# ChibiOS/HAL

## 1. ChibiOS/HAL

1.1 Copyright ................................................. 1
1.2 Introduction .............................................. 1
1.3 Related Documents ....................................... 1

## 2. Deprecated List ........................................... 3

## 3. Module Index ............................................... 5

3.1 Modules .................................................. 5

## 4. Hierarchical Index ....................................... 7

4.1 Class Hierarchy ........................................... 7

## 5. Data Structure Index .................................... 11

5.1 Data Structures ......................................... 11

## 6. File Index ................................................ 17

6.1 File List ................................................ 17

## 7. Module Documentation .................................... 23

7.1 HAL Driver ................................................ 23

7.1.1 Detailed Description .................................. 23

7.1.2 Macro Definition Documentation ....................... 25

7.1.3 Typedef Documentation .................................. 27

7.1.4 Function Documentation ................................ 28

7.1.11 HAL_RET_HW_BUSY .................................... 27

7.1.12 HAL_RET_HW_FAILURE ................................. 27

7.1.13 HAL_RET_UNKNOWN_CTL ............................... 27

7.1.14 HAL_LLTD_SELECT_SPI_V2 ............................ 27

7.1.15 HAL_LLTD_USE_CLOCK_MANAGEMENT .................. 27

7.1.16 Typedef Documentation ................................. 27

7.1.17 Macro Definition Documentation ...................... 29

7.1.18 Function Documentation ................................ 29

7.1.19 HAL_Init() ........................................... 29

7.1.20 HAL_ClockSwitchMode() ............................... 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.4.3</td>
<td>ibqGetEmptyBuffer()</td>
</tr>
<tr>
<td>7.4.4.4</td>
<td>ibqPostFullBuffer()</td>
</tr>
<tr>
<td>7.4.4.5</td>
<td>ibqGetFullBufferTimeout()</td>
</tr>
<tr>
<td>7.4.4.6</td>
<td>ibqGetFullBufferTimeoutS()</td>
</tr>
<tr>
<td>7.4.4.7</td>
<td>ibqReleaseEmptyBuffer()</td>
</tr>
<tr>
<td>7.4.4.8</td>
<td>ibqReleaseEmptyBufferS()</td>
</tr>
<tr>
<td>7.4.4.9</td>
<td>ibqGetTimeout()</td>
</tr>
<tr>
<td>7.4.4.10</td>
<td>ibqReadTimeout()</td>
</tr>
<tr>
<td>7.4.4.11</td>
<td>obqObjectInit()</td>
</tr>
<tr>
<td>7.4.4.12</td>
<td>obqReset()</td>
</tr>
<tr>
<td>7.4.4.13</td>
<td>obqGetFullBuffer()</td>
</tr>
<tr>
<td>7.4.4.14</td>
<td>obqReleaseEmptyBuffer()</td>
</tr>
<tr>
<td>7.4.4.15</td>
<td>obqGetEmptyBufferTimeout()</td>
</tr>
<tr>
<td>7.4.4.16</td>
<td>obqGetEmptyBufferTimeoutS()</td>
</tr>
<tr>
<td>7.4.4.17</td>
<td>obqPostFullBuffer()</td>
</tr>
<tr>
<td>7.4.4.18</td>
<td>obqPostFullBufferS()</td>
</tr>
<tr>
<td>7.4.4.19</td>
<td>obqPutTimeout()</td>
</tr>
<tr>
<td>7.4.4.20</td>
<td>obqWriteTimeout()</td>
</tr>
<tr>
<td>7.4.4.21</td>
<td>obqTryFlush()</td>
</tr>
<tr>
<td>7.4.4.22</td>
<td>obqFlush()</td>
</tr>
</tbody>
</table>

7.5 CAN Driver

7.5.1 Detailed Description

7.5.2 Driver State Machine

7.5.3 Macro Definition Documentation

7.5.3.1 CAN_LIMIT_WARNING
7.5.3.2 CAN_LIMIT_ERROR
7.5.3.3 CAN_BUS_OFF_ERROR
7.5.3.4 CAN_FRAMING_ERROR
7.5.3.5 CAN_OVERFLOW_ERROR
7.5.3.6 CAN_ANY_MAILBOX
7.5.3.7 CAN_USE_SLEEP_MODE
7.5.3.8 CAN_ENFORCE_USE_CALLBACKS
7.5.3.9 CAN_MAILBOX_TO_MASK
7.5.3.10 canTransmit
7.5.3.11 canReceive
7.5.3.12 _can_tx_empty_isr
7.5.3.13 _can_rx_full_isr
7.5.3.14 _can_wakeup_isr
7.5.3.15 _can_error_isr
7.5.3.16 CAN_TX_MAILBOXES
7.5.3.17 CAN_RX_MAILBOXES
7.5.3.18 PLATFORM_CAN_USE_CAN1
7.6.4 Enumeration Type Documentation ............................................. 116
  7.6.4.1 crystate_t ............................................................... 116
  7.6.4.2 cryerror_t ............................................................... 116
  7.6.4.3 cryalgorithm_t ......................................................... 117
7.6.5 Function Documentation ..................................................... 117
  7.6.5.1 cryInit() .................................................................. 117
  7.6.5.2 cryObjectInit() ............................................................ 118
  7.6.5.3 cryStart() .................................................................. 118
  7.6.5.4 cryStop() ................................................................... 119
  7.6.5.5 cryLoadAESTransientKey() .............................................. 120
  7.6.5.6 cryEncryptAES() ............................................................. 121
  7.6.5.7 cryDecryptAES() ............................................................. 122
  7.6.5.8 cryEncryptAES_ECB() ..................................................... 123
  7.6.5.9 cryDecryptAES_ECB() ..................................................... 124
  7.6.5.10 cryEncryptAES_CBC() .................................................... 125
  7.6.5.11 cryDecryptAES_CBC() .................................................... 126
  7.6.5.12 cryEncryptAES_CFB() .................................................... 127
  7.6.5.13 cryDecryptAES_CFB() .................................................... 128
  7.6.5.14 cryEncryptAES_CTR() .................................................... 129
  7.6.5.15 cryDecryptAES_CTR() .................................................... 130
  7.6.5.16 cryEncryptAES_GCM() ................................................... 131
  7.6.5.17 cryDecryptAES_GCM() ................................................... 132
  7.6.5.18 cryLoadDESTransientKey() ............................................ 134
  7.6.5.19 cryEncryptDES() ........................................................... 135
  7.6.5.20 cryDecryptDES() ........................................................... 136
  7.6.5.21 cryEncryptDES_ECB() .................................................... 137
  7.6.5.22 cryDecryptDES_ECB() .................................................... 138
  7.6.5.23 cryEncryptDES_CBC() .................................................... 139
  7.6.5.24 cryDecryptDES_CBC() .................................................... 140
  7.6.5.25 crySHA1Init() ............................................................... 141
  7.6.5.26 crySHA1Update() .......................................................... 142
  7.6.5.27 crySHA1Final() ............................................................. 143
  7.6.5.28 crySHA256Init() ........................................................... 144
  7.6.5.29 crySHA256Update() ....................................................... 145
  7.6.5.30 crySHA256Final() .......................................................... 146
  7.6.5.31 crySHA512Init() ............................................................ 147
  7.6.5.32 crySHA512Update() ....................................................... 148
  7.6.5.33 crySHA512Final() .......................................................... 149
  7.6.5.34 cryLoadHMACTransientKey() ......................................... 150
  7.6.5.35 cryHMACSHA256Init() ................................................... 151
  7.6.5.36 cryHMACSHA256Update() ............................................. 152
  7.6.5.37 cryHMACSHA256Final() ................................................ 153
7.9.1 Detailed Description ............................................. 223
7.9.2 Driver State Machine ........................................... 223
7.9.3 Macro Definition Documentation ............................... 225
  7.9.3.1 FLASH_ATTR_ERASED_IS_ONE ............................ 225
  7.9.3.2 FLASH_ATTR_MEMORY_MAPPED ............................. 226
  7.9.3.3 FLASH_ATTR_REWRITABLE ............................... 226
  7.9.3.4 FLASH_ATTR_ECC_CAPABLE .............................. 226
  7.9.3.5 FLASH_ATTR_ECC_ZERO_LINE_CAPABLE ...................... 226
  7.9.3.6 FLASH_ATTR_SUSPEND_ERASE_CAPABLE .................... 227
  7.9.3.7 _base_flash_methods_alone .............................. 227
  7.9.3.8 _base_flash_methods .................................... 227
  7.9.3.9 _base_flash_data ........................................ 227
  7.9.3.10 getBaseFlash ............................................ 228
  7.9.3.11 flashGetDescriptor ..................................... 228
  7.9.3.12 flashRead ............................................... 228
  7.9.3.13 flashProgram ............................................ 229
  7.9.3.14 flashStartEraseAll ..................................... 230
  7.9.3.15 flashStartEraseSector .................................. 230
  7.9.3.16 flashQueryErase ........................................ 231
  7.9.3.17 flashVerifyErase ........................................ 231
  7.9.3.18 flashAcquireExclusive ................................. 232
  7.9.3.19 flashReleaseExclusive ................................. 232
7.9.4 Typedef Documentation .......................................... 233
  7.9.4.1 flash_offset_t .......................................... 233
  7.9.4.2 flash_sector_t .......................................... 233
7.9.5 Enumeration Type Documentation ............................... 233
  7.9.5.1 flash_state_t ............................................ 233
  7.9.5.2 flash_error_t ............................................ 234
7.9.6 Function Documentation ........................................ 234
  7.9.6.1 flashWaitErase() ........................................ 234
  7.9.6.2 flashGetSectorOffset() ................................ 234
  7.9.6.3 flashGetSectorSize() ................................... 235
  7.9.6.4 flashGetOffsetSector() .................................. 235
  7.9.6.5 flashGetOffsetAddress() ................................. 236
  7.9.6.6 flashGetAddressOffset() ................................. 236
7.10 GPT Driver ......................................................... 237
  7.10.1 Detailed Description ...................................... 237
  7.10.2 Driver State Machine ..................................... 237
  7.10.3 GPT Operations ............................................. 238
  7.10.4 Macro Definition Documentation ........................... 240
    7.10.4.1 gptChangeInterval() .................................. 240
    7.10.4.2 gptGetInterval() ...................................... 240
7.12.3.6 PLATFORM_I2S_USE_I2S1 ................................................................................. 277
7.12.3.7 i2s_lld_driver_fields .................................................................................. 278
7.12.3.8 i2s_lld_config_fields . .................................................................................. 278
7.12.4 Typedef Documentation .................................................................................. 278
7.12.4.1 I2SDriver . .................................................................................................. 278
7.12.4.2 I2SConfig ................................................................................................... 278
7.12.4.3 i2scallback_t . ............................................................................................. 278
7.12.5 Enumeration Type Documentation .................................................................. 279
7.12.5.1 i2sstate_t . .................................................................................................. 279
7.12.6 Function Documentation ............................................................................... 279
7.12.6.1 i2sInit() ..................................................................................................... 279
7.12.6.2 i2sObjectInit() ........................................................................................... 280
7.12.6.3 i2sStart() . ................................................................................................. 280
7.12.6.4 i2sStop() . .................................................................................................. 281
7.12.6.5 i2sStartExchange() .................................................................................... 282
7.12.6.6 i2sStopExchange() ..................................................................................... 283
7.12.6.7 i2s_lld_init() .............................................................................................. 283
7.12.6.8 i2s_lld_start() ........................................................................................... 284
7.12.6.9 i2s_lld_stop() ............................................................................................. 284
7.12.6.10 i2s_lld_start_exchange() . ...................................................................... 284
7.12.6.11 i2s_lld_stop_exchange() ......................................................................... 285
7.12.7 Variable Documentation ................................................................................. 285
7.12.7.1 I2SD1 . ......................................................................................................... 285
7.13 ICU Driver ........................................................................................................ 286
7.13.1 Detailed Description ....................................................................................... 286
7.13.2 Driver State Machine ..................................................................................... 286
7.13.3 ICU Operations. .............................................................................................. 286
7.13.4 Macro Definition Documentation ................................................................... 289
7.13.4.1 icuStartCaptureI ......................................................................................... 289
7.13.4.2 icuStopCaptureI ......................................................................................... 290
7.13.4.3 icuEnableNotificationsI ............................................................................ 290
7.13.4.4 icuDisableNotificationsI ........................................................................... 291
7.13.4.5 icuAreNotificationsEnabledI . ................................................................... 291
7.13.4.6 icuGetWidthI ............................................................................................. 292
7.13.4.7 icuGetPeriodI ............................................................................................ 292
7.13.4.8 _icu_isr_invoke_width_cb ........................................................................ 293
7.13.4.9 _icu_isr_invoke_period_cb ....................................................................... 293
7.13.4.10 _icu_isr_invoke_overflow_cb ................................................................... 294
7.13.4.11 PLATFORM_ICU_USE_ICU1 ..................................................................... 294
7.13.4.12 icu_lld_get_width ..................................................................................... 295
7.13.4.13 icu_lld_get_period . .................................................................................. 295
7.13.4.14 icu_lld_are_notifications_enabled . ......................................................... 295
7.13.5 Typedef Documentation ............................................................. 296
7.13.5.1 ICUDriver ........................................................................... 296
7.13.5.2 icucallback_t ..................................................................... 296
7.13.5.3 icufreq_t ........................................................................... 296
7.13.5.4 icucnt_t ............................................................................. 297
7.13.6 Enumeration Type Documentation ............................................. 297
7.13.6.1 icustate_t .......................................................................... 297
7.13.6.2 icumode_t ......................................................................... 297
7.13.7 Function Documentation ............................................................ 297
7.13.7.1 icuInit() ........................................................................... 298
7.13.7.2 icuObjectInit() .................................................................. 298
7.13.7.3 icuStart() .......................................................................... 298
7.13.7.4 icuStop() .......................................................................... 300
7.13.7.5 icuStartCapture() ................................................................. 301
7.13.7.6 icuWaitCapture() ................................................................. 302
7.13.7.7 icuStopCapture() ................................................................. 303
7.13.7.8 icuEnableNotifications() ..................................................... 304
7.13.7.9 icuDisableNotifications() .................................................... 304
7.13.7.10 icu_lld_init() .................................................................. 305
7.13.7.11 icu_lld_start() ................................................................. 305
7.13.7.12 icu_lld_stop() ................................................................. 306
7.13.7.13 icu_lld_start_capture() .................................................... 306
7.13.7.14 icu_lld_wait_capture() ..................................................... 306
7.13.7.15 icu_lld_stop_capture() ..................................................... 307
7.13.7.16 icu_lld_enable_notifications() ........................................... 307
7.13.7.17 icu_lld_disable_notifications() .......................................... 308
7.13.8 Variable Documentation ........................................................... 308
7.13.8.1 ICUD1 ............................................................................. 309
7.14 Blocks ......................................................................................... 310
7.14.1 Detailed Description .................................................................. 310
7.15 Streams ....................................................................................... 311
7.15.1 Detailed Description .................................................................. 311
7.16 Abstract Streams Interface ............................................................ 312
7.16.1 Detailed Description .................................................................. 312
7.16.2 Macro Definition Documentation .............................................. 313
7.16.2.1 _base_sequential_stream_methods ................................... 313
7.16.2.2 _base_sequential_stream_data .......................................... 313
7.16.2.3 streamWrite ...................................................................... 313
7.16.2.4 streamRead ....................................................................... 314
7.16.2.5 streamPut ......................................................................... 314
7.16.2.6 streamGet ......................................................................... 315
7.16.2.7 _base_buffered_stream_methods ....................................... 316
7.22.3.1 BufferedSerial ................................................. 347
7.22.4 Enumeration Type Documentation ......................... 347
  7.22.4.1 bstate_t .................................................. 347
7.22.5 Function Documentation ..................................... 348
  7.22.5.1 bsIncomingDataI() .................................... 348
  7.22.5.2 bsRequestDataI() ..................................... 349
  7.22.5.3 __buffered_serial_objinit_impl() ..................... 349
7.23 Buffered Streams Class ......................................... 351
  7.23.1 Detailed Description ..................................... 351
7.23.2 Macro Definition Documentation ......................... 351
    7.23.2.1 BUFSTREAM_BUFFER_SIZE .......................... 351
    7.23.2.2 _buffered_stream_adapter_data .................. 351
7.23.3 Function Documentation ................................... 352
    7.23.3.1 bsaObjectInit() ................................... 352
7.24 Output Formatter Utility ....................................... 353
  7.24.1 Detailed Description ..................................... 353
7.24.2 Macro Definition Documentation ......................... 353
    7.24.2.1 CHPRINTF_USE_FLOAT .................................. 353
7.24.3 Function Documentation ................................... 353
    7.24.3.1 chvprintf() .......................................... 354
    7.24.3.2 chprintf() .......................................... 354
    7.24.3.3 chsnprintf() ........................................ 356
    7.24.3.4 chvsnprintf() ....................................... 357
7.25 Input Formatter Utility ........................................ 359
  7.25.1 Detailed Description ..................................... 359
7.25.2 Macro Definition Documentation ......................... 359
    7.25.2.1 CHSCANF_USE_FLOAT .................................. 359
7.25.3 Function Documentation ................................... 359
    7.25.3.1 chvscanf() .......................................... 360
    7.25.3.2 chscanf() .......................................... 360
    7.25.3.3 chsnscanf() ........................................ 362
    7.25.3.4 chvsnscanf() ....................................... 363
7.26 Abstract I/O Block Device .................................... 365
  7.26.1 Detailed Description ..................................... 365
7.26.2 Driver State Machine ....................................... 365
7.26.3 Macro Definition Documentation ......................... 366
    7.26.3.1 _base_block_device_methods ......................... 366
    7.26.3.2 _base_block_device_data .......................... 367
    7.26.3.3 blkGetDriverState .................................. 367
    7.26.3.4 blkIsTransferring .................................. 367
    7.26.3.5 blkIsInserted ....................................... 368
    7.26.3.6 blkIsWriteProtected .................................. 369
7.26.3.7 blkConnect .............................................. 369
7.26.3.8 blkDisconnect ......................................... 370
7.26.3.9 blkRead .................................................. 370
7.26.3.10 blkWrite ............................................... 371
7.26.3.11 blkSync ................................................ 372
7.26.3.12 blkGetInfo ........................................... 372
7.26.4 Enumeration Type Documentation ...................... 373
  7.26.4.1 blkstate_t ........................................... 373

7.27 MAC Driver .................................................... 374
  7.27.1 Detailed Description .................................... 374
  7.27.2 Macro Definition Documentation ....................... 377
    7.27.2.1 MAC_USE_ZERO_COPY ................................ 377
    7.27.2.2 MAC_USE_EVENTS .................................... 377
    7.27.2.3 __mac_callback ..................................... 377
    7.27.2.4 __mac_tx_wakeup ................................... 378
    7.27.2.5 __mac_rx_wakeup ................................... 378
    7.27.2.6 macSetCallbackX ..................................... 378
    7.27.2.7 macGetEventSource .................................. 379
    7.27.2.8 macGetTransmitDescriptorX ......................... 379
    7.27.2.9 macReleaseTransmitDescriptorX ...................... 380
    7.27.2.10 macGetReceiveDescriptorX .......................... 380
    7.27.2.11 macReleaseReceiveDescriptorX ....................... 381
    7.27.2.12 macWriteTransmitDescriptor ......................... 381
    7.27.2.13 macReadReceiveDescriptor .......................... 382
    7.27.2.14 macGetNextTransmitBuffer ......................... 382
    7.27.2.15 macGetNextReceiveBuffer ........................... 383
    7.27.2.16 MAC_SUPPORTS_ZERO_COPY ............................ 383
    7.27.2.17 PLATFORM_MAC_USE_MAC1 ............................. 384
    7.27.2.18 mac_lld_driver_fields ............................... 384
    7.27.2.19 mac_lld_config_fields .............................. 384
    7.27.2.20 mac_lld_transmit_descriptor_fields ................ 384
    7.27.2.21 mac_lld_receive_descriptor_fields .................. 384
  7.27.3 Typedef Documentation ................................ 384
    7.27.3.1 MACDriver .......................................... 385
    7.27.3.2 MACConfig ........................................... 385
    7.27.3.3 MACTransmitDescriptor .............................. 385
    7.27.3.4 MACReceiveDescriptor ............................... 385
    7.27.3.5 maccb_t ............................................. 385
  7.27.4 Enumeration Type Documentation ....................... 385
    7.27.4.1 macstate_t ......................................... 386
  7.27.5 Function Documentation ................................ 386
    7.27.5.1 macInit() ........................................... 386
7.27.5.2 macObjectInit() ................................................. 386
7.27.5.3 macStart() .................................................. 387
7.27.5.4 macStop() .................................................. 388
7.27.5.5 macGetAndClearEventsI() .................................... 389
7.27.5.6 macWaitTransmitDescriptor() .................................. 389
7.27.5.7 macWaitReceiveDescriptor() .................................. 390
7.27.5.8 macPollLinkStatus() ........................................ 391
7.27.5.9 mac_lld_init() .............................................. 392
7.27.5.10 mac_lld_start() ............................................ 392
7.27.5.11 mac_lld_stop() ............................................. 393
7.27.5.12 mac_lld_get_transmit_descriptor() ......................... 393
7.27.5.13 mac_lld_release_transmit_descriptor() ..................... 394
7.27.5.14 mac_lld_get_receive_descriptor() ......................... 394
7.27.5.15 mac_lld_release_receive_descriptor() ....................... 395
7.27.5.16 mac_lld_poll_link_status() .................................. 395
7.27.5.17 mac_lld_write_transmit_descriptor() ....................... 396
7.27.5.18 mac_lld_read_receive_descriptor() ......................... 396
7.27.5.19 mac_lld_get_next_transmit_buffer() ......................... 397
7.27.5.20 mac_lld_get_next_receive_buffer() ......................... 397
7.27.6 Variable Documentation ........................................ 398
7.27.6.1 ETHD1 ...................................................... 398
7.28 Managed Flash Storage Driver .................................... 399
7.28.1 Detailed Description ........................................... 399
7.28.2 Macro Definition Documentation ............................... 401
7.28.2.1 ALIGNED_REC_SIZE ......................................... 402
7.28.2.2 ALIGNED_DHDR_SIZE ....................................... 402
7.28.2.3 ALIGNED_SIZEOF ........................................... 402
7.28.2.4 PAIR ....................................................... 402
7.28.2.5 RET_ON_ERROR .............................................. 402
7.28.2.6 MFS_CFG_MAX_RECORDS ..................................... 403
7.28.2.7 MFS_CFG_MAX_REPAIR_ATTEMPTS ............................ 403
7.28.2.8 MFS_CFG_WRITE_VERIFY ..................................... 403
7.28.2.9 MFS_CFG_STRONG_CHECKING .................................. 403
7.28.2.10 MFS_CFG_BUFFER_SIZE .................................... 403
7.28.2.11 MFS_CFG_MEMORY_ALIGNMENT ............................... 404
7.28.2.12 MFS_CFG_TRANSACTION_MAX ................................. 404
7.28.3 Typedef Documentation ......................................... 404
7.28.3.1 mfs_id_t .................................................. 404
7.28.3.2 mfs_nocache_buffer_t ...................................... 404
7.28.4 Enumeration Type Documentation ............................... 404
7.28.4.1 mfs_bank_t ................................................ 404
7.28.4.2 mfs_state_t ............................................... 405
7.28.5 Function Documentation

