_pipelining_metrics.h, 1869
transition_metrics_should_speculate
physics_pipelining_metrics.c, 2376
physics_pipelining_metrics.h, 1870
transition_metrics_stats_string
physics_pipelining_metrics.c, 2377
physics_pipelining_metrics.h, 1871
transition_metrics_update_cache
physics_pipelining_metrics.c, 2378
physics_pipelining_metrics.h, 1872
transition_metrics_update_context_window
physics_pipelining_metrics.c, 2379
physics_pipelining_metrics.h, 1873
TRANSITION_WINDOW_SIZE
starforth_config.h, 1997
traverse_name_field
dictionary_manipulation_words.c, 2878
TRUE
uefi.h, 725
TrueType Text Words, 77
trust
time_trust_state, 282
timer_calibration_record, 284
tsc_hz_locked
timer.c, 1051
tsc_hz_mean
timer_calibration_record, 284
ttf.c
decode_composite_glyph, 1339
decode_glyph_r, 1339
decode_simple_glyph, 1340
find_table, 1341
fits, 1342
flat_push, 1343
flatten_contour, 1343
flatten_quad, 1344
in_bounds, 1345

q48_from_i32, 1346
q48_mid, 1346
q48_smul, 1346
raw_midpoint, 1347
rd_i16, 1347
rd_i8, 1348
rd_u16, 1348
rd_u32, 1348
rd_u8, 1349
TAG, 1336
to_pixel_space, 1349
ttf_codepoint_to_glyph, 1350
TTF_COMP_ARGS_ARE_WORDS, 1336
TTF_COMP_ARGS_ARE_XY_VALUES, 1336
TTF_COMP_MORE_COMPONENTS, 1337
TTF_COMP_WE_HAVE_2X2, 1337
TTF_COMP_WE_HAVE_A_SCALE, 1337
TTF_COMP_WE_HAVE_XY_SCALE, 1337
TTF_F2DOT14_ONE, 1337
TTF_FLAG_ON_CURVE, 1337
TTF_FLAG_REPEAT, 1337
TTF_FLAG_X_SAME_OR_POS, 1337
TTF_FLAG_X_SHORT, 1337
TTF_FLAG_Y_SAME_OR_POS, 1337
TTF_FLAG_Y_SHORT, 1338
ttf_glyph_advance_width, 1350
ttf_glyph_header, 1351
ttf_glyph_outline, 1352
TTF_MAX_SIMPLE_POINTS, 1338
ttf_parse, 1353
TTF_QUAD_SEGMENTS, 1338
ttf_raster_cache_get, 1354
ttf_raster_cache_init, 1355
TTF_RASTER_FLAT_MAX_CONTOURS, 1338
TTF_RASTER_FLAT_MAX_POINTS, 1338
TTF_RASTER_RAW_MAX_CONTOURS, 1338
TTF_RASTER_RAW_MAX_POINTS, 1338
ttf_rasterize_glyph, 1355

ttf.h
TTF_CACHE_BITMAP_BYTES, 709
TTF_CACHE_BITMAP_DIM, 709
TTF_CACHE_MARGIN, 709
TTF_CACHE_MAX_SIZE_PX, 709
ttf_codepoint_to_glyph, 711
TTF_ERR_BAD_GLYPH_INDEX, 709
TTF_ERR_BAD_SFNT, 709
TTF_ERR_BAD_TABLE, 709
TTF_ERR_OUT_OF_BOUNDS, 709
TTF_ERR_TABLE_MISSING, 709
TTF_ERR_TOO_DEEP, 709
TTF_ERR_TOO_MANY_CONTOURS, 709
TTF_ERR_TOO_MANY_POINTS, 710
TTF_ERR_UNSUPPORTED, 710
ttf_glyph_advance_width, 711
ttf_glyph_header, 712
TTF_GLYPH_MISSING, 710
ttf_glyph_outline, 713
TTF_MAX_COMPOSITE_DEPTH, 710
TTF_OK, 710
ttf_parse, 714
ttf_raster_cache_get, 715
ttf_raster_cache_init, 716
ttf_rasterize_glyph, 716
ttf_bitmap_t, 287
height, 287
pixels, 287
width, 288
TTF_CACHE_BITMAP_BYTES
ttf.h, 709
TTF_CACHE_BITMAP_DIM
ttf.h, 709
TTF_CACHE_MARGIN
ttf.h, 709
TTF_CACHE_MAX_SIZE_PX
ttf.h, 709
ttf_codepoint_to_glyph
ttf.c, 1350
ttf.h, 711
TTF_COMP_ARGS_ARE_WORDS
ttf.c, 1336
TTF_COMP_ARGS_ARE_XY_VALUES
ttf.c, 1336
TTF_COMP_MORE_COMPONENTS
ttf.c, 1337
TTF_COMP_WE_HAVE_2X2
ttf.c, 1337
TTF_COMP_WE_HAVE_A_SCALE
ttf.c, 1337
TTF_COMP_WE_HAVE_XY_SCALE
ttf.c, 1337
ttf_draw_glyph_cell
vt100.c, 1381
TTF_ERR_BAD_GLYPH_INDEX
ttf.h, 709
TTF_ERR_BAD_SFNT
ttf.h, 709
TTF_ERR_BAD_TABLE
ttf.h, 709
TTF_ERR_OUT_OF_BOUNDS
ttf.h, 709
TTF_ERR_TABLE_MISSING
ttf.h, 709
TTF_ERR_TOO_DEEP
ttf.h, 709
TTF_ERR_TOO_MANY_CONTOURS
ttf.h, 709
TTF_ERR_TOO_MANY_POINTS
ttf.h, 710
TTF_ERR_UNSUPPORTED
ttf.h, 710
TTF_F2DOT14_ONE
ttf.c, 1337
TTF_FLAG_ON_CURVE
ttf.c, 1337
TTF_FLAG_REPEAT
ttf.c, 1337

TTF_FLAG_X_SAME_OR_POS
ttf.c, 1337
TTF_FLAG_X_SHORT
ttf.c, 1337
TTF_FLAG_Y_SAME_OR_POS
ttf.c, 1337
TTF_FLAG_Y_SHORT
ttf.c, 1338
ttf_flat_point_t, 288
x, 288
y, 288
ttf_font_t, 288
cmap_subtable_off, 289
data, 289
glyf_len, 289
glyf_off, 289
has_cmap, 289
head_off, 290
hmtx_off, 290
index_to_loc_format, 290
loca_len, 290
loca_off, 290
maxp_off, 290
num_glyphs, 290
num_h_metrics, 290
size, 291
units_per_em, 291
ttf_glyph_advance_width
ttf.c, 1350
ttf.h, 711
ttf_glyph_header
ttf.c, 1351
ttf.h, 712
ttf_glyph_header_t, 291
glyf_length, 292
glyf_offset, 292
num_contours, 292
x_max, 292
x_min, 292
y_max, 292
y_min, 292
TTF_GLYPH_MISSING
ttf.h, 710
ttf_glyph_outline
ttf.c, 1352
ttf.h, 713
TTF_MAX_COMPOSITE_DEPTH
ttf.h, 710
TTF_MAX_SIMPLE_POINTS
ttf.c, 1338
TTF_OK
ttf.h, 710
ttf_outline_t, 293
contour_count, 293
contour_ends, 293
max_contours, 294
max_points, 294
point_count, 294
points, 294
ttf_parse
ttf.c, 1353
ttf.h, 714
ttf_point_t, 294
on_curve, 295
x, 295
y, 295
TTF_QUAD_SEGMENTS
ttf.c, 1338
ttf_raster_cache_get
ttf.c, 1354
ttf.h, 715
ttf_raster_cache_init
ttf.c, 1355
ttf.h, 716
ttf_raster_cache_slot_t, 295
codepoint, 296
font, 296
height, 296
hits, 296
pixels, 296
size_px, 296
valid, 296
width, 296
ttf_raster_cache_t, 297
evict_next, 297
slot_count, 297
slots, 298
TTF_RASTER_FLAT_MAX_CONTOURS
ttf.c, 1338
TTF_RASTER_FLAT_MAX_POINTS
ttf.c, 1338
TTF_RASTER_RAW_MAX_CONTOURS
ttf.c, 1338
TTF_RASTER_RAW_MAX_POINTS
ttf.c, 1338
ttf_rasterize_glyph
ttf.c, 1355
ttf.h, 716
ttf_words.c
register_ttf_words, 3248
ttf_words.h
register_ttf_words, 3023
tuning_lock
VM, 335
tv_nsec
timespec, 285
tv_sec
timespec, 285
timeval, 286
tv_usec
timeval, 286
Type
EFI_DEVICE_PATH_PROTOCOL, 164
EFI_MEMORY_DESCRIPTOR, 173
type
__attribute__, 83

BlockBuffer, 125
gpt_part_t, 194
TestCase, 278
VirtBlkReq, 303
VirtioInputEvent, 312
type_attr
idt_entry, 214
interrupts.c, 988
types.h
gid_t, 608
mode_t, 608
off_t, 608
pid_t, 608
ssize_t, 608
uid_t, 608
tz_dsttime
timezone, 287
tz_minuteswest
timezone, 287

u128
fe25519.c, 1207
u32_to_dec
capsule_loader.c, 1126
UART16550_LSR
console.c, 1252
UART16550_LSR_DR
console.c, 1252
UART16550_LSR_THRE
console.c, 1252
ucb_k
CDTuning, 140
ucell_t
vm.h, 2011
udiv_floor
block_subsystem.c, 2208
uefi.h
AllHandles, 731
AllocateAddress, 730
AllocateAnyPages, 730
AllocateMaxAddress, 730
BOOLEAN, 725
ByProtocol, 731
ByRegisterNotify, 731
CHAR16, 725
EFI_ABORTED, 721
EFI_ACPI_20_TABLE_GUID, 732
EFI_ACPI_TABLE_GUID, 732
EFI_ALLOCATE_PAGES, 725
EFI_ALLOCATE_POOL, 725
EFI_ALLOCATE_TYPE, 730
EFI_BAD_BUFFER_SIZE, 721
EFI_BLOCK_IO_PROTOCOL, 725
EFI_BLOCK_IO_PROTOCOL_GUID, 732
EFI_BLOCK_READ, 725
EFI_BOOT_SERVICES, 726
EFI_BUFFER_TOO_SMALL, 721
EFI_DEVICE_ERROR, 721
EFI_DEVICE_PATH_PROTOCOL_GUID, 732
EFI_DEVICE_PATH_SUBTYPE_END_ENTIRE, 721
EFI_DEVICE_PATH_SUBTYPE_HARDDRIVE, 721
EFI_DEVICE_PATH_TYPE_END, 721
EFI_DEVICE_PATH_TYPE_MEDIA, 722
EFI_DTB_TABLE_GUID, 732
EFI_EXIT_BOOT_SERVICES, 726
EFI_FILE_CLOSE, 726
EFI_FILE_GET_INFO, 726
EFI_FILE_INFO_GUID, 732
EFI_FILE_MODE_READ, 722
EFI_FILE_OPEN, 726
EFI_FILE_PROTOCOL, 726
EFI_FILE_READ, 726
EFI_FREE_PAGES, 726
EFI_FREE_POOL, 726
EFI_GET_MEMORY_MAP, 726
EFI_GET_VARIABLE, 727
EFI_GRAPHICS_OUTPUT_PROTOCOL, 727
EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID, 733
EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE, 727
EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE, 727
EFI_GRAPHICS_PIXEL_FORMAT, 731
EFI_HANDLE, 727
EFI_HANDLE_PROTOCOL, 727
EFI_INVALID_PARAMETER, 722
EFI_LOAD_ERROR, 722
EFI_LOADED_IMAGE_PROTOCOL_GUID, 733
EFI_LOCATE_DEVICE_PATH, 727
EFI_LOCATE_HANDLE, 727
EFI_LOCATE_PROTOCOL, 727
EFI_LOCATE_SEARCH_TYPE, 731
EFI_MEMORY_RP, 722
EFI_MEMORY_RUNTIME, 722
EFI_MEMORY_TYPE, 731
EFI_MEMORY_UC, 722
EFI_MEMORY_UCE, 722
EFI_MEMORY_WB, 722
EFI_MEMORY_WC, 722
EFI_MEMORY_WP, 723
EFI_MEMORY_WT, 723
EFI_MEMORY_XP, 723
EFI_NOT_FOUND, 723
EFI_NOT_READY, 723
EFI_OUT_OF_RESOURCES, 723
EFI_PHYSICAL_ADDRESS, 728
EFI_RAISE_TPL, 728
EFI_RESET_SYSTEM, 728
EFI_RESET_TYPE, 731
EFI_RESTORE_TPL, 728
EFI_RUNTIME_SERVICES, 728
EFI_SET_VARIABLE, 728
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL, 728

EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID, 733
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_OPEN_VOLUME, 728
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL, 728
EFI_STALL, 728
EFI_STATUS, 729
EFI_SUCCESS, 723
EFI_SYSTEM_TABLE, 729
EFI_TEXT_RESET, 729
EFI_TEXT_STRING, 729
EFI_TPL, 729
EFI_UNSUPPORTED, 723
EFI_VARIABLE_BOOTSERVICE_ACCESS, 723
EFI_VARIABLE_NON_VOLATILE, 723
EFI_VARIABLE_RUNTIME_ACCESS, 724
EFI_WRITE_PROTECTED, 724
EfiACPIMemoryNVS, 731
EfiACPIReclaimMemory, 731
EFIAPI, 724
EfiBootServicesCode, 731
EfiBootServicesData, 731
EfiConventionalMemory, 731
EfiLoaderCode, 731
EfiLoaderData, 731
EfiMaxMemoryType, 731
EfiMemoryMappedIO, 731
EfiMemoryMappedIOPortSpace, 731
EfiPalCode, 731
EfiPersistentMemory, 731
EfiReservedMemoryType, 731
EfiResetCold, 732
EfiResetShutdown, 732
EfiResetWarm, 732
EfiRuntimeServicesCode, 731
EfiRuntimeServicesData, 731
EfiUnusableMemory, 731
FALSE, 724
INT16, 729
INT32, 729
INT64, 729
INT8, 729
INTN, 729
MaxAllocateType, 730
PixelBitMask, 731
PixelBltOnly, 731
PixelBlueGreenRedReserved8BitPerColor, 731
PixelFormatMax, 731
PixelRedGreenBlueReserved8BitPerColor, 731
SF_VAR_BOOT_ARGS, 724
SF_VAR_REBOOT_TRIES, 724
SF_VAR_ZUSE_CERT, 724
STARFORTH_VENDOR_GUID, 724
TPL_APPLICATION, 724
TPL_CALLBACK, 725
TPL_HIGH_LEVEL, 725
TPL_NOTIFY, 725
TRUE, 725
UINT16, 730
UINT32, 730
UINT64, 730
UINT8, 730
UINTN, 730
VOID, 730
uefi_boot_services_exited
BootInfo, 128
UEFI_DEBUG_CHECKPOINTS
uefi_loader.c, 1082
uefi_loader.c
debug_checkpoint, 1083
DEBUG_CHECKPOINT_STALL_US, 1082
efi_main, 1083
g_boot_info, 1088
guid_equals, 1085
kernel_elf_buffer, 1088
kernel_elf_size, 1089
KERNEL_MAX_SIZE, 1082
KernelEntry, 1082
load_cfg_from_esp, 1086
load_kernel_from_esp, 1087
RAW_LOG, 1082
UEFI_DEBUG_CHECKPOINTS, 1082
uid_t
types.h, 608
UINT16
uefi.h, 730
UINT32
uefi.h, 730
UINT64
uefi.h, 730
UINT8
uefi.h, 730
UINTN
uefi.h, 730
underline
VT100State, 353
UNDERLINE_ROW
vt100.c, 1358
Unified API Wrappers, 19
UninstallProtocolInterface
_EFI_BOOT_SERVICES, 90
units_per_em
ttf_font_t, 291
UNLIKELY
vm.h, 2009
Unload
EFI_LOADED_IMAGE_PROTOCOL, 172
UnloadImage
_EFI_BOOT_SERVICES, 90
update_free_bytes
kmalloc.c, 1455
update_stats
timer.c, 1048
usage_to_key
usb_kbd.c, 1514
usb_blk_close

blkio_usb.c, 1504
usb_blk_flush
blkio_usb.c, 1505
usb_blk_info
blkio_usb.c, 1505
usb_blk_open
blkio_usb.c, 1505
usb_blk_read
blkio_usb.c, 1505
usb_blk_state_for
blkio_usb.c, 1506
usb_blk_write
blkio_usb.c, 1507
USB_BOT_CBW_FLAG_DATA_IN
xhci.h, 805
USB_BOT_CBW_LENGTH
xhci.h, 806
USB_BOT_CBW_LUN_DEFAULT
xhci.h, 806
USB_BOT_CBW_SIGNATURE
xhci.h, 806
usb_bot_cbw_t, 298
bCBWCBLength, 298
bCBWLUN, 298
bmCBWFlags, 298
CBWCB, 299
dCBWDataTransferLength, 299
dCBWSignature, 299
dCBWTag, 299
USB_BOT_CSW_LENGTH
xhci.h, 806
USB_BOT_CSW_SIGNATURE
xhci.h, 806
USB_BOT_CSW_STATUS_FAILED
xhci.h, 806
USB_BOT_CSW_STATUS_PASS
xhci.h, 806
USB_BOT_CSW_STATUS_PHASE_ERROR
xhci.h, 806
usb_bot_csw_t, 299
bCSWStatus, 300
dCSWDataResidue, 300
dCSWSignature, 300
dCSWTag, 300
USB_BOT_MASS_STORAGE_RESET
xhci.h, 806
USB_CLASS_HID
xhci.h, 806
USB_CLASS_MASS_STORAGE
xhci.h, 807
usb_cmd
xhci_op_regs_t, 388
USB_CONFIG_OFF_CONFIG_VALUE
xhci.h, 807
USB_CONFIG_OFF_TOTAL_LENGTH
xhci.h, 807
USB_DESC_OFF_LENGTH
xhci.h, 807
USB_DESC_OFF_TYPE
xhci.h, 807
USB_DESC_TYPE_CONFIG
xhci.h, 807
USB_DESC_TYPE_DEVICE
xhci.h, 807
USB_DESC_TYPE_ENDPOINT
xhci.h, 807
USB_DESC_TYPE_INTERFACE
xhci.h, 807
USB_DIR_DEVICE_TO_HOST
xhci.h, 807
USB_DIR_HOST_TO_DEVICE
xhci.h, 808
USB_EP_ADDR_DIR_MASK
xhci.h, 808
USB_EP_ADDR_NUM_MASK
xhci.h, 808
USB_EP_ATTR_TYPE_MASK
xhci.h, 808
USB_EP_OFF_ADDRESS
xhci.h, 808
USB_EP_OFF_ATTRIBUTES
xhci.h, 808
USB_EP_OFF_INTERVAL
xhci.h, 808
USB_EP_OFF_MAX_PACKET_SIZE
xhci.h, 808
USB_EP_TYPE_BULK
xhci.h, 808
USB_EP_TYPE_INTERRUPT
xhci.h, 808
USB_FEATURE_ENDPOINT_HALT
xhci.h, 809
USB_HID_PROTOCOL_BOOT
xhci.h, 809
USB_HID_REPORT_OUTPUT
xhci.c, 1517
USB_HID_REQ_SET_PROTOCOL
xhci.h, 809
USB_HID_REQ_SET_REPORT
xhci.c, 1517
USB_IFACE_OFF_CLASS
xhci.h, 809
USB_IFACE_OFF_NUMBER
xhci.h, 809
USB_IFACE_OFF_PROTOCOL
xhci.h, 809
USB_IFACE_OFF_SUBCLASS
xhci.h, 809
usb_kbd.c
g_count, 1514
g_events, 1514
g_head, 1514
g_leds_want, 1514
held, 1510
KBD_EVENTS, 1510
key_for, 1510

keycode, 1514
modifier_key, 1514
pressed, 1514
push, 1510
usage_to_key, 1514
usb_kbd_forget, 1511
usb_kbd_leds_wanted, 1511
usb_kbd_pop_event, 1512
usb_kbd_report, 1512
usb_kbd_set_leds, 1513
usb_kbd.h
usb_kbd_forget, 735
USB_KBD_LED_CAPS, 735
USB_KBD_LED_NUM, 735
USB_KBD_LED_SCROLL, 735
usb_kbd_leds_wanted, 735
usb_kbd_pop_event, 736
usb_kbd_report, 736
usb_kbd_set_leds, 737
usb_kbd_forget
usb_kbd.c, 1511
usb_kbd.h, 735
USB_KBD_LED_CAPS
usb_kbd.h, 735
USB_KBD_LED_NUM
usb_kbd.h, 735
USB_KBD_LED_SCROLL
usb_kbd.h, 735
usb_kbd_leds_wanted
usb_kbd.c, 1511
usb_kbd.h, 735
usb_kbd_pop_event
usb_kbd.c, 1512
usb_kbd.h, 736
usb_kbd_report
usb_kbd.c, 1512
usb_kbd.h, 736
usb_kbd_set_leds
usb_kbd.c, 1513
usb_kbd.h, 737
USB_PROTOCOL_BOT
xhci.h, 809
USB_PROTOCOL_HID_KEYBOARD
xhci.h, 809
USB_RECIP_DEVICE
xhci.h, 809
USB_RECIP_ENDPOINT
xhci.h, 810
USB_RECIP_INTERFACE
xhci.h, 810
USB_REQ_CLEAR_FEATURE
xhci.h, 810
USB_REQ_GET_DESCRIPTOR
xhci.h, 810
USB_REQ_SET_CONFIGURATION
xhci.h, 810
USB_REQ_TYPE_CLASS
xhci.h, 810
USB_REQ_TYPE_STANDARD
xhci.h, 810
usb_setup_packet_t, 300
bmRequestType, 301
bRequest, 301
wIndex, 301
wLength, 301
wValue, 301
usb_sts
xhci_op_regs_t, 388
USB_SUBCLASS_HID_BOOT
xhci.h, 810
USB_SUBCLASS_SCSI
xhci.h, 810
used
sk_v4_drive_t, 269
VirtBlkState, 306
VirtInputState, 308
VirtRngState, 317
used_bytes
kmalloc_stats_t, 221
pmm_stats_t, 248
sf_alloc_stats, 261
used_event
VirtqAvail, 313
used_pages
pmm_stats_t, 248
user_blocks
blk_dev_slot, 107
USER_BLOCKS_START
vm.h, 2009
USER_IDENTITY_EMAIL_MAX
user_identity_seed.h, 739
USER_IDENTITY_FULL_NAME_MAX
user_identity_seed.h, 739
USER_IDENTITY_PHONE_MAX
user_identity_seed.h, 739
user_identity_seed.h
USER_IDENTITY_EMAIL_MAX, 739
USER_IDENTITY_FULL_NAME_MAX, 739
USER_IDENTITY_PHONE_MAX, 739
USER_IDENTITY_SEED_MAGIC, 739
user_identity_seed_size_check, 739
USER_IDENTITY_SEED_VERSION, 739
USER_IDENTITY_USERNAME_MAX, 739
USER_IDENTITY_SEED_MAGIC
user_identity_seed.h, 739
user_identity_seed_size_check
user_identity_seed.h, 739
user_identity_seed_t, 301
_pad, 302
crc, 302
email, 302
full_name, 302
magic, 302
phone, 302
pubkey, 302
seed, 302

username, 303
version, 303
USER_IDENTITY_SEED_VERSION
user_identity_seed.h, 739
USER_IDENTITY_USERNAME_MAX
user_identity_seed.h, 739
user_vm_id
wirebind_live_entry_t, 354
username
user_identity_seed_t, 303
wirebind_live_entry_t, 354

v
fe25519, 189
v1
sk_v4_marker.c, 1599
XXHash64State, 393
v2
sk_v4_marker.c, 1599
XXHash64State, 393
v3
XXHash64State, 393
v3/include/arch_detect.h, 1708
v3/include/blkcfg.h, 1716
v3/include/blkio.h, 1719
v3/include/blkio_factory.h, 1724
v3/include/block_subsystem.h, 1727
v3/include/cli.h, 1759
v3/include/compudynamics.h, 1763
v3/include/dictionary_heat_optimization.h, 1770
v3/include/dictionary_management.h, 1780
v3/include/doe_metrics.h, 1782
v3/include/inference_engine.h, 1791
v3/include/io.h, 1800
v3/include/log.h, 1806
v3/include/math_portable.h, 1812
v3/include/memory_management.h, 1816
v3/include/physics_execution_hooks.h, 1820
v3/include/physics_hotwords_cache.h, 1826
v3/include/physics_metadata.h, 1845
v3/include/physics_pipelining_metrics.h, 1857
v3/include/physics_runtime.h, 1874
v3/include/platform_alloc.h, 1882
v3/include/platform_io.h, 1891
v3/include/platform_lock.h, 1892
v3/include/platform_time.h, 1897
v3/include/profiler.h, 1904
v3/include/q48_16.h, 1926
v3/include/README.md, 1943
v3/include/repl.h, 1958
v3/include/rolling_window_knobs.h, 1961
v3/include/rolling_window_of_truth.h, 1962
v3/include/ssm_jacquard.h, 1977
v3/include/starforth_config.h, 1988
v3/include/startup.h, 1998
v3/include/version.h, 2000
v3/include/vm.h, 2001
v3/include/vm_api.h, 2077
v3/include/vm_asm_opt.h, 2078
v3/include/vm_asm_opt_arm64.h, 2080
v3/include/vm_asm_opt_riscv64.h, 2087
v3/include/vm_debug.h, 2092
v3/include/vm_host.h, 2095
v3/include/vm_inner_interp_arm64.h, 2100
v3/include/vm_inner_interp_asm.h, 2102
v3/include/vm_inner_interp_riscv64.h, 2112
v3/include/word_registry.h, 2120
v3/src/blkio_factory.c, 2124
v3/src/blkio_file.c, 2132
v3/src/blkio_ram.c, 2141
v3/src/block_subsystem.c, 2148
v3/src/cli.c, 2214
v3/src/compudynamics.c, 2217
v3/src/dictionary_heat_optimization.c, 2226
v3/src/dictionary_management.c, 2235
v3/src/doe_metrics.c, 2261
v3/src/heartbeat_export.c, 2270
v3/src/inference_engine.c, 2272
v3/src/io.c, 2288
v3/src/log.c, 2292
v3/src/main.c, 2301
v3/src/math_portable.c, 2312
v3/src/memory_management.c, 2316
v3/src/physics_execution_hooks.c, 2321
v3/src/physics_hotwords_cache.c, 2326
v3/src/physics_metadata.c, 2348
v3/src/physics_pipelining_metrics.c, 2362
v3/src/physics_runtime.c, 2380
v3/src/platform/alloc_host.c, 2397
v3/src/platform/alloc_kernel.c, 2409
v3/src/platform/l4re/README.md, 1943
v3/src/platform/l4re/time.c, 2414
v3/src/platform/linux/io.c, 2291
v3/src/platform/linux/README.md, 1943
v3/src/platform/linux/time.c, 2415
v3/src/platform/platform_init.c, 2418
v3/src/platform/README.md, 1943
v3/src/platform/threading.c, 2419
v3/src/profiler.c, 2424
v3/src/README.md, 1943
v3/src/repl.c, 2468
v3/src/rolling_window_of_truth.c, 2471
v3/src/ssm_jacquard.c, 2494
v3/src/stack_management.c, 2506
v3/src/test_runner/include/README.md, 1943
v3/src/test_runner/include/test_common.h, 2512
v3/src/test_runner/include/test_contracts.h, 2534
v3/src/test_runner/include/test_runner.h, 2545
v3/src/test_runner/modules/acl_words_test.c, 2580
v3/src/test_runner/modules/arithmetic_words_test.c, 2599
v3/src/test_runner/modules/block_words_test.c, 2601
v3/src/test_runner/modules/break_me_tests.c, 2603
v3/src/test_runner/modules/control_words_test.c, 2607
v3/src/test_runner/modules/defining_words_tests.c, 2609

v3/src/test_runner/modules/dictionary_manipulation_words_test.c, 2611
v3/src/test_runner/modules/dictionary_words_test.c, 2613
v3/src/test_runner/modules/double_words_test.c, 2615
v3/src/test_runner/modules/format_words_test.c, 2616
v3/src/test_runner/modules/inference_words_test.c, 2618
v3/src/test_runner/modules/integration_tests.c, 2620
v3/src/test_runner/modules/io_words_test.c, 2623
v3/src/test_runner/modules/logical_words_test.c, 2625
v3/src/test_runner/modules/mama_forth_words_test.c, 2626
v3/src/test_runner/modules/memory_words_test.c, 2629
v3/src/test_runner/modules/mixed_arithmetic_words_test.c, 2631
v3/src/test_runner/modules/physics_freeze_words_test.c, 2633
v3/src/test_runner/modules/q48_words_test.c, 2635
v3/src/test_runner/modules/README.md, 1943
v3/src/test_runner/modules/return_stack_words_test.c, 2636
v3/src/test_runner/modules/stack_words_test.c, 2638
v3/src/test_runner/modules/starforth_words_test.c, 2641
v3/src/test_runner/modules/stress_tests.c, 2643
v3/src/test_runner/modules/string_words_test.c, 2645
v3/src/test_runner/modules/system_words_test.c, 2647
v3/src/test_runner/modules/vocabulary_words_test.c, 2649
v3/src/test_runner/README.md, 1943
v3/src/test_runner/test_common.c, 2651
v3/src/test_runner/test_contracts.c, 2674
v3/src/test_runner/test_runner.c, 2686
v3/src/vm.c, 2694
v3/src/vm_api.c, 2712
v3/src/vm_bootstrap.c, 2726
v3/src/vm_debug.c, 2728
v3/src/vm_internal.h, 2746
v3/src/vm_time.c, 2765
v3/src/word_registry.c, 2773
v3/src/word_source/acl_words.c, 2777
v3/src/word_source/arithmetic_words.c, 2792
v3/src/word_source/block_words.c, 2806
v3/src/word_source/control_words.c, 2834
v3/src/word_source/defer_words.c, 2836
v3/src/word_source/defining_words.c, 2841
v3/src/word_source/dictionary_heat_diagnostic_words.c, 2860
v3/src/word_source/dictionary_manipulation_words.c, 2866
v3/src/word_source/dictionary_words.c, 2879
v3/src/word_source/double_words.c, 2888
v3/src/word_source/editor_words.c, 2902
v3/src/word_source/format_words.c, 2909
v3/src/word_source/framebuffer_words.c, 2926
v3/src/word_source/include/acl_words.h, 2930
v3/src/word_source/include/arithmetic_words.h, 2933
v3/src/word_source/include/block_words.h, 2936
v3/src/word_source/include/control_words.h, 2948
v3/src/word_source/include/defer_words.h, 2949
v3/src/word_source/include/defining_words.h, 2951
v3/src/word_source/include/dictionary_heat_diagnostic_words.h, 2955
v3/src/word_source/include/dictionary_manipulation_words.h, 2957
v3/src/word_source/include/dictionary_words.h, 2959
v3/src/word_source/include/double_words.h, 2961
v3/src/word_source/include/editor_words.h, 2964
v3/src/word_source/include/format_words.h, 2966
v3/src/word_source/include/framebuffer_words.h, 2968
v3/src/word_source/include/inference_words.h, 2970
v3/src/word_source/include/io_words.h, 2972
v3/src/word_source/include/keyboard_words.h, 2975
v3/src/word_source/include/log_words.h, 2978
v3/src/word_source/include/logical_words.h, 2980
v3/src/word_source/include/mama_forth_words.h, 2983
v3/src/word_source/include/memory_words.h, 2984
v3/src/word_source/include/mixed_arithmetic_words.h, 2986
v3/src/word_source/include/physics_benchmark_words.h, 2988
v3/src/word_source/include/physics_diagnostic_words.h, 2990
v3/src/word_source/include/physics_freeze_words.h, 2995
v3/src/word_source/include/physics_pipelining_diagnostic_words.h, 2997
v3/src/word_source/include/q48_words.h, 2999
v3/src/word_source/include/README.md, 1943
v3/src/word_source/include/return_stack_words.h, 3002
v3/src/word_source/include/scroll_words.h, 3004
v3/src/word_source/include/stack_words.h, 3005
v3/src/word_source/include/starforth_words.h, 3007
v3/src/word_source/include/string_words.h, 3017
v3/src/word_source/include/system_words.h, 3020
v3/src/word_source/include/ttf_words.h, 3023
v3/src/word_source/include/vocabulary_words.h, 3024
v3/src/word_source/inference_words.c, 3029
v3/src/word_source/io_words.c, 3045
v3/src/word_source/keyboard_words.c, 3055
v3/src/word_source/lifecycle_words_hosted.c, 3062
v3/src/word_source/log_words.c, 3068
v3/src/word_source/logical_words.c, 3085
v3/src/word_source/memory_words.c, 3100
v3/src/word_source/mixed_arithmetic_words.c, 3114
v3/src/word_source/physics_benchmark_words.c, 3123
v3/src/word_source/physics_diagnostic_words.c, 3131
v3/src/word_source/physics_freeze_words.c, 3135
v3/src/word_source/physics_pipelining_diagnostic_words.c, 3145
v3/src/word_source/q48_16_words.c, 3151
v3/src/word_source/q48_words.c, 3159
v3/src/word_source/README.md, 1943
v3/src/word_source/return_stack_words.c, 3179