7.28.5.1 mfs_flash_read()

7.28.5.2 mfs_flash_write()

7.28.5.3 mfs_flash_copy()

7.28.5.4 mfs_bank_erase()

7.28.5.5 mfs_bank_verify_erase()

7.28.5.6 mfs_bank_write_header()

7.28.5.7 mfs_bank_check_header()

7.28.5.8 mfs_bank_scan_records()

7.28.5.9 mfs_bank_get_state()

7.28.5.10 mfs_garbage_collect()

7.28.5.11 mfs_try_mount()

7.28.5.12 mfs_mount()

7.28.5.13 mfsObjectInit()

7.28.5.14 mfsStart()

7.28.5.15 mfsStop()

7.28.5.16 mfsErase()

7.28.5.17 mfsReadRecord()

7.28.5.18 mfsWriteRecord()

7.28.5.19 mfsEraseRecord()

7.28.5.20 mfsPerformGarbageCollection()

7.28.5.21 mfsStartTransaction()

7.28.5.22 mfsCommitTransaction()

7.28.5.23 mfsRollbackTransaction()

7.29 MII/RMII Header

7.29.1 Detailed Description

7.29.2 Macro Definition Documentation

7.29.2.1 MII_BMCR

7.29.2.2 MII_BMSR

7.29.2.3 MII_PHYSID1

7.29.2.4 MII_PHYSID2

7.29.2.5 MII_ADVERTISE

7.29.2.6 MII_LPA

7.29.2.7 MII_EXPANSION

7.29.2.8 MII_ANNPTR

7.29.2.9 MII_CTRL1000

7.29.2.10 MII_STAT1000

7.29.2.11 MII_ESTATUS

7.29.2.12 MII_PHYSTS

7.29.2.13 MII_MICR
7.29.2.14 MII_DCOUNTER ........................................ 424
7.29.2.15 MII_FSCOUNTER ....................................... 424
7.29.2.16 MII_NWAYTEST .......................................... 424
7.29.2.17 MII_RERRCOUNTER ...................................... 425
7.29.2.18 MII_SREVOLUTION ........................................ 425
7.29.2.19 MII_RESV1 .............................................. 425
7.29.2.20 MII_LBRERROR ........................................... 425
7.29.2.21 MII_PHYADDR ............................................. 425
7.29.2.22 MII_RESV2 ............................................... 425
7.29.2.23 MII_TPISTATUS .......................................... 425
7.29.2.24 MII_NCONFIG ............................................. 425
7.29.2.25 BMCR_RESV .............................................. 426
7.29.2.26 BMCR_CTST .............................................. 426
7.29.2.27 BMCR_FULLDPLX ......................................... 426
7.29.2.28 BMCR_ANRESTART ....................................... 426
7.29.2.29 BMCR_ISOLATE .......................................... 426
7.29.2.30 BMCR_PDOWN ............................................ 426
7.29.2.31 BMCR_ANENABLE ......................................... 427
7.29.2.32 BMCR_SPEED100 ......................................... 427
7.29.2.33 BMCR_LOOPBACK ........................................ 427
7.29.2.34 BMCR_RESET ............................................. 427
7.29.2.35 BMSR_ERCAP ............................................ 427
7.29.2.36 BMSR_JCD .............................................. 427
7.29.2.37 BMSR_LSTATUS .......................................... 427
7.29.2.38 BMSR_ANECAPABLE ...................................... 428
7.29.2.39 BMSR_RFAULT ........................................... 428
7.29.2.40 BMSR_ANECOMPLETE .................................... 428
7.29.2.41 BMSR_MFPRESUPPCAP .................................. 428
7.29.2.42 BMSR_RESV .............................................. 428
7.29.2.43 BMSR_10HALF ............................................ 428
7.29.2.44 BMSR_10FULL ............................................ 428
7.29.2.45 BMSR_100HALF ........................................... 429
7.29.2.46 BMSR_100FULL ........................................... 429
7.29.2.47 BMSR_100BASE4 ......................................... 429
7.29.2.48 ADVERTISE_SLCT ....................................... 429
7.29.2.49 ADVERTISE_CSMA ....................................... 429
7.29.2.50 ADVERTISE_10HALF ..................................... 429
7.29.2.51 ADVERTISE_10FULL ...................................... 429
7.29.2.52 ADVERTISE_100HALF .................................... 430
7.29.2.53 ADVERTISE_100FULL .................................... 430
7.29.2.54 ADVERTISE_100BASE4 .................................. 430
7.29.2.55 ADVERTISE_PAUSE_CAP ................................. 430
7.29.2.56 ADVERTISE_PAUSEASYM ............................................. 430
7.29.2.57 ADVERTISE_RESV .................................................. 430
7.29.2.58 ADVERTISE_RFAL U T .............................................. 430
7.29.2.59 ADVERTISE_LPACK ............................................... 431
7.29.2.60 ADVERTISE_NPAGE ............................................... 431
7.29.2.61 LPA_SLCT ......................................................... 431
7.29.2.62 LPA_10HALF ....................................................... 431
7.29.2.63 LPA_10FULL ......................................................... 431
7.29.2.64 LPA_100HALF ....................................................... 431
7.29.2.65 LPA_100FULL ....................................................... 431
7.29.2.66 LPA_100BASE4 ..................................................... 432
7.29.2.67 LPA_PAUSE_CAP ................................................... 432
7.29.2.68 LPA_PAUSEASYM .................................................. 432
7.29.2.69 LPA_RESV .......................................................... 432
7.29.2.70 LPA_RFAL U T ....................................................... 432
7.29.2.71 LPA_LPACK .......................................................... 432
7.29.2.72 LPA_NPAGE ........................................................ 432
7.29.2.73 EXPANSION_NWAY ............................................... 433
7.29.2.74 EXPANSION_LCW P ............................................... 433
7.29.2.75 EXPANSION_ENABLEPAGE ...................................... 433
7.29.2.76 EXPANSION_NPCAPABLE ....................................... 433
7.29.2.77 EXPANSION_MFAULTS .......................................... 433
7.29.2.78 EXPANSION_RESV ............................................... 433
7.29.2.79 NWAYTEST_RESV1 .............................................. 433
7.29.2.80 NWAYTEST_LOOPBACK ......................................... 434
7.29.2.81 NWAYTEST_RESV2 ............................................... 434

7.30 MMC over SPI Driver ..................................................... 435
    7.30.1 Detailed Description ............................................. 435
    7.30.2 Driver State Machine ........................................... 435
    7.30.3 Driver Operations ............................................... 435
    7.30.4 Macro Definition Documentation ............................... 437
        7.30.4.1 MMC_BUFFER_SIZE ........................................ 437
        7.30.4.2 MMC_IDLE_TIMEOUT_MS ................................... 437
        7.30.4.3 MMC_USE_MUTUAL_EXCLUSION .............................. 438
        7.30.4.4 __mmc_driver_methods .................................... 438
        7.30.4.5 mmcIsCardInserted ........................................ 438
        7.30.4.6 mmcIsWriteProtected ...................................... 439
    7.30.5 Typedef Documentation .......................................... 439
        7.30.5.1 MMCConfig .................................................. 439
        7.30.5.2 MMCDriver .................................................. 439
    7.30.6 Function Documentation ........................................ 440
        7.30.6.1 mmc_crc7() ................................................ 440
7.30.6.2 mmc_wait_idle() ................................................. 440
7.30.6.3 mmc_send_hdr() ................................................. 441
7.30.6.4 mmc_recrv1() .................................................... 442
7.30.6.5 mmc_recrv3() .................................................... 442
7.30.6.6 mmc_send_command_R1() ......................................... 443
7.30.6.7 mmc_send_command_R3() ......................................... 444
7.30.6.8 mmc_read_CxD() ................................................... 444
7.30.6.9 mmcInit() ......................................................... 445
7.30.6.10 mmcObjectInit() ................................................ 445
7.30.6.11 mmcStart() ....................................................... 446
7.30.6.12 mmcStop() ......................................................... 446
7.30.6.13 mmcConnect() .................................................... 446
7.30.6.14 mmcDisconnect() ................................................ 447
7.30.6.15 mmcStartSequentialRead() .................................... 448
7.30.6.16 mmcSequentialRead() .......................................... 449
7.30.6.17 mmcStopSequentialRead() ...................................... 449
7.30.6.18 mmcStartSequentialWrite() ................................... 450
7.30.6.19 mmcSequentialWrite() ......................................... 450
7.30.6.20 mmcStopSequentialWrite() ..................................... 451
7.30.6.21 mmcSync() ....................................................... 451
7.30.6.22 mmcGetInfo() .................................................... 452
7.30.6.23 mmcErase() ....................................................... 453

7.30.7 Variable Documentation ............................................ 453
7.30.7.1 mmc_vmt .......................................................... 453
7.30.7.2 mmc_crc7_lookup_table ......................................... 454

7.31 MMC/SD Block Device .................................................. 455
7.31.1 Detailed Description ................................................ 455
7.31.2 Macro Definition Documentation .................................. 459
7.31.2.1 MMC_BLOG_SIZE ................................................. 459
7.31.2.2 MMCSD_R1_ERROR_MASK ........................................ 460
7.31.2.3 MMC_SD8_PATTERN ............................................... 460
7.31.2.4 _mmcsd_block_device_methods ................................ 460
7.31.2.5 _mmcsd_block_device_data .................................... 460
7.31.2.6 MMCSD_R1_ERROR ................................................ 460
7.31.2.7 MMCSD_R1_STS .................................................. 461
7.31.2.8 MMCSD_R1_IS_CARD_LOCKED ................................... 461
7.31.2.9 mmcsdGetCardCapacity ........................................ 461

7.31.3 Function Documentation ............................................ 462
7.31.3.1 _mmcsd_get_slice() ............................................. 462
7.31.3.2 _mmcsd_get_capacity() ......................................... 462
7.31.3.3 _mmcsd_get_capacity_ext() .................................... 463
7.31.3.4 _mmcsd_unpack_sdc_cid() ...................................... 464
7.31.3.5_mmcsd_unpack_mmc_cid() ................................................................. 464
7.31.3.6_mmcsd_unpack_csd_mmc() ............................................................. 465
7.31.3.7_mmcsd_unpack_csd_v10() ............................................................... 466
7.31.3.8_mmcsd_unpack_csd_v20() ............................................................... 466

7.32 Serial NOR Flash Driver ................................................................. 468
7.32.1 Detailed Description ........................................................................ 468
7.32.2 Driver State Machine ..................................................................... 468
7.32.3 Macro Definition Documentation .................................................. 470
  7.32.3.1 SNOR_BUFFER_SIZE ................................................................. 470
  7.32.3.2 SNOR_BUS_DRIVER ................................................................. 470
  7.32.3.3 SNOR_SHARED_BUS ............................................................... 470
  7.32.3.4 SNOR_USE_MUTUAL_EXCLUSION ............................................ 471
  7.32.3.5 SNOR_SPI_4BYTES_ADDRESS .................................................. 471
  7.32.3.6 _snor_flash_methods_alone ....................................................... 471
  7.32.3.7 _snor_flash_methods ............................................................... 471
7.32.4 Function Documentation .............................................................. 471
  7.32.4.1 snor_get_descriptor() .............................................................. 471
  7.32.4.2 bus_acquire() ......................................................................... 472
  7.32.4.3 bus_release() ........................................................................... 472
  7.32.4.4 bus_stop() ................................................................................ 473
  7.32.4.5 bus_cmd() ................................................................................ 473
  7.32.4.6 bus_cmd_send() ....................................................................... 473
  7.32.4.7 bus_cmd_receive() ................................................................. 474
  7.32.4.8 bus_cmd_addr() ....................................................................... 474
  7.32.4.9 bus_cmd_addr_send() ............................................................... 474
  7.32.4.10 bus_cmd_addr_receive() .......................................................... 475
  7.32.4.11 bus_cmd_dummy_receive() ...................................................... 476
  7.32.4.12 bus_cmd_addr_dummy_receive() ............................................. 476
  7.32.4.13 snorObjectInit() ..................................................................... 477
  7.32.4.14 snorStart() ............................................................................. 477
  7.32.4.15 snorStop() ............................................................................. 477
  7.32.4.16 snorMemoryMap() .................................................................. 478
  7.32.4.17 snorMemoryUnmap() .............................................................. 478
7.32.5 Variable Documentation ................................................................ 479
  7.32.5.1 snor_vmt .................................................................................. 479

7.33 Base Object ....................................................................................... 480
7.33.1 Detailed Description ...................................................................... 480
7.33.2 Macro Definition Documentation ................................................ 480
  7.33.2.1__base_object_methods ............................................................ 480
  7.33.2.2__base_object_data ................................................................. 481
  7.33.2.3 objGetInstance ........................................................................... 481

7.34 PAL Driver .......................................................................................... 482
7.34.1 Detailed Description ................................. 482
7.34.2 Implementation Rules .............................. 482
  7.34.2.1 Writing on input pads ......................... 482
  7.34.2.2 Reading from output pads .................... 482
  7.34.2.3 Writing unused or unimplemented port bits .... 483
  7.34.2.4 Reading from unused or unimplemented port bits 483
  7.34.2.5 Reading or writing on pins associated to other functionalities 483
7.34.3 Macro Definition Documentation ................. 488
  7.34.3.1 PAL_MODE_RESET ............................. 488
  7.34.3.2 PAL_MODE_UNCONNECTED ..................... 489
  7.34.3.3 PAL_MODE_INPUT ............................. 489
  7.34.3.4 PAL_MODE_INPUT_PULLUP ..................... 489
  7.34.3.5 PAL_MODE_INPUT_PULLDOWN ................. 489
  7.34.3.6 PAL_MODE_INPUT_ANALOG .................... 489
  7.34.3.7 PAL_MODE_OUTPUT_PUSHPULL ................. 489
  7.34.3.8 PAL_MODE_OUTPUT_OPENDRAIN ............... 490
  7.34.3.9 PAL_LOW .................................. 490
  7.34.3.10 PAL_HIGH ................................ 490
  7.34.3.11 PAL_EVENT_MODE_EDGES_MASK ............. 490
  7.34.3.12 PAL_EVENT_MODE_DISABLED ................. 490
  7.34.3.13 PAL_EVENT_MODE_RISING_EDGE .......... 490
  7.34.3.14 PAL_EVENT_MODE_FALLING_EDGE .......... 491
  7.34.3.15 PAL_EVENT_MODE_BOTH_EDGES ............ 491
  7.34.3.16 PAL_USE_CALLBACKS ....................... 491
  7.34.3.17 PAL_USE_WAIT ............................ 491
  7.34.3.18 PAL_PORT_BIT ............................ 491
  7.34.3.19 PAL_GROUP_MASK .......................... 492
  7.34.3.20 _IOBUS_DATA .............................. 492
  7.34.3.21 IOBUS_DECL ................................ 493
  7.34.3.22_pal_init_event [1/3] ...................... 493
  7.34.3.23_pal_init_event [2/3] ...................... 493
  7.34.3.24_pal_init_event [3/3] ...................... 494
  7.34.3.25_pal_clear_event [1/3] .................... 494
  7.34.3.26_pal_clear_event [2/3] .................... 495
  7.34.3.27_pal_clear_event [3/3] .................... 495
  7.34.3.28_pal_isr_code [1/3] ....................... 496
  7.34.3.29_pal_isr_code [2/3] ....................... 496
  7.34.3.30_pal_isr_code [3/3] ....................... 497
  7.34.3.31 palInit .................................. 497
  7.34.3.32 palReadPort ............................. 498
  7.34.3.33 palReadLatch ............................ 498
  7.34.3.34 palWritePort ............................ 499
7.34.3.77 pal_lld_setport
7.34.3.78 pal_lld_clearport
7.34.3.79 pal_lld_toggleport
7.34.3.80 pal_lld_readgroup
7.34.3.81 pal_lld_writegroup
7.34.3.82 pal_lld_setgroupmode
7.34.3.83 pal_lld_readpad
7.34.3.84 pal_lld_writepad
7.34.3.85 pal_lld_setpad
7.34.3.86 pal_lld_clearpad
7.34.3.87 pal_lld_togglepad
7.34.3.88 pal_lld_setpadmode
7.34.3.89 pal_lld_get_pad_event
7.34.3.90 pal_lld_get_line_event

7.34.4 Typedef Documentation
7.34.4.1 palcallback_t
7.34.4.2 iomask_t
7.34.4.3 iomode_t
7.34.4.4 ioline_t
7.34.4.5 iopid_t
7.34.4.6 iopid_t

7.34.5 Function Documentation
7.34.5.1 palReadBus()
7.34.5.2 palWriteBus()
7.34.5.3 palSetBusMode()
7.34.5.4 palSetPadCallback()
7.34.5.5 palSetLineCallback()
7.34.5.6 palWaitPadTimeoutS()
7.34.5.7 palWaitPadTimeout()
7.34.5.8 palWaitLineTimeoutS()
7.34.5.9 palWaitLineTimeout()
7.34.5.10 _pal_lld_init()
7.34.5.11 _pal_lld_setgroupmode()

7.35 Peripheral Interfaces
7.35.1 Detailed Description

7.36 PWM Driver
7.36.1 Detailed Description
7.36.2 Driver State Machine
7.36.3 PWM Operations
7.36.4 Macro Definition Documentation
7.36.4.1 PWM_OUTPUT_MASK
7.36.4.2 PWM_OUTPUT_DISABLED
### 7.36.3 PWM_OUTPUT_ACTIVE_HIGH
- Page 541

### 7.36.4 PWM_OUTPUT_ACTIVE_LOW
- Page 541

### 7.36.5 PWM_FRACTION_TO_WIDTH
- Page 541

### 7.36.6 PWM_DEGREES_TO_WIDTH
- Page 542

### 7.36.7 PWM_PERCENTAGE_TO_WIDTH
- Page 542

### 7.36.8 pwmChangePeriodI
- Page 543

### 7.36.9 pwmEnableChannelI
- Page 544

### 7.36.10 pwmDisableChannelI
- Page 544

### 7.36.11 pwmIsChannelEnabledI
- Page 545

### 7.36.12 pwmEnablePeriodicNotificationI
- Page 546

### 7.36.13 pwmDisablePeriodicNotificationI
- Page 546

### 7.36.14 pwmEnableChannelNotificationI
- Page 547

### 7.36.15 pwmDisableChannelNotificationI
- Page 547

### 7.36.16 PWM_CHANNELS
- Page 548

### 7.36.17 PLATFORM_PWM_USE_PWM1
- Page 548

### 7.36.18 pwm_lld_change_period
- Page 549

### 7.36.5 Typedef Documentation
- Page 549

#### 7.36.5.1 PWMDriver
- Page 549

#### 7.36.5.2 pwmcallback_t
- Page 549

#### 7.36.5.3 pwmmode_t
- Page 550

#### 7.36.5.4 pwmmode_t
- Page 550

#### 7.36.5.5 PWMchannel_t
- Page 550

#### 7.36.5.6 PWMchannel_t
- Page 550

### 7.36.6 Enumeration Type Documentation
- Page 550

#### 7.36.6.1 pwmstate_t
- Page 550

### 7.36.7 Function Documentation
- Page 551

#### 7.36.7.1 pwmInit()
- Page 551

#### 7.36.7.2 pwmObjectInit()
- Page 551

#### 7.36.7.3 pwmStart()
- Page 552

#### 7.36.7.4 pwmStop()
- Page 553

#### 7.36.7.5 pwmChangePeriod()
- Page 553

#### 7.36.7.6 pwmEnableChannel()
- Page 554

#### 7.36.7.7 pwmDisableChannel()
- Page 554

#### 7.36.7.8 pwmEnablePeriodicNotification()
- Page 555

#### 7.36.7.9 pwmDisablePeriodicNotification()
- Page 556

#### 7.36.7.10 pwmEnableChannelNotification()
- Page 557

#### 7.36.7.11 pwmDisableChannelNotification()
- Page 558

#### 7.36.7.12 pwm_lld_init()
- Page 559

#### 7.36.7.13 pwm_lld_start()
- Page 560

#### 7.36.7.14 pwm_lld_stop()
- Page 561

#### 7.36.7.15 pwm_lld_enable_channel()
- Page 561

#### 7.36.7.16 pwm_lld_disable_channel()
- Page 562
7.36.7.17 pwm_lld_enable_periodic_notification() ........................................... 562
7.36.7.18 pwm_lld_disable_periodic_notification() ........................................... 563
7.36.7.19 pwm_lld_enable_channel_notification() ............................................. 563
7.36.7.20 pwm_lld_disable_channel_notification() ............................................. 564
7.36.8 Variable Documentation ........................................................................... 564
7.36.8.1 PWMD1 ................................................................................................. 565
7.37 I/O Bytes Queues ......................................................................................... 566
7.37.1 Detailed Description ................................................................................. 566
7.37.2 Macro Definition Documentation ............................................................. 568
7.37.2.1 Q_OK ..................................................................................................... 568
7.37.2.2 Q_TIMEOUT ......................................................................................... 568
7.37.2.3 Q_RESET ............................................................................................... 568
7.37.2.4 Q_EMPTY ............................................................................................... 569
7.37.2.5 Q_FULL .................................................................................................. 569
7.37.2.6 qSizeX .................................................................................................... 569
7.37.2.7 qSpaceI ................................................................................................. 569
7.37.2.8 qGetLink ............................................................................................... 570
7.37.2.9 qSetLink ............................................................................................... 570
7.37.2.10 iqGetFullI ............................................................................................ 571
7.37.2.11 iqGetEmptyI ....................................................................................... 571
7.37.2.12 iqIsEmptyI ............................................................................................ 572
7.37.2.13 iqIsFullI ............................................................................................... 572
7.37.2.14 iqGet .................................................................................................... 573
7.37.2.15 oqGetFullI ............................................................................................ 574
7.37.2.16 oqGetEmptyI ....................................................................................... 574
7.37.2.17 oqIsEmptyI ............................................................................................ 575
7.37.2.18 oqIsFullI ............................................................................................... 575
7.37.2.19 oqPut .................................................................................................... 576
7.37.3 Typedef Documentation ............................................................................. 576
7.37.3.1 io_queue_t ............................................................................................ 577
7.37.3.2 qnotify_t ............................................................................................... 577
7.37.3.3 input_queue_t .......................................................................................... 577
7.37.3.4 output_queue_t ....................................................................................... 577
7.37.4 Function Documentation .......................................................................... 577
7.37.4.1 iq_read() .............................................................................................. 577
7.37.4.2 oq_write() ........................................................................................... 578
7.37.4.3 iqObjectInit() ....................................................................................... 578
7.37.4.4 iqResetI() ............................................................................................. 579
7.37.4.5 iqPutI() .................................................................................................. 580
7.37.4.6 iqGetI() ................................................................................................. 581
7.37.4.7 iqGetTimeout() ..................................................................................... 582
7.37.4.8 iqReadI() .............................................................................................. 583
7.37.4.9 iqReadTimeout() ................................................. 583
7.37.4.10 oqObjectInit() ................................................ 584
7.37.4.11 oqResetI() ..................................................... 585
7.37.4.12 oqPutI() ....................................................... 586
7.37.4.13 oqPutTimeout() ............................................... 586
7.37.4.14 oqGetI() ....................................................... 588
7.37.4.15 oqWriteI() .................................................... 588
7.37.4.16 oqWriteTimeout() ............................................. 589