v3/src/word_source/scroll_words.c, 3184
v3/src/word_source/stack_words.c, 3185
v3/src/word_source/starforth_words.c, 3195
v3/src/word_source/string_words.c, 3211
v3/src/word_source/system_words.c, 3233
v3/src/word_source/ttf_words.c, 3248
v3/src/word_source/vocabulary_words.c, 3249
v4
    XXHash64State, 393
vaddr_from_ptr
    string_words.c, 3232
vaddr_t
    vm.h, 2011
valid
    cache_slot_t, 129
    ttf_raster_cache_slot_t, 296
    xhci_dev_t, 377
validate_result_strings
    capsule_validate.c, 1153
value
    DiagEvent, 147
    sbiret_t, 260
    VirtioInputEvent, 312
variance
    time_trust_state, 282
VBLK_ARTEMIS_SECTORS
    virtio_blk.h, 759
vblk_close
    virtio_blk.c, 1609
vblk_flush
    virtio_blk.c, 1609
vblk_info
    virtio_blk.c, 1609
vblk_init_device
    virtio_blk.c, 1609
vblk_io
    virtio_blk.c, 1610
VBLK_IO_TIMEOUT_NS
    virtio_blk.c, 1606
vblk_open
    virtio_blk.c, 1611
vblk_read
    virtio_blk.c, 1611
VBLK_SECTOR_SIZE
    virtio_blk.h, 759
VBLK_SECTORS_PER_BLOK
    virtio_blk.h, 759
vblk_write
    virtio_blk.c, 1612
VCAP_OFF_BAR
    virtio_blk.c, 1606
    virtio_input.c, 1617
    virtio_rng.c, 1626
VCAP_OFF_CAP_LEN
    virtio_blk.c, 1606
VCAP_OFF_CAP_NEXT
    virtio_blk.c, 1606
    virtio_input.c, 1617
    virtio_rng.c, 1626
VCAP_OFF_CAP_VNDR
    virtio_blk.c, 1606
    virtio_input.c, 1617
    virtio_rng.c, 1627
VCAP_OFF_CFG_TYPE
    virtio_blk.c, 1607
    virtio_input.c, 1617
    virtio_rng.c, 1627
VCAP_OFF_LENGTH
    virtio_blk.c, 1607
    virtio_input.c, 1618
    virtio_rng.c, 1627
VCAP_OFF_NOTIFY_MULT
    virtio_blk.c, 1607
    virtio_input.c, 1618
    virtio_rng.c, 1627
VCAP_OFF_OFFSET
    virtio_blk.c, 1607
    virtio_input.c, 1618
    virtio_rng.c, 1627
vendor_id
    PciDevice, 228
VendorGuid
    EFI_CONFIGURATION_TABLE, 164
VendorTable
    EFI_CONFIGURATION_TABLE, 164
verbose
    VMHostServices, 343
verify_mint
    capsule_mint.c, 1134
Version
    EFI_GRAPHICS_OUTPUT_MODE_INFORMATION, 168
version
    blk_volume_meta_t, 115
    fdt_header_t, 188
    user_identity_seed_t, 303
    zuse_cert_devblock_t, 394
    zuse_genesis_marker_t, 395
    zuse_genesis_marker_v2_t, 396
version.h
    LITHOS_VERSION, 2000
    LITHOS_VERSION_STR, 2000
    STARFORTH_ARCH, 2000
    STARFORTH_TARGET, 2000
    STARFORTH_TIMESTAMP, 2000
    STARFORTH_VERSION, 2001
    STARFORTH_VERSION_FULL, 2001
version_major
    physics_analytics_header, 232
version_minor
    physics_analytics_header, 233
VerticalResolution
    EFI_GRAPHICS_OUTPUT_MODE_INFORMATION, 168
vfprintf
    stdio.h, 590

vinput_init_device
virtio_input.c, 1620
VirtBlkReq, 303
reserved, 303
sector, 303
type, 303
VirtBlkState, 304
avail, 304
avail_idx, 304
capacity_sectors, 305
common, 305
data_buf, 305
desc, 305
last_used_idx, 305
notify, 305
notify_off_mult, 305
queue_notify_off, 305
req_buf, 306
status_buf, 306
total_forth_blocks, 306
used, 306
VirtInputState, 306
avail, 307
avail_idx, 307
bufs, 307
common, 307
desc, 307
isr_status, 307
last_used_idx, 308
notify, 308
notify_off_mult, 308
queue_notify_off, 308
used, 308
virtio_blk.c
g_vblk, 1615
g_vblk_dead, 1615
g_vblk_ready, 1615
g_vblk_vtable, 1615
rmb, 1609
vblk_close, 1609
vblk_flush, 1609
vblk_info, 1609
vblk_init_device, 1609
vblk_io, 1610
VBLK_IO_TIMEOUT_NS, 1606
vblk_open, 1611
vblk_read, 1611
vblk_write, 1612
VCAP_OFF_BAR, 1606
VCAP_OFF_CAP_LEN, 1606
VCAP_OFF_CAP_NEXT, 1606
VCAP_OFF_CAP_VNDR, 1606
VCAP_OFF_CFG_TYPE, 1607
VCAP_OFF_LENGTH, 1607
VCAP_OFF_NOTIFY_MULT, 1607
VCAP_OFF_OFFSET, 1607
virtio_blk_find_artemis, 1613
VIRTIO_F_VERSION_1, 1607
VIRTIO_PCI_CAP_COMMON_CFG, 1607
VIRTIO_PCI_CAP_DEVICE_CFG, 1607
VIRTIO_PCI_CAP_ISR_CFG, 1607
VIRTIO_PCI_CAP_NOTIFY_CFG, 1607
VIRTIO_PCI_CAP_VENDOR_ID, 1607
VIRTIO_STATUS_ACKNOWLEDGE, 1608
VIRTIO_STATUS_DRIVER, 1608
VIRTIO_STATUS_DRIVER_OK, 1608
VIRTIO_STATUS_FAILED, 1608
VIRTIO_STATUS_FEATURES_OK, 1608
VQUEUE_SIZE, 1608
VRING_DESC_F_NEXT, 1608
VRING_DESC_F_WRITE, 1608
walk_virtio_caps, 1613
wmb, 1614
virtio_blk.h
VBLK_ARTEMIS_SECTORS, 759
VBLK_SECTOR_SIZE, 759
VBLK_SECTORS_PER_BLOK, 759
VIRTIO_BLK_DEVICE_LEGACY, 759
VIRTIO_BLK_DEVICE_MODERN, 759
virtio_blk_find_artemis, 760
VIRTIO_BLK_S_IOERR, 759
VIRTIO_BLK_S_OK, 759
VIRTIO_BLK_S_UNSUPP, 760
VIRTIO_BLK_T_IN, 760
VIRTIO_BLK_T_OUT, 760
VIRTIO_PCI_VENDOR_ID, 760
VIRTIO_BLK_DEVICE_LEGACY
virtio_blk.h, 759
VIRTIO_BLK_DEVICE_MODERN
virtio_blk.h, 759
virtio_blk_find_artemis
virtio_blk.c, 1613
virtio_blk.h, 760
VIRTIO_BLK_S_IOERR
virtio_blk.h, 759
VIRTIO_BLK_S_OK
virtio_blk.h, 759
VIRTIO_BLK_S_UNSUPP
virtio_blk.h, 760
VIRTIO_BLK_T_IN
virtio_blk.h, 760
VIRTIO_BLK_T_OUT
virtio_blk.h, 760
VIRTIO_F_VERSION_1
virtio_blk.c, 1607
virtio_input.c, 1618
virtio_rng.c, 1627
virtio_input.c
diag_push, 1619
DIAG_RING_SIZE, 1617
enable_interrupt_route, 1619
EVENTQ_SIZE, 1617
g_diag_head, 1624
g_diag_ring, 1624
g_diag_tail, 1624
g_vinput, 1624

g_vinput_ready, 1624
g_virtio_input_gic_intid, 1624
g_virtio_input_isr_count, 1625
g_virtio_input_plic_source, 1625
VCAP_OFF_BAR, 1617
VCAP_OFF_CAP_NEXT, 1617
VCAP_OFF_CAP_VNDR, 1617
VCAP_OFF_CFG_TYPE, 1617
VCAP_OFF_LENGTH, 1618
VCAP_OFF_NOTIFY_MULT, 1618
VCAP_OFF_OFFSET, 1618
vinput_init_device, 1620
VIRTIO_F_VERSION_1, 1618
virtio_input_find_keyboard, 1621
virtio_input_isr, 1621
virtio_input_pop_event, 1622
VIRTIO_PCI_CAP_COMMON_CFG, 1618
VIRTIO_PCI_CAP_ISR_CFG, 1618
VIRTIO_PCI_CAP_NOTIFY_CFG, 1618
VIRTIO_PCI_CAP_VENDOR_ID, 1618
VIRTIO_STATUS_ACKNOWLEDGE, 1618
VIRTIO_STATUS_DRIVER, 1618
VIRTIO_STATUS_DRIVER_OK, 1619
VIRTIO_STATUS_FAILED, 1619
VIRTIO_STATUS_FEATURES_OK, 1619
VRING_DESC_F_WRITE, 1619
walk_virtio_caps, 1623
wmb, 1623
virtio_input.h
VIRTIO_INPUT_EV_KEY, 762
virtio_input_find_keyboard, 762
virtio_input_isr, 763
VIRTIO_INPUT_PCI_DEVICE_ID, 762
VIRTIO_INPUT_PCI_VENDOR_ID, 762
virtio_input_pop_event, 763
VIRTIO_INPUT_EV_KEY
virtio_input.h, 762
virtio_input_find_keyboard
virtio_input.c, 1621
virtio_input.h, 762
virtio_input_isr
interrupts.c, 976, 995
virtio_input.c, 1621
virtio_input.h, 763
VIRTIO_INPUT_PCI_DEVICE_ID
virtio_input.h, 762
VIRTIO_INPUT_PCI_VENDOR_ID
virtio_input.h, 762
virtio_input_pop_event
virtio_input.c, 1622
virtio_input.h, 763
VIRTIO_PCI_CAP_COMMON_CFG
virtio_blk.c, 1607
virtio_input.c, 1618
virtio_rng.c, 1627
VIRTIO_PCI_CAP_DEVICE_CFG
virtio_blk.c, 1607
VIRTIO_PCI_CAP_ISR_CFG
virtio_blk.c, 1607
virtio_input.c, 1618
VIRTIO_PCI_CAP_NOTIFY_CFG
virtio_blk.c, 1607
virtio_input.c, 1618
virtio_rng.c, 1627
VIRTIO_PCI_CAP_VENDOR_ID
virtio_blk.c, 1607
virtio_input.c, 1618
virtio_rng.c, 1627
VIRTIO_PCI_VENDOR_ID
virtio_blk.h, 760
virtio_rng.c
g_vrng, 1633
g_vrng_ready, 1633
rmb, 1628
VCAP_OFF_BAR, 1626
VCAP_OFF_CAP_NEXT, 1626
VCAP_OFF_CAP_VNDR, 1627
VCAP_OFF_CFG_TYPE, 1627
VCAP_OFF_LENGTH, 1627
VCAP_OFF_NOTIFY_MULT, 1627
VCAP_OFF_OFFSET, 1627
VIRTIO_F_VERSION_1, 1627
VIRTIO_PCI_CAP_COMMON_CFG, 1627
VIRTIO_PCI_CAP_NOTIFY_CFG, 1627
VIRTIO_PCI_CAP_VENDOR_ID, 1627
virtio_rng_get_bytes, 1628
virtio_rng_init, 1629
virtio_rng_ready, 1630
VIRTIO_STATUS_ACKNOWLEDGE, 1627
VIRTIO_STATUS_DRIVER, 1628
VIRTIO_STATUS_DRIVER_OK, 1628
VIRTIO_STATUS_FAILED, 1628
VIRTIO_STATUS_FEATURES_OK, 1628
VQUEUE_SIZE, 1628
VRING_DESC_F_WRITE, 1628
VRNG_BUF_SIZE, 1628
vrng_init_device, 1630
vrng_request, 1631
walk_virtio_caps, 1631
wmb, 1632
virtio_rng.h
VIRTIO_RNG_DEVICE_LEGACY, 765
VIRTIO_RNG_DEVICE_MODERN, 765
virtio_rng_get_bytes, 765
virtio_rng_init, 765
VIRTIO_RNG_PCI_VENDOR_ID, 765
virtio_rng_ready, 766
VIRTIO_RNG_DEVICE_LEGACY
virtio_rng.h, 765
VIRTIO_RNG_DEVICE_MODERN
virtio_rng.h, 765
virtio_rng_get_bytes
virtio_rng.c, 1628
virtio_rng.h, 765
virtio_rng_init
virtio_rng.c, 1629

virtio_rng.h, 765
VIRTIO_RNG_PCI_VENDOR_ID
virtio_rng.h, 765
virtio_rng_ready
virtio_rng.c, 1630
virtio_rng.h, 766
VIRTIO_STATUS_ACKNOWLEDGE
virtio_blk.c, 1608
virtio_input.c, 1618
virtio_rng.c, 1627
VIRTIO_STATUS_DRIVER
virtio_blk.c, 1608
virtio_input.c, 1618
virtio_rng.c, 1628
VIRTIO_STATUS_DRIVER_OK
virtio_blk.c, 1608
virtio_input.c, 1619
virtio_rng.c, 1628
VIRTIO_STATUS_FAILED
virtio_blk.c, 1608
virtio_input.c, 1619
virtio_rng.c, 1628
VIRTIO_STATUS_FEATURES_OK
virtio_blk.c, 1608
virtio_input.c, 1619
virtio_rng.c, 1628
VirtioCommonCfg, 309
config_generation, 309
config_msix_vector, 309
device_feature, 309
device_feature_select, 309
device_status, 309
driver_feature, 309
driver_feature_select, 310
num_queues, 310
queue_desc, 310
queue_device, 310
queue_driver, 310
queue_enable, 310
queue_msix_vector, 310
queue_notify_data, 310
queue_notify_off, 311
queue_reset, 311
queue_select, 311
queue_size, 311
VirtioInputEvent, 311
code, 312
type, 312
value, 312
VirtqAvail, 312
flags, 312
idx, 312
ring, 312
used_event, 313
VirtqDesc, 313
addr, 313
flags, 313
len, 313
next, 313
VirtqUsed, 314
avail_event, 314
flags, 314
idx, 314
ring, 315
VirtqUsedElem, 315
id, 315
len, 315
VirtRngState, 316
avail, 316
avail_idx, 316
common, 316
data_buf, 317
desc, 317
last_used_idx, 317
notify, 317
notify_off_mult, 317
queue_notify_off, 317
used, 317
VirtualStart
EFI_MEMORY_DESCRIPTOR, 173
VM, 318
abort_requested, 321
acl_skip, 321
base, 321
base_addr, 321
blk_vm_cbuf, 322
blk_vm_dirty, 322
blk_vm_epoch, 322
blk_vm_lbn, 322
blk_vm_next, 322
cf, 322
compile_buffer, 322
compile_pos, 323
compile_size, 323
compiling_word, 323
current_executing_entry, 323
current_word_name, 323
data_stack, 323
decay_slope_direction, 323
decay_slope_q48, 324
dict_fence_here, 324
dict_fence_latest, 324
dict_lock, 324
doe_row_printing, 324
dsp, 324
ecw_nesting, 325
emergency_console, 326
error, 326
exit_colon, 327
halted, 327
heartbeat, 327
heartbeat_decay_cursor_id, 328
heat_threshold_25th, 328
heat_threshold_50th, 328
heat_threshold_75th, 328
here, 328

hold_addr, 329
hold_pos, 329
host, 329
hot_word_count_at_check, 329
hotwords_cache, 329
in_var, 330
input_buffer, 330
input_length, 330
input_pos, 330
interpreter_enabled, 330
ip, 330
last_bucket_reorg_ns, 331
last_decay_check_ns, 331
last_inference_outputs, 331
latest, 331
lookup_strategy, 331
memory, 332
mode, 332
next_word_id, 332
pipeline_metrics, 332
profiler.h, 1907
recycled_word_id_count, 333
recycled_word_ids, 333
return_stack, 333
rolling_window, 333
rsp, 333
scr_addr, 334
span_var, 334
ssm_config, 334
ssm_l8_state, 334
stale_word_count_at_check, 334
state_addr, 335
state_var, 335
tib_buf, 335
tib_cap, 335
total_heat_at_last_check, 335
tuning_lock, 335
vm.h, 2011
vocab, 336
word_count_at_check, 336
word_id_map, 336
zuse_cert_installed, 336
zuse_cert_pubkey, 336
zuse_cert_seed, 336
zuse_root_pubkey, 337
zuse_root_pubkey_known, 337
zuse_session, 337
vm
InferenceInputs, 216
VM Input Operations, 20
vm_addr_from_ptr, 20
vm_input_ensure, 21
vm_input_in, 22
vm_input_load_line, 22
vm_input_source, 23
vm_input_span, 24
vm_input_tib, 24
vm.c
acl_recheck, 2695
execute_colon_word, 2696
g_vm_interpret_depth, 2712
vm_addr_ok, 2697
vm_compile_call, 2698
vm_compile_exit, 2699
vm_compile_literal, 2699
vm_compile_word, 2700
vm_enter_compile_mode, 2701
vm_exit_compile_mode, 2702
vm_get_base, 2702
vm_interpret, 2703
vm_interpret_word, 2703
vm_load_cell, 2705
vm_load_u8, 2706
vm_make_immediate, 2706
vm_parse_number, 2707
vm_parse_word, 2708
vm_ptr, 2709
vm_set_base, 2710
vm_store_cell, 2710
vm_store_u8, 2711
vm_zuse_cert_install, 2711
vm.h
__attribute__, 2012, 2075
acl_allow, 2075
acl_default, 2075
acl_mode, 2075
ACL_MODE_STRICT, 2006
ACL_MODE_TTL, 2006
acl_pinned, 2075
acl_ttl, 2075
ACL_TTL_OPEN, 2006
ACL_USER_DEFAULT, 2006
BLK_VM_SLOTS, 2006
BLK_VM_WINDOW_BASE, 2006
BLK_VM_WINDOW_SIZE, 2006
BLOCK_SIZE, 2006
CELL, 2012
cell_t, 2010
CF_BEGIN, 2011
CF_CASE, 2011
CF_DO, 2011
CF_ELSE, 2011
CF_IF, 2011
CF_OF, 2011
CF_STACK_MAX, 2006
cf_tag_t, 2011
CF_WHILE, 2011
CHARGE_NEUTRAL, 2006
COMPILE_BUFFER_SIZE, 2006
DICTIONARY_BLOCKS, 2007
DICTIONARY_MEMORY_SIZE, 2007
DICTIONARY_SIZE, 2007
execute_colon_word, 2012
execution_heat, 2075
flags, 2076
func, 2076

heartbeat_capture_tick_snapshot, 2014
heartbeat_emit_tick_row, 2015
HEARTBEAT_TICK_BUFFER_SIZE, 2007
HeartbeatWorker, 2010
HEAT_CACHE_DEMOTION_THRESHOLD, 2007
HotwordsCache, 2010
InferenceOutputs, 2011
INPUT_BUFFER_SIZE, 2007
LIKELY, 2007
link, 2076
LOG_BLOCKS, 2007
LOG_BLOCKS_END, 2007
LOG_BLOCKS_START, 2007
LOG_LAYER1_MAX_LINES, 2008
LOG_LINE_MAX, 2008
LOG_LINES_PER_BLOCK, 2008
MAX_BLOCKS, 2008
MODE_COMPILE, 2012
MODE_INTERPRET, 2012
name, 2076
name_len, 2076
physics, 2076
PHYSICS_STATE_COMPILED, 2008
PHYSICS_STATE_HIDDEN, 2008
PHYSICS_STATE_IMMEDIATE, 2008
PHYSICS_STATE_PINNED, 2008
SPIN_IDLE, 2008
STACK_SIZE, 2008
transition_metrics, 2077
ucell_t, 2011
UNLIKELY, 2009
USER_BLOCKS_START, 2009
vaddr_t, 2011
VM, 2011
VM_ADDR, 2016
vm_addr_ok, 2017
vm_addr_to_block, 2019
vm_align, 2020
vm_allot, 2020
vm_bootstrap_root_vocabulary, 2021
vm_cleanup, 2022
vm_compile_call, 2023
vm_compile_exit, 2024
vm_compile_literal, 2025
vm_compile_word, 2026
vm_create_word, 2027
vm_dict_entry_ok, 2029
vm_dict_resolve_in_bucket, 2030
vm_dictionary_find_by_func, 2031
vm_dictionary_find_latest_by_func, 2032
vm_dictionary_get_data_field, 2032
vm_dictionary_lookup_by_word_id, 2033
vm_dictionary_track_entry, 2034
vm_dictionary_untrack_entry, 2035
vm_enable_interpreter, 2036
vm_enter_compile_mode, 2036
vm_exit_compile_mode, 2038
vm_find_word, 2039
vm_get_base, 2041
vm_get_block_addr, 2041
vm_heartbeat_start, 2042
vm_hide_word, 2042
vm_init, 2043
vm_init_with_host, 2046
vm_interpret, 2048
vm_interpret_word, 2049
vm_load_cell, 2051
vm_load_u8, 2052
vm_make_immediate, 2053
VM_MEMORY_SIZE, 2009
vm_mode_t, 2012
vm_parse_number, 2054
vm_parse_word, 2055
vm_pin_execution_heat, 2057
VM_POP, 2009
vm_pop, 2058
vm_pop_fast, 2059
vm_ptr, 2059
VM_PUSH, 2009
vm_push, 2061
vm_push_fast, 2062
vm_repl, 2063
VM_RPOP, 2009
vm_rpop, 2063
vm_rpop_fast, 2064
VM_RPUSH, 2009
vm_rpush, 2064
vm_rpush_fast, 2065
vm_run_smoke_tests, 2065
vm_set_base, 2065
vm_smudge_word, 2066
vm_snapshot_read, 2067
vm_store_cell, 2067
vm_store_u8, 2068
vm_tick, 2070
vm_tick_inference_engine, 2071
vm_tick_slope_validator, 2072
vm_tick_window_tuner, 2073
vm_unpin_execution_heat, 2073
vm_zuse_cert_install, 2074
VMHostServices, 2011
VOCAB_FC_BUCKETS, 2009
WORD_COMPILED, 2009
WORD_FROZEN, 2010
word_func_t, 2011
WORD_HIDDEN, 2010
word_id, 2077
WORD_ID_INVALID, 2010
WORD_IMMEDIATE, 2010
WORD_NAME_MAX, 2010
WORD_PINNED, 2010
WORD_SMUDGED, 2010
WordTransitionMetrics, 2011
vm_abs_asm
vm_asm_opt_arm64.h, 2081
vm_asm_opt_riscv64.h, 2088

vm_abs_opt
arch_detect.h, 1711
vm_add_check_overflow
Stack Operations, 27
vm_asm_opt_arm64.h, 2081
vm_asm_opt_riscv64.h, 2088
vm_add_overflow_opt
arch_detect.h, 1711
VM_ADDR
vm.h, 2016
vm_addr_from_ptr
VM Input Operations, 20
vm_addr_ok
vm.c, 2697
vm.h, 2017
vm_core.c, 1659
vm_addr_to_block
memory_management.c, 2316
vm.h, 2019
vm_align
memory_management.c, 2317
memory_management.h, 1817
vm.h, 2020
vm_alloc_fn
capsule_birth.c, 1109
vm_allot
memory_management.c, 2318
memory_management.h, 1818
vm.h, 2020
vm_api.c
vm_api_create_word, 2714
vm_api_find_word, 2714
vm_api_hide_word, 2715
vm_api_pop, 2716
vm_api_push, 2716
vm_api_rpop, 2717
vm_api_rpush, 2717
vm_api_smudge_word, 2719
vm_api_create_word
vm_api.c, 2714
vm_api_find_word
vm_api.c, 2714
vm_api_hide_word
vm_api.c, 2715
vm_api_pop
vm_api.c, 2716
vm_api_push
vm_api.c, 2716
vm_api_rpop
vm_api.c, 2717
vm_api_rpush
vm_api.c, 2717
vm_api_smudge_word
vm_api.c, 2719
vm_asm_opt_arm64.h
vm_abs_asm, 2081
vm_add_check_overflow, 2081
vm_clz, 2082
vm_ctz, 2082
vm_dc_zva, 2082
vm_divmod, 2082
vm_has_neon, 2082
vm_is_cortex_a72, 2082
vm_max_asm, 2082
vm_memcpy_asm, 2083
vm_memzero_asm, 2083
vm_min_asm, 2083
vm_mul_double, 2083
vm_pop_asm, 2084
vm_popcnt, 2084
vm_prefetch, 2084
vm_prefetch_stream, 2084
vm_push_asm, 2084
vm_read_midr, 2085
vm_rpop_asm, 2085
vm_rpush_asm, 2085
vm_strcmp_asm, 2086
vm_strcmp_short_asm, 2086
vm_sub_check_overflow, 2087
vm_asm_opt_riscv64.h
vm_abs_asm, 2088
vm_add_check_overflow, 2088
vm_clz, 2089
vm_ctz, 2089
vm_divmod, 2089
vm_has_riscv_b_ext, 2089
vm_has_riscv_m_ext, 2089
vm_max_asm, 2089
vm_memcpy_asm, 2089
vm_memzero_asm, 2089
vm_min_asm, 2090
vm_mul_double, 2090
vm_pop_asm, 2090
vm_popcnt, 2090
vm_prefetch, 2090
vm_prefetch_stream, 2090
vm_push_asm, 2090
vm_rpop_asm, 2091
vm_rpush_asm, 2091
vm_strcmp_asm, 2091
vm_sub_check_overflow, 2091
vm_assert_interpreter_enabled
vm_core.c, 1660
vm_bootstrap.c
vm_bootstrap_root_vocabulary, 2720, 2726
vm_bootstrap_scr, 2721, 2727
vm_cleanup, 2727
vm_init, 2722, 2727
vm_init_with_host, 2723
vm_bootstrap_root_vocabulary
vm.h, 2021
vm_bootstrap.c, 2720, 2726
vm_bootstrap_scr
vm_bootstrap.c, 2721, 2727
vm_check_arena
vm_core.c, 1651

vm_cleanup
vm.h, 2022
vm_bootstrap.c, 2727
vm_core.c, 1661
vm_internal.h, 2751
vm_clz
vm_asm_opt_arm64.h, 2082
vm_asm_opt_riscv64.h, 2089
vm_clz_opt
arch_detect.h, 1711
vm_compile_call
vm.c, 2698
vm.h, 2023
vm_core.c, 1662
vm_compile_exit
vm.c, 2699
vm.h, 2024
vm_core.c, 1663
vm_compile_literal
vm.c, 2699
vm.h, 2025
vm_core.c, 1664
vm_compile_word
vm.c, 2700
vm.h, 2026
vm_core.c, 1665
vm_core.c
acl_recheck, 1651
execute_colon_word, 1652
g_log_attrib_vm, 1693
g_vm_interpret_depth, 1693
hosted_alloc, 1654
hosted_fake_ns, 1693
hosted_free, 1655
hosted_mutex_destroy, 1655
hosted_mutex_init, 1655
hosted_mutex_lock, 1656
hosted_mutex_unlock, 1656
hosted_panic, 1656
hosted_putc, 1657
hosted_puts, 1657
hosted_services, 1693
hosted_time_ns, 1658
hosted_xt_entry_owned, 1658
hosted_xt_is_executable, 1658
sk_vm_at_outermost_interpret, 1658
vm_addr_ok, 1659
vm_assert_interpreter_enabled, 1660
vm_check_arena, 1651
vm_cleanup, 1661
vm_compile_call, 1662
vm_compile_exit, 1663
vm_compile_literal, 1664
vm_compile_word, 1665
vm_default_host_services, 1666
vm_enable_interpreter, 1667
vm_enter_compile_mode, 1667
vm_exit_compile_mode, 1669
vm_get_base, 1670
vm_host, 1672
vm_host_alloc, 1673
vm_host_calloc, 1674
vm_host_free, 1674
vm_interpret, 1675
vm_interpret_word, 1677
vm_load_cell, 1679
vm_load_u8, 1680
vm_log_attributed_vm, 1681
vm_make_immediate, 1681
vm_monotonic_ns, 1682
vm_parse_number, 1683
vm_parse_word, 1684
vm_ptr, 1686
vm_reset_hosted_fake_ns, 1688
vm_set_base, 1689
vm_store_cell, 1689
vm_store_u8, 1691
vm_zuse_cert_install, 1692
vm_cpuid
Stack Operations, 27
vm_create_word
dictionary_management.c, 2246
vm.h, 2027
vm_ctz
vm_asm_opt_arm64.h, 2082
vm_asm_opt_riscv64.h, 2089
vm_ctz_opt
arch_detect.h, 1712
vm_cycles
Profiler, 250
vm_dc_zva
vm_asm_opt_arm64.h, 2082
vm_debug.c
dbg_get_base, 2729
dump_block_preview, 2729
g_vm, 2734
mode_str, 2730
sig_handler, 2731
vm_debug_dump_state, 2731
vm_debug_install_signal_handlers, 2732
vm_debug_set_current_vm, 2733
vm_debug.h
vm_debug_dump_state, 2092
vm_debug_install_signal_handlers, 2093
vm_debug_set_current_vm, 2094
vm_debug_dump_state
vm_debug.c, 2731
vm_debug.h, 2092
vm_debug_install_signal_handlers
vm_debug.c, 2732
vm_debug.h, 2093
vm_debug_set_current_vm
vm_debug.c, 2733
vm_debug.h, 2094
vm_default_host_services
vm_core.c, 1666

vm_internal.h, 2738
vm_dict_entry_ok
dictionary_management.c, 2248
vm.h, 2029
vm_dict_hash_fn
capsule_birth.c, 1109
vm_dict_resolve_in_bucket
dictionary_management.c, 2249
vm.h, 2030
vm_dictionary_acquire_word_id
dictionary_management.c, 2250
vm_dictionary_find_by_func
dictionary_management.c, 2251
vm.h, 2031
vm_dictionary_find_latest_by_func
dictionary_management.c, 2252
vm.h, 2032
vm_dictionary_get_data_field
dictionary_management.c, 2252
vm.h, 2032
vm_dictionary_index_reset
dictionary_management.c, 2253
vm_dictionary_lookup_by_word_id
dictionary_management.c, 2253
vm.h, 2033
vm_dictionary_track_entry
dictionary_management.c, 2254
vm.h, 2034
vm_dictionary_untrack_entry
dictionary_management.c, 2255
vm.h, 2035
vm_divmod
Stack Operations, 28
vm_asm_opt_arm64.h, 2082
vm_asm_opt_riscv64.h, 2089
vm_divmod_opt
arch_detect.h, 1712
vm_docol_arm64
vm_inner_interp_arm64.h, 2102
vm_docol_asm
vm_inner_interp_asm.h, 2111
vm_docol_rv64
vm_inner_interp_riscv64.h, 2120
vm_enable_interpreter
vm.h, 2036
vm_core.c, 1667
vm_enter_compile_mode
vm.c, 2701
vm.h, 2036
vm_core.c, 1667
vm_exec_fn
capsule_birth.c, 1109
vm_exit_arm64
vm_inner_interp_arm64.h, 2102
vm_exit_asm
vm_inner_interp_asm.h, 2111
vm_exit_compile_mode
vm.c, 2702
vm.h, 2038
vm_core.c, 1669
vm_exit_rv64
vm_inner_interp_riscv64.h, 2120
vm_find_entry_ptr
capsule_birth.c, 1106
vm_find_word
dictionary_management.c, 2256
vm.h, 2039
VM_FLEET_WINDOW_DEPTH
capsule_vm_physics.c, 1159
vm_get_arch_name
arch_detect.h, 1714
vm_get_base
vm.c, 2702
vm.h, 2041
vm_core.c, 1670
vm_internal.h, 2739, 2752
vm_get_block_addr
memory_management.c, 2320
vm.h, 2041
vm_get_build_info
arch_detect.h, 1714
vm_has_asm_opt
arch_detect.h, 1715
vm_has_direct_threading
arch_detect.h, 1715
vm_has_neon
vm_asm_opt_arm64.h, 2082
vm_has_riscv_b_ext
vm_asm_opt_riscv64.h, 2089
vm_has_riscv_m_ext
vm_asm_opt_riscv64.h, 2089
vm_has_sse42
Stack Operations, 28
vm_has_sse42_opt
arch_detect.h, 1716
vm_heartbeat_publish_snapshot
vm_internal.h, 2752
vm_time.c, 2766
vm_heartbeat_run_cycle
vm_internal.h, 2740, 2753
vm_runtime.c, 1698
vm_time.c, 2766
vm_heartbeat_start
vm.h, 2042
vm_heartbeat_start_thread
vm_internal.h, 2755
vm_time.c, 2767
vm_heartbeat_stop_thread
vm_internal.h, 2756
vm_time.c, 2767
vm_heartbeat_update_l8
vm_runtime.c, 1700
vm_time.c, 2768
vm_hide_word
dictionary_management.c, 2258
vm.h, 2042

vm_host
vm_core.c, 1672
vm_internal.h, 2741
vm_host.h
SK_ALLOC, 2097
SK_ALLOC_ALIGNED, 2097
sk_alloc_fn, 2098
SK_FREE, 2097
sk_free_fn, 2098
sk_host_services, 2100
SK_MUTEX_DESTROY, 2097
sk_mutex_destroy_fn, 2098
SK_MUTEX_INIT, 2097
sk_mutex_init_fn, 2098
SK_MUTEX_LOCK, 2097
sk_mutex_lock_fn, 2098
SK_MUTEX_UNLOCK, 2097
sk_mutex_unlock_fn, 2098
sk_panic_fn, 2099
SK_PUTC, 2097
sk_putc_fn, 2099
SK_PUTS, 2098
sk_puts_fn, 2099
sk_time_fn, 2099
SK_TIME_NS, 2098
sk_xt_entry_owns_fn, 2099
sk_xt_validate_fn, 2099
VMHostServices, 2099
vm_host_alloc
vm_core.c, 1673
vm_internal.h, 2742
vm_host_calloc
vm_core.c, 1674
vm_internal.h, 2743
vm_host_free
vm_core.c, 1674
vm_internal.h, 2743
vm_id
CapsuleRunRecord, 133
vm_physics_node, 339
VMRegistryEntry, 347
vm_identity.c
vm_identity_from_cert, 1193
vm_identity_has_cap, 1194
vm_identity_from_cert
vm_identity.c, 1193
vm_identity_has_cap
vm_identity.c, 1194
vm_init
vm.h, 2043
vm_bootstrap.c, 2722, 2727
vm_internal.h, 2757
vm_init_with_host
vm.h, 2046
vm_bootstrap.c, 2723
vm_inner_interp_arm64.h
NEXT_ARM64, 2101
vm_docol_arm64, 2102
vm_exit_arm64, 2102
vm_save_registers_arm64, 2102
vm_setup_registers_arm64, 2102
vm_inner_interp_asm.h
NEXT_ASM, 2104
PRIM_2DROP, 2104
PRIM_2DUP, 2104
PRIM_AND, 2105
PRIM_BRANCH, 2105
PRIM_C_FETCH, 2105
PRIM_C_STORE, 2105
PRIM_DROP, 2105
PRIM_DUP, 2106
PRIM_EQUALS, 2106
PRIM_EXECUTE, 2106
PRIM_FETCH, 2106
PRIM_GREATER, 2106
PRIM_INVERT, 2107
PRIM_LESS, 2107
PRIM_MINUS, 2107
PRIM_MOD, 2107
PRIM_NEGATE, 2107
PRIM_OR, 2108
PRIM_OVER, 2108
PRIM_PLUS, 2108
PRIM_R_FETCH, 2108
PRIM_R_FROM, 2108
PRIM_ROT, 2109
PRIM_SLASH, 2109
PRIM_STAR, 2109
PRIM_STORE, 2109
PRIM_SWAP, 2109
PRIM_TO_R, 2110
PRIM_XOR, 2110
PRIM_ZBRANCH, 2110
PRIM_ZERO_EQUALS, 2110
PRIM_ZERO_LESS, 2110
vm_docol_asm, 2111
vm_exit_asm, 2111
vm_save_registers, 2111
vm_setup_registers, 2111
vm_inner_interp_riscv64.h
NEXT_RV64, 2113
PRIM_2DROP_RV64, 2113
PRIM_2DUP_RV64, 2113
PRIM_AND_RV64, 2113
PRIM_BRANCH_RV64, 2114
PRIM_C_FETCH_RV64, 2114
PRIM_C_STORE_RV64, 2114
PRIM_DROP_RV64, 2114
PRIM_DUP_RV64, 2114
PRIM_EQUALS_RV64, 2115
PRIM_EXECUTE_RV64, 2115
PRIM_FETCH_RV64, 2115
PRIM_GREATER_RV64, 2115
PRIM_INVERT_RV64, 2115
PRIM_LESS_RV64, 2116
PRIM_MINUS_RV64, 2116