7.38 RTC Driver .............................................................. 591

7.38.1 Detailed Description .............................................. 591

7.38.2 Macro Definition Documentation .................................. 593
    7.38.2.1 RTC_BASE_YEAR ........................................... 593
    7.38.2.2 _rtc_driver_methods ...................................... 593
    7.38.2.3 RTC_SUPPORTS_CALLBACKS ................................ 593
    7.38.2.4 RTC_ALARMS ................................................. 593
    7.38.2.5 RTC_HAS_STORAGE .......................................... 594
    7.38.2.6 PLATFORM_RTC_USE_RTC1 ................................ 594
    7.38.2.7 rtc_lld_driver_fields ..................................... 594

7.38.3 Typedef Documentation ......................................... 594
    7.38.3.1 RTCDriver .................................................. 594
    7.38.3.2 rtcalarm_t ................................................ 594
    7.38.3.3 rtccb_t .................................................... 595

7.38.4 Enumeration Type Documentation ............................... 595
    7.38.4.1 rtcevent_t ................................................. 595

7.38.5 Function Documentation ........................................... 595
    7.38.5.1 rtcInit() ................................................... 595
    7.38.5.2 rtcObjectInit() ........................................... 595
    7.38.5.3 rtcSetTime() .............................................. 596
    7.38.5.4 rtcGetTime() .............................................. 597
    7.38.5.5 rtcSetAlarm() .............................................. 597
    7.38.5.6 rtcGetAlarm() .............................................. 598
    7.38.5.7 rtcSetCallback() .......................................... 599
    7.38.5.8 rtcConvertDateTimeToStructTm() .......................... 599
    7.38.5.9 rtcConvertStructTmToDateTm() ......................... 600
    7.38.5.10 rtcConvertDateTimeToFAT() .............................. 600
    7.38.5.11 rtc_lld_init() ............................................ 601
    7.38.5.12 rtc_lld_set_time() ...................................... 601
    7.38.5.13 rtc_lld_get_time() ...................................... 602
    7.38.5.14 rtc_lld_set_alarm() .................................... 602
    7.38.5.15 rtc_lld_get_alarm() .................................... 603

7.39 SDC Driver ............................................................. 604

7.39.1 Detailed Description ............................................. 604
7.39.2 Driver State Machine .................................................. 604
7.39.3 Driver Operations .................................................... 604
7.39.4 Macro Definition Documentation .................................. 608
  7.39.4.1 SDC_INIT_RETRY ................................................. 608
  7.39.4.2 SDC_MMC_SUPPORT ............................................ 608
  7.39.4.3 SDC_NICE_WAITING ........................................... 608
  7.39.4.4 SDC_INIT_OCR_V20 ............................................ 608
  7.39.4.5 SDC_INIT_OCR ................................................ 609
  7.39.4.6 sdcIsCardInserted ............................................ 609
  7.39.4.7 sdcIsWriteProtected ......................................... 610
  7.39.4.8 PLATFORM_SDC_USE_SDC1 .................................... 611
  7.39.4.9 _sdc_driver_methods .......................................... 611
7.39.5 Typedef Documentation ............................................. 611
  7.39.5.1 sdcmode_t .................................................... 611
  7.39.5.2 sdcflags_t .................................................. 611
  7.39.5.3 SDCDriver .................................................... 611
7.39.6 Enumeration Type Documentation .................................. 612
  7.39.6.1 mmc_switch_t ............................................... 612
  7.39.6.2 sd_switch_t ................................................. 612
  7.39.6.3 sd_switch_function_t ....................................... 612
  7.39.6.4 sdcbusmode_t ................................................ 612
  7.39.6.5 sdcbusclk_t ................................................ 612
7.39.7 Function Documentation ............................................ 612
  7.39.7.1 mode_detect() ............................................... 612
  7.39.7.2 mmc_init() ................................................ 613
  7.39.7.3 sdc_init() ................................................ 614
  7.39.7.4 mmc_cmd6_construct() ..................................... 614
  7.39.7.5 sdc_cmd6_construct() ..................................... 616
  7.39.7.6 sdc_cmd6_extract_info() ................................... 616
  7.39.7.7 sdc_cmd6_check_status() .................................. 617
  7.39.7.8 sdc_detect_bus_clk() ...................................... 617
  7.39.7.9 mmc_detect_bus_clk() ...................................... 618
  7.39.7.10 detect_bus_clk() ......................................... 619
  7.39.7.11 sdc_set_bus_width() ..................................... 619
  7.39.7.12 mmc_set_bus_width() ..................................... 620
  7.39.7.13 _sdc_wait_for_transfer_state() ......................... 620
  7.39.7.14 sdcInit() ................................................ 621
  7.39.7.15 sdcObjectInit() ......................................... 622
  7.39.7.16 sdcStart() ............................................... 622
  7.39.7.17 sdcStop() ................................................ 623
  7.39.7.18 sdcConnect() ............................................. 624
  7.39.7.19 sdcDisconnect() .......................................... 625
7.39.7.20 sdcRead() ................................................. 625
7.39.7.21 sdcWrite() ............................................. 626
7.39.7.22 sdcGetAndClearErrors() ................................. 627
7.39.7.23 sdcSync() .............................................. 627
7.39.7.24 sdcGetInfo() .......................................... 628
7.39.7.25 sdcErase() ............................................. 629
7.39.7.26 sdc_lld_init() ......................................... 629
7.39.7.27 sdc_lld_start() ....................................... 630
7.39.7.28 sdc_lld_stop() ....................................... 630
7.39.7.29 sdc_lld_start_clk() ................................... 631
7.39.7.30 sdc_lld_set_data_clk() ................................. 631
7.39.7.31 sdc_lld_stop_clk() ................................... 631
7.39.7.32 sdc_lld_set_bus_mode() ............................... 632
7.39.7.33 sdc_lld_send_cmd_none() ............................... 632
7.39.7.34 sdc_lld_send_cmd_short() ............................. 632
7.39.7.35 sdc_lld_send_cmd_short_crc() ......................... 633
7.39.7.36 sdc_lld_send_cmd_long_crc() .......................... 634
7.39.7.37 sdc_lld_read() ....................................... 635
7.39.7.38 sdc_lld_write() ...................................... 635
7.39.7.39 sdc_lld_sync() ...................................... 636
7.39.8 Variable Documentation .................................... 636
  7.39.8.1 sdc_vmt .............................................. 636
  7.39.8.2 SDCD1 ................................................. 637

7.40 Serial Driver .................................................. 638
  7.40.1 Detailed Description ..................................... 638
  7.40.2 Driver State Machine .................................... 638
  7.40.3 Macro Definition Documentation ......................... 641
    7.40.3.1 SERIAL_DEFAULT_BITRATE .......................... 641
    7.40.3.2 SERIAL_BUFFERS_SIZE .............................. 641
    7.40.3.3 _serial_driver_methods ............................. 641
    7.40.3.4 sdPutI ............................................. 642
    7.40.3.5 sdPut ............................................. 642
    7.40.3.6 sdPutTimeout ...................................... 642
    7.40.3.7 sdGetI ............................................ 642
    7.40.3.8 sdGet ............................................. 642
    7.40.3.9 sdGetTimeout ..................................... 642
    7.40.3.10 sdWriteI ......................................... 643
    7.40.3.11 sdWrite .......................................... 643
    7.40.3.12 sdWriteTimeout ................................... 643
    7.40.3.13 sdAsynchronousWrite ............................... 644
    7.40.3.14 sdReadl ........................................... 644
    7.40.3.15 sdRead ........................................... 644
7.40.3.16 sdReadTimeout ........................................ 646
7.40.3.17 sdAsynchronousRead ................................ 646
7.40.3.18 PLATFORM_SERIAL_USE_USART1 .................. 646
7.40.3.19 _serial_driver_data .................................. 647
7.40.4 Typedef Documentation .................................. 647
7.40.4.1 SerialDriver ........................................... 647
7.40.4.2 SerialConfig ........................................... 647
7.40.5 Enumeration Type Documentation ....................... 647
7.40.5.1 sdstate_t ............................................... 647
7.40.6 Function Documentation .................................. 648
7.40.6.1 sdInit() ................................................ 648
7.40.6.2 sdObjectInit() ......................................... 648
7.40.6.3 sdStart() .............................................. 649
7.40.6.4 sdStop() .............................................. 650
7.40.6.5 sdIncomingData() ....................................... 650
7.40.6.6 sdRequestData() ......................................... 651
7.40.6.7 sdPutWouldBlock() ..................................... 652
7.40.6.8 sdGetWouldBlock() ..................................... 653
7.40.6.9 sdControl() ............................................ 654
7.40.6.10 sd_lld_init() .......................................... 655
7.40.6.11 sd_lld_start() ......................................... 655
7.40.6.12 sd_lld_stop() .......................................... 655
7.40.7 Variable Documentation .................................. 656
7.40.7.1 SD1 ..................................................... 656
7.40.7.2 default_config ......................................... 656
7.41 Serial over USB Driver ...................................... 657
7.41.1 Detailed Description ..................................... 657
7.41.2 Driver State Machine ..................................... 657
7.41.3 Macro Definition Documentation ....................... 659
7.41.3.1 SERIAL_USB_BUFFERS_SIZE ......................... 659
7.41.3.2 SERIAL_USB_BUFFERS_NUMBER ....................... 659
7.41.3.3 _serial_usb_driver_data ............................... 659
7.41.3.4 _serial_usb_driver_methods ........................... 660
7.41.4 Typedef Documentation .................................. 660
7.41.4.1 SerialUSBDriver ....................................... 660
7.41.5 Enumeration Type Documentation ....................... 660
7.41.5.1 sdustate_t ............................................. 660
7.41.6 Function Documentation .................................. 660
7.41.6.1 ibnotify() ............................................. 660
7.41.6.2 obnotify() ............................................. 661
7.41.6.3 sdulInit() ............................................. 661
7.41.6.4 sdulObjectInit() ..................................... 662
7.41.6.5 sduStart() ....................................................... 662
7.41.6.6 sduStop() ....................................................... 663
7.41.6.7 sduSuspendHook() ........................................... 663
7.41.6.8 sduWakeupHook() ........................................... 664
7.41.6.9 sduConfigureHook() ......................................... 664
7.41.6.10 sduRequestsHook() ......................................... 665
7.41.6.11 sduSOFHook() .............................................. 666
7.41.6.12 sduDataTransmitted() ...................................... 666
7.41.6.13 sduDataReceived() .......................................... 667
7.41.6.14 sduInterruptTransmitted() ................................. 668
7.41.6.15 sduControl() .............................................. 668

7.42 SIO Driver .......................................................... 669
  7.42.1 Detailed Description ........................................... 669
  7.42.2 Driver State Machine ......................................... 669
  7.42.3 Macro Definition Documentation .............................. 674
    7.42.3.1 SIO_DEFAULT_BITRATE ................................ 674
    7.42.3.2 SIO_USE_SYNCHRONIZATION .............................. 674
    7.42.3.3 SIO_USE_STREAMS_INTERFACE .......................... 674
    7.42.3.4 _sio_driver_methods .................................... 675
    7.42.3.5 sioSetCallbackX ......................................... 675
    7.42.3.6 siolsXRXEmptyX .......................................... 675
    7.42.3.7 siolsRXIdleX ........................................... 676
    7.42.3.8 siolsHasRXErrorsX ...................................... 676
    7.42.3.9 siolsTXFullX ............................................ 677
    7.42.3.10 siolsTXOngoingX ....................................... 677
    7.42.3.11 siWriteEnableFlagsX .................................. 678
    7.42.3.12 sioSetEnableFlagsX .................................... 678
    7.42.3.13 sioClearEnableFlagsX .................................. 679
    7.42.3.14 sioGetEnableFlagsX .................................... 679
    7.42.3.15 sioGetAndClearErrorsX ................................ 679
    7.42.3.16 sioGetAndClearEventsX ................................ 680
    7.42.3.17 sioGetEventsX ......................................... 680
    7.42.3.18 sioGetX .................................................. 681
    7.42.3.19 sioPutX .................................................. 681
    7.42.3.20 sioAsyncReadX .......................................... 682
    7.42.3.21 sioAsyncWriteX ......................................... 682
    7.42.3.22 sioControlX ............................................ 683
    7.42.3.23 _sio_callback .......................................... 684
    7.42.3.24 __sio_wakeup_errors .................................. 684
    7.42.3.25 __sio_wakeup_rx ....................................... 684
    7.42.3.26 __sio_wakeup_rxidle ................................... 685
    7.42.3.27 __sio_wakeup_tx ....................................... 685
7.42.3.28 __sio_wakeup_txend .............................. 686
7.42.3.29 __sio_reloc_field ............................. 686
7.42.3.30 PLATFORM_SIO_USE_SIO1 ......................... 687
7.42.3.31 sio_lld_driver_fields ......................... 687
7.42.3.32 sio_lld_config_fields ......................... 687
7.42.3.33 sio_lld_is_rxi_empty ......................... 687
7.42.3.34 sio_lld_is_rxi_idle ......................... 688
7.42.3.35 sio_lld_has_rx_errors ......................... 688
7.42.3.36 sio_lld_is_tx_full ......................... 689
7.42.3.37 sio_lld_is_tx_ongoing ......................... 689
7.42.4 Typedef Documentation ........................................ 690
7.42.4.1 sioevents_t .................................. 690
7.42.4.2 SIODriver ...................................... 690
7.42.4.3 SIOConfig .................................... 690
7.42.4.4 siocb_t ...................................... 690
7.42.5 Enumeration Type Documentation .............................. 691
7.42.5.1 siostate_t .................................. 691
7.42.6 Function Documentation .................................... 691
7.42.6.1 sioInit() .................................. 691
7.42.6.2 sioObjectInit() .................................. 692
7.42.6.3 sioStart() .................................. 692
7.42.6.4 sioStop() .................................. 693
7.42.6.5 sioWriteEnableFlags() ......................... 694
7.42.6.6 sioSetEnableFlags() ............................ 695
7.42.6.7 sioClearEnableFlags() ......................... 695
7.42.6.8 sioGetAndClearErrors() ......................... 696
7.42.6.9 sioGetAndClearEvents() ......................... 697
7.42.6.10 sioGetEvents() .................................. 697
7.42.6.11 sioAsyncRead() .................................. 698
7.42.6.12 sioAsyncWrite() .................................. 699
7.42.6.13 sioSynchronizeRX() ......................... 700
7.42.6.14 sioSynchronizeRXIdle() ....................... 701
7.42.6.15 sioSynchronizeTX() ......................... 702
7.42.6.16 sioSynchronizeTXEnd() ....................... 703
7.42.6.17 sio_lld_init() .................................. 704
7.42.6.18 sio_lld_start() .................................. 704
7.42.6.19 sio_lld_stop() .................................. 705
7.42.6.20 sio_lld_update_enable_flags() .................. 705
7.42.6.21 sio_lld_get_and_clear_errors() .................. 705
7.42.6.22 sio_lld_get_and_clear_events() .................. 706
7.42.6.23 sio_lld_get_events() ......................... 706
7.42.6.24 sio_lld_read() .................................. 707
7.42.6.25 sio_lld_write() .................................................. 707
7.42.6.26 sio_lld_get() .................................................. 708
7.42.6.27 sio_lld_put() .................................................. 708
7.42.6.28 sio_lld_control() .............................................. 709
7.42.6.29 sio_lld_serve_interrupt() .................................. 709
7.42.7 Variable Documentation ........................................... 710
   7.42.7.1 SIOD1 ......................................................... 710

7.43 SPI Driver Selector .................................................. 711
   7.43.1 Detailed Description .......................................... 711

7.44 SPIv1 Driver .......................................................... 712
   7.44.1 Detailed Description .......................................... 712
   7.44.2 Driver State Machine ........................................ 712
   7.44.3 Macro Definition Documentation .............................. 715
      7.44.3.1 SPI_USE_WAIT ........................................... 715
      7.44.3.2 SPI_USE_MUTUAL_EXCLUSION .......................... 715
      7.44.3.3 SPI_SELECT_MODE ...................................... 715
      7.44.3.4 spilsBufferComplete .................................. 716
      7.44.3.5 spiSelectI .............................................. 716
      7.44.3.6 spiUnselectI ........................................... 717
      7.44.3.7 spiStartIgnoreI ....................................... 717
      7.44.3.8 spiStartExchangeI .................................... 718
      7.44.3.9 spiStartSendI ......................................... 719
      7.44.3.10 spiStartReceiveI ..................................... 720
      7.44.3.11 spiPolledExchange ................................... 720
      7.44.3.12 _spi_wakeup_isr ..................................... 721
      7.44.3.13 _spi_isr_code ....................................... 721
      7.44.3.14 _spi_isr_half_code ................................. 722
      7.44.3.15 _spi_isr_full_code ................................ 723
      7.44.3.16 SPI_SUPPORTS_CIRCULAR ............................. 723
      7.44.3.17 PLATFORM_SPI_USE_SPI1 ............................ 724
      7.44.3.18 spi_lld_driver_fields ............................... 724
      7.44.3.19 spi_lld_config_fields ............................... 724

7.44.4 Typedef Documentation ......................................... 724
   7.44.4.1 SPIDriver ................................................. 724
   7.44.4.2 SPIConfig ............................................... 724
   7.44.4.3 spicallback_t ........................................... 724

7.44.5 Enumeration Type Documentation ............................... 725
   7.44.5.1 spistate_t ............................................... 725

7.44.6 Function Documentation ....................................... 725
   7.44.6.1 spi_lld_init() .......................................... 725
   7.44.6.2 spi_lld_start() ......................................... 726
   7.44.6.3 spi_lld_stop() ........................................... 726
7.45.2 spi_lld_start() ........................................ 744
7.45.3 spi_lld_stop() ........................................ 745
7.45.4 spi_lld_select() ..................................... 745
7.45.5 spi_lld_unselect() ................................... 746
7.45.6 spi_lld_ignore() ...................................... 746
7.45.7 spi_lld_exchange() ................................... 747
7.45.8 spi_lld_send() ........................................ 747
7.45.9 spi_lld_receive() ..................................... 749
7.45.10 spi_lld_stop_transfer() ............................. 750
7.45.11 spi_lld_polled_exchange() ......................... 750
7.45.7 Variable Documentation ................................ 750
7.45.7.1 SPID1 .................................................. 751
7.46 ST Driver .................................................. 752
7.46.1 Detailed Description ................................... 752
7.46.2 Function Documentation ............................... 753
7.46.2.1 stInit() ............................................. 753
7.46.2.2 stBind() ............................................ 753
7.46.2.3 stGetCounter() .................................... 754
7.46.2.4 stStartAlarm() .................................... 754
7.46.2.5 stStopAlarm() ..................................... 755
7.46.2.6 stSetAlarm() ....................................... 755
7.46.2.7 stGetAlarm() ....................................... 756
7.46.2.8 stlsAlarmActive() ................................ 757
7.46.2.9 stSetCallback() .................................... 757
7.46.2.10 stBindAlarmN() ................................ 758
7.46.2.11 stStartAlarmN() ................................ 758
7.46.2.12 stStopAlarmN() .................................. 759
7.46.2.13 stSetAlarmN() ................................... 759
7.46.2.14 stGetAlarmN() ................................... 760
7.46.2.15 stlsAlarmActiveN() ................................ 760
7.46.2.16 st_lld_init() ...................................... 761
7.46.2.17 st_lld_get_counter() ............................... 761
7.46.2.18 st_lld_start_alarm() ............................... 761
7.46.2.19 st_lld_stop_alarm() ............................... 762
7.46.2.20 st_lld_set_alarm() ................................. 762
7.46.2.21 st_lld_get_alarm() ................................. 762
7.46.2.22 st_lld_is_alarm_active() ......................... 763
7.46.3 Variable Documentation ................................ 763
7.46.3.1 st_callbacks ...................................... 763
7.47 TRNG Driver ............................................... 764
7.47.1 Detailed Description ................................... 764
7.47.2 Macro Definition Documentation ....................... 765
7.47.2.1 PLATFORM_TRNG_USE_TRNG1 ........................................ 765
7.47.2.2 trng_lld_driver_fields ............................................. 766
7.47.2.3 trng_lld_config_fields ............................................ 766

7.47.3 Typedef Documentation .............................................. 766
   7.47.3.1 TRNGDriver ..................................................... 766
   7.47.3.2 TRNGConfig .................................................... 766

7.47.4 Enumeration Type Documentation ................................. 766
   7.47.4.1 trngstate_t .................................................... 766