PRIM_MOD_RV64, 2116
PRIM_NEGATE_RV64, 2116
PRIM_OR_RV64, 2116
PRIM_OVER_RV64, 2117
PRIM_PLUS_RV64, 2117
PRIM_R_FETCH_RV64, 2117
PRIM_R_FROM_RV64, 2117
PRIM_ROT_RV64, 2117
PRIM_SLASH_RV64, 2118
PRIM_STAR_RV64, 2118
PRIM_STORE_RV64, 2118
PRIM_SWAP_RV64, 2118
PRIM_TO_R_RV64, 2118
PRIM_XOR_RV64, 2119
PRIM_ZBRANCH_RV64, 2119
PRIM_ZERO_EQUALS_RV64, 2119
PRIM_ZERO_LESS_RV64, 2119
vm_docol_rv64, 2120
vm_exit_rv64, 2120
vm_save_registers_rv64, 2120
vm_setup_registers_rv64, 2120
vm_input_ensure
VM Input Operations, 21
vm_input_in
VM Input Operations, 22
vm_input_load_line
VM Input Operations, 22
vm_input_source
VM Input Operations, 23
vm_input_span
VM Input Operations, 24
vm_input_tib
VM Input Operations, 24
vm_internal.h
execute_colon_word, 2748
HEARTBEAT_DECAY_BATCH, 2735, 2747
HEARTBEAT_HAS_THREADS, 2735, 2747
heartbeat_publish_snapshot, 2736
heartbeat_snapshot_index_load, 2737, 2750
heartbeat_snapshot_index_store, 2737, 2751
HeartbeatWorker, 2736, 2748
PARITY_MODE, 2735
SK_PARITY_DEBUG, 2735
vm_cleanup, 2751
vm_default_host_services, 2738
vm_get_base, 2739, 2752
vm_heartbeat_publish_snapshot, 2752
vm_heartbeat_run_cycle, 2740, 2753
vm_heartbeat_start_thread, 2755
vm_heartbeat_stop_thread, 2756
vm_host, 2741
vm_host_alloc, 2742
vm_host_calloc, 2743
vm_host_free, 2743
vm_init, 2757
vm_monotonic_ns, 2744
vm_reset_hosted_fake_ns, 2745
vm_set_base, 2745, 2758
vm_snapshot_read, 2760
vm_tick, 2760
vm_tick_inference_engine, 2761
vm_tick_slope_validator, 2763
vm_tick_window_tuner, 2764
vm_interpret
vm.c, 2703
vm.h, 2048
vm_core.c, 1675
vm_interpret_word
vm.c, 2703
vm.h, 2049
vm_core.c, 1677
vm_is_cortex_a72
vm_asm_opt_arm64.h, 2082
vm_load_cell
vm.c, 2705
vm.h, 2051
vm_core.c, 1679
vm_load_u8
vm.c, 2706
vm.h, 2052
vm_core.c, 1680
vm_log_attributed_vm
log_attrib.h, 642
vm_core.c, 1681
vm_make_immediate
vm.c, 2706
vm.h, 2053
vm_core.c, 1681
vm_max_asm
Stack Operations, 29
vm_asm_opt_arm64.h, 2082
vm_asm_opt_riscv64.h, 2089
vm_max_opt
arch_detect.h, 1712
vm_memcpy_asm
Stack Operations, 29
vm_asm_opt_arm64.h, 2083
vm_asm_opt_riscv64.h, 2089
vm_memcpy_opt
arch_detect.h, 1712
VM_MEMORY_SIZE
vm.h, 2009
vm_memzero_asm
Stack Operations, 31
vm_asm_opt_arm64.h, 2083
vm_asm_opt_riscv64.h, 2089
vm_memzero_opt
arch_detect.h, 1712
vm_min_asm
Stack Operations, 31
vm_asm_opt_arm64.h, 2083
vm_asm_opt_riscv64.h, 2090
vm_min_opt
arch_detect.h, 1712
vm_mode
timer_calibration_record, 285

vm_mode_t
vm.h, 2012
vm_monotonic_ns
vm_core.c, 1682
vm_internal.h, 2744
vm_mul_double
Stack Operations, 31
vm_asm_opt_arm64.h, 2083
vm_asm_opt_riscv64.h, 2090
vm_mul_double_opt
arch_detect.h, 1712
vm_name_copy
capsule_birth.c, 1106
vm_name_eq
capsule_birth.c, 1106
vm_name_eq_nocase
capsule_birth.c, 1107
VM_NAME_MAX
capsule_run.h, 464
vm_name_prefix_eq_nocase
capsule_birth.c, 1107
vm_node, 338
entry, 338
next, 338
vm_node_t
capsule_birth.c, 1091
vm_ns_base
timer.c, 1051
vm_parse_number
vm.c, 2707
vm.h, 2054
vm_core.c, 1683
vm_parse_word
vm.c, 2708
vm.h, 2055
vm_core.c, 1684
vm_physics_conserved
capsule_vm_physics.c, 1161
capsule_vm_physics.h, 476
VM_PHYSICS_EPSILON_Q48
capsule_vm_physics.c, 1159
vm_physics_find
capsule_vm_physics.c, 1162
vm_physics_find_root_id
capsule_vm_physics.c, 1163
vm_physics_fleet_heat_sum
capsule_vm_physics.c, 1163
capsule_vm_physics.h, 476
vm_physics_head
capsule_vm_physics.c, 1171
vm_physics_heartbeat_tick
capsule_vm_physics.c, 1164
capsule_vm_physics.h, 477
vm_physics_heat_of
capsule_vm_physics.c, 1165
capsule_vm_physics.h, 478
vm_physics_init
capsule_vm_physics.c, 1166
capsule_vm_physics.h, 479
VM_PHYSICS_MIN_INFORMATIVE_SAMPLES
capsule_vm_physics.c, 1160
vm_physics_node, 339
next, 339
physics, 339
vm_id, 339
vm_physics_node_t
capsule_vm_physics.c, 1160
vm_physics_retire
capsule_vm_physics.c, 1167
capsule_vm_physics.h, 480
vm_physics_status
capsule_vm_physics.c, 1167
capsule_vm_physics.h, 480
vm_physics_tick
capsule_vm_physics.c, 1168
capsule_vm_physics.h, 481
vm_physics_touch
capsule_vm_physics.c, 1169
capsule_vm_physics.h, 482
vm_physics_transfer
capsule_vm_physics.c, 1170
vm_pin_execution_heat
dictionary_management.c, 2259
vm.h, 2057
vm_pm_start
timer.c, 1051
VM_POP
vm.h, 2009
vm_pop
stack_management.c, 2507
vm.h, 2058
vm_pop_asm
Stack Operations, 32
vm_asm_opt_arm64.h, 2084
vm_asm_opt_riscv64.h, 2090
vm_pop_fast
vm.h, 2059
vm_pop_opt
arch_detect.h, 1713
vm_popcnt
vm_asm_opt_arm64.h, 2084
vm_asm_opt_riscv64.h, 2090
vm_popcnt_opt
arch_detect.h, 1713
vm_prefetch
vm_asm_opt_arm64.h, 2084
vm_asm_opt_riscv64.h, 2090
vm_prefetch_opt
arch_detect.h, 1713
vm_prefetch_stream
vm_asm_opt_arm64.h, 2084
vm_asm_opt_riscv64.h, 2090
vm_prefetch_stream_opt
arch_detect.h, 1713
vm_ptr
vm.c, 2709

vm.h, 2059
vm_core.c, 1686
VMRegistryEntry, 347
VM_PUSH
vm.h, 2009
vm_push
stack_management.c, 2509
vm.h, 2061
vm_push_asm
Stack Operations, 33
vm_asm_opt_arm64.h, 2084
vm_asm_opt_riscv64.h, 2090
vm_push_fast
vm.h, 2062
vm_push_opt
arch_detect.h, 1713
vm_read_midr
vm_asm_opt_arm64.h, 2085
vm_registry_alloc
capsule_birth.c, 1108
vm_registry_count
capsule_birth.c, 1110
vm_registry_head
capsule_birth.c, 1110
vm_registry_live_count
capsule_birth.c, 1108
vm_repl
repl.c, 2469
repl.h, 1959
vm.h, 2063
vm_reset_hosted_fake_ns
vm_core.c, 1688
vm_internal.h, 2745
VM_RPOP
vm.h, 2009
vm_rpop
stack_management.c, 2510
vm.h, 2063
vm_rpop_asm
Stack Operations, 35
vm_asm_opt_arm64.h, 2085
vm_asm_opt_riscv64.h, 2091
vm_rpop_fast
vm.h, 2064
vm_rpop_opt
arch_detect.h, 1713
VM_RPUSH
vm.h, 2009
vm_rpush
stack_management.c, 2511
vm.h, 2064
vm_rpush_asm
Stack Operations, 36
vm_asm_opt_arm64.h, 2085
vm_asm_opt_riscv64.h, 2091
vm_rpush_fast
vm.h, 2065
vm_rpush_opt
arch_detect.h, 1713
vm_run_smoke_tests
vm.h, 2065
vm_runtime.c
heartbeat_publish_snapshot, 1695
heartbeat_snapshot_index_load, 1695
heartbeat_snapshot_index_store, 1696
heartbeat_thread_main, 1697
vm_heartbeat_run_cycle, 1698
vm_heartbeat_update_l8, 1700
vm_snapshot_read, 1701
vm_tick, 1702
vm_tick_apply_background_decay, 1703
vm_tick_inference_engine, 1704
vm_tick_slope_validator, 1706
vm_tick_window_tuner, 1707
vm_save_registers
vm_inner_interp_asm.h, 2111
vm_save_registers_arm64
vm_inner_interp_arm64.h, 2102
vm_save_registers_opt
arch_detect.h, 1713
vm_save_registers_rv64
vm_inner_interp_riscv64.h, 2120
vm_set_base
vm.c, 2710
vm.h, 2065
vm_core.c, 1689
vm_internal.h, 2745, 2758
vm_setup_registers
vm_inner_interp_asm.h, 2111
vm_setup_registers_arm64
vm_inner_interp_arm64.h, 2102
vm_setup_registers_opt
arch_detect.h, 1714
vm_setup_registers_rv64
vm_inner_interp_riscv64.h, 2120
vm_smudge_word
dictionary_management.c, 2259
vm.h, 2066
vm_snapshot_read
vm.h, 2067
vm_internal.h, 2760
vm_runtime.c, 1701
vm_time.c, 2769
VM_STATE_DEAD
capsule_run.h, 465
VM_STATE_EMBRYO
capsule_run.h, 465
VM_STATE_LIVE
capsule_run.h, 465
VM_STATE_STILLBORN
capsule_run.h, 465
VM_STATE_STOPPED
capsule_run.h, 465
VM_STATE_SWITCHED_OUT
capsule_run.h, 465
vm_store_cell

vm.c, 2710
vm.h, 2067
vm_core.c, 1689
vm_store_u8
vm.c, 2711
vm.h, 2068
vm_core.c, 1691
vm_strcmp_asm
Stack Operations, 37
vm_asm_opt_arm64.h, 2086
vm_asm_opt_riscv64.h, 2091
vm_strcmp_opt
arch_detect.h, 1714
vm_strcmp_short_asm
vm_asm_opt_arm64.h, 2086
VM_STRICT_PTR
starforth_config.h, 1997
vm_sub_check_overflow
Stack Operations, 37
vm_asm_opt_arm64.h, 2087
vm_asm_opt_riscv64.h, 2091
vm_sub_overflow_opt
arch_detect.h, 1714
vm_tick
vm.h, 2070
vm_internal.h, 2760
vm_runtime.c, 1702
vm_time.c, 2770
vm_tick_apply_background_decay
vm_runtime.c, 1703
vm_time.c, 2770
vm_tick_inference_engine
vm.h, 2071
vm_internal.h, 2761
vm_runtime.c, 1704
vm_time.c, 2771
vm_tick_slope_validator
vm.h, 2072
vm_internal.h, 2763
vm_runtime.c, 1706
vm_time.c, 2772
vm_tick_window_tuner
vm.h, 2073
vm_internal.h, 2764
vm_runtime.c, 1707
vm_time.c, 2772
vm_time.c
vm_heartbeat_publish_snapshot, 2766
vm_heartbeat_run_cycle, 2766
vm_heartbeat_start_thread, 2767
vm_heartbeat_stop_thread, 2767
vm_heartbeat_update_l8, 2768
vm_snapshot_read, 2769
vm_tick, 2770
vm_tick_apply_background_decay, 2770
vm_tick_inference_engine, 2771
vm_tick_slope_validator, 2772
vm_tick_window_tuner, 2772
vm_timebase_init_from_pmtimer
timer.c, 1049
vm_to_lower
capsule_birth.c, 1109
vm_unpin_execution_heat
dictionary_management.c, 2260
vm.h, 2073
vm_vocabulary_find_word
vocabulary_words.c, 3254
vm_workload_duration_ns_q48
DoeMetrics, 160
vm_zuse_cert_install
vm.c, 2711
vm.h, 2074
vm_core.c, 1692
VMFleetTouchSample, 340
amount, 340
clamped, 340
elapsed_ticks, 340
VMFleetWindow, 340
count, 341
head, 341
is_warm, 341
touch_samples, 341
VMHostServices, 341
alloc, 342
free, 342
is_executable_ptr, 342
monotonic_ns, 342
mutex_destroy, 342
mutex_init, 343
mutex_lock, 343
mutex_unlock, 343
owns_xt_entry, 343
panic, 343
parity_mode, 343
putc, 343
puts, 343
verbose, 343
vm.h, 2011
vm_host.h, 2099
vmm.c
ADDR_MASK, 1470
alloc_page_zeroed, 1471
get_or_alloc_table, 1471
get_table, 1472
hpet_phys_base, 1481
invlpg, 1473
is_ram_type, 1473
lapic_phys_base, 1481
load_cr3, 1473
make_pte, 1474
paddr_to_virt, 1474
pd_index, 1474
pdpt_index, 1475
pml4_index, 1475
pt_index, 1475
PTE_ACCESSED, 1470

PTE_DIRTY, 1470
PTE_GLOBAL, 1470
PTE_NX, 1470
PTE_PCD, 1470
PTE_PRESENT, 1470
PTE_PS, 1471
PTE_PWT, 1471
PTE_USER, 1471
PTE_WRITABLE, 1471
resolve_pt, 1476
vmm_get_paddr, 1477
vmm_init, 1477
vmm_map_page, 1478
vmm_map_range, 1479
vmm_query_page, 1480
vmm_root_pml4_phys, 1481
vmm_unmap_page, 1480
vmm.h
VMM_FLAG_CACHE_DISABLE, 779
VMM_FLAG_NX, 779
VMM_FLAG_PRESENT, 779
VMM_FLAG_USER, 779
VMM_FLAG_WRITABLE, 779
vmm_get_paddr, 780
vmm_init, 780
vmm_map_page, 781
vmm_map_range, 782
VMM_PAGE_SIZE, 779
vmm_query_page, 783
vmm_unmap_page, 783
VMM_FLAG_CACHE_DISABLE
vmm.h, 779
VMM_FLAG_NX
vmm.h, 779
VMM_FLAG_PRESENT
vmm.h, 779
VMM_FLAG_USER
vmm.h, 779
VMM_FLAG_WRITABLE
vmm.h, 779
vmm_get_paddr
vmm.c, 1477
vmm.h, 780
vmm_init
vmm.c, 1477
vmm.h, 780
vmm_map_page
vmm.c, 1478
vmm.h, 781
vmm_map_range
vmm.c, 1479
vmm.h, 782
vmm_page_info_t, 344
executable, 344
present, 344
writable, 344
VMM_PAGE_SIZE
vmm.h, 779
vmm_query_page
vmm.c, 1480
vmm.h, 783
vmm_root_pml4_phys
vmm.c, 1481
vmm_unmap_page
vmm.c, 1480
vmm.h, 783
VMPhysics, 345
execution_heat_q48, 345
is_live, 345
last_active_tick, 345
touched, 345
VMRegistryEntry, 345
birth_capsule_id, 346
birth_dict_hash, 346
birth_timestamp_ns, 346
flags, 346
name, 346
parent_vm_id, 346
pending_reap, 347
stadium_patron_cell, 347
state, 347
vm_id, 347
vm_ptr, 347
VMState
capsule_run.h, 465
vocab
VM, 336
VOCAB_FC_BUCKETS
vm.h, 2009
vocab_find_word
vocabulary_words.c, 3255
vocab_safe_len
vocabulary_words.c, 3256
vocab_sync_vm_vars
vocabulary_words.c, 3257
vocabulary_create_vocabulary_direct
vocabulary_words.c, 3257
vocabulary_words.h, 3026
vocabulary_debug_print_hex
vocabulary_words.c, 3258
vocabulary_select_runtime
vocabulary_words.c, 3259
vocabulary_word_context
vocabulary_words.c, 3259
vocabulary_word_current
vocabulary_words.c, 3260
vocabulary_word_definitions
vocabulary_words.c, 3261
vocabulary_words.h, 3027
vocabulary_word_forth
vocabulary_words.c, 3262
vocabulary_words.h, 3028
vocabulary_word_order
vocabulary_words.c, 3262
vocabulary_word_paren_find
vocabulary_words.c, 3263

vocabulary_word_suites
vocabulary_words_test.c, 2651
vocabulary_word_vocabulary
vocabulary_words.c, 3264
vocabulary_words.c
fc_free, 3250
fc_rebuild, 3251
init_vocabulary_system, 3252
register_vocabulary_words, 3253
vm_vocabulary_find_word, 3254
vocab_find_word, 3255
vocab_safe_len, 3256
vocab_sync_vm_vars, 3257
vocabulary_create_vocabulary_direct, 3257
vocabulary_debug_print_hex, 3258
vocabulary_select_runtime, 3259
vocabulary_word_context, 3259
vocabulary_word_current, 3260
vocabulary_word_definitions, 3261
vocabulary_word_forth, 3262
vocabulary_word_order, 3262
vocabulary_word_paren_find, 3263
vocabulary_word_vocabulary, 3264
vocabulary_words.h
register_vocabulary_words, 3025
vocabulary_create_vocabulary_direct, 3026
vocabulary_word_definitions, 3027
vocabulary_word_forth, 3028
vocabulary_words_test.c
run_vocabulary_words_tests, 2650
vocabulary_word_suites, 2651
VocabularyState, 348
context_var_addr, 348
context_vocab, 348
ctx_cached_head, 348
ctx_fc, 349
ctx_n, 349
current_var_addr, 349
current_vocab, 349
forth_cached_head, 349
forth_fc, 349
forth_n, 349
forth_vocab, 349
initialized, 350
VOID
uefi.h, 730
vol_meta
blk_dev_slot, 107
vol_meta_dirty
blk_dev_slot, 107
volatility_high_threshold_q16
CDTuning, 140
volatility_q16
CDRegimeMetrics, 138
volmeta_from_buf
block_subsystem.c, 2208
volmeta_to_buf
block_subsystem.c, 2209
VQUEUE_SIZE
virtio_blk.c, 1608
virtio_rng.c, 1628
VRING_DESC_F_NEXT
virtio_blk.c, 1608
VRING_DESC_F_WRITE
virtio_blk.c, 1608
virtio_input.c, 1619
virtio_rng.c, 1628
VRNG_BUF_SIZE
virtio_rng.c, 1628
vrng_init_device
virtio_rng.c, 1630
vrng_request
virtio_rng.c, 1631
vsnprintf
stdio.h, 591
vt
blkio_dev, 117
vt100.c
advance_cursor, 1360
ansi256_to_rgb, 1361
apply_sgr, 1362
cell_h, 1363
cell_w, 1364
dispatch_csi, 1365
draw_cursor_glyph, 1366
eff_bg, 1368
eff_fg, 1369
erase_display, 1370
erase_line, 1371
erase_line_range, 1372
g_boot_line_stride, 1392
g_boot_ring, 1392
g_boot_ring_cap, 1393
g_boot_ring_count, 1393
g_boot_ring_head, 1393
g_boot_rows, 1393
g_boot_shadow, 1393
g_glyph_mode, 1393
g_line_stride, 1393
g_ring, 1394
g_ring_cap, 1394
g_ring_count, 1394
g_ring_head, 1394
g_scroll_offset, 1394
g_shadow, 1394
g_terminal_visible, 1394
g_ttf_cache, 1395
g_ttf_font, 1395
g_ttf_ready, 1395
g_vt, 1395
param, 1373
ParseState, 1359
put_char, 1374
px_of, 1375
py_of, 1376
reset_attrs, 1377

reset_parser, 1377
scroll_up, 1378
scrollback_line_at, 1379
scrollback_redraw, 1380
set_cursor, 1381
ST_CSI, 1360
ST_ESC, 1360
ST_ESC_CHAR, 1360
ST_ESC_HASH, 1360
ST_NORMAL, 1360
ttf_draw_glyph_cell, 1381
UNDERLINE_ROW, 1358
VT100_BOOT_SCROLLBACK_MAX_LINES, 1358
vt100_cols, 1382
vt100_draw_cursor, 1382
vt100_enable_ttf, 1383
vt100_erase_cursor, 1384
vt100_init, 1385
VT100_MAX_PARAMS, 1358
vt100_putc, 1386
vt100_puts, 1388
vt100_reset, 1389
vt100_rows, 1390
vt100_scroll_back, 1390
vt100_scroll_fwd, 1391
VT100_SCROLLBACK_MAX_LINES, 1359
vt100_toggle_graphics, 1391
VT100_TTF_CACHE_SLOTS, 1359
VT100_TTF_CELL_H_PX, 1359
VT100_TTF_CELL_W_PX, 1359
VT100_TTF_FONT_CAPSULE, 1359
VT100_TTF_SIZE_PX, 1359
VT_GLYPH_BITMAP, 1360
vt_glyph_mode_t, 1360
VT_GLYPH_TTF, 1360
vt100.h
vt100_cols, 786
VT100_DEFAULT_BG, 786
VT100_DEFAULT_FG, 786
vt100_draw_cursor, 786
vt100_enable_ttf, 787
vt100_erase_cursor, 788
vt100_init, 789
vt100_putc, 790
vt100_puts, 792
vt100_reset, 793
vt100_rows, 794
vt100_scroll_back, 794
vt100_scroll_fwd, 795
vt100_toggle_graphics, 795
VT100_BOOT_SCROLLBACK_MAX_LINES
vt100.c, 1358
vt100_cols
vt100.c, 1382
vt100.h, 786
VT100_DEFAULT_BG
vt100.h, 786
VT100_DEFAULT_FG
vt100.h, 786
vt100_draw_cursor
vt100.c, 1382
vt100.h, 786
vt100_enable_ttf
vt100.c, 1383
vt100.h, 787
vt100_erase_cursor
vt100.c, 1384
vt100.h, 788
vt100_init
vt100.c, 1385
vt100.h, 789
VT100_MAX_PARAMS
vt100.c, 1358
vt100_putc
vt100.c, 1386
vt100.h, 790
vt100_puts
vt100.c, 1388
vt100.h, 792
vt100_reset
vt100.c, 1389
vt100.h, 793
vt100_rows
vt100.c, 1390
vt100.h, 794
vt100_scroll_back
vt100.c, 1390
vt100.h, 794
vt100_scroll_fwd
vt100.c, 1391
vt100.h, 795
VT100_SCROLLBACK_MAX_LINES
vt100.c, 1359
vt100_toggle_graphics
vt100.c, 1391
vt100.h, 795
VT100_TTF_CACHE_SLOTS
vt100.c, 1359
VT100_TTF_CELL_H_PX
vt100.c, 1359
VT100_TTF_CELL_W_PX
vt100.c, 1359
VT100_TTF_FONT_CAPSULE
vt100.c, 1359
VT100_TTF_SIZE_PX
vt100.c, 1359
VT100State, 350
bg, 350
bold, 350
cols, 351
cx, 351
cy, 351
def_bg, 351
def_fg, 351
fg, 351
initialized, 351

param_count, 352
param_has_value, 352
params, 352
reverse, 352
rows, 352
saved_cx, 352
saved_cy, 352
state, 353
underline, 353
VT_GLYPH_BITMAP
vt100.c, 1360
vt_glyph_mode_t
vt100.c, 1360
VT_GLYPH_TTF
vt100.c, 1360

WaitForEvent
_EFI_BOOT_SERVICES, 90
walk_virtio_caps
virtio_blk.c, 1613
virtio_input.c, 1623
virtio_rng.c, 1631
WARMUP_WINDOWS
timer.c, 1017
WBITS
fe25519.c, 1207
width
FbState, 186
FramebufferInfo, 191
Rpi5FramebufferInfo, 260
ttf_bitmap_t, 288
ttf_raster_cache_slot_t, 296
wIndex
usb_setup_packet_t, 301
window
InferenceInputs, 216
time_trust_state, 282
window_diversity_percent
DoeMetrics, 160
window_final_size_bytes
DoeMetrics, 160
window_hash64
ParityPacket, 226
window_pos
rolling_window_view_t, 255
RollingWindowOfTruth, 258
WINDOW_TICKS_HPET
timer.c, 1017
WINDOW_TICKS_PIT
timer.c, 1017
window_tuning_checks
DoeMetrics, 160
PipelineGlobalMetrics, 246
window_variance_q48
DoeMetrics, 160
InferenceOutputs, 218
window_width
HeartbeatSnapshot, 197
HeartbeatTickSnapshot, 204
windows_used
timer.c, 1051
timer_calibration_record, 285
WIREBIND_CERT_MAX_DEVBLOCKS
capsule_wirebind.c, 1173
wirebind_live_alloc
capsule_wirebind.c, 1180
wirebind_live_entry_t, 353
dev, 354
in_use, 354
switch_slot, 354
user_vm_id, 354
username, 354
wirebind_live_find_by_dev
capsule_wirebind.c, 1180
wirebind_live_remove
capsule_wirebind.c, 1180
WIREBIND_MAX_LIVE
capsule_wirebind.c, 1173
WIREBIND_OVERFLOW_CLAIM_DEVBLOCKS
capsule_wirebind.c, 1173
WIREBIND_OVERFLOW_FREE_THRESHOLD_PCT
capsule_wirebind.c, 1173
wirebind_resolve_attached
capsule_wirebind.c, 1181
wirebind_teardown_console
capsule_wirebind.c, 1181
wLength
usb_setup_packet_t, 301
wmb
virtio_blk.c, 1614
virtio_input.c, 1623
virtio_rng.c, 1632
word_capacity
profiler.c, 2441
WORD_COMPILED
vm.h, 2009
word_count
InferenceInputs, 217
ParityPacket, 226
profiler.c, 2442
word_count_at_check
VM, 336
word_defer
defer_words.c, 2839
word_defer_fetch
defer_words.c, 2839
word_divide
arithmetic_words.c, 2804
word_divmod
arithmetic_words.c, 2805
word_executions_delta
HeartbeatTickSnapshot, 204
WORD_FROZEN
vm.h, 2010
word_func_t
vm.h, 2011
WORD_HIDDEN

vm.h, 2010
word_id
DictEntry, 150
vm.h, 2077
WORD_ID_INVALID
vm.h, 2010
word_id_map
VM, 336
WORD_IMMEDIATE
vm.h, 2010
word_is
defer_words.c, 2840
word_name
ProfileScope, 251
WordProfile, 356
WordTestSuite, 360
WORD_NAME_MAX
vm.h, 2010
WORD_PINNED
vm.h, 2010
word_profile_capacity
Profiler, 250
word_profile_count
Profiler, 250
word_profiles
Profiler, 250
word_registry.c
register_forth79_words, 2774
register_word, 2776
word_registry.h
register_forth79_words, 2121
register_word, 2123
word_scratch_addr
string_words.c, 3232
WORD_SCRATCH_CAP
string_words.c, 3213
WORD_SMUDGED
vm.h, 2010
word_stats
profiler.c, 2442
WordContract, 354
flags, 355
fuzz_rounds, 355
WordProfile, 355
avg_time, 356
call_count, 356
max_time, 356
min_time, 356
percentage, 356
total_time, 356
word_name, 356
words_executed
DoeMetrics, 160
HeartbeatState, 201
WordStats, 357
call_count, 358
entry, 358
max_time_ns, 358
min_time_ns, 358
total_time_ns, 358
WordTestSuite, 359
suite_contract, 359
test_count, 359
tests, 360
word_name, 360
WordTransitionMetrics, 360
actual_window_size, 362
context_transitions, 362
context_window, 362
context_window_pos, 362
max_transition_probability_q48, 363
misprediction_cost_q48, 363
most_likely_next_word_id, 363
physics_pipelining_metrics.h, 1859
prefetch_attempts, 363
prefetch_hits, 363
prefetch_latency_saved_q48, 364
prefetch_misses, 364
total_context_transitions, 364
total_transitions, 364
transition_heat, 364
vm.h, 2011
worker
HeartbeatState, 201
writable
vmm_page_info_t, 344
Write
_EFI_FILE_PROTOCOL, 92
write
blkio_vtable_t, 122
write_count
blk_meta_t, 111
write_devblock
capsule_mint.c, 1135
write_devblock_4k
block_subsystem.c, 2210
write_header_4k
block_subsystem.c, 2211
write_offset
physics_analytics_header, 233
write_ramdrive_block
capsule_loader.c, 1126
WriteBlocks
_EFI_BLOCK_IO_PROTOCOL, 85
WriteCaching
EFI_BLOCK_IO_MEDIA, 163
writes
MemoryProfile, 224
wrmsr64
apic.c, 937
wValue
usb_setup_packet_t, 301

x
ed_point_t, 161
ttf_flat_point_t, 288
ttf_point_t, 295

x509_build_user_cert
x509_ed25519.c, 1227
x509_ed25519.h, 797
x509_ed25519.c
der_next, 1225
der_put_ed25519_algid, 1225
der_put_tlv, 1226
der_put_uint_integer, 1226
x509_build_user_cert, 1227
x509_extract_ed25519_pubkey, 1228
x509_extract_serial, 1229
x509_verify_signature, 1230
x509_ed25519.h
x509_build_user_cert, 797
x509_extract_ed25519_pubkey, 798
x509_extract_serial, 799
x509_verify_signature, 800
x509_extract_ed25519_pubkey
x509_ed25519.c, 1228
x509_ed25519.h, 798
x509_extract_serial
x509_ed25519.c, 1229
x509_ed25519.h, 799
x509_verify_signature
x509_ed25519.c, 1230
x509_ed25519.h, 800
x_max
ttf_glyph_header_t, 292
x_min
ttf_glyph_header_t, 292
xdev
blkio_usb.c, 1508
BlkioUsbState, 124
xhci.c
g_xhci_dev_count, 1560
g_xhci_devs, 1560
g_xhci_keyboards, 1560
USB_HID_REPORT_OUTPUT, 1517
USB_HID_REQ_SET_REPORT, 1517
XHCI_BAR0_MIN_MAP_SIZE, 1518
xhci_bot_get_capacity, 1519
xhci_bot_read_block, 1519
xhci_bot_read_data_in, 1520
xhci_bot_receive_csw, 1521
xhci_bot_send_read10, 1521
xhci_bot_send_read_capacity10, 1522
xhci_bot_send_test_unit_ready, 1523
xhci_bot_send_write10, 1524
xhci_bot_wait_for_idle, 1525
xhci_bot_write_block, 1526
xhci_bot_write_data_out, 1527
xhci_bringup, 1528
xhci_bulk_in_enqueue_and_ring, 1529
xhci_bulk_out_enqueue_and_ring, 1530
xhci_bulk_purpose_stalled, 1530
xhci_cmd_address_device, 1531
xhci_cmd_configure_endpoint, 1532
xhci_cmd_configure_hid_endpoint, 1533
xhci_cmd_disable_slot, 1534
xhci_cmd_enable_slot, 1535
xhci_cmd_reset_endpoint, 1535
xhci_cmd_set_tr_dequeue_pointer, 1536
xhci_ctx_bytes, 1537
xhci_default_ep0_max_packet, 1537
xhci_dev_at, 1537
xhci_dev_count, 1538
xhci_drain_pending_event, 1538
xhci_ep0_bot_mass_storage_reset, 1539
xhci_ep0_clear_endpoint_halt, 1540
xhci_ep0_control_read, 1540
xhci_ep0_control_write_nodata, 1541
xhci_ep0_enqueue_trb, 1542
xhci_ep0_get_config_descriptor, 1543
xhci_ep0_get_device_descriptor, 1543
xhci_ep0_hid_set_leds, 1544
xhci_ep0_hid_set_protocol, 1545
xhci_ep0_set_configuration, 1546
xhci_find_and_map, 1546
xhci_handle_bulk_stall, 1547
xhci_handle_port_connected, 1548
xhci_hid_queue_report, 1549
xhci_init_bulk_ring, 1549
xhci_keyboard_running, 1549
XHCI_LEGSUP_BIOS_OWNED, 1518
XHCI_LEGSUP_OS_OWNED, 1518
xhci_log_hex32, 1550
xhci_msc_slot_for, 1550
XHCI_PCI_CLASS, 1518
XHCI_PCI_PROG_IF, 1518
XHCI_PCI_SUBCLASS, 1518
xhci_poll_events, 1551
xhci_poll_one, 1553
xhci_port_regs, 1556
xhci_queue_pending_event, 1556
xhci_scan_ports_for_already_connected, 1557
xhci_stall_fail, 1557
xhci_submit_command, 1558
xhci_take_from_bios, 1558
xhci_wait_bit, 1559
XHCI_XCAP_LEGACY, 1518
xhci.h
SCSI_CDB_LEN_READ10, 805
SCSI_CDB_LEN_READ_CAPACITY10, 805
SCSI_CDB_LEN_TEST_UNIT_READY, 805
SCSI_CDB_LEN_WRITE10, 805
SCSI_CMD_READ10, 805
SCSI_CMD_READ_CAPACITY10, 805
SCSI_CMD_TEST_UNIT_READY, 805
SCSI_CMD_WRITE10, 805
SCSI_READ_CAPACITY10_DATA_LEN, 805
USB_BOT_CBW_FLAG_DATA_IN, 805
USB_BOT_CBW_LENGTH, 806
USB_BOT_CBW_LUN_DEFAULT, 806
USB_BOT_CBW_SIGNATURE, 806
USB_BOT_CSW_LENGTH, 806
USB_BOT_CSW_SIGNATURE, 806