7.47.5 Function Documentation .......................................... 767
   7.47.5.1 trngInit() ..................................................... 767
   7.47.5.2 trngObjectInit() .............................................. 767
   7.47.5.3 trngStart() .................................................... 768
   7.47.5.4 trngStop() ..................................................... 769
   7.47.5.5 trngGenerate() .................................................. 770
   7.47.5.6 trng_lld_init() ............................................... 771
   7.47.5.7 trng_lld_start() .............................................. 771
   7.47.5.8 trng_lld_stop() ............................................... 772
   7.47.5.9 trng_lld_generate() ......................................... 772

7.47.6 Variable Documentation ......................................... 773
   7.47.6.1 TRNGD1 .......................................................... 773

7.48 UART Driver ............................................................ 774
   7.48.1 Detailed Description ............................................. 774
   7.48.2 Driver State Machine ............................................ 774
      7.48.2.1 Transmitter sub State Machine .......................... 775
      7.48.2.2 Receiver sub State Machine ............................... 775
   7.48.3 Macro Definition Documentation ............................... 778
      7.48.3.1 UART_NO_ERROR ............................................ 779
      7.48.3.2 UART_PARITY_ERROR ...................................... 779
      7.48.3.3 UART_FRAMING_ERROR .................................... 779
      7.48.3.4 UART_OVERRUN_ERROR ................................... 779
      7.48.3.5 UART_NOISE_ERROR ....................................... 779
      7.48.3.6 UART_BREAK_DETECTED ................................... 779
      7.48.3.7 UART_USE_WAIT ........................................... 780
      7.48.3.8 UART_USE_MUTUAL_EXCLUSION ............................ 780
      7.48.3.9 _uart_wakeup_tx1_isr ................................... 780
      7.48.3.10 _uart_wakeup_tx2_isr .................................. 781
      7.48.3.11 _uart_wakeup_rx_complete_isr ........................ 781
      7.48.3.12 _uart_wakeup_rx_error_isr ............................... 781
      7.48.3.13 _uart_wakeup_rx_cm_isr .................................. 782
      7.48.3.14 _uart_wakeup_rx_timeout_isr ............................ 782
      7.48.3.15 _uart_tx1_isr_code ...................................... 783
      7.48.3.16 _uart_tx2_isr_code ..................................... 783
7.48.3.17 _uart_rx_complete_isr_code ................................................. 784
7.48.3.18 _uart_rx_error_isr_code .................................................... 785
7.48.3.19 _uart_rx_idle_code .......................................................... 785
7.48.3.20 _uart_timeout_isr_code ...................................................... 786
7.48.3.21 _uart_rx_char_match_isr_code ........................................... 787
7.48.3.22 PLATFORM_UART_USE_UART1 ...... .................................... 787
7.48.4 Typedef Documentation ......................................................... 788
  7.48.4.1 uartflags_t ................................................................. 788
  7.48.4.2 UARTDriver ................................................................. 788
  7.48.4.3 uartcb_t ................................................................. 788
  7.48.4.4 uartccb_t ............................................................. 788
  7.48.4.5 uartecb_t ............................................................. 789
  7.48.4.6 UARTConfig .............................................................. 789
7.48.5 Enumeration Type Documentation ........................................ 789
  7.48.5.1 uartstate_t ............................................................. 789
  7.48.5.2 uarttxstate_t .......................................................... 790
  7.48.5.3 uartrxstate_t .......................................................... 790
7.48.6 Function Documentation .................................................... 790
  7.48.6.1 uartInit() ............................................................... 790
  7.48.6.2 uartObjectInit() ....................................................... 791
  7.48.6.3 uartStart() ............................................................. 791
  7.48.6.4 uartStop() ............................................................... 792
  7.48.6.5 uartStartSend() ....................................................... 793
  7.48.6.6 uartStartSendI() ...................................................... 794
  7.48.6.7 uartStopSend() .......................................................... 795
  7.48.6.8 uartStopSendI() ........................................................ 796
  7.48.6.9 uartStartReceive() ..................................................... 797
  7.48.6.10 uartStartReceiveI() .................................................. 798
  7.48.6.11 uartStopReceive() ..................................................... 799
  7.48.6.12 uartStopReceiveI() .................................................... 800
  7.48.6.13 uartSendTimeout() ..................................................... 801
  7.48.6.14 uartSendFullTimeout() ................................................. 802
  7.48.6.15 uartReceiveTimeout() .................................................. 803
  7.48.6.16 uartAcquireBus() ....................................................... 804
  7.48.6.17 uartReleaseBus() ....................................................... 805
  7.48.6.18 uart_lld_init() ......................................................... 806
  7.48.6.19 uart_lld_start() ......................................................... 806
  7.48.6.20 uart_lld_stop() .......................................................... 806
  7.48.6.21 uart_lld_start_send() .................................................. 807
  7.48.6.22 uart_lld_stop_send() ................................................... 807
  7.48.6.23 uart_lld_start_receive() ............................................. 808
  7.48.6.24 uart_lld_stop_receive() ............................................. 808
7.48.7 Variable Documentation .................................................. 809
  7.48.7.1 UARTD1 .............................................................. 809
7.49 USB Driver ................................................................. 810
  7.49.1 Detailed Description ................................................ 810
  7.49.2 Driver State Machine .............................................. 810
  7.49.3 USB Operations ...................................................... 810
    7.49.3.1 USB Implementation ....................................... 811
    7.49.3.2 USB Endpoints ............................................. 811
    7.49.3.3 USB Callbacks ............................................. 812
  7.49.4 Macro Definition Documentation .................................. 817
    7.49.4.1 USB_DESC_INDEX ........................................... 817
    7.49.4.2 USB_DESC_BYTE ........................................... 817
    7.49.4.3 USB_DESC_WORD .......................................... 817
    7.49.4.4 USB_DESC_BCD ............................................ 818
    7.49.4.5 USB_DESC_DEVICE ........................................ 818
    7.49.4.6 USB_DESC_CONFIGURATION_SIZE ........................... 818
    7.49.4.7 USB_DESC_CONFIGURATION ................................ 818
    7.49.4.8 USB_DESC_INTERFACE_SIZE ................................ 819
    7.49.4.9 USB_DESC_INTERFACE ...................................... 819
    7.49.4.10 USB_DESC_INTERFACE_ASSOCIATION_SIZE ............... 820
    7.49.4.11 USB_DESC_INTERFACE_ASSOCIATION ...................... 820
    7.49.4.12 USB_DESC_ENDPOINT_SIZE ................................ 820
    7.49.4.13 USB_DESC_ENDPOINT ...................................... 820
    7.49.4.14 USB_EP_MODE_TYPE ........................................ 821
    7.49.4.15 USB_EP_MODE_TYPE_CTRL .................................. 821
    7.49.4.16 USB_EP_MODE_TYPE_ISO ..................................... 821
    7.49.4.17 USB_EP_MODE_TYPE_BULK ................................ 821
    7.49.4.18 USB_EP_MODE_TYPE_INTR ................................ 821
    7.49.4.19 USB_USE_WAIT ............................................ 821
  7.49.4.20 usbGetDriverStateI ......................................... 821
  7.49.4.21 usbConnectBus ................................................ 821
  7.49.4.22 usbDisconnectBus ........................................... 821
  7.49.4.23 usbGetFrameNumberX ....................................... 822
  7.49.4.24 usbGetTransmitStatusI .................................... 822
  7.49.4.25 usbGetReceiveStatusI ..................................... 823
  7.49.4.26 usbGetReceiveTransactionSizeX ........................... 823
  7.49.4.27 usbSetupTransfer .......................................... 824
  7.49.4.28 usbReadSetup .............................................. 824
  7.49.4.29 _usb_isr_invoke_event_cb ................................ 825
  7.49.4.30 _usb_isr_invoke_sof_cb .................................. 825
  7.49.4.31 _usb_isr_invoke_setup_cb ................................ 825
  7.49.4.32 _usb_isr_invoke_in_cb ................................... 826
7.51.5.3 wdgStop() ........................................ 864
7.51.5.4 wdgReset() ....................................... 865
7.51.5.5 wdg_lld_init() .................................... 866
7.51.5.6 wdg_lld_start() .................................... 866
7.51.5.7 wdg_lld_stop() ..................................... 866
7.51.5.8 wdg_lld_reset() .................................... 867

7.52 WSPI Driver ........................................... 868

7.52.1 Detailed Description ................................ 868

7.52.2 Macro Definition Documentation .................... 871

7.52.2.1 WSPI_USE_WAIT ................................... 872
7.52.2.2 WSPI_USE_MUTUAL_EXCLUSION ..................... 872
7.52.2.3 wspiStartCommandI ................................ 872
7.52.2.4 wspiStartSendI ..................................... 873
7.52.2.5 wspiStartReceiveI ................................... 873
7.52.2.6 wspiMapFlashI ...................................... 874
7.52.2.7 wspiUnmapFlashI ................................... 875
7.52.2.8 _wspi_wakeup_isr .................................. 875
7.52.2.9 _wspi_isr_code .................................... 876
7.52.2.10 _wspi_error_code ................................ 876
7.52.2.11 PLATFORM_WSPI_USE_WSPI1 ...................... 877
7.52.2.12 wspi_lld_driver_fields ......................... 877
7.52.2.13 wspi_lld_config_fields ......................... 878

7.52.3 Typedef Documentation .................................. 878

7.52.3.1 WSPIDriver ........................................ 878
7.52.3.2 WSPIConfig ........................................ 878
7.52.3.3 wspicallback_t .................................... 878

7.52.4 Enumeration Type Documentation ...................... 878

7.52.4.1 wspistate_t ........................................ 878

7.52.5 Function Documentation .................................. 879

7.52.5.1 wspilInit() ........................................ 879
7.52.5.2 wspilObjectInit() ................................... 880
7.52.5.3 wspilStart() ....................................... 881
7.52.5.4 wspilStop() ........................................ 882
7.52.5.5 wspilStartCommand() .............................. 883
7.52.5.6 wspilStartSend() .................................... 884
7.52.5.7 wspilStartReceive() ............................... 885
7.52.5.8 wspilCommand() .................................... 886
7.52.5.9 wspilSend() ......................................... 887
7.52.5.10 wspilReceive() ..................................... 887
7.52.5.11 wspilMapFlash() ................................... 888
7.52.5.12 wspilUnmapFlash() ............................... 889
7.52.5.13 wspilAcquireBus() ................................ 889
7.54.2.21 HAL_USE_UART .................................................. 904
7.54.2.22 HAL_USE_USB ................................................. 904
7.54.2.23 HAL_USE_WDG .................................................. 904
7.54.2.24 HAL_USE_WSPI ............................................... 904
7.54.2.25 PAL_USE_CALLBACKS ......................................... 904
7.54.2.26 PAL_USE_WAIT ............................................... 905
7.54.2.27 ADC_USE_WAIT ............................................... 905
7.54.2.28 ADC_USE_MUTUAL EXCLUSION ............................. 905
7.54.2.29 CAN_USE_SLEEP_MODE ...................................... 905
7.54.2.30 CAN_ENFORCE_USE_CALLBACKS ............................ 905
7.54.2.31 HAL_CRY_USE_FALLBACK .................................. 906
7.54.2.32 HAL_CRY_ENFORCE_FALLBACK ............................. 906
7.54.2.33 DAC_USE_WAIT ............................................... 906
7.54.2.34 DAC_USE_MUTUAL EXCLUSION ............................. 906
7.54.2.35 I2C_USE_MUTUAL EXCLUSION .............................. 907
7.54.2.36 MAC_USE_ZERO_COPY ....................................... 907
7.54.2.37 MAC_USE_EVENTS ............................................ 907
7.54.2.38 MMC_IDLE_TIMEOUT_MS ................................... 907
7.54.2.39 MMC_USE_MUTUAL EXCLUSION ............................ 907
7.54.2.40 SDC_INIT_RETRY ............................................. 908
7.54.2.41 SDC_MMC_SUPPORT .......................................... 908
7.54.2.42 SDC_NICE_WAITING ......................................... 908
7.54.2.43 SDC_INIT_OCR_V20 ......................................... 908
7.54.2.44 SDC_INIT_OCR ............................................... 908
7.54.2.45 SERIAL_DEFAULT_BITRATE ................................ 909
7.54.2.46 SERIAL_BUFFERS_SIZE ................................... 909
7.54.2.47 SIO_DEFAULT_BITRATE ..................................... 909
7.54.2.48 SIO_USE_SYNCHRONIZATION ................................ 909
7.54.2.49 SERIAL_USB_BUFFERS_SIZE ............................... 909
7.54.2.50 SERIAL_USB_BUFFERS_NUMBER ............................. 910
7.54.2.51 SPI_USE_WAIT ................................................. 910
7.54.2.52 SPI_USE_ASSERT_ON_ERROR ................................. 910
7.54.2.53 SPI_USE_MUTUAL EXCLUSION ............................. 910
7.54.2.54 SPI_SELECT_MODE ............................................ 911
7.54.2.55 UART_USE_WAIT .............................................. 911
7.54.2.56 UART_USE_MUTUAL EXCLUSION ........................... 911
7.54.2.57 USB_USE_WAIT ................................................. 911
7.54.2.58 WSPI_USE_WAIT .............................................. 912
7.54.2.59 WSPI_USE_MUTUAL EXCLUSION ............................ 912

7.55 Normal Drivers .................................................... 913
  7.55.1 Detailed Description ........................................ 913

7.56 Complex Drivers .................................................. 915
7.56.1 Detailed Description .............................................. 915
7.57 Interfaces and Classes .............................................. 916
  7.57.1 Detailed Description ........................................... 916
7.58 Inner Code .......................................................... 917
  7.58.1 Detailed Description ........................................... 917
7.59 Support Code ........................................................ 918
  7.59.1 Detailed Description ........................................... 918
7.60 OSAL ................................................................. 919
  7.60.1 Detailed Description ........................................... 919
  7.60.2 Macro Definition Documentation ................................ 924
    7.60.2.1 OSAL_ST_RESOLUTION .................................... 924
    7.60.2.2 OSAL_ST_FREQUENCY ..................................... 924
    7.60.2.3 OSAL_ST_MODE ............................................ 925
    7.60.2.4 OSAL_IRQ_PRIORITY_LEVELS ............................... 925
    7.60.2.5 OSAL_IRQ_MAXIMUM_PRIORITY ......................... 925
    7.60.2.6 OSAL_DBG_ENABLE_ASSERTS ................................ 925
    7.60.2.7 OSAL_DBG_ENABLE_CHECKS ................................ 925
    7.60.2.8 osalDbgAssert ............................................ 925
    7.60.2.9 osalDbgCheck ............................................. 926
    7.60.2.10 osalDbgCheckClassI ..................................... 926
    7.60.2.11 osalDbgCheckClassS ..................................... 927
    7.60.2.12 OSAL_IRQ_IS_VALID_PRIORITY ........................... 927
    7.60.2.13 OSAL_IRQ_PROLOGUE .................................... 927
    7.60.2.14 OSAL_IRQ_EPILOGUE ..................................... 927
    7.60.2.15 OSAL_IRQ_HANDLER ....................................... 927
    7.60.2.16 OSAL_S2I ................................................ 928
    7.60.2.17 OSAL_MS2I ................................................ 928
    7.60.2.18 OSAL_US2I ................................................ 929
    7.60.2.19 OSAL_I2S ................................................ 930
    7.60.2.20 OSAL_I2MS ............................................... 930
    7.60.2.21 OSAL_I2US ............................................... 931
    7.60.2.22 OSAL_S2RTC .............................................. 932
    7.60.2.23 OSAL_MS2RTC ............................................. 932
    7.60.2.24 OSAL_US2RTC ............................................. 933
    7.60.2.25 osalThreadSleepSeconds ............................... 933
    7.60.2.26 osalThreadSleepMilliseconds .......................... 934
    7.60.2.27 osalThreadSleepMicroseconds ....................... 934
  7.60.3 Typedef Documentation ....................................... 935
    7.60.3.1 syssts_t ................................................ 935
    7.60.3.2 msg_t ................................................... 935
    7.60.3.3 systime_t ............................................... 935
    7.60.3.4 sysinterval_t ........................................... 935
7.60.3.5 time_conv_t .................................................. 935
7.60.3.6 rtcnt_t ......................................................... 936
7.60.3.7 thread_reference_t ........................................... 936
7.60.3.8 eventflags_t .................................................. 936
7.60.3.9 event_source_t ................................................ 936
7.60.3.10 eventcallback_t ............................................. 936
7.60.3.11 mutex_t ....................................................... 937

7.60.4 Function Documentation ........................................ 937
7.60.4.1 osalInit() ..................................................... 937
7.60.4.2 osalSysHalt() .................................................. 937
7.60.4.3 osalSysPolledDelayX() ....................................... 938
7.60.4.4 osalOsTimerHandlerI() ....................................... 938
7.60.4.5 osalOsRescheduleS() ......................................... 939
7.60.4.6 osalOsGetSystemTimeX() .................................... 939
7.60.4.7 osalThreadSleepS() ........................................... 939
7.60.4.8 osalThreadSleep() ............................................. 940
7.60.4.9 osalThreadSuspendS() .......................................... 940
7.60.4.10 osalThreadSuspendTimeoutS() ................................. 941
7.60.4.11 osalThreadResumeI() ......................................... 941
7.60.4.12 osalThreadResumeS() ......................................... 942
7.60.4.13 osalThreadEnqueueTimeoutS() ................................. 942
7.60.4.14 osalThreadDequeueNextI() .................................. 943
7.60.4.15 osalThreadDequeueAllI() .................................... 943
7.60.4.16 osalEventBroadcastFlagsI() ................................ 945
7.60.4.17 osalEventBroadcastFlags() .................................. 945
7.60.4.18 osalEventSetCallback() ...................................... 946
7.60.4.19 osalMutexLock() ............................................... 947
7.60.4.20 osalMutexUnlock() ............................................ 947
7.60.4.21 osalSysDisable() ............................................. 947
7.60.4.22 osalSysEnable() ............................................... 948
7.60.4.23 osalSysLock() .................................................. 948
7.60.4.24 osalSysUnlock() ............................................... 948
7.60.4.25 osalSysLockFromISR() ....................................... 949
7.60.4.26 osalSysUnlockFromISR() .................................... 949
7.60.4.27 osalSysGetStatusAndLockX() ................................ 949
7.60.4.28 osalSysRestoreStatusX() .................................... 950
7.60.4.29 osalTimeAddX() ............................................... 950
7.60.4.30 osalTimeDiffX() .............................................. 950
7.60.4.31 osalTimeIsInRangeX() ....................................... 951
7.60.4.32 osalThreadQueueObjectInit() ................................. 952
7.60.4.33 osalEventObjectInit() ....................................... 952
7.60.4.34 osalMutexObjectInit() ...................................... 952
8 Data Structure Documentation

8.1 BaseAsynchronousChannel Struct Reference

8.1.1 Detailed Description

8.1.2 Field Documentation

8.1.2.1 vmt

8.2 BaseAsynchronousChannelVMT Struct Reference

8.2.1 Detailed Description

8.3 BaseBlockDevice Struct Reference

8.3.1 Detailed Description

8.3.2 Field Documentation

8.3.2.1 vmt

8.4 BaseBlockDeviceVMT Struct Reference

8.4.1 Detailed Description

8.5 BaseBufferedStream Struct Reference

8.5.1 Detailed Description

8.5.2 Field Documentation

8.5.2.1 vmt

8.6 BaseBufferedStreamVMT Struct Reference

8.6.1 Detailed Description

8.7 BaseChannel Struct Reference

8.7.1 Detailed Description

8.7.2 Field Documentation

8.7.2.1 vmt

8.8 BaseChannelVMT Struct Reference

8.8.1 Detailed Description

8.9 BaseFlash Struct Reference

8.9.1 Detailed Description

8.9.2 Field Documentation

8.9.2.1 vmt

8.10 BaseFlashVMT Struct Reference

8.10.1 Detailed Description

8.11 BaseObject Struct Reference

8.11.1 Detailed Description

8.11.2 Field Documentation

8.11.2.1 vmt

8.12 BaseObjectVMT Struct Reference

8.12.1 Detailed Description

8.13 BasePersistentStorage Struct Reference

8.13.1 Detailed Description
8.13.1 Detailed Description .............................................. 975
8.13.2 Field Documentation ............................................. 975
8.13.2.1 vmt ................................................................. 975

8.14 BasePersistentStorageVMT Struct Reference .................. 975
8.14.1 Detailed Description .............................................. 975

8.15 BaseSequentialStream Struct Reference ......................... 976
8.15.1 Detailed Description .............................................. 977
8.15.2 Field Documentation ............................................. 977
8.15.2.1 vmt ................................................................. 977

8.16 BaseSequentialStreamVMT Struct Reference ..................... 978
8.16.1 Detailed Description .............................................. 978

8.17 BlockDeviceInfo Struct Reference ................................. 979
8.17.1 Detailed Description .............................................. 979
8.17.2 Field Documentation ............................................. 979
8.17.2.1 blk_size ......................................................... 979
8.17.2.2 blk_num ........................................................ 980

8.18 BufferedSerialVMT Struct Reference .............................. 980
8.18.1 Detailed Description .............................................. 981

8.19 BufferedStreamAdapter Struct Reference ......................... 981
8.19.1 Detailed Description .............................................. 982
8.19.2 Field Documentation ............................................. 982
8.19.2.1 vmt ................................................................. 982

8.20 BufferedStreamAdapterVMT Struct Reference ................... 983
8.20.1 Detailed Description .............................................. 983

8.21 CANRxFrame Struct Reference .................................... 983
8.21.1Detailed Description ............................................. 984
8.21.2 Field Documentation ............................................. 984
8.21.2.1 FMI ................................................................. 984
8.21.2.2 TIME ............................................................ 985
8.21.2.3 DLC ............................................................... 985
8.21.2.4 RTR ................................................................. 985
8.21.2.5 IDE ................................................................. 985
8.21.2.6 SID ................................................................. 985
8.21.2.7 EID ................................................................. 985
8.21.2.8 data8 ............................................................. 986
8.21.2.9 data16 ........................................................... 986
8.21.2.10 data32 ........................................................ 986