USB_BOT_CSW_STATUS_FAILED, 806
USB_BOT_CSW_STATUS_PASS, 806
USB_BOT_CSW_STATUS_PHASE_ERROR, 806
USB_BOT_MASS_STORAGE_RESET, 806
USB_CLASS_HID, 806
USB_CLASS_MASS_STORAGE, 807
USB_CONFIG_OFF_CONFIG_VALUE, 807
USB_CONFIG_OFF_TOTAL_LENGTH, 807
USB_DESC_OFF_LENGTH, 807
USB_DESC_OFF_TYPE, 807
USB_DESC_TYPE_CONFIG, 807
USB_DESC_TYPE_DEVICE, 807
USB_DESC_TYPE_ENDPOINT, 807
USB_DESC_TYPE_INTERFACE, 807
USB_DIR_DEVICE_TO_HOST, 807
USB_DIR_HOST_TO_DEVICE, 808
USB_EP_ADDR_DIR_MASK, 808
USB_EP_ADDR_NUM_MASK, 808
USB_EP_ATTR_TYPE_MASK, 808
USB_EP_OFF_ADDRESS, 808
USB_EP_OFF_ATTRIBUTES, 808
USB_EP_OFF_INTERVAL, 808
USB_EP_OFF_MAX_PACKET_SIZE, 808
USB_EP_TYPE_BULK, 808
USB_EP_TYPE_INTERRUPT, 808
USB_FEATURE_ENDPOINT_HALT, 809
USB_HID_PROTOCOL_BOOT, 809
USB_HID_REQ_SET_PROTOCOL, 809
USB_IFACE_OFF_CLASS, 809
USB_IFACE_OFF_NUMBER, 809
USB_IFACE_OFF_PROTOCOL, 809
USB_IFACE_OFF_SUBCLASS, 809
USB_PROTOCOL_BOT, 809
USB_PROTOCOL_HID_KEYBOARD, 809
USB_RECIP_DEVICE, 809
USB_RECIP_ENDPOINT, 810
USB_RECIP_INTERFACE, 810
USB_REQ_CLEAR_FEATURE, 810
USB_REQ_GET_DESCRIPTOR, 810
USB_REQ_SET_CONFIGURATION, 810
USB_REQ_TYPE_CLASS, 810
USB_REQ_TYPE_STANDARD, 810
USB_SUBCLASS_HID_BOOT, 810
USB_SUBCLASS_SCSI, 810
XHCI_BOT_STALL_MAX_RECOVERIES, 810
XHCI_BOT_TUR_MAX_RETRIES, 811
XHCI_COMPLETION_CODE_SHORT_PACKET, 811
XHCI_COMPLETION_CODE_STALL_ERROR, 811
XHCI_COMPLETION_CODE_SUCCESS, 811
XHCI_CONFIG_MAX_SLOTS_EN, 811
XHCI_CRCR_CA, 811
XHCI_CRCR_CRR, 811
XHCI_CRCR_CS, 811
XHCI_CRCR_PTR_MASK, 811
XHCI_CRCR_RCS, 811
XHCI_DB_STREAM_ID, 812
XHCI_DB_TARGET, 812
xhci_doorbell_t, 820
XHCI_EP_ADDR_TO_DCI, 812
XHCI_EP_CTX_CERR_SHIFT, 812
XHCI_EP_CTX_INTERVAL_SHIFT, 812
XHCI_EP_CTX_MAX_PACKET_SHIFT, 812
XHCI_EP_CTX_TYPE_BULK_IN, 812
XHCI_EP_CTX_TYPE_BULK_OUT, 812
XHCI_EP_CTX_TYPE_CONTROL_BIDI, 812
XHCI_EP_CTX_TYPE_INTERRUPT_IN, 813
XHCI_EP_CTX_TYPE_SHIFT, 813
XHCI_ERDP_EHB, 813
XHCI_ERDP_PTR_MASK, 813
XHCI_EVT_COMPLETION_CODE, 813
XHCI_EVT_EP_ID, 813
XHCI_EVT_RING_MAX_DRAIN, 813
XHCI_EVT_SLOT_ID, 813
XHCI_HCCPARAMS1_CSZ, 813
XHCI_HCSPARAMS1_MAX_INTRS, 813
XHCI_HCSPARAMS1_MAX_PORTS, 814
XHCI_HCSPARAMS1_MAX_SLOTS, 814
XHCI_HCSPARAMS2_MAX_SCRATCHPAD_BUFS, 814
XHCI_IMAN_IE, 814
XHCI_IMAN_IP, 814
XHCI_INPUT_CTRL_ADD_EP0, 814
XHCI_INPUT_CTRL_ADD_SLOT, 814
XHCI_MAX_TRACKED_PORTS, 814
XHCI_PORT_REGS_OFFSET, 814
XHCI_PORTSC_CCS, 814
XHCI_PORTSC_CSC, 815
XHCI_PORTSC_PEC, 815
XHCI_PORTSC_PED, 815
XHCI_PORTSC_PLS_MASK, 815
XHCI_PORTSC_PP, 815
XHCI_PORTSC_PR, 815
XHCI_PORTSC_PRC, 815
XHCI_PORTSC_SPEED, 815
XHCI_PORTSC_SPEED_MASK, 815
XHCI_PSC_EVT_PORT_ID, 815
XHCI_RING_BYTES, 816
XHCI_RING_TRB_COUNT, 816
XHCI_SETUP_TRT_IN_DATA, 816
XHCI_SETUP_TRT_NO_DATA, 816
XHCI_SETUP_TRT_OUT_DATA, 816
XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT, 816
XHCI_SLOT_CTX_INTR_TARGET_SHIFT, 816
XHCI_SLOT_CTX_ROOT_PORT_SHIFT, 816
XHCI_SLOT_CTX_SPEED_SHIFT, 816
XHCI_TRB_CONTROL_CYCLE, 816
XHCI_TRB_CONTROL_DIR_IN, 817
XHCI_TRB_CONTROL_IDT, 817
XHCI_TRB_CONTROL_IOC, 817
XHCI_TRB_CONTROL_TC, 817
XHCI_TRB_CONTROL_TRT_SHIFT, 817
XHCI_TRB_CONTROL_TYPE_MASK, 817
XHCI_TRB_CONTROL_TYPE_SHIFT, 817

XHCI_TRB_TYPE, 817
XHCI_TRB_TYPE_ADDRESS_DEVICE_CMD, 817
XHCI_TRB_TYPE_COMMAND_COMPLETION_EVT, 817
XHCI_TRB_TYPE_CONFIGURE_ENDPOINT_CMD, 818
XHCI_TRB_TYPE_DATA_STAGE, 818
XHCI_TRB_TYPE_DISABLE_SLOT_CMD, 818
XHCI_TRB_TYPE_ENABLE_SLOT_CMD, 818
XHCI_TRB_TYPE_LINK, 818
XHCI_TRB_TYPE_NORMAL, 818
XHCI_TRB_TYPE_PORT_STATUS_CHANGE_EVT, 818
XHCI_TRB_TYPE_RESET_ENDPOINT_CMD, 818
XHCI_TRB_TYPE_SET_TR_DEQUEUE_POINTER_CMD, 818
XHCI_TRB_TYPE_SETUP_STAGE, 818
XHCI_TRB_TYPE_STATUS_STAGE, 819
XHCI_TRB_TYPE_TRANSFER_EVENT, 819
XHCI_USBCMD_HCRST, 819
XHCI_USBCMD_HSEE, 819
XHCI_USBCMD_INTE, 819
XHCI_USBCMD_RUN, 819
XHCI_USBSTS_CNR, 819
XHCI_USBSTS_EINT, 819
XHCI_USBSTS_HCE, 819
XHCI_USBSTS_HCH, 819
XHCI_USBSTS_HSE, 820
XHCI_USBSTS_PCD, 820
XHCI_BAR0_MIN_MAP_SIZE
xhci.c, 1518
xhci_bot_get_capacity
xhci.c, 1519
xhci_driver.h, 823
xhci_bot_read_block
xhci.c, 1519
xhci_driver.h, 824
xhci_bot_read_data_in
xhci.c, 1520
xhci_driver.h, 825
xhci_bot_receive_csw
xhci.c, 1521
xhci_driver.h, 825
xhci_bot_send_read10
xhci.c, 1521
xhci_driver.h, 826
xhci_bot_send_read_capacity10
xhci.c, 1522
xhci_driver.h, 827
xhci_bot_send_test_unit_ready
xhci.c, 1523
xhci_driver.h, 828
xhci_bot_send_write10
xhci.c, 1524
xhci_driver.h, 828
XHCI_BOT_STALL_MAX_RECOVERIES
xhci.h, 810
XHCI_BOT_TUR_MAX_RETRIES
xhci.h, 811
xhci_bot_wait_for_idle
xhci.c, 1525
xhci_driver.h, 829
xhci_bot_write_block
xhci.c, 1526
xhci_driver.h, 830
xhci_bot_write_data_out
xhci.c, 1527
xhci_driver.h, 831
xhci_bringup
xhci.c, 1528
xhci_driver.h, 832
xhci_bulk_in_enqueue_and_ring
xhci.c, 1529
xhci_bulk_out_enqueue_and_ring
xhci.c, 1530
xhci_bulk_purpose_stalled
xhci.c, 1530
xhci_cap_regs_t, 365
cap_length, 365
db_off, 365
hcc_params1, 365
hcc_params2, 366
hci_version, 366
hcs_params1, 366
hcs_params2, 366
hcs_params3, 366
reserved0, 366
rts_off, 366
xhci_cmd_address_device
xhci.c, 1531
xhci_driver.h, 833
xhci_cmd_configure_endpoint
xhci.c, 1532
xhci_driver.h, 834
xhci_cmd_configure_hid_endpoint
xhci.c, 1533
xhci_driver.h, 835
xhci_cmd_disable_slot
xhci.c, 1534
xhci_driver.h, 836
xhci_cmd_enable_slot
xhci.c, 1535
xhci_driver.h, 837
xhci_cmd_reset_endpoint
xhci.c, 1535
xhci_driver.h, 837
xhci_cmd_set_tr_dequeue_pointer
xhci.c, 1536
xhci_driver.h, 838
XHCI_COMPLETION_CODE_SHORT_PACKET
xhci.h, 811
XHCI_COMPLETION_CODE_STALL_ERROR
xhci.h, 811
XHCI_COMPLETION_CODE_SUCCESS

xhci.h, 811
XHCI_CONFIG_MAX_SLOTS_EN
xhci.h, 811
XHCI_CONN_AWAIT_ADDRESS_DEVICE
xhci_dev_t, 369
XHCI_CONN_AWAIT_CONFIGURE_ENDPOINT
xhci_dev_t, 370
XHCI_CONN_AWAIT_DISABLE_SLOT
xhci_dev_t, 370
XHCI_CONN_AWAIT_ENABLE_SLOT
xhci_dev_t, 369
XHCI_CONN_AWAIT_PORT_RESET
xhci_dev_t, 369
XHCI_CONN_AWAIT_RESET_ENDPOINT
xhci_dev_t, 370
XHCI_CONN_AWAIT_SET_TR_DEQUEUE
xhci_dev_t, 370
XHCI_CONN_IDLE
xhci_dev_t, 369
XHCI_CRCR_CA
xhci.h, 811
XHCI_CRCR_CRR
xhci.h, 811
XHCI_CRCR_CS
xhci.h, 811
XHCI_CRCR_PTR_MASK
xhci.h, 811
XHCI_CRCR_RCS
xhci.h, 811
xhci_ctx_bytes
xhci.c, 1537
XHCI_DB_STREAM_ID
xhci.h, 812
XHCI_DB_TARGET
xhci.h, 812
xhci_default_ep0_max_packet
xhci.c, 1537
xhci_dev_at
xhci.c, 1537
xhci_driver.h, 839
xhci_dev_count
xhci.c, 1538
xhci_driver.h, 839
xhci_dev_t, 367
bar0_phys, 370
bot_cap_block_size, 370
bot_cap_last_lba, 370
bot_cbw, 370
bot_cmd_kind, 370
BOT_CMD_NONE, 369
BOT_CMD_READ10, 369
BOT_CMD_READ_CAPACITY10, 369
BOT_CMD_TEST_UNIT_READY, 369
BOT_CMD_WRITE10, 369
bot_csw, 370
bot_data_buf, 370
bot_expected_data_len, 371
bot_last_status, 371
bot_last_tag, 371
bot_next_tag, 371
bot_read10_block_size, 371
bot_read10_lba, 371
bot_read10_num_blocks, 371
bot_reset_clear_remaining, 372
bot_stall_recoveries, 372
BOT_STATUS_FAILED, 369
BOT_STATUS_IDLE, 369
BOT_STATUS_PASS, 369
BOT_STATUS_TIMEOUT, 369
BOT_TUR_CHAIN_NONE, 369
BOT_TUR_CHAIN_READ10, 369
BOT_TUR_CHAIN_READ_CAPACITY10, 369
bot_tur_chain_target, 372
BOT_TUR_CHAIN_WRITE10, 369
bot_tur_retries, 372
bot_write10_block_size, 372
bot_write10_lba, 372
bot_write10_num_blocks, 372
cap, 373
cmd_ring, 373
cmd_ring_cycle, 373
cmd_ring_enq, 373
connect_state, 373
dcbaa, 373
doorbell, 373
evt_ring, 374
evt_ring_cycle, 374
evt_ring_deq, 374
evt_ring_seg_table, 374
intr0, 374
is_connect, 374
max_intrs, 374
max_ports, 375
max_scratchpad_bufs, 375
max_slots, 375
msc_slots, 375
op, 375
pci, 375
pending_connect_port_id, 375
pending_connect_slot_id, 376
pending_connect_speed, 376
pending_disable_slot_id, 376
pending_events, 376
port_id, 376
port_slot_id, 376
portsc, 376
runtime, 376
scratchpad_arr, 377
slot_id, 377
stall_dci, 377
stall_ep_addr, 377
stall_retry_action, 377
valid, 377
XHCI_CONN_AWAIT_ADDRESS_DEVICE, 369
XHCI_CONN_AWAIT_CONFIGURE_ENDPOINT, 370

XHCI_CONN_AWAIT_DISABLE_SLOT, 370
XHCI_CONN_AWAIT_ENABLE_SLOT, 369
XHCI_CONN_AWAIT_PORT_RESET, 369
XHCI_CONN_AWAIT_RESET_ENDPOINT, 370
XHCI_CONN_AWAIT_SET_TR_DEQUEUE, 370
XHCI_CONN_IDLE, 369
xhci_doorbell_t
xhci.h, 820
xhci_drain_pending_event
xhci.c, 1538
xhci_driver.h
xhci_bot_get_capacity, 823
xhci_bot_read_block, 824
xhci_bot_read_data_in, 825
xhci_bot_receive_csw, 825
xhci_bot_send_read10, 826
xhci_bot_send_read_capacity10, 827
xhci_bot_send_test_unit_ready, 828
xhci_bot_send_write10, 828
xhci_bot_wait_for_idle, 829
xhci_bot_write_block, 830
xhci_bot_write_data_out, 831
xhci_bringup, 832
xhci_cmd_address_device, 833
xhci_cmd_configure_endpoint, 834
xhci_cmd_configure_hid_endpoint, 835
xhci_cmd_disable_slot, 836
xhci_cmd_enable_slot, 837
xhci_cmd_reset_endpoint, 837
xhci_cmd_set_tr_dequeue_pointer, 838
xhci_dev_at, 839
xhci_dev_count, 839
xhci_ep0_bot_mass_storage_reset, 839
xhci_ep0_clear_endpoint_halt, 840
xhci_ep0_get_config_descriptor, 841
xhci_ep0_get_device_descriptor, 842
xhci_ep0_hid_set_protocol, 842
xhci_ep0_set_configuration, 843
xhci_find_and_map, 844
xhci_keyboard_running, 845
XHCI_MAX_CONTROLLERS, 822
xhci_msc_slot_for, 845
XHCI_NEXT_ACTION_BOT_CSW_RECEIVE, 823
XHCI_NEXT_ACTION_BOT_DATA_IN, 822
XHCI_NEXT_ACTION_BOT_DATA_OUT, 823
XHCI_NEXT_ACTION_BOT_RESET, 823
XHCI_NEXT_ACTION_BOT_SEND_READ10, 823
XHCI_NEXT_ACTION_BOT_SEND_READ_CAPACITY10, 823
XHCI_NEXT_ACTION_BOT_SEND_TUR, 823
XHCI_NEXT_ACTION_BOT_SEND_WRITE10, 823
XHCI_NEXT_ACTION_CLEAR_HALT, 823
XHCI_NEXT_ACTION_CONFIGURE_ENDPOINT, 822
XHCI_NEXT_ACTION_GET_CONFIG_DESC, 822
XHCI_NEXT_ACTION_GET_DEVICE_DESC, 822
XHCI_NEXT_ACTION_HID_POLL, 823
XHCI_NEXT_ACTION_HID_SET_PROTOCOL, 823
XHCI_NEXT_ACTION_NONE, 822
XHCI_NEXT_ACTION_SET_CONFIG, 822
xhci_next_action_t, 822
xhci_poll_events, 846
xhci_transfer_purpose_t, 823
XHCI_XFER_BOT_DATA_IN, 823
XHCI_XFER_BOT_DATA_OUT, 823
XHCI_XFER_BOT_RESET, 823
XHCI_XFER_CBW_SENT, 823
XHCI_XFER_CLEAR_HALT, 823
XHCI_XFER_CONFIG_DESC_FULL, 823
XHCI_XFER_CONFIG_DESC_SHORT, 823
XHCI_XFER_CSW_RECEIVED, 823
XHCI_XFER_DEVICE_DESC, 823
XHCI_XFER_HID_SET_LEDS, 823
XHCI_XFER_HID_SET_PROTOCOL, 823
XHCI_XFER_NONE, 823
XHCI_XFER_SET_CONFIG, 823
xhci_ep0_bot_mass_storage_reset
xhci.c, 1539
xhci_driver.h, 839
xhci_ep0_clear_endpoint_halt
xhci.c, 1540
xhci_driver.h, 840
xhci_ep0_control_read
xhci.c, 1540
xhci_ep0_control_write_nodata
xhci.c, 1541
xhci_ep0_enqueue_trb
xhci.c, 1542
xhci_ep0_get_config_descriptor
xhci.c, 1543
xhci_driver.h, 841
xhci_ep0_get_device_descriptor
xhci.c, 1543
xhci_driver.h, 842
xhci_ep0_hid_set_leds
xhci.c, 1544
xhci_ep0_hid_set_protocol
xhci.c, 1545
xhci_driver.h, 842
xhci_ep0_set_configuration
xhci.c, 1546
xhci_driver.h, 843
XHCI_EP_ADDR_TO_DCI
xhci.h, 812
xhci_ep_ctx32_t, 378
dword0, 378
dword1, 378
dword4, 378
reserved, 378
tr_dequeue_ptr, 378
XHCI_EP_CTX_CERR_SHIFT
xhci.h, 812
XHCI_EP_CTX_INTERVAL_SHIFT
xhci.h, 812

XHCI_EP_CTX_MAX_PACKET_SHIFT
xhci.h, 812
XHCI_EP_CTX_TYPE_BULK_IN
xhci.h, 812
XHCI_EP_CTX_TYPE_BULK_OUT
xhci.h, 812
XHCI_EP_CTX_TYPE_CONTROL_BIDI
xhci.h, 812
XHCI_EP_CTX_TYPE_INTERRUPT_IN
xhci.h, 813
XHCI_EP_CTX_TYPE_SHIFT
xhci.h, 813
XHCI_ERDP_EHB
xhci.h, 813
XHCI_ERDP_PTR_MASK
xhci.h, 813
XHCI_EVT_COMPLETION_CODE
xhci.h, 813
XHCI_EVT_EP_ID
xhci.h, 813
XHCI_EVT_RING_MAX_DRAIN
xhci.h, 813
XHCI_EVT_SLOT_ID
xhci.h, 813
xhci_find_and_map
xhci.c, 1546
xhci_driver.h, 844
xhci_handle_bulk_stall
xhci.c, 1547
xhci_handle_port_connected
xhci.c, 1548
XHCI_HCCPARAMS1_CSZ
xhci.h, 813
XHCI_HCSPARAMS1_MAX_INTRS
xhci.h, 813
XHCI_HCSPARAMS1_MAX_PORTS
xhci.h, 814
XHCI_HCSPARAMS1_MAX_SLOTS
xhci.h, 814
XHCI_HCSPARAMS2_MAX_SCRATCHPAD_BUFS
xhci.h, 814
xhci_hid_queue_report
xhci.c, 1549
XHCI_IMAN_IE
xhci.h, 814
XHCI_IMAN_IP
xhci.h, 814
xhci_init_bulk_ring
xhci.c, 1549
XHCI_INPUT_CTRL_ADD_EP0
xhci.h, 814
XHCI_INPUT_CTRL_ADD_SLOT
xhci.h, 814
xhci_input_ctrl_ctx32_t, 379
add_flags, 379
config_word, 379
drop_flags, 379
reserved, 379
xhci_intr_regs_t, 379
erdp, 380
erstba, 380
erstsz, 380
iman, 380
imod, 380
reserved0, 380
xhci_keyboard_running
xhci.c, 1549
xhci_driver.h, 845
XHCI_LEGSUP_BIOS_OWNED
xhci.c, 1518
XHCI_LEGSUP_OS_OWNED
xhci.c, 1518
xhci_log_hex32
xhci.c, 1550
XHCI_MAX_CONTROLLERS
xhci_driver.h, 822
XHCI_MAX_TRACKED_PORTS
xhci.h, 814
xhci_msc_slot_for
xhci.c, 1550
xhci_driver.h, 845
xhci_msc_slot_t, 381
bot_msc_attach_pending, 382
bot_msc_attached, 382
bot_msc_detach_pending, 382
bulk_in_ep_addr, 382
bulk_in_max_packet, 382
bulk_in_ring, 382
bulk_in_ring_cycle, 383
bulk_in_ring_enq, 383
bulk_out_ep_addr, 383
bulk_out_max_packet, 383
bulk_out_ring, 383
bulk_out_ring_cycle, 383
bulk_out_ring_enq, 383
config_descriptor, 384
config_total_length, 384
device_ctx, 384
device_descriptor, 384
ep0_ring, 384
ep0_ring_cycle, 384
ep0_ring_enq, 384
hid_buf, 385
hid_ep_addr, 385
hid_iface, 385
hid_interval, 385
hid_kbd, 385
hid_leds_sent, 385
hid_max_packet, 385
hid_prev, 386
hid_ring, 386
hid_ring_cycle, 386
hid_ring_enq, 386
hid_running, 386
in_use, 386
input_ctx, 386

next_action, 386
next_action_config_value, 387
next_action_length, 387
transfer_purpose, 387
XHCI_NEXT_ACTION_BOT_CSW_RECEIVE
xhci_driver.h, 823
XHCI_NEXT_ACTION_BOT_DATA_IN
xhci_driver.h, 822
XHCI_NEXT_ACTION_BOT_DATA_OUT
xhci_driver.h, 823
XHCI_NEXT_ACTION_BOT_RESET
xhci_driver.h, 823
XHCI_NEXT_ACTION_BOT_SEND_READ10
xhci_driver.h, 823
XHCI_NEXT_ACTION_BOT_SEND_READ_CAPACITY10
xhci_driver.h, 823
XHCI_NEXT_ACTION_BOT_SEND_TUR
xhci_driver.h, 823
XHCI_NEXT_ACTION_BOT_SEND_WRITE10
xhci_driver.h, 823
XHCI_NEXT_ACTION_CLEAR_HALT
xhci_driver.h, 823
XHCI_NEXT_ACTION_CONFIGURE_ENDPOINT
xhci_driver.h, 822
XHCI_NEXT_ACTION_GET_CONFIG_DESC
xhci_driver.h, 822
XHCI_NEXT_ACTION_GET_DEVICE_DESC
xhci_driver.h, 822
XHCI_NEXT_ACTION_HID_POLL
xhci_driver.h, 823
XHCI_NEXT_ACTION_HID_SET_PROTOCOL
xhci_driver.h, 823
XHCI_NEXT_ACTION_NONE
xhci_driver.h, 822
XHCI_NEXT_ACTION_SET_CONFIG
xhci_driver.h, 822
xhci_next_action_t
xhci_driver.h, 822
xhci_op_regs_t, 387
config, 388
crcr, 388
dcbaap, 388
dn_ctrl, 388
page_size, 388
reserved0, 388
reserved1, 388
usb_cmd, 388
usb_sts, 388
XHCI_PCI_CLASS
xhci.c, 1518
XHCI_PCI_PROG_IF
xhci.c, 1518
XHCI_PCI_SUBCLASS
xhci.c, 1518
xhci_poll_events
xhci.c, 1551
xhci_driver.h, 846
xhci_poll_one
xhci.c, 1553
xhci_port_regs
xhci.c, 1556
XHCI_PORT_REGS_OFFSET
xhci.h, 814
xhci_port_regs_t, 389
porthlpmc, 389
portli, 389
portpmsc, 389
portsc, 389
XHCI_PORTSC_CCS
xhci.h, 814
XHCI_PORTSC_CSC
xhci.h, 815
XHCI_PORTSC_PEC
xhci.h, 815
XHCI_PORTSC_PED
xhci.h, 815
XHCI_PORTSC_PLS_MASK
xhci.h, 815
XHCI_PORTSC_PP
xhci.h, 815
XHCI_PORTSC_PR
xhci.h, 815
XHCI_PORTSC_PRC
xhci.h, 815
XHCI_PORTSC_SPEED
xhci.h, 815
XHCI_PORTSC_SPEED_MASK
xhci.h, 815
XHCI_PSC_EVT_PORT_ID
xhci.h, 815
xhci_queue_pending_event
xhci.c, 1556
XHCI_RING_BYTES
xhci.h, 816
XHCI_RING_TRB_COUNT
xhci.h, 816
xhci_runtime_regs_t, 390
mf_index, 390
reserved0, 390
xhci_scan_ports_for_already_connected
xhci.c, 1557
XHCI_SETUP_TRT_IN_DATA
xhci.h, 816
XHCI_SETUP_TRT_NO_DATA
xhci.h, 816
XHCI_SETUP_TRT_OUT_DATA
xhci.h, 816
xhci_slot_ctx32_t, 390
dword0, 390
dword1, 390
dword2, 391
dword3, 391
reserved, 391
XHCI_SLOT_CTX_CONTEXT_ENTRIES_SHIFT
xhci.h, 816
XHCI_SLOT_CTX_INTR_TARGET_SHIFT

xhci.h, 816
XHCI_SLOT_CTX_ROOT_PORT_SHIFT
xhci.h, 816
XHCI_SLOT_CTX_SPEED_SHIFT
xhci.h, 816
xhci_stall_fail
xhci.c, 1557
xhci_submit_command
xhci.c, 1558
xhci_take_from_bios
xhci.c, 1558
xhci_transfer_purpose_t
xhci_driver.h, 823
XHCI_TRB_CONTROL_CYCLE
xhci.h, 816
XHCI_TRB_CONTROL_DIR_IN
xhci.h, 817
XHCI_TRB_CONTROL_IDT
xhci.h, 817
XHCI_TRB_CONTROL_IOC
xhci.h, 817
XHCI_TRB_CONTROL_TC
xhci.h, 817
XHCI_TRB_CONTROL_TRT_SHIFT
xhci.h, 817
XHCI_TRB_CONTROL_TYPE_MASK
xhci.h, 817
XHCI_TRB_CONTROL_TYPE_SHIFT
xhci.h, 817
xhci_trb_t, 391
control, 391
parameter, 391
status, 392
XHCI_TRB_TYPE
xhci.h, 817
XHCI_TRB_TYPE_ADDRESS_DEVICE_CMD
xhci.h, 817
XHCI_TRB_TYPE_COMMAND_COMPLETION_EVT
xhci.h, 817
XHCI_TRB_TYPE_CONFIGURE_ENDPOINT_CMD
xhci.h, 818
XHCI_TRB_TYPE_DATA_STAGE
xhci.h, 818
XHCI_TRB_TYPE_DISABLE_SLOT_CMD
xhci.h, 818
XHCI_TRB_TYPE_ENABLE_SLOT_CMD
xhci.h, 818
XHCI_TRB_TYPE_LINK
xhci.h, 818
XHCI_TRB_TYPE_NORMAL
xhci.h, 818
XHCI_TRB_TYPE_PORT_STATUS_CHANGE_EVT
xhci.h, 818
XHCI_TRB_TYPE_RESET_ENDPOINT_CMD
xhci.h, 818
XHCI_TRB_TYPE_SET_TR_DEQUEUE_POINTER_CMD
xhci.h, 818
XHCI_TRB_TYPE_SETUP_STAGE
xhci.h, 818
XHCI_TRB_TYPE_STATUS_STAGE
xhci.h, 819
XHCI_TRB_TYPE_TRANSFER_EVENT
xhci.h, 819
XHCI_USBCMD_HCRST
xhci.h, 819
XHCI_USBCMD_HSEE
xhci.h, 819
XHCI_USBCMD_INTE
xhci.h, 819
XHCI_USBCMD_RUN
xhci.h, 819
XHCI_USBSTS_CNR
xhci.h, 819
XHCI_USBSTS_EINT
xhci.h, 819
XHCI_USBSTS_HCE
xhci.h, 819
XHCI_USBSTS_HCH
xhci.h, 819
XHCI_USBSTS_HSE
xhci.h, 820
XHCI_USBSTS_PCD
xhci.h, 820
xhci_wait_bit
xhci.c, 1559
XHCI_XCAP_LEGACY
xhci.c, 1518
XHCI_XFER_BOT_DATA_IN
xhci_driver.h, 823
XHCI_XFER_BOT_DATA_OUT
xhci_driver.h, 823
XHCI_XFER_BOT_RESET
xhci_driver.h, 823
XHCI_XFER_CBW_SENT
xhci_driver.h, 823
XHCI_XFER_CLEAR_HALT
xhci_driver.h, 823
XHCI_XFER_CONFIG_DESC_FULL
xhci_driver.h, 823
XHCI_XFER_CONFIG_DESC_SHORT
xhci_driver.h, 823
XHCI_XFER_CSW_RECEIVED
xhci_driver.h, 823
XHCI_XFER_DEVICE_DESC
xhci_driver.h, 823
XHCI_XFER_HID_SET_LEDS
xhci_driver.h, 823
XHCI_XFER_HID_SET_PROTOCOL
xhci_driver.h, 823
XHCI_XFER_NONE
xhci_driver.h, 823
XHCI_XFER_SET_CONFIG
xhci_driver.h, 823
xsdt_address
__attribute__, 83
xxh64_avalanche

xxhash64.c, 1397
xxh64_merge_round
xxhash64.c, 1397
xxh64_read32
xxhash64.c, 1398
xxh64_read64
xxhash64.c, 1398
xxh64_rotl
xxhash64.c, 1398
xxh64_round
xxhash64.c, 1399
xxhash64
xxhash64.c, 1399
xxhash64.h, 849
xxhash64.c
xxh64_avalanche, 1397
xxh64_merge_round, 1397
xxh64_read32, 1398
xxh64_read64, 1398
xxh64_rotl, 1398
xxh64_round, 1399
xxhash64, 1399
xxhash64_digest, 1400
XXHASH64_PRIME1, 1396
XXHASH64_PRIME2, 1396
XXHASH64_PRIME3, 1396
XXHASH64_PRIME4, 1397
XXHASH64_PRIME5, 1397
xxhash64_reset, 1401
xxhash64_update, 1402
xxhash64.h
xxhash64, 849
xxhash64_capsule, 850
XXHASH64_CAPSULE_SEED, 849
xxhash64_digest, 851
xxhash64_reset, 852
xxhash64_update, 852
xxhash64_capsule
xxhash64.h, 850
XXHASH64_CAPSULE_SEED
xxhash64.h, 849
xxhash64_digest
xxhash64.c, 1400
xxhash64.h, 851
XXHASH64_PRIME1
xxhash64.c, 1396
XXHASH64_PRIME2
xxhash64.c, 1396
XXHASH64_PRIME3
xxhash64.c, 1396
XXHASH64_PRIME4
xxhash64.c, 1397
XXHASH64_PRIME5
xxhash64.c, 1397
xxhash64_reset
xxhash64.c, 1401
xxhash64.h, 852
xxhash64_update
xxhash64.c, 1402
xxhash64.h, 852
XXHash64State, 392
buffer, 392
buffer_size, 392
seed, 393
total_len, 393
v1, 393
v2, 393
v3, 393
v4, 393

y
ed_point_t, 161
ttf_flat_point_t, 288
ttf_point_t, 295
y_max
ttf_glyph_header_t, 292
y_min
ttf_glyph_header_t, 292
Year
EFI_TIME, 176

z
ed_point_t, 162
zero
idt_entry, 214
interrupts.c, 988
zuse_cert_devblock.h
ZUSE_CERT_DEVBLOCK_MAGIC, 854
ZUSE_CERT_DEVBLOCK_VERSION, 854
ZUSE_CERT_DEVBLOCK_MAGIC
zuse_cert_devblock.h, 854
zuse_cert_devblock_t, 394
_pad, 394
crc, 394
magic, 394
pubkey, 394
seed, 394
version, 394
ZUSE_CERT_DEVBLOCK_VERSION
zuse_cert_devblock.h, 854
zuse_cert_installed
VM, 336
zuse_cert_pubkey
VM, 336
zuse_cert_seed
VM, 336
zuse_genesis_marker.h
ZUSE_GENESIS_MARKER_MAGIC, 856
ZUSE_GENESIS_MARKER_VERSION, 856
ZUSE_GENESIS_MARKER_VERSION_2, 856
ZUSE_GENESIS_MARKER_MAGIC
zuse_genesis_marker.h, 856
zuse_genesis_marker_t, 395
_pad, 395
crc, 395
magic, 395
version, 395

zuse_pubkey, 395
zuse_genesis_marker_v2_t, 396
_pad, 396
_reserved, 396
crc, 396
has_real, 396
magic, 396
proxy_pubkey, 396
real_pubkey, 396
version, 396
ZUSE_GENESIS_MARKER_VERSION
zuse_genesis_marker.h, 856
ZUSE_GENESIS_MARKER_VERSION_2
zuse_genesis_marker.h, 856
zuse_pubkey
zuse_genesis_marker_t, 395
zuse_root_pubkey
VM, 337
zuse_root_pubkey_known
VM, 337
zuse_session
VM, 337