8.22 CANTxFrame Struct Reference .................................... 986
8.22.1 Detailed Description ............................................. 987
8.22.2 Field Documentation ............................................. 987
8.22.2.1 DLC ................................................................. 987
8.22.2.2 RTR ................................................................. 988
8.68.2.4 period_cb .................................................. 1056
8.68.2.5 overflow_cb .................................................. 1056
8.69 ICUDriver Struct Reference ........................................ 1057
  8.69.1 Detailed Description ......................................... 1057
  8.69.2 Field Documentation ......................................... 1057
    8.69.2.1 state .................................................. 1057
    8.69.2.2 config .................................................. 1058
8.70 io_buffers_queue Struct Reference ................................ 1058
  8.70.1 Detailed Description ......................................... 1059
  8.70.2 Field Documentation ......................................... 1059
    8.70.2.1 waiting .................................................. 1059
    8.70.2.2 suspended .............................................. 1059
    8.70.2.3 bcounter ................................................. 1059
    8.70.2.4 bwrptr .................................................. 1060
    8.70.2.5 brdptr .................................................. 1060
    8.70.2.6 btop ...................................................... 1060
    8.70.2.7 bsize .................................................... 1060
    8.70.2.8 bn ....................................................... 1060
    8.70.2.9 buffers .................................................. 1061
    8.70.2.10 ptr ...................................................... 1061
    8.70.2.11 top ...................................................... 1061
    8.70.2.12 notify .................................................. 1061
    8.70.2.13 link .................................................... 1061
8.71 io_queue Struct Reference ......................................... 1062
  8.71.1 Detailed Description ......................................... 1063
  8.71.2 Field Documentation ......................................... 1063
    8.71.2.1 q_waiting .................................................. 1063
    8.71.2.2 q_counter ................................................. 1063
    8.71.2.3 q_buffer .................................................. 1063
    8.71.2.4 q_top .................................................... 1063
    8.71.2.5 q_wrptr .................................................. 1064
    8.71.2.6 q_rdptr .................................................. 1064
    8.71.2.7 q_notify .................................................. 1064
    8.71.2.8 q_link .................................................... 1064
8.72 IOBus Struct Reference ........................................... 1064
  8.72.1 Detailed Description ......................................... 1065
  8.72.2 Field Documentation ......................................... 1065
    8.72.2.1 portid ................................................... 1065
    8.72.2.2 mask ..................................................... 1065
    8.72.2.3 offset ................................................... 1065
8.73 MemoryStream Struct Reference .................................. 1066
  8.73.1 Detailed Description ......................................... 1067
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.97.1 Detailed Description</td>
<td>1107</td>
</tr>
<tr>
<td>8.97.2 Field Documentation</td>
<td>1107</td>
</tr>
<tr>
<td>8.97.2.1 bus_width</td>
<td>1107</td>
</tr>
<tr>
<td>8.98 SDCDriver Struct Reference</td>
<td>1107</td>
</tr>
<tr>
<td>8.98.1 Detailed Description</td>
<td>1108</td>
</tr>
<tr>
<td>8.98.2 Field Documentation</td>
<td>1109</td>
</tr>
<tr>
<td>8.98.2.1 vmt</td>
<td>1109</td>
</tr>
<tr>
<td>8.98.2.2 config</td>
<td>1109</td>
</tr>
<tr>
<td>8.98.2.3 cardmode</td>
<td>1109</td>
</tr>
<tr>
<td>8.98.2.4 errors</td>
<td>1109</td>
</tr>
<tr>
<td>8.98.2.5 rca</td>
<td>1109</td>
</tr>
<tr>
<td>8.98.2.6 buf</td>
<td>1109</td>
</tr>
<tr>
<td>8.99 SDCDriverVMT Struct Reference</td>
<td>1110</td>
</tr>
<tr>
<td>8.99.1 Detailed Description</td>
<td>1111</td>
</tr>
<tr>
<td>8.100 SerialDriverVMT Struct Reference</td>
<td>1111</td>
</tr>
<tr>
<td>8.100.1 Detailed Description</td>
<td>1113</td>
</tr>
<tr>
<td>8.101 SerialUSBCConfig Struct Reference</td>
<td>1113</td>
</tr>
<tr>
<td>8.101.1 Detailed Description</td>
<td>1114</td>
</tr>
<tr>
<td>8.101.2 Field Documentation</td>
<td>1114</td>
</tr>
<tr>
<td>8.101.2.1 usbp</td>
<td>1114</td>
</tr>
<tr>
<td>8.101.2.2 bulk_in</td>
<td>1114</td>
</tr>
<tr>
<td>8.101.2.3 bulk_out</td>
<td>1114</td>
</tr>
<tr>
<td>8.101.2.4 int_in</td>
<td>1114</td>
</tr>
<tr>
<td>8.102 SerialUSBDriver Struct Reference</td>
<td>1115</td>
</tr>
<tr>
<td>8.102.1 Detailed Description</td>
<td>1116</td>
</tr>
<tr>
<td>8.102.2 Field Documentation</td>
<td>1116</td>
</tr>
<tr>
<td>8.102.2.1 vmt</td>
<td>1116</td>
</tr>
<tr>
<td>8.103 SerialUSBDriverVMT Struct Reference</td>
<td>1117</td>
</tr>
<tr>
<td>8.103.1 Detailed Description</td>
<td>1118</td>
</tr>
<tr>
<td>8.104 SHA1Context Struct Reference</td>
<td>1118</td>
</tr>
<tr>
<td>8.104.1 Detailed Description</td>
<td>1119</td>
</tr>
<tr>
<td>8.105 SHA256Context Struct Reference</td>
<td>1119</td>
</tr>
<tr>
<td>8.105.1 Detailed Description</td>
<td>1119</td>
</tr>
<tr>
<td>8.106 SHA512Context Struct Reference</td>
<td>1119</td>
</tr>
<tr>
<td>8.106.1 Detailed Description</td>
<td>1120</td>
</tr>
<tr>
<td>8.107 sio_driver_vmt Struct Reference</td>
<td>1120</td>
</tr>
<tr>
<td>8.107.1 Detailed Description</td>
<td>1121</td>
</tr>
<tr>
<td>8.108 SNORConfig Struct Reference</td>
<td>1121</td>
</tr>
<tr>
<td>8.108.1 Detailed Description</td>
<td>1122</td>
</tr>
<tr>
<td>8.109 SNORDriver Struct Reference</td>
<td>1122</td>
</tr>
<tr>
<td>8.109.1 Detailed Description</td>
<td>1123</td>
</tr>
<tr>
<td>8.109.2 Field Documentation</td>
<td>1123</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>8.109.2.1</td>
<td>vmt</td>
</tr>
<tr>
<td>8.109.2.2</td>
<td>config</td>
</tr>
<tr>
<td>8.109.2.3</td>
<td>nocache</td>
</tr>
<tr>
<td>8.109.2.4</td>
<td>mutex</td>
</tr>
<tr>
<td>8.110</td>
<td>SNORDriverVMT Struct Reference</td>
</tr>
<tr>
<td>8.110.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.111</td>
<td>threads_queue_t Struct Reference</td>
</tr>
<tr>
<td>8.111.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.112</td>
<td>unpacked_mmc_cid_t Struct Reference</td>
</tr>
<tr>
<td>8.112.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.113</td>
<td>unpacked_mmc_csd_t Struct Reference</td>
</tr>
<tr>
<td>8.113.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.114</td>
<td>unpacked_sdc_cid_t Struct Reference</td>
</tr>
<tr>
<td>8.114.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.115</td>
<td>unpacked_sdc_csd_10_t Struct Reference</td>
</tr>
<tr>
<td>8.115.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.116</td>
<td>unpacked_sdc_csd_20_t Struct Reference</td>
</tr>
<tr>
<td>8.116.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.117</td>
<td>USBConfig Struct Reference</td>
</tr>
<tr>
<td>8.117.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.117.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.117.2.1</td>
<td>event_cb</td>
</tr>
<tr>
<td>8.117.2.2</td>
<td>get_descriptor_cb</td>
</tr>
<tr>
<td>8.117.2.3</td>
<td>requests_hook_cb</td>
</tr>
<tr>
<td>8.117.2.4</td>
<td>sof_cb</td>
</tr>
<tr>
<td>8.118</td>
<td>USBDescriptor Struct Reference</td>
</tr>
<tr>
<td>8.118.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.118.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.118.2.1</td>
<td>ud_size</td>
</tr>
<tr>
<td>8.118.2.2</td>
<td>ud_string</td>
</tr>
<tr>
<td>8.119</td>
<td>USBDriver Struct Reference</td>
</tr>
<tr>
<td>8.119.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.119.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.119.2.1</td>
<td>state</td>
</tr>
<tr>
<td>8.119.2.2</td>
<td>config</td>
</tr>
<tr>
<td>8.119.2.3</td>
<td>transmitting</td>
</tr>
<tr>
<td>8.119.2.4</td>
<td>receiving</td>
</tr>
<tr>
<td>8.119.2.5</td>
<td>epc</td>
</tr>
<tr>
<td>8.119.2.6</td>
<td>in_params</td>
</tr>
<tr>
<td>8.119.2.7</td>
<td>out_params</td>
</tr>
<tr>
<td>8.119.2.8</td>
<td>ep0state</td>
</tr>
<tr>
<td>8.119.2.9</td>
<td>ep0next</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>8.119.2.10</td>
<td>ep0n</td>
</tr>
<tr>
<td>8.119.2.11</td>
<td>ep0endcb</td>
</tr>
<tr>
<td>8.119.2.12</td>
<td>setup</td>
</tr>
<tr>
<td>8.119.2.13</td>
<td>status</td>
</tr>
<tr>
<td>8.119.2.14</td>
<td>address</td>
</tr>
<tr>
<td>8.119.2.15</td>
<td>configuration</td>
</tr>
<tr>
<td>8.119.2.16</td>
<td>saved_state</td>
</tr>
<tr>
<td>8.120</td>
<td>USBEndpointConfig Struct Reference</td>
</tr>
<tr>
<td>8.120.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.120.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.120.2.1</td>
<td>ep_mode</td>
</tr>
<tr>
<td>8.120.2.2</td>
<td>setup_cb</td>
</tr>
<tr>
<td>8.120.2.3</td>
<td>in_cb</td>
</tr>
<tr>
<td>8.120.2.4</td>
<td>out_cb</td>
</tr>
<tr>
<td>8.120.2.5</td>
<td>in_maxsize</td>
</tr>
<tr>
<td>8.120.2.6</td>
<td>out_maxsize</td>
</tr>
<tr>
<td>8.120.2.7</td>
<td>in_state</td>
</tr>
<tr>
<td>8.120.2.8</td>
<td>out_state</td>
</tr>
<tr>
<td>8.121</td>
<td>USBInEndpointState Struct Reference</td>
</tr>
<tr>
<td>8.121.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.121.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.121.2.1</td>
<td>txsize</td>
</tr>
<tr>
<td>8.121.2.2</td>
<td>txcnt</td>
</tr>
<tr>
<td>8.121.2.3</td>
<td>txbuf</td>
</tr>
<tr>
<td>8.121.2.4</td>
<td>thread</td>
</tr>
<tr>
<td>8.122</td>
<td>USBOutEndpointState Struct Reference</td>
</tr>
<tr>
<td>8.122.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.122.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.122.2.1</td>
<td>rxsize</td>
</tr>
<tr>
<td>8.122.2.2</td>
<td>rxcnt</td>
</tr>
<tr>
<td>8.122.2.3</td>
<td>rxbuf</td>
</tr>
<tr>
<td>8.122.2.4</td>
<td>thread</td>
</tr>
<tr>
<td>8.123</td>
<td>WDGConfig Struct Reference</td>
</tr>
<tr>
<td>8.123.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.124</td>
<td>WDGDriver Struct Reference</td>
</tr>
<tr>
<td>8.124.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.124.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.124.2.1</td>
<td>state</td>
</tr>
<tr>
<td>8.124.2.2</td>
<td>config</td>
</tr>
<tr>
<td>8.125</td>
<td>wspi_command_t Struct Reference</td>
</tr>
<tr>
<td>8.125.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.125.2</td>
<td>Field Documentation</td>
</tr>
</tbody>
</table>
9 File Documentation

9.1 bufstreams.c File Reference .................................................. 1147
  9.1.1 Detailed Description ................................................... 1147
9.2 bufstreams.h File Reference .................................................. 1147
  9.2.1 Detailed Description ................................................... 1148
9.3 chprintf.c File Reference ..................................................... 1148
  9.3.1 Detailed Description ................................................... 1148
9.4 chprintf.h File Reference ..................................................... 1148
  9.4.1 Detailed Description ................................................... 1149
9.5 chscanf.c File Reference ...................................................... 1149
  9.5.1 Detailed Description ................................................... 1149
9.6 chscanf.h File Reference ...................................................... 1150
  9.6.1 Detailed Description ................................................... 1150
9.7 hal.c File Reference ........................................................ 1150
  9.7.1 Detailed Description ................................................... 1150
9.8 hal.h File Reference ........................................................ 1151
  9.8.1 Detailed Description ................................................... 1151
9.9 hal_adc.c File Reference ..................................................... 1152
  9.9.1 Detailed Description ................................................... 1152
9.10 hal_adc.h File Reference .................................................... 1153
  9.10.1 Detailed Description ................................................... 1155
9.11 hal_adc_lld.c File Reference ............................................... 1155
  9.11.1 Detailed Description ................................................... 1156
9.12 hal_adc_lld.h File Reference ............................................... 1156
  9.12.1 Detailed Description ................................................... 1157
9.13 hal_buffered_serial.h File Reference ................................... 1157
  9.13.1 Detailed Description ................................................... 1158
9.14 hal_buffers.c File Reference ............................................... 1158
  9.14.1 Detailed Description ................................................... 1160
9.15 hal_buffers.h File Reference ............................................... 1160
  9.15.1 Detailed Description ................................................... 1162
9.16 hal_can.c File Reference .................................................... 1162
  9.16.1 Detailed Description ................................................... 1163
9.17 hal_can.h File Reference .................................................... 1163
  9.17.1 Detailed Description ................................................... 1164
9.18 hal_can_lld.c File Reference ............................................... 1164
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.18.1 Detailed Description</td>
<td>1165</td>
</tr>
<tr>
<td>9.19 hal_can_lld.h File Reference</td>
<td>1165</td>
</tr>
<tr>
<td>9.19.1 Detailed Description</td>
<td>1166</td>
</tr>
<tr>
<td>9.20 hal_channels.h File Reference</td>
<td>1167</td>
</tr>
<tr>
<td>9.20.1 Detailed Description</td>
<td>1168</td>
</tr>
<tr>
<td>9.21 hal_crypto.c File Reference</td>
<td>1168</td>
</tr>
<tr>
<td>9.21.1 Detailed Description</td>
<td>1170</td>
</tr>
<tr>
<td>9.22 hal_crypto.h File Reference</td>
<td>1171</td>
</tr>
<tr>
<td>9.22.1 Detailed Description</td>
<td>1173</td>
</tr>
<tr>
<td>9.23 hal_crypto_lld.c File Reference</td>
<td>1174</td>
</tr>
<tr>
<td>9.23.1 Detailed Description</td>
<td>1176</td>
</tr>
<tr>
<td>9.24 hal_crypto_lld.h File Reference</td>
<td>1176</td>
</tr>
<tr>
<td>9.24.1 Detailed Description</td>
<td>1179</td>
</tr>
<tr>
<td>9.25 hal_dac.c File Reference</td>
<td>1179</td>
</tr>
<tr>
<td>9.25.1 Detailed Description</td>
<td>1180</td>
</tr>
<tr>
<td>9.26 hal_dac.h File Reference</td>
<td>1180</td>
</tr>
<tr>
<td>9.26.1 Detailed Description</td>
<td>1182</td>
</tr>
<tr>
<td>9.27 hal_dac_lld.c File Reference</td>
<td>1182</td>
</tr>
<tr>
<td>9.27.1 Detailed Description</td>
<td>1183</td>
</tr>
<tr>
<td>9.28 hal_dac_lld.h File Reference</td>
<td>1183</td>
</tr>
<tr>
<td>9.28.1 Detailed Description</td>
<td>1184</td>
</tr>
<tr>
<td>9.29 hal_efl.c File Reference</td>
<td>1184</td>
</tr>
<tr>
<td>9.29.1 Detailed Description</td>
<td>1184</td>
</tr>
<tr>
<td>9.30 hal_efl.h File Reference</td>
<td>1184</td>
</tr>
<tr>
<td>9.30.1 Detailed Description</td>
<td>1185</td>
</tr>
<tr>
<td>9.31 hal_efl_lld.c File Reference</td>
<td>1185</td>
</tr>
<tr>
<td>9.31.1 Detailed Description</td>
<td>1186</td>
</tr>
<tr>
<td>9.32 hal_efl_lld.h File Reference</td>
<td>1186</td>
</tr>
<tr>
<td>9.32.1 Detailed Description</td>
<td>1187</td>
</tr>
<tr>
<td>9.33 hal_files.h File Reference</td>
<td>1187</td>
</tr>
<tr>
<td>9.33.1 Detailed Description</td>
<td>1188</td>
</tr>
<tr>
<td>9.34 hal_flash.c File Reference</td>
<td>1188</td>
</tr>
<tr>
<td>9.34.1 Detailed Description</td>
<td>1189</td>
</tr>
<tr>
<td>9.35 hal_flash.h File Reference</td>
<td>1189</td>
</tr>
<tr>
<td>9.35.1 Detailed Description</td>
<td>1191</td>
</tr>
<tr>
<td>9.36 hal_gpt.c File Reference</td>
<td>1191</td>
</tr>
<tr>
<td>9.36.1 Detailed Description</td>
<td>1192</td>
</tr>
<tr>
<td>9.37 hal_gpt.h File Reference</td>
<td>1192</td>
</tr>
<tr>
<td>9.37.1 Detailed Description</td>
<td>1193</td>
</tr>
<tr>
<td>9.38 hal_gpt_lld.c File Reference</td>
<td>1193</td>
</tr>
<tr>
<td>9.38.1 Detailed Description</td>
<td>1194</td>
</tr>
<tr>
<td>9.39 hal_gpt_lld.h File Reference</td>
<td>1194</td>
</tr>
</tbody>
</table>
9.39.1 Detailed Description ........................................ 1195
9.40 hal_i2c.c File Reference ........................................ 1195
  9.40.1 Detailed Description ......................................... 1195
9.41 hal_i2c.h File Reference ........................................ 1195
  9.41.1 Detailed Description ......................................... 1197
9.42 hal_i2c_lld.c File Reference ................................. 1197
  9.42.1 Detailed Description ......................................... 1198
9.43 hal_i2c_lld.h File Reference ................................. 1198
  9.43.1 Detailed Description ......................................... 1199
9.44 hal_i2s.c File Reference ........................................ 1199
  9.44.1 Detailed Description ......................................... 1199
9.45 hal_i2s.h File Reference ........................................ 1200
  9.45.1 Detailed Description ......................................... 1201
9.46 hal_i2s_lld.c File Reference ................................. 1201
  9.46.1 Detailed Description ......................................... 1201
9.47 hal_i2s_lld.h File Reference ................................. 1202
  9.47.1 Detailed Description ......................................... 1202
9.48 hal_icu.c File Reference ........................................ 1202
  9.48.1 Detailed Description ......................................... 1203
9.49 hal_icu.h File Reference ........................................ 1203
  9.49.1 Detailed Description ......................................... 1204
9.50 hal_icu_lld.c File Reference ................................. 1204
  9.50.1 Detailed Description ......................................... 1205
9.51 hal_icu_lld.h File Reference ................................. 1205
  9.51.1 Detailed Description ......................................... 1206
9.52 hal_ioblock.h File Reference ................................ 1207
  9.52.1 Detailed Description ......................................... 1208
9.53 hal_lld.c File Reference ........................................ 1208
  9.53.1 Detailed Description ......................................... 1208
9.54 hal_lld.h File Reference ........................................ 1208
  9.54.1 Detailed Description ......................................... 1209
9.55 hal_mac.c File Reference ...................................... 1209
  9.55.1 Detailed Description ......................................... 1210
9.56 hal_mac.h File Reference ...................................... 1210
  9.56.1 Detailed Description ......................................... 1212
9.57 hal_mac_lld.c File Reference ................................. 1212
  9.57.1 Detailed Description ......................................... 1212
9.58 hal_mac_lld.h File Reference ................................. 1213
  9.58.1 Detailed Description ......................................... 1214
9.59 hal_mfs.c File Reference ...................................... 1214
  9.59.1 Detailed Description ......................................... 1215
9.60 hal_mfs.h File Reference ...................................... 1215
<table>
<thead>
<tr>
<th>Section</th>
<th>File Reference</th>
<th>Detailed Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.60.1</td>
<td>Detailed Description</td>
<td>1217</td>
</tr>
<tr>
<td>9.61</td>
<td>hal_mii.h</td>
<td>1217</td>
</tr>
<tr>
<td>9.61.1</td>
<td>Detailed Description</td>
<td>1220</td>
</tr>
<tr>
<td>9.62</td>
<td>hal_mmc_spi.c</td>
<td>1220</td>
</tr>
<tr>
<td>9.62.1</td>
<td>Detailed Description</td>
<td>1221</td>
</tr>
<tr>
<td>9.63</td>
<td>hal_mmc_spi.h</td>
<td>1221</td>
</tr>
<tr>
<td>9.63.1</td>
<td>Detailed Description</td>
<td>1223</td>
</tr>
<tr>
<td>9.64</td>
<td>hal_mmcsd.c</td>
<td>1223</td>
</tr>
<tr>
<td>9.64.1</td>
<td>Detailed Description</td>
<td>1223</td>
</tr>
<tr>
<td>9.65</td>
<td>hal_mmcsd.h</td>
<td>1224</td>
</tr>
<tr>
<td>9.65.1</td>
<td>Detailed Description</td>
<td>1228</td>
</tr>
<tr>
<td>9.66</td>
<td>hal_objects.h</td>
<td>1228</td>
</tr>
<tr>
<td>9.66.1</td>
<td>Detailed Description</td>
<td>1228</td>
</tr>
<tr>
<td>9.67</td>
<td>hal_pal.c</td>
<td>1228</td>
</tr>
<tr>
<td>9.67.1</td>
<td>Detailed Description</td>
<td>1229</td>
</tr>
<tr>
<td>9.68</td>
<td>hal_pal.h</td>
<td>1229</td>
</tr>
<tr>
<td>9.68.1</td>
<td>Detailed Description</td>
<td>1233</td>
</tr>
<tr>
<td>9.69</td>
<td>hal_pal_lld.c</td>
<td>1233</td>
</tr>
<tr>
<td>9.69.1</td>
<td>Detailed Description</td>
<td>1233</td>
</tr>
<tr>
<td>9.70</td>
<td>hal_pal_lld.h</td>
<td>1233</td>
</tr>
<tr>
<td>9.70.1</td>
<td>Detailed Description</td>
<td>1235</td>
</tr>
<tr>
<td>9.71</td>
<td>hal_persistent.h</td>
<td>1235</td>
</tr>
<tr>
<td>9.71.1</td>
<td>Detailed Description</td>
<td>1236</td>
</tr>
<tr>
<td>9.72</td>
<td>hal_pwm.c</td>
<td>1236</td>
</tr>
<tr>
<td>9.72.1</td>
<td>Detailed Description</td>
<td>1237</td>
</tr>
<tr>
<td>9.73</td>
<td>hal_pwm.h</td>
<td>1237</td>
</tr>
<tr>
<td>9.73.1</td>
<td>Detailed Description</td>
<td>1239</td>
</tr>
<tr>
<td>9.74</td>
<td>hal_pwm_lld.c</td>
<td>1239</td>
</tr>
<tr>
<td>9.74.1</td>
<td>Detailed Description</td>
<td>1240</td>
</tr>
<tr>
<td>9.75</td>
<td>hal_pwm_lld.h</td>
<td>1240</td>
</tr>
<tr>
<td>9.75.1</td>
<td>Detailed Description</td>
<td>1241</td>
</tr>
<tr>
<td>9.76</td>
<td>hal_queues.c</td>
<td>1241</td>
</tr>
<tr>
<td>9.76.1</td>
<td>Detailed Description</td>
<td>1242</td>
</tr>
<tr>
<td>9.77</td>
<td>hal_queues.h</td>
<td>1242</td>
</tr>
<tr>
<td>9.77.1</td>
<td>Detailed Description</td>
<td>1244</td>
</tr>
<tr>
<td>9.78</td>
<td>hal_rtc.c</td>
<td>1244</td>
</tr>
<tr>
<td>9.78.1</td>
<td>Detailed Description</td>
<td>1245</td>
</tr>
<tr>
<td>9.79</td>
<td>hal_rtc.h</td>
<td>1245</td>
</tr>
<tr>
<td>9.79.1</td>
<td>Detailed Description</td>
<td>1247</td>
</tr>
<tr>
<td>9.80</td>
<td>hal_rtc_lld.c</td>
<td>1247</td>
</tr>
<tr>
<td>9.80.1</td>
<td>Detailed Description</td>
<td>1247</td>
</tr>
<tr>
<td>9.81</td>
<td>hal_rtc_lld.h</td>
<td>1247</td>
</tr>
</tbody>
</table>
Chapter 1

ChibiOS/HAL

1.1 Copyright

Copyright (C) 2006..2015 Giovanni Di Sirio. All rights reserved.

Neither the whole nor any part of the information contained in this document may be adapted or reproduced in any form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Giovanni Di Sirio in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Giovanni Di Sirio shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

1.2 Introduction

This document is the Reference Manual for the ChibiOS/HAL hardware abstraction layer.

1.3 Related Documents

- ChibiOS/HAL General Architecture
Chapter 2

Deprecated List

Global `canReceive` (canp, mailbox, crfp, timeout)

Global `canTransmit` (canp, mailbox, ctfp, timeout)

Global `flashGetAddressOffset` (BaseFlash *devp, void *addr)

Global `flashGetOffsetAddress` (BaseFlash *devp, flash_offset_t offset)

Global `HAL_FAILED`

Global `HAL_SUCCESS`

Global `MMCCConfig`

Global `MMCDriver`

Global `sdGetWouldBlock` (SerialDriver *sdp)

Global `sdPutWouldBlock` (SerialDriver *sdp)
Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>895</td>
</tr>
<tr>
<td>Peripheral Interfaces</td>
<td>536</td>
</tr>
<tr>
<td>Generic Flash Interface</td>
<td>223</td>
</tr>
<tr>
<td>Configuration</td>
<td>898</td>
</tr>
<tr>
<td>Normal Drivers</td>
<td></td>
</tr>
<tr>
<td>HAL Driver</td>
<td>23</td>
</tr>
<tr>
<td>ADC Driver</td>
<td>33</td>
</tr>
<tr>
<td>CAN Driver</td>
<td>85</td>
</tr>
<tr>
<td>Crypto Driver</td>
<td>109</td>
</tr>
<tr>
<td>DAC Driver</td>
<td>186</td>
</tr>
<tr>
<td>EFL Driver</td>
<td>210</td>
</tr>
<tr>
<td>GPT Driver</td>
<td>237</td>
</tr>
<tr>
<td>I2C Driver</td>
<td>256</td>
</tr>
<tr>
<td>I2S Driver</td>
<td>273</td>
</tr>
<tr>
<td>ICU Driver</td>
<td>286</td>
</tr>
<tr>
<td>MAC Driver</td>
<td>374</td>
</tr>
<tr>
<td>PAL Driver</td>
<td>482</td>
</tr>
<tr>
<td>PWM Driver</td>
<td>537</td>
</tr>
<tr>
<td>RTC Driver</td>
<td>591</td>
</tr>
<tr>
<td>SDC Driver</td>
<td>604</td>
</tr>
<tr>
<td>Serial Driver</td>
<td>638</td>
</tr>
<tr>
<td>SIO Driver</td>
<td>669</td>
</tr>
<tr>
<td>SPI Driver Selector</td>
<td>711</td>
</tr>
<tr>
<td>SPIv1 Driver</td>
<td>712</td>
</tr>
<tr>
<td>SPIv2 Driver</td>
<td>732</td>
</tr>
<tr>
<td>ST Driver</td>
<td>752</td>
</tr>
<tr>
<td>TRNG Driver</td>
<td>764</td>
</tr>
<tr>
<td>UART Driver</td>
<td>774</td>
</tr>
<tr>
<td>USB Driver</td>
<td>810</td>
</tr>
<tr>
<td>WDG Driver</td>
<td>861</td>
</tr>
<tr>
<td>WSPI Driver</td>
<td>868</td>
</tr>
<tr>
<td>Complex Drivers</td>
<td>915</td>
</tr>
<tr>
<td>Buffered SIO Driver</td>
<td>57</td>
</tr>
<tr>
<td>Managed Flash Storage Driver</td>
<td>399</td>
</tr>
<tr>
<td>MMC over SPI Driver</td>
<td>435</td>
</tr>
<tr>
<td>Module Index</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>Serial NOR Flash Driver ........................................ 468</td>
<td></td>
</tr>
<tr>
<td>Serial over USB Driver .......................................... 657</td>
<td></td>
</tr>
<tr>
<td>Interfaces and Classes ........................................ 916</td>
<td></td>
</tr>
<tr>
<td>Blocks ................................................................. 310</td>
<td></td>
</tr>
<tr>
<td>Abstract I/O Block Device ....................................... 365</td>
<td></td>
</tr>
<tr>
<td>Streams ................................................................. 311</td>
<td></td>
</tr>
<tr>
<td>Abstract Streams Interface ...................................... 312</td>
<td></td>
</tr>
<tr>
<td>Abstract I/O Channel Interface ................................ 318</td>
<td></td>
</tr>
<tr>
<td>Abstract Files Interface ......................................... 329</td>
<td></td>
</tr>
<tr>
<td>Abstract Persistent Storage Interface ......................... 337</td>
<td></td>
</tr>
<tr>
<td>Memory Streams Class ............................................ 342</td>
<td></td>
</tr>
<tr>
<td>Null Streams Class ................................................ 344</td>
<td></td>
</tr>
<tr>
<td>Buffered Serial Class ............................................ 346</td>
<td></td>
</tr>
<tr>
<td>Buffered Streams Class .......................................... 351</td>
<td></td>
</tr>
<tr>
<td>Output Formatter Utility ......................................... 353</td>
<td></td>
</tr>
<tr>
<td>Input Formatter Utility .......................................... 359</td>
<td></td>
</tr>
<tr>
<td>Base Object ........................................................... 480</td>
<td></td>
</tr>
<tr>
<td>Inner Code ............................................................ 917</td>
<td></td>
</tr>
<tr>
<td>I/O Buffers Queues ............................................... 58</td>
<td></td>
</tr>
<tr>
<td>MMC/SD Block Device ............................................. 455</td>
<td></td>
</tr>
<tr>
<td>I/O Bytes Queues .................................................. 566</td>
<td></td>
</tr>
<tr>
<td>Support Code .......................................................... 918</td>
<td></td>
</tr>
<tr>
<td>MII/RMII Header .................................................... 420</td>
<td></td>
</tr>
<tr>
<td>USB CDC Header ..................................................... 859</td>
<td></td>
</tr>
<tr>
<td>OSAL ................................................................. 919</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BaseBlockDeviceVMT .......................................................... 961
MMCSDBlockDeviceVMT ............................................... 1090
mmc_spi_driver_vmt ....................................................... 1086
SDCDriverVMT ............................................................ 1110
BaseFlashVMT ............................................................ 971
EFlashDriverVMT .......................................................... 991
SNORDriverVMT ........................................................... 1124
BaseObject ............................................................... 971
BaseBlockDevice .......................................................... 958
  MMCSDBlockDevice .................................................. 1087
  mmc_spi_driver_t ................................................... 1082
BaseFlash ............................................................... 968
  hal_efl_driver ...................................................... 1019
  SNORDriver .......................................................... 1122
BasePersistentStorage .................................................. 973
BaseSequentialStream ................................................... 976
  BaseBufferedStream ................................................ 962
  NullStream .......................................................... 1091
  BaseChannel .......................................................... 965
  BaseAsynchronousChannel .......................................... 955
    hal_buffered_serial ............................................ 1008
    hal_serial_driver .............................................. 1032
    SerialUSBDriver ................................................ 1115
  FileStream .......................................................... 994
  RTCDriverVMT ......................................................... 1105
  MemoryStream ........................................................ 1066
BaseObjectVMT .......................................................... 973
BasePersistentStorageVMT ................................................ 975
BaseSequentialStreamVMT ................................................. 978
  BaseBufferedStreamVMT ........................................... 964
  BaseChannelVMT ...................................................... 967
  BaseAsynchronousChannelVMT ..................................... 957
    BufferedSerialVMT ............................................... 980
4.1 Class Hierarchy

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFSConfig</td>
<td>1074</td>
</tr>
<tr>
<td>MFSDriver</td>
<td>1077</td>
</tr>
<tr>
<td>mmc_spi_config_t</td>
<td>1080</td>
</tr>
<tr>
<td>NullStreamVMT</td>
<td>1093</td>
</tr>
<tr>
<td>PALConfig</td>
<td>1093</td>
</tr>
<tr>
<td>paevent_t</td>
<td>1094</td>
</tr>
<tr>
<td>PWMChannelConfig</td>
<td>1095</td>
</tr>
<tr>
<td>PWMConfig</td>
<td>1097</td>
</tr>
<tr>
<td>PWMDriver</td>
<td>1099</td>
</tr>
<tr>
<td>RTCAlarm</td>
<td>1101</td>
</tr>
<tr>
<td>RTCDateTime</td>
<td>1101</td>
</tr>
<tr>
<td>RTCDriver</td>
<td>1103</td>
</tr>
<tr>
<td>SDCConfig</td>
<td>1106</td>
</tr>
<tr>
<td>SDCDriver</td>
<td>1107</td>
</tr>
<tr>
<td>SerialUSBConfig</td>
<td>1113</td>
</tr>
<tr>
<td>SHA1Context</td>
<td>1118</td>
</tr>
<tr>
<td>SHA256Context</td>
<td>1119</td>
</tr>
<tr>
<td>SHA512Context</td>
<td>1119</td>
</tr>
<tr>
<td>SNORConfig</td>
<td>1121</td>
</tr>
<tr>
<td>threads_queue_t</td>
<td>1125</td>
</tr>
<tr>
<td>unpacked_mmc_cid_t</td>
<td>1126</td>
</tr>
<tr>
<td>unpacked_mmc_csd_t</td>
<td>1126</td>
</tr>
<tr>
<td>unpacked_sdc_cid_t</td>
<td>1127</td>
</tr>
<tr>
<td>unpacked_sdc_csd_10_t</td>
<td>1127</td>
</tr>
<tr>
<td>unpacked_sdc_csd_20_t</td>
<td>1128</td>
</tr>
<tr>
<td>USBConfig</td>
<td>1129</td>
</tr>
<tr>
<td>USBDescriptor</td>
<td>1131</td>
</tr>
<tr>
<td>USBDriver</td>
<td>1132</td>
</tr>
<tr>
<td>USBEndpointConfig</td>
<td>1136</td>
</tr>
<tr>
<td>USBInEndpointState</td>
<td>1140</td>
</tr>
<tr>
<td>USBOutEndpointState</td>
<td>1141</td>
</tr>
<tr>
<td>WDGConfig</td>
<td>1143</td>
</tr>
<tr>
<td>WDGDriver</td>
<td>1143</td>
</tr>
<tr>
<td>wspi_command_t</td>
<td>1144</td>
</tr>
</tbody>
</table>
## 5.1 Data Structures

Here are the data structures with brief descriptions:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseAsynchronousChannel</td>
<td>Base asynchronous channel class</td>
<td>955</td>
</tr>
<tr>
<td>BaseAsynchronousChannelVMT</td>
<td>BaseAsynchronousChannel virtual methods table</td>
<td>957</td>
</tr>
<tr>
<td>BaseBlockDevice</td>
<td>Base block device class</td>
<td>958</td>
</tr>
<tr>
<td>BaseBlockDeviceVMT</td>
<td>BaseBlockDevice virtual methods table</td>
<td>961</td>
</tr>
<tr>
<td>BaseBufferedStream</td>
<td>Buffered stream class</td>
<td>962</td>
</tr>
<tr>
<td>BaseBufferedStreamVMT</td>
<td>BaseBufferedStream virtual methods table</td>
<td>964</td>
</tr>
<tr>
<td>BaseChannel</td>
<td>Base channel class</td>
<td>965</td>
</tr>
<tr>
<td>BaseChannelVMT</td>
<td>BaseChannel virtual methods table</td>
<td>967</td>
</tr>
<tr>
<td>BaseFlash</td>
<td>Base flash class</td>
<td>968</td>
</tr>
<tr>
<td>BaseFlashVMT</td>
<td>BaseFlash virtual methods table</td>
<td>971</td>
</tr>
<tr>
<td>BaseObject</td>
<td>Base object class</td>
<td>971</td>
</tr>
<tr>
<td>BaseObjectVMT</td>
<td>BaseObject virtual methods table</td>
<td>973</td>
</tr>
<tr>
<td>BasePersistentStorage</td>
<td>Base persistent storage class</td>
<td>973</td>
</tr>
<tr>
<td>BasePersistentStorageVMT</td>
<td>BasePersistentStorage virtual methods table</td>
<td>975</td>
</tr>
<tr>
<td>BaseSequentialStream</td>
<td>Base stream class</td>
<td>976</td>
</tr>
<tr>
<td>BaseSequentialStreamVMT</td>
<td>BaseSequentialStream virtual methods table</td>
<td>978</td>
</tr>
<tr>
<td>BlockDeviceInfo</td>
<td>Block device info</td>
<td>979</td>
</tr>
<tr>
<td>BufferedSerialVMT</td>
<td>BufferedSerial virtual methods table</td>
<td>980</td>
</tr>
<tr>
<td>Structure Name</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>BufferedStreamAdapter</td>
<td>Buffered stream adapter object</td>
<td>981</td>
</tr>
<tr>
<td>BufferedStreamAdapterVMT</td>
<td>BufferedStreamAdapter virtual methods table</td>
<td>983</td>
</tr>
<tr>
<td>CANRxFrame</td>
<td>CAN received frame</td>
<td>983</td>
</tr>
<tr>
<td>CANTxFrame</td>
<td>CAN transmission frame</td>
<td>986</td>
</tr>
<tr>
<td>cdc_linecoding_t</td>
<td>Type of Line Coding structure</td>
<td>989</td>
</tr>
<tr>
<td>CRYConfig</td>
<td>Driver configuration structure</td>
<td>989</td>
</tr>
<tr>
<td>CRYDriver</td>
<td>Structure representing an CRY driver</td>
<td>990</td>
</tr>
<tr>
<td>EFlashDriverVMT</td>
<td>EFlash virtual methods table</td>
<td>991</td>
</tr>
<tr>
<td>event_source</td>
<td>Events source object</td>
<td>992</td>
</tr>
<tr>
<td>FileStream</td>
<td>Base file stream class</td>
<td>994</td>
</tr>
<tr>
<td>FileStreamVMT</td>
<td>FileStream virtual methods table</td>
<td>996</td>
</tr>
<tr>
<td>flash_descriptor_t</td>
<td>Type of a flash device descriptor</td>
<td>997</td>
</tr>
<tr>
<td>flash_sector_descriptor_t</td>
<td>Flash sector descriptor</td>
<td>999</td>
</tr>
<tr>
<td>GPTConfig</td>
<td>Driver configuration structure</td>
<td>1000</td>
</tr>
<tr>
<td>GPTDriver</td>
<td>Structure representing a GPT driver</td>
<td>1002</td>
</tr>
<tr>
<td>hal_adc_config</td>
<td>Driver configuration structure</td>
<td>1003</td>
</tr>
<tr>
<td>hal_adc_configuration_group</td>
<td>Conversion group configuration structure</td>
<td>1003</td>
</tr>
<tr>
<td>hal_adc_driver</td>
<td>Structure representing an ADC driver</td>
<td>1006</td>
</tr>
<tr>
<td>hal_buffered_serial</td>
<td>Buffered serial channel class</td>
<td>1008</td>
</tr>
<tr>
<td>hal_can_config</td>
<td>Type of a CAN configuration structure</td>
<td>1010</td>
</tr>
<tr>
<td>hal_can_driver</td>
<td>Structure representing an CAN driver</td>
<td>1010</td>
</tr>
<tr>
<td>hal_dac_config</td>
<td>Driver configuration structure</td>
<td>1013</td>
</tr>
<tr>
<td>hal_dac_conversion_group</td>
<td>DAC Conversion group structure</td>
<td>1014</td>
</tr>
<tr>
<td>hal_dac_driver</td>
<td>Structure representing a DAC driver</td>
<td>1015</td>
</tr>
<tr>
<td>hal_efl_config</td>
<td>Type of a structure representing a flash driver configuration</td>
<td>1018</td>
</tr>
<tr>
<td>hal_efl_driver</td>
<td>Structure representing an embedded flash driver</td>
<td>1019</td>
</tr>
<tr>
<td>hal_i2c_config</td>
<td>I2C driver configuration structure</td>
<td>1021</td>
</tr>
<tr>
<td>hal_i2c_driver</td>
<td>Structure representing an I2C driver</td>
<td>1021</td>
</tr>
<tr>
<td>hal_i2s_config</td>
<td>Driver configuration structure</td>
<td>1023</td>
</tr>
<tr>
<td>Structure/Driver</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>hal_i2s_driver</td>
<td>Structure representing an I2S driver</td>
<td></td>
</tr>
<tr>
<td>hal_mac_config</td>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>hal_mac_driver</td>
<td>Structure representing a MAC driver</td>
<td></td>
</tr>
<tr>
<td>hal_mac_receive_descriptor</td>
<td>Structure representing a MAC receive descriptor</td>
<td></td>
</tr>
<tr>
<td>hal_mac_transmit_descriptor</td>
<td>Structure representing a MAC transmit descriptor</td>
<td></td>
</tr>
<tr>
<td>hal_serial_config</td>
<td>PLATFORM Serial Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>hal_serial_driver</td>
<td>Full duplex serial driver class</td>
<td></td>
</tr>
<tr>
<td>hal_sio_config</td>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>hal_sio_driver</td>
<td>Structure representing a SIO driver</td>
<td></td>
</tr>
<tr>
<td>hal_spi_config</td>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>hal_spi_driver</td>
<td>Structure representing an SPI driver</td>
<td></td>
</tr>
<tr>
<td>hal_trng_config</td>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>hal_trng_driver</td>
<td>Structure representing a TRNG driver</td>
<td></td>
</tr>
<tr>
<td>hal_uart_config</td>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>hal_uart_driver</td>
<td>Structure representing an UART driver</td>
<td></td>
</tr>
<tr>
<td>hal_wspi_config</td>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>hal_wspi_driver</td>
<td>Structure representing an WSPI driver</td>
<td></td>
</tr>
<tr>
<td>halclkcfg_t</td>
<td>Type of a clock configuration structure</td>
<td></td>
</tr>
<tr>
<td>HMACSHA256Context</td>
<td>Type of a HMAC_SHA256 context</td>
<td></td>
</tr>
<tr>
<td>HMACSHA512Context</td>
<td>Type of a HMAC_SHA512 context</td>
<td></td>
</tr>
<tr>
<td>ICUConfig</td>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>ICU.Driver</td>
<td>Structure representing an ICU driver</td>
<td></td>
</tr>
<tr>
<td>io_buffers_queue</td>
<td>Structure of a generic buffers queue</td>
<td></td>
</tr>
<tr>
<td>io_queue</td>
<td>Generic I/O queue structure</td>
<td></td>
</tr>
<tr>
<td>IOBus</td>
<td>I/O bus descriptor</td>
<td></td>
</tr>
<tr>
<td>MemoryStream</td>
<td>Memory stream object</td>
<td></td>
</tr>
<tr>
<td>MemStreamVMT</td>
<td>MemStream virtual methods table</td>
<td></td>
</tr>
<tr>
<td>mfs_bank_header_t</td>
<td>Type of a bank header</td>
<td></td>
</tr>
<tr>
<td>mfs_data_header_t</td>
<td>Type of a data block header</td>
<td></td>
</tr>
<tr>
<td>Symbol/Type</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>mfs_nocache_buffer</td>
<td>Type of an non-cacheable MFS buffer</td>
<td>1072</td>
</tr>
<tr>
<td>mfs_transaction_op_t</td>
<td>Type of a buffered write/erase operation within a transaction</td>
<td>1072</td>
</tr>
<tr>
<td>MFSCConfig</td>
<td>Type of a MFS configuration structure</td>
<td>1074</td>
</tr>
<tr>
<td>MFSDriver</td>
<td>Type of an MFS instance</td>
<td>1077</td>
</tr>
<tr>
<td>mmc_spi_config_t</td>
<td>Type of a MMC/SD over SPI driver configuration structure</td>
<td>1080</td>
</tr>
<tr>
<td>mmc_spi_driver_t</td>
<td>Structure representing a MMC/SD over SPI driver</td>
<td>1082</td>
</tr>
<tr>
<td>mmc_spi_driver_vmt</td>
<td>MMCDriver virtual methods table</td>
<td>1086</td>
</tr>
<tr>
<td>MMCSDBlockDevice</td>
<td>MCC/SD block device class</td>
<td>1087</td>
</tr>
<tr>
<td>MCMCSDBlockDeviceVMT</td>
<td>MCMCSDBlockDevice virtual methods table</td>
<td>1090</td>
</tr>
<tr>
<td>NullStream</td>
<td>Null stream object</td>
<td>1091</td>
</tr>
<tr>
<td>NullStreamVMT</td>
<td>NullStream virtual methods table</td>
<td>1093</td>
</tr>
<tr>
<td>PALConfig</td>
<td>Generic I/O ports static initializer</td>
<td>1093</td>
</tr>
<tr>
<td>palevent_t</td>
<td>Type of a PAL event record</td>
<td>1094</td>
</tr>
<tr>
<td>PWMChannelConfig</td>
<td>Type of a PWM driver channel configuration structure</td>
<td>1095</td>
</tr>
<tr>
<td>PWMConfig</td>
<td>Type of a PWM driver configuration structure</td>
<td>1097</td>
</tr>
<tr>
<td>PWMDriver</td>
<td>Structure representing a PWM driver</td>
<td>1099</td>
</tr>
<tr>
<td>RTCAlarm</td>
<td>Type of a structure representing an RTC alarm time stamp</td>
<td>1101</td>
</tr>
<tr>
<td>RTCDateTime</td>
<td>Type of a structure representing an RTC date/time stamp</td>
<td>1101</td>
</tr>
<tr>
<td>RTCDriver</td>
<td>Structure representing an RTC driver</td>
<td>1103</td>
</tr>
<tr>
<td>RTCDriverVMT</td>
<td>RTCDriver virtual methods table</td>
<td>1105</td>
</tr>
<tr>
<td>SDCConfig</td>
<td>Driver configuration structure</td>
<td>1106</td>
</tr>
<tr>
<td>SDCDriver</td>
<td>Structure representing an SDC driver</td>
<td>1107</td>
</tr>
<tr>
<td>SDCDriverVMT</td>
<td>SDCDriver virtual methods table</td>
<td>1110</td>
</tr>
<tr>
<td>SerialDriverVMT</td>
<td>SerialDriver virtual methods table</td>
<td>1111</td>
</tr>
<tr>
<td>SerialUSBConfig</td>
<td>Serial over USB Driver configuration structure</td>
<td>1113</td>
</tr>
<tr>
<td>SerialUSBDriver</td>
<td>Full duplex serial driver class</td>
<td>1115</td>
</tr>
<tr>
<td>SerialUSBDriverVMT</td>
<td>SerialDriver virtual methods table</td>
<td>1117</td>
</tr>
<tr>
<td>SHA1Context</td>
<td>Type of a SHA1 context</td>
<td>1118</td>
</tr>
<tr>
<td>SHA256Context</td>
<td>Type of a SHA256 context</td>
<td>1119</td>
</tr>
<tr>
<td>Type</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>SHA512Context</td>
<td>1119</td>
<td></td>
</tr>
<tr>
<td>sio_driver_vmt</td>
<td>1120</td>
<td></td>
</tr>
<tr>
<td>SNORConfig</td>
<td>1121</td>
<td></td>
</tr>
<tr>
<td>SNORDriver</td>
<td>1122</td>
<td></td>
</tr>
<tr>
<td>threads_queue_t</td>
<td>1124</td>
<td></td>
</tr>
<tr>
<td>unpacked_mmc_cid_t</td>
<td>1125</td>
<td></td>
</tr>
<tr>
<td>unpacked_mmc_csd_t</td>
<td>1126</td>
<td></td>
</tr>
<tr>
<td>unpacked_sdc_cid_t</td>
<td>1127</td>
<td></td>
</tr>
<tr>
<td>unpacked_sdc_csd_1_0_t</td>
<td>1127</td>
<td></td>
</tr>
<tr>
<td>unpacked_sdc_csd_2_0_t</td>
<td>1128</td>
<td></td>
</tr>
<tr>
<td>USBConfig</td>
<td>1129</td>
<td></td>
</tr>
<tr>
<td>USBDescriptor</td>
<td>1131</td>
<td></td>
</tr>
<tr>
<td>USBDriver</td>
<td>1132</td>
<td></td>
</tr>
<tr>
<td>USBEndpointConfig</td>
<td>1136</td>
<td></td>
</tr>
<tr>
<td>USBInEndpointState</td>
<td>1140</td>
<td></td>
</tr>
<tr>
<td>USBOutEndpointState</td>
<td>1141</td>
<td></td>
</tr>
<tr>
<td>WDGConfig</td>
<td>1143</td>
<td></td>
</tr>
<tr>
<td>WDGDriver</td>
<td>1143</td>
<td></td>
</tr>
<tr>
<td>wspi_command_t</td>
<td>1144</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

- **bufstreams.c**
  - Buffered streams code .................................................. 1147
- **bufstreams.h**
  - Buffered streams structures and macros ............................. 1147
- **chprintf.c**
  - Mini printf-like functionality ........................................ 1148
- **chprintf.h**
  - Mini printf-like functionality ........................................ 1148
- **chscanf.c**
  - Mini scanf-like functionality ........................................ 1149
- **chscanf.h**
  - Mini scanf-like functionality ........................................ 1150
- **hal.c**
  - HAL subsystem code ................................................... 1150
- **hal.h**
  - HAL subsystem header ................................................ 1151
- **hal_adc.c**
  - ADC Driver code ....................................................... 1152
- **hal_adc.h**
  - ADC Driver macros and structures .................................. 1153
- **hal_adc_lld.c**
  - PLATFORM ADC subsystem low level driver source .................. 1155
- **hal_adc_lld.h**
  - PLATFORM ADC subsystem low level driver header ................. 1156
- **hal_buffered_serial.h**
  - Buffered Serial Driver code ......................................... 1157
- **hal_buffers.c**
  - I/O Buffers code ...................................................... 1158
- **hal_buffers.h**
  - I/O Buffers macros and structures .................................. 1160
- **hal_can.c**
  - CAN Driver code ...................................................... 1162
- **hal_can.h**
  - CAN Driver macros and structures .................................. 1163
- **hal_can_lld.c**
  - PLATFORM CAN subsystem low level driver source ............... 1164
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>hal_can_lld.h</td>
<td>PLATFORM CAN subsystem low level driver header</td>
<td>1165</td>
</tr>
<tr>
<td>hal_channels.h</td>
<td>I/O channels access</td>
<td>1167</td>
</tr>
<tr>
<td>hal_crypto.c</td>
<td>Cryptographic Driver code</td>
<td>1168</td>
</tr>
<tr>
<td>hal_crypto.h</td>
<td>Cryptographic Driver macros and structures</td>
<td>1171</td>
</tr>
<tr>
<td></td>
<td>PLATFORM cryptographic subsystem low level driver code</td>
<td>1174</td>
</tr>
<tr>
<td></td>
<td>PLATFORM cryptographic subsystem low level driver header</td>
<td>1176</td>
</tr>
<tr>
<td>hal_dac.c</td>
<td>DAC Driver code</td>
<td>1179</td>
</tr>
<tr>
<td>hal_dac.h</td>
<td>DAC Driver macros and structures</td>
<td>1180</td>
</tr>
<tr>
<td></td>
<td>PLATFORM DAC subsystem low level driver source</td>
<td>1182</td>
</tr>
<tr>
<td></td>
<td>PLATFORM DAC subsystem low level driver header</td>
<td>1183</td>
</tr>
<tr>
<td>hal_efl.c</td>
<td>Embedded Flash Driver code</td>
<td>1184</td>
</tr>
<tr>
<td>hal_efl.h</td>
<td>Embedded Flash Driver macros and structures</td>
<td>1184</td>
</tr>
<tr>
<td></td>
<td>PLATFORM Embedded Flash subsystem low level driver source</td>
<td>1185</td>
</tr>
<tr>
<td></td>
<td>PLATFORM Embedded Flash subsystem low level driver header</td>
<td>1186</td>
</tr>
<tr>
<td>hal_files.h</td>
<td>Data files</td>
<td>1187</td>
</tr>
<tr>
<td>hal_flash.c</td>
<td>Generic flash driver class code</td>
<td>1188</td>
</tr>
<tr>
<td>hal_flash.h</td>
<td>Generic flash driver class header</td>
<td>1189</td>
</tr>
<tr>
<td>hal_gpt.c</td>
<td>GPT Driver code</td>
<td>1191</td>
</tr>
<tr>
<td>hal_gpt.h</td>
<td>GPT Driver macros and structures</td>
<td>1192</td>
</tr>
<tr>
<td></td>
<td>PLATFORM GPT subsystem low level driver source</td>
<td>1193</td>
</tr>
<tr>
<td></td>
<td>PLATFORM GPT subsystem low level driver header</td>
<td>1194</td>
</tr>
<tr>
<td>hal_i2c.c</td>
<td>I2C Driver code</td>
<td>1195</td>
</tr>
<tr>
<td>hal_i2c.h</td>
<td>I2C Driver macros and structures</td>
<td>1195</td>
</tr>
<tr>
<td></td>
<td>PLATFORM I2C subsystem low level driver source</td>
<td>1197</td>
</tr>
<tr>
<td></td>
<td>PLATFORM I2C subsystem low level driver header</td>
<td>1198</td>
</tr>
<tr>
<td>hal_i2s.c</td>
<td>I2S Driver code</td>
<td>1199</td>
</tr>
<tr>
<td>hal_i2s.h</td>
<td>I2S Driver macros and structures</td>
<td>1200</td>
</tr>
<tr>
<td></td>
<td>PLATFORM I2S subsystem low level driver source</td>
<td>1201</td>
</tr>
<tr>
<td></td>
<td>PLATFORM I2S subsystem low level driver header</td>
<td>1202</td>
</tr>
<tr>
<td>File Name</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>hal_icu.c</td>
<td>ICU Driver code</td>
<td>1202</td>
</tr>
<tr>
<td>hal_icu.h</td>
<td>ICU Driver macros and structures</td>
<td>1203</td>
</tr>
<tr>
<td>hal_icu_lld.c</td>
<td>PLATFORM ADC subsystem low level driver source</td>
<td>1205</td>
</tr>
<tr>
<td>hal_icu_lld.h</td>
<td>PLATFORM ICU subsystem low level driver header</td>
<td>1205</td>
</tr>
<tr>
<td>hal_ioblock.h</td>
<td>I/O block devices access</td>
<td>1207</td>
</tr>
<tr>
<td>hal_lld.c</td>
<td>PLATFORM HAL subsystem low level driver source</td>
<td>1208</td>
</tr>
<tr>
<td>hal_lld.h</td>
<td>PLATFORM HAL subsystem low level driver header</td>
<td>1208</td>
</tr>
<tr>
<td>hal_mac.c</td>
<td>MAC Driver code</td>
<td>1209</td>
</tr>
<tr>
<td>hal_mac.h</td>
<td>MAC Driver macros and structures</td>
<td>1210</td>
</tr>
<tr>
<td>hal_mac_lld.c</td>
<td>PLATFORM MAC subsystem low level driver source</td>
<td>1212</td>
</tr>
<tr>
<td>hal_mac_lld.h</td>
<td>PLATFORM MAC subsystem low level driver header</td>
<td>1213</td>
</tr>
<tr>
<td>hal_mfs.c</td>
<td>Managed Flash Storage module code</td>
<td>1214</td>
</tr>
<tr>
<td>hal_mfs.h</td>
<td>Managed Flash Storage module header</td>
<td>1215</td>
</tr>
<tr>
<td>hal_mii.h</td>
<td>MII macros and structures</td>
<td>1217</td>
</tr>
<tr>
<td>hal_mmc_spi.c</td>
<td>MMC over SPI driver code</td>
<td>1220</td>
</tr>
<tr>
<td>hal_mmc_spi.h</td>
<td>MMC over SPI driver header</td>
<td>1221</td>
</tr>
<tr>
<td>hal_mmcsd.c</td>
<td>MMC/SD cards common code</td>
<td>1223</td>
</tr>
<tr>
<td>hal_mmcsd.h</td>
<td>MMC/SD cards common header</td>
<td>1224</td>
</tr>
<tr>
<td>hal_objects.h</td>
<td>Base object</td>
<td>1228</td>
</tr>
<tr>
<td>hal_pal.c</td>
<td>I/O Ports Abstraction Layer code</td>
<td>1228</td>
</tr>
<tr>
<td>hal_pal.h</td>
<td>I/O Ports Abstraction Layer macros, types and structures</td>
<td>1229</td>
</tr>
<tr>
<td>hal_pal_lld.c</td>
<td>PLATFORM PAL subsystem low level driver source</td>
<td>1233</td>
</tr>
<tr>
<td>hal_pal_lld.h</td>
<td>PLATFORM PAL subsystem low level driver header</td>
<td>1233</td>
</tr>
<tr>
<td>hal_persistent.h</td>
<td>Generic persistent storage class header</td>
<td>1235</td>
</tr>
<tr>
<td>hal_pwm.c</td>
<td>PWM Driver code</td>
<td>1236</td>
</tr>
<tr>
<td>hal_pwm.h</td>
<td>PWM Driver macros and structures</td>
<td>1237</td>
</tr>
<tr>
<td>hal_pwm_lld.c</td>
<td>PLATFORM PWM subsystem low level driver source</td>
<td>1239</td>
</tr>
<tr>
<td>hal_pwm_lld.h</td>
<td>PLATFORM PWM subsystem low level driver header</td>
<td>1240</td>
</tr>
<tr>
<td>hal_queues.c</td>
<td>I/O Queues code</td>
<td>1241</td>
</tr>
<tr>
<td>File Name</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>hal_queues.h</td>
<td>I/O Queues macros and structures</td>
<td>1242</td>
</tr>
<tr>
<td>hal_rtc.c</td>
<td>RTC Driver code</td>
<td>1244</td>
</tr>
<tr>
<td>hal_rtc.h</td>
<td>RTC Driver macros and structures</td>
<td>1245</td>
</tr>
<tr>
<td>hal_rtc_lld.c</td>
<td>PLATFORM RTC subsystem low level driver source</td>
<td>1247</td>
</tr>
<tr>
<td>hal_rtc_lld.h</td>
<td>PLATFORM RTC subsystem low level driver header</td>
<td>1247</td>
</tr>
<tr>
<td>hal_sdc.c</td>
<td>SDC Driver code</td>
<td>1249</td>
</tr>
<tr>
<td>hal_sdc.h</td>
<td>SDC Driver macros and structures</td>
<td>1250</td>
</tr>
<tr>
<td>hal_sdc_lld.c</td>
<td>PLATFORM SDC subsystem low level driver source</td>
<td>1252</td>
</tr>
<tr>
<td>hal_sdc_lld.h</td>
<td>PLATFORM SDC subsystem low level driver header</td>
<td>1253</td>
</tr>
<tr>
<td>hal_serial.c</td>
<td>Serial Driver code</td>
<td>1255</td>
</tr>
<tr>
<td>hal_serial.h</td>
<td>Serial Driver macros and structures</td>
<td>1256</td>
</tr>
<tr>
<td>hal_serial_lld.c</td>
<td>PLATFORM serial subsystem low level driver source</td>
<td>1258</td>
</tr>
<tr>
<td>hal_serial_lld.h</td>
<td>PLATFORM serial subsystem low level driver header</td>
<td>1258</td>
</tr>
<tr>
<td>hal_serial_nor.c</td>
<td>Serial NOR serial flash driver code</td>
<td>1258</td>
</tr>
<tr>
<td>hal_serial_nor.h</td>
<td>Serial NOR driver header</td>
<td>1260</td>
</tr>
<tr>
<td>hal_serial_usb.c</td>
<td>Serial over USB Driver code</td>
<td>1262</td>
</tr>
<tr>
<td>hal_serial_usb.h</td>
<td>Serial over USB Driver macros and structures</td>
<td>1263</td>
</tr>
<tr>
<td>hal_sio.c</td>
<td>SIO Driver code</td>
<td>1264</td>
</tr>
<tr>
<td>hal_sio.h</td>
<td>SIO Driver macros and structures</td>
<td>1265</td>
</tr>
<tr>
<td>hal_sio_lld.c</td>
<td>PLATFORM SIO subsystem low level driver source</td>
<td>1269</td>
</tr>
<tr>
<td>hal_sio_lld.h</td>
<td>PLATFORM SIO subsystem low level driver header</td>
<td>1270</td>
</tr>
<tr>
<td>hal_spi.c</td>
<td>SPI Driver selector code</td>
<td>1271</td>
</tr>
<tr>
<td>hal_spi.h</td>
<td>SPI Driver selector macros and structures</td>
<td>1271</td>
</tr>
<tr>
<td>hal_spi_lld.c</td>
<td>PLATFORM SPI (v1) subsystem low level driver source</td>
<td>1271</td>
</tr>
<tr>
<td>hal_spi_lld.h</td>
<td>PLATFORM SPI (v1) subsystem low level driver header</td>
<td>1272</td>
</tr>
<tr>
<td>hal_spi_v1.c</td>
<td>SPI (v1) Driver macros and structures</td>
<td>1273</td>
</tr>
<tr>
<td>hal_spi_v2.c</td>
<td>SPI (v2) Driver macros and structures</td>
<td>1275</td>
</tr>
<tr>
<td>hal_spi_v1_lld.c</td>
<td>PLATFORM SPI (v1) subsystem low level driver source</td>
<td>1276</td>
</tr>
<tr>
<td>hal_spi_v2_lld.c</td>
<td>PLATFORM SPI (v2) subsystem low level driver header</td>
<td>1277</td>
</tr>
</tbody>
</table>
6.1 File List

hal_st.c
   ST Driver code ........................................ 1278
hal_st.h
   ST Driver macros and structures ...................... 1279
hal_st_lld.c
   PLATFORM ST subsystem low level driver source .... 1280
hal_st_lld.h
   PLATFORM ST subsystem low level driver header .... 1280
hal_streams.h
   Data streams ........................................... 1281
hal_trng.c
   TRNG Driver code ...................................... 1282
hal_trng.h
   TRNG Driver macros and structures ................... 1282
hal_trng_lld.c
   PLATFORM TRNG subsystem low level driver source ... 1283
hal_trng_lld.h
   PLATFORM TRNG subsystem low level driver header ... 1284
hal_uart.c
   UART Driver code ...................................... 1284
hal_uart.h
   UART Driver macros and structures .................... 1285
hal_uart_lld.c
   PLATFORM UART subsystem low level driver source ... 1288
hal_uart_lld.h
   PLATFORM UART subsystem low level driver header ... 1288
hal_usb.c
   USB Driver code ...................................... 1289
hal_usb.h
   USB Driver macros and structures ..................... 1291
hal_usb_cdc.h
   USB CDC macros and structures ....................... 1294
hal_usb_lld.c
   PLATFORM USB subsystem low level driver source .... 1295
hal_usb_lld.h
   PLATFORM USB subsystem low level driver header .... 1297
hal_wdg.c
   WDG Driver code ...................................... 1299
hal_wdg.h
   WDG Driver macros and structures ..................... 1299
hal_wdg_lld.c
   WDG Driver subsystem low level driver source template ... 1300
hal_wdg_lld.h
   WDG Driver subsystem low level driver header template ... 1301
hal_wspi.c
   WSPI Driver code ...................................... 1301
hal_wspi.h
   WSPI Driver macros and structures .................... 1302
hal_wspi_lld.c
   PLATFORM WSPI subsystem low level driver source ... 1305
hal_wspi_lld.h
   PLATFORM WSPI subsystem low level driver header ... 1306
halconf.h
   HAL configuration header ................................ 1307
memstreams.c
   Memory streams code .................................... 1309
memstreams.h
   Memory streams structures and macros ................ 1310
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nullstreams.c</td>
<td>Null streams code</td>
<td>1310</td>
</tr>
<tr>
<td>nullstreams.h</td>
<td>Null streams structures and macros</td>
<td>1311</td>
</tr>
<tr>
<td>osal.c</td>
<td>OSAL module code</td>
<td>1311</td>
</tr>
<tr>
<td>osal.h</td>
<td>OSAL module header</td>
<td>1313</td>
</tr>
</tbody>
</table>
Chapter 7

Module Documentation

7.1 HAL Driver

Hardware Abstraction Layer.

7.1.1 Detailed Description

Hardware Abstraction Layer.

The HAL (Hardware Abstraction Layer) driver performs the system initialization and includes the platform support code shared by the other drivers. This driver does contain any API function except for a general initialization function `halInit()` that must be invoked before any HAL service can be used, usually the HAL initialization should be performed immediately before the kernel initialization.

Some HAL driver implementations also offer a custom early clock setup function that can be invoked before the C runtime initialization in order to accelerate the startup time.

Macros

- `#define __CHIBIOS_HAL__`
  
  ChibiOS/HAL identification macro.
- `#define CH_HAL_STABLE 1`
  
  Stable release flag.
- `#define HAL_LLD_SELECT_SPI_V2 TRUE`
  
  Requires use of SPIv2 driver model.
- `#define HAL_LLD_USE_CLOCK_MANAGEMENT`
  
  Specifies implementation of dynamic clock management.

ChibiOS/HAL version identification

- `#define CH_HAL_VERSION "8.4.0"`
  
  HAL version string.
- `#define CH_HAL_MAJOR 8`
  
  HAL version major number.
- `#define CH_HAL_MINOR 4`
  
  HAL version minor number.
- `#define CH_HAL_PATCH 0`
  
  HAL version patch number.
Return codes

- #define HAL_SUCCESS false
  HAL operation success.
- #define HAL_FAILED true
  HAL operation failed.

Return codes for HAL functions

- #define HAL_RET_SUCCESS MSG_OK
- #define HAL_RET_CONFIG_ERROR (msg_t)-16
  Configuration error.
- #define HAL_RET_NO_RESOURCE (msg_t)-17
  A required resource is not available.
- #define HAL_RET_HW_BUSY (msg_t)-18
  The peripheral is busy.
- #define HAL_RET_HW_FAILURE (msg_t)-19
  Peripheral failure.
- #define HAL_RET_UNKNOWN_CTL (msg_t)-20
  Unknown control code.

Platform identification macros

- #define PLATFORM_NAME "templates"

TypeDefs

- typedef unsigned halclkpt_t
  Type of a clock point identifier.
- typedef uint32_t halfreq_t
  Type of a clock point frequency in Hz.

Data Structures

- struct halclkcfg_t
  Type of a clock configuration structure.

Functions

- void halInit (void)
  HAL initialization.
- static bool halClockSwitchMode (const halclkcfg_t *ccp)
  Switches to a different clock configuration.
- static halfreq_t halClockGetPointX (halclkpt_t clkpt)
  Returns the frequency of a clock point in Hz.
- void hal_lld_init (void)
  Low level HAL driver initialization.
- bool hal_lld_clock_switch_mode (const halclkcfg_t *ccp)
  Switches to a different clock configuration.
- halfreq_t hal_lld_get_clock_point (halclkpt_t clkpt)
  Returns the frequency of a clock point in Hz.
7.1 HAL Driver

7.1.2 Macro Definition Documentation

7.1.2.1 __CHIBIOS_HAL__

#define __CHIBIOS_HAL__

ChibiOS/HAL identification macro.

7.1.2.2 CH_HAL_STABLE

#define CH_HAL_STABLE 1

Stable release flag.

7.1.2.3 CH_HAL_VERSION

#define CH_HAL_VERSION "8.4.0"

HAL version string.

7.1.2.4 CH_HAL_MAJOR

#define CH_HAL_MAJOR 8

HAL version major number.

7.1.2.5 CH_HAL_MINOR

#define CH_HAL_MINOR 4

HAL version minor number.
7.1.2.6 CH_HAL_PATCH

#define CH_HAL_PATCH 0

HAL version patch number.

7.1.2.7 HAL_SUCCESS

#define HAL_SUCCESS false

HAL operation success.

Deprecated

7.1.2.8 HAL_FAILED

#define HAL_FAILED true

HAL operation failed.

Deprecated

7.1.2.9 HAL_RET_CONFIG_ERROR

#define HAL_RET_CONFIG_ERROR (msg_t)-16

Configuration error.

An error has been detected in the driver configuration structure.

7.1.2.10 HAL_RET_NO_RESOURCE

#define HAL_RET_NO_RESOURCE (msg_t)-17

A required resource is not available.

One of the resources required for driver operations is not available.
7.1 HAL Driver

7.1.2.11 HAL_RET_HW_BUSY

#define HAL_RET_HW_BUSY (msg_t)-18

The peripheral is busy.
The peripheral is not available or taken by some other system actor.

7.1.2.12 HAL_RET_HW_FAILURE

#define HAL_RET_HW_FAILURE (msg_t)-19

Peripheral failure.
Peripheral failed, for example HW timeouts.

7.1.2.13 HAL_RET_UNKNOWN_CTL

#define HAL_RET_UNKNOWN_CTL (msg_t)-20

Unknown control code.

7.1.2.14 HAL_LLD_SELECT_SPI_V2

#define HAL_LLD_SELECT_SPI_V2 TRUE

Requires use of SPIv2 driver model.

7.1.2.15 HAL_LLD_USE_CLOCK_MANAGEMENT

#define HAL_LLD_USE_CLOCK_MANAGEMENT

Specifies implementation of dynamic clock management.

7.1.3 Typedef Documentation

7.1.3.1 halclkpt_t

typedef unsigned halclkpt_t

Type of a clock point identifier.
7.1.3.2 halfreq_t

typedef uint32_t halfreq_t

Type of a clock point frequency in Hz.

7.1.4 Function Documentation

7.1.4.1 halInit()

void halInit ( void )

HAL initialization.

This function invokes the low level initialization code then initializes all the drivers enabled in the HAL. Finally the board-specific initialization is performed by invoking boardInit() (usually defined in board.c).
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:
### 7.1.4.2 halClockSwitchMode()

```c
static bool halClockSwitchMode (const halclkcfg_t * ccp ) [inline], [static]
```

Switches to a different clock configuration.

**Parameters**

| in    | ccp | pointer to clock a halclkcfg_t structure |

**Returns**

The clock switch result.

**Return values**

| false | if the clock switch succeeded |
| true  | if the clock switch failed    |

**Function Class:**

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

![Call Graph](image)

### 7.1.4.3 halClockGetPointX()

```c
static halfreq_t halClockGetPointX (halclkpt_t clkpt ) [inline], [static]
```

Returns the frequency of a clock point in Hz.

**Parameters**

| in    | clkpt | clock point to be returned |

ChibiOS/HAL
7.1 HAL Driver

Returns

The clock point frequency in Hz or zero if the frequency is unknown.

Function Class:

This is an X-Class API, this function can be invoked from any context.

Here is the call graph for this function:

```
halClockGetPointX  hal_lld_get_clock_point
```

7.1.4.4 hal_lld_init()

```c
void hal_lld_init ( 
    void )
```

Low level HAL driver initialization.

Function Class:

Not an API, this function is for internal use only.

7.1.4.5 hal_lld_clock_switch_mode()

```c
bool hal_lld_clock_switch_mode ( 
    const halclkcfg_t * ccp )
```

Switches to a different clock configuration.

Parameters

| in | ccp | pointer to clock a halclkcfg_t structure |

Returns

The clock switch result.
Return values

| false | if the clock switch succeeded |
| true  | if the clock switch failed    |

Function Class:

Not an API, this function is for internal use only.

7.1.4.6 hal_lld_get_clock_point()

`hal_freq_t hal_lld_get_clock_point (halclkpt_t clkpt)`

Returns the frequency of a clock point in Hz.

Parameters

| in  | clkpt | clock point to be returned |

Returns

The clock point frequency in Hz or zero if the frequency is unknown.

Function Class:

Not an API, this function is for internal use only.
7.2 ADC Driver

Generic ADC Driver.

7.2.1 Detailed Description

Generic ADC Driver.

This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

Precondition

In order to use the ADC driver the `HAL_USE_ADC` option must be enabled in `halconf.h`.

7.2.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

7.2.3 ADC Operations

The ADC driver is quite complex, an explanation of the terminology and of the operational details follows.
7.2.3.1 ADC Conversion Groups

The ADCConversionGroup is the objects that specifies a physical conversion operation. This structure contains some standard fields and several implementation-dependent fields. The standard fields define the CG mode, the number of channels belonging to the CG and the optional callbacks. The implementation-dependent fields specify the physical ADC operation mode, the analog channels belonging to the group and any other implementation-specific setting. Usually the extra fields just mirror the physical ADC registers, please refer to the vendor's MCU Reference Manual for details about the available settings. Details are also available into the documentation of the ADC low level drivers and in the various sample applications.

7.2.3.2 ADC Conversion Modes

The driver supports several conversion modes:

- **One Shot**, the driver performs a single group conversion then stops.
- **Linear Buffer**, the driver performs a series of group conversions then stops. This mode is like a one shot conversion repeated N times, the buffer pointer increases after each conversion. The buffer is organized as an S(CG)×N samples matrix, when S(CG) is the conversion group size (number of channels) and N is the buffer depth (number of repeated conversions).
- **Circular Buffer**, much like the linear mode but the operation does not stop when the buffer is filled, it is automatically restarted with the buffer pointer wrapping back to the buffer base.

7.2.3.3 ADC Callbacks

The driver is able to invoke callbacks during the conversion process. A callback is invoked when the operation has been completed or, in circular mode, when the buffer has been filled and the operation is restarted. In circular mode a callback is also invoked when the buffer is half filled. The “half filled” and “filled” callbacks in circular mode allow to implement “streaming processing” of the sampled data, while the driver is busy filling one half of the buffer the application can process the other half, this allows for continuous interleaved operations.

The driver is not thread safe for performance reasons, if you need to access the ADC bus from multiple threads then use the `adcAcquireBus()` and `adcReleaseBus()` APIs in order to gain exclusive access.

**Macros**

- `#define adc_lld_driver_fields`
  
  Low level fields of the ADC driver structure.

- `#define adc_lld_config_fields`
  
  Low level fields of the ADC configuration structure.

- `#define adc_lld_configuration_group_fields`
  
  Low level fields of the ADC configuration structure.

**ADC configuration options**

- `#define ADC_USE_WAIT TRUE`
  
  Enables synchronous APIs.

- `#define ADC_USE_MUTUAL_EXCLUSION TRUE`
  
  Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.
Macro Functions

- `#define adcIsBufferComplete(adcp) ((bool)((adcp)->state == ADC_COMPLETE))`
  
  Buffer state.

Low level driver helper macros

- `#define _adc_reset_i(adcp) osalThreadResumeI(&(adcp)->thread, MSG_RESET)`
  
  Resumes a thread waiting for a conversion completion.

- `#define _adc_reset_s(adcp) osalThreadResumeS(&(adcp)->thread, MSG_RESET)`
  
  Resumes a thread waiting for a conversion completion.

- `#define _adc_wakeup_isr(adcp)`
  
  Wakes up the waiting thread.

- `#define _adc_timeout_isr(adcp)`
  
  Wakes up the waiting thread with a timeout message.

- `#define _adc_isr_half_code(adcp)`
  
  Common ISR code, half buffer event.

- `#define _adc_isr_full_code(adcp)`
  
  Common ISR code, full buffer event.

- `#define _adc_isr_error_code(adcp, err)`
  
  Common ISR code, error event.

Possible ADC errors mask bits.

- `#define ADC_ERR_DMAFAILURE 1U`
- `#define ADC_ERR_OVERFLOW 2U`
- `#define ADC_ERR_AWD 4U`

PLATFORM configuration options

- `#define PLATFORM_ADC_USE_ADC1 FALSE`
  
  ADC1 driver enable switch.

Typedefs

- `typedef struct hal_adc_driver ADCDriver`
  
  Type of a structure representing an ADC driver.

- `typedef struct hal_adc_config ADCCConfig`
  
  Type of a structure representing an ADC driver configuration.

- `typedef struct hal_adc_configuration_group ADCCConversionGroup`
  
  Conversion group configuration structure.

- `typedef void(∗adccallback_t)(ADCDriver ∗adcp)`
  
  Type of an ADC notification callback.

- `typedef void(∗adcerrorcallback_t)(ADCDriver ∗adcp, adcerror_t err)`
  
  Type of an ADC error callback.

- `typedef uint16_t adcsample_t`
  
  ADC sample data type.

- `typedef uint16_t adc_channels_num_t`
  
  Channels number in a conversion group.

- `typedef uint32_t adcerror_t`
  
  Type of an ADC error mask.
Data Structures

- struct hal_adc_configuration_group
  Conversion group configuration structure.
- struct hal_adc_config
  Driver configuration structure.
- struct hal_adc_driver
  Structure representing an ADC driver.

Functions

- void adcInit (void)
  ADC Driver initialization.
- void adcObjectInit (ADCDriver *adcp)
  Initializes the standard part of an ADC Driver structure.
- msg_t adcStart (ADCDriver *adcp, const ADCConfig *config)
  Configures and activates the ADC peripheral.
- void adcStop (ADCDriver *adcp)
  Deactivates the ADC peripheral.
- void adcStartConversion (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Starts an ADC conversion.
- void adcStartConversionI (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Starts an ADC conversion.
- void adcStopConversion (ADCDriver *adcp)
  Stops an ongoing conversion.
- void adcStopConversionI (ADCDriver *adcp)
  Stops an ongoing conversion.
- msg_t adcConvert (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Performs an ADC conversion.
- void adcAcquireBus (ADCDriver *adcp)
  Gains exclusive access to the ADC peripheral.
- void adcReleaseBus (ADCDriver *adcp)
  Releases exclusive access to the ADC peripheral.
- void adc_lld_init (void)
  Low level ADC driver initialization.
- void adc_lld_start (ADCDriver *adcp)
  Configures and activates the ADC peripheral.
- void adc_lld_stop (ADCDriver *adcp)
  Deactivates the ADC peripheral.
- void adc_lld_start_conversion (ADCDriver *adcp)
  Starts an ADC conversion.
- void adc_lld_stop_conversion (ADCDriver *adcp)
  Stops an ongoing conversion.

Enumerations

- enum adcstate_t {
  ADC_UNINIT = 0, ADC_STOP = 1, ADC_READY = 2, ADC_ACTIVE = 3,
  ADC_COMPLETE = 4, ADC_ERROR = 5
}
  Driver state machine possible states.
Variables

- **ADCDriver ADCD1**
  
  ADC1 driver identifier.

7.2.4 Macro Definition Documentation

7.2.4.1 ADC_USE_WAIT

```c
#define ADC_USE_WAIT TRUE
```

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

7.2.4.2 ADC_USE_MUTUAL_EXCLUSION

```c
#define ADC_USE_MUTUAL_EXCLUSION TRUE
```

Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

**Note**

Disabling this option saves both code and data space.

7.2.4.3 adcIsBufferComplete

```c
#define adcIsBufferComplete(
    adcp) ((bool)((adcp)->state == ADC_COMPLETE))
```

Buffer state.

**Note**

This function is meant to be called from the ADC callback only.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in adcp</strong></td>
<td>pointer to the ADCDriver object</td>
</tr>
</tbody>
</table>
Returns

The buffer state.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if the driver filled/sent the first half of the buffer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if the driver filled/sent the second half of the buffer.</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

7.2.4.4 _adc_reset_i

```c
#define _adc_reset_i(adcp) osalThreadResumeI(&(adcp)->thread, MSG_RESET)
```

Resumes a thread waiting for a conversion completion.

Parameters

| in  | adcp | pointer to the ADCDriver object |

Function Class:

Not an API, this function is for internal use only.

7.2.4.5 _adc_reset_s

```c
#define _adc_reset_s(adcp) osalThreadResumeS(&(adcp)->thread, MSG_RESET)
```

Resumes a thread waiting for a conversion completion.

Parameters

| in  | adcp | pointer to the ADCDriver object |

Function Class:

Not an API, this function is for internal use only.
7.2.4.6  _adc_wakeup_isr

#define _adc_wakeup_isr(adc)
{
  //
  osalSysLockFromISR();
  osalThreadResumeI(&(adc)->thread, MSG_OK);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread.

Parameters

| in | adc | pointer to the ADCDriver object |

Function Class:

Not an API, this function is for internal use only.

7.2.4.7  _adc_timeout_isr

#define _adc_timeout_isr(adc)
{
  //
  osalSysLockFromISR();
  osalThreadResumeI(&(adc)->thread, MSG_TIMEOUT);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread with a timeout message.

Parameters

| in | adc | pointer to the ADCDriver object |

Function Class:

Not an API, this function is for internal use only.

7.2.4.8  _adc_isr_half_code

#define _adc_isr_half_code(adc)

Value:
```c
if ((adcp)->grpp->end_cb != NULL) {
    (adcp)->grpp->end_cb(adcp);
}
```

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

Note
This macro is meant to be used in the low level drivers implementation only.

Parameters
```
in adc pointer to the ADCDriver object
```

Function Class:
Not an API, this function is for internal use only.

7.2.4.9 _adc_isr_full_code

```c
#define _adc_isr_full_code(adcp )
```

Value:
```c
if ((adcp)->grpp->circular) {
    /* Callback handling.*/
    if ((adcp)->grpp->end_cb != NULL) {
        (adcp)->state = ADC_COMPLETE;
        (adcp)->grpp->end_cb(adcp);
        if ((adcp)->state == ADC_COMPLETE) {
            (adcp)->state = ADC_ACTIVE;
        }
    }
    else {
        /* End conversion.*/
        adc_lld_stop_conversion(adcp);
        if ((adcp)->grpp->end_cb != NULL) {
            (adcp)->state = ADC_COMPLETE;
            (adcp)->grpp->end_cb(adcp);
            if ((adcp)->state == ADC_COMPLETE) {
                (adcp)->state = ADC_READY;
                (adcp)->grpp = NULL;
            }
            else {
                (adcp)->state = ADC_READY;
                (adcp)->grpp = NULL;
            }
        }
    }
    _adc_wakeup_isr(adcp);
}
```

Common ISR code, full buffer event.

This code handles the portable part of the ISR code:
7.2 ADC Driver

• Callback invocation.
• Waiting thread wakeup, if any.
• Driver state transitions.

Note
This macro is meant to be used in the low level drivers implementation only.

Parameters

\[
\text{in } \text{adcp} \quad \text{pointer to the ADCDriver object}
\]

Function Class:
Not an API, this function is for internal use only.

7.2.4.10 _adc_isr_error_code

#define _adc_isr_error_code(
    adcp,
    err)

Value:

    adc_lld_stop_conversion(adcp);
    \[
    \text{if } ((adcp)->grpp)->error_cb != NULL \{ \\
    (adcp)->state = ADC_ERROR;
    (adcp)->grpp->error_cb(adcp, err);
    \text{if } ((adcp)->state == ADC_ERROR) \{ \\
    (adcp)->state = ADC_READY;
    (adcp)->grpp = NULL;
    \}
    \text{else } \\
    (adcp)->state = ADC_READY;
    (adcp)->grpp = NULL;
    \}
    \_adc_timeout_isr(adcp);
    \]

Common ISR code, error event.

This code handles the portable part of the ISR code:

• Callback invocation.
• Waiting thread timeout signaling, if any.
• Driver state transitions.

Note
This macro is meant to be used in the low level drivers implementation only.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in adcp</td>
<td>pointer to the ADCDriver object</td>
</tr>
<tr>
<td>in err</td>
<td>platform dependent error code</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.2.4.11 ADC_ERR_DMAFAILURE

#define ADC_ERR_DMAFAILURE 1U

DMA operations failure.

7.2.4.12 ADC_ERR_OVERFLOW

#define ADC_ERR_OVERFLOW 2U

ADC overflow condition.

7.2.4.13 ADC_ERR_AWD

#define ADC_ERR_AWD 4U

Watchdog triggered.

7.2.4.14 PLATFORM_ADC_USE_ADC1

#define PLATFORM_ADC_USE_ADC1 FALSE

ADC1 driver enable switch.
If set to TRUE the support for ADC1 is included.

Note
The default is FALSE.
7.2 ADC Driver

7.2.4.15 adc_lld_driver_fields

#define adc_lld_driver_fields

Value:

    /**< Dummy field, it is not needed. */
    uint32_t    dummy

Low level fields of the ADC driver structure.

7.2.4.16 adc_lld_config_fields

#define adc_lld_config_fields

Value:

    /**< Dummy configuration, it is not needed. */
    uint32_t    dummy

Low level fields of the ADC configuration structure.

7.2.4.17 adc_lld_configuration_group_fields

#define adc_lld_configuration_group_fields

Value:

    /**< Dummy configuration, it is not needed. */
    uint32_t    dummy

Low level fields of the ADC configuration structure.

7.2.5 Typedef Documentation

7.2.5.1 ADCDriver

typedef struct hal_adc_driver ADCDriver

Type of a structure representing an ADC driver.

7.2.5.2 ADCConfig

typedef struct hal_adc_config ADCConfig

Type of a structure representing an ADC driver configuration.
7.2.5.3 ADCConversionGroup

typedef struct hal_adc_configuration_group ADCConversionGroup

Conversion group configuration structure.

This implementation-dependent structure describes a conversion operation.

**Note**

The use of this configuration structure requires knowledge of STM32 ADC cell registers interface, please refer to the STM32 reference manual for details.

7.2.5.4 adccallback_t

typedef void(* adccallback_t) (ADCDriver *adcp)

Type of an ADC notification callback.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adcp</code></td>
<td>pointer to the ADCDriver object triggering the callback</td>
</tr>
</tbody>
</table>

7.2.5.5 adcerrorcallback_t

typedef void(* adcerrorcallback_t) (ADCDriver *adcp, adcerror_t err)

Type of an ADC error callback.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adcp</code></td>
<td>pointer to the ADCDriver object triggering the callback</td>
</tr>
<tr>
<td><code>err</code></td>
<td>ADC error code</td>
</tr>
</tbody>
</table>

7.2.5.6 adcsample_t

typedef uint16_t adcsample_t

ADC sample data type.
7.2.5.7 adc_channels_num_t

typedef uint16_t adc_channels_num_t

Channels number in a conversion group.

7.2.5.8adcerror_t

typedef uint32_t adcerror_t

Type of an ADC error mask.

7.2.6 Enumeration Type Documentation

7.2.6.1 adcstate_t

defined

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>ADC_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>ADC_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>ADC_ACTIVE</td>
<td>Converting.</td>
</tr>
<tr>
<td>ADC_COMPLETE</td>
<td>Conversion complete.</td>
</tr>
<tr>
<td>ADC_ERROR</td>
<td>Conversion error.</td>
</tr>
</tbody>
</table>

7.2.7 Function Documentation

7.2.7.1 adcInit()

void adcInit ( void )

ADC Driver initialization.
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
adcInit → adc_lld_init → adcObjectInit
```

### 7.2.7.2 `adcObjectInit()`

```c
void adcObjectInit (ADCDriver *adcp)
```

Initializes the standard part of a `ADCDriver` structure.

**Parameters**

| Out | `adcp` | Pointer to the `ADCDriver` object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
adcObjectInit → osalMutexObjectInit
```
7.2 ADC Driver

7.2.7.3 adcStart()

```c
msg_t adcStart ( 
    ADCDriver * adcp, 
    const ADCConfig * config )
```

Configures and activates the ADC peripheral.

**Parameters**

| in | adcp | pointer to the ADCDriver object |
| in | config | pointer to the ADCConfig object. Depending on the implementation the value can be NULL. |

**Returns**

The operation status.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.2.7.4 adcStop()

```c
void adcStop ( 
    ADCDriver * adcp )
```

Deactivates the ADC peripheral.

**Parameters**

| in | adcp | pointer to the ADCDriver object |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
adcStop
  adc_lld_stop
  osalSysLock
  osalSysUnlock
```

### 7.2.7.5 adcStartConversion()

```c
void adcStartConversion (  
  ADCDriver * adcp,  
  const ADCConversionGroup * grpp,  
  adcsample_t * samples,  
  size_t depth )
```

Starts an ADC conversion.

Starts an asynchronous conversion operation.

**Note**

The buffer is organized as a matrix of M×N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>adcp</td>
<td>pointer to the ADCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>grpp</td>
<td>pointer to a ADCConversionGroup object</td>
</tr>
<tr>
<td>out</td>
<td>samples</td>
<td>pointer to the samples buffer</td>
</tr>
<tr>
<td>in</td>
<td>depth</td>
<td>buffer depth (matrix rows number). The buffer depth must be one or an even number.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

```
adcStartConversionI
  └── adcStartConversion
     └── osalSysLock

adcStartConversion
  └── osalSysUnlock

adc_lld_start_conversion
```

### 7.2.7.6 adcStartConversionI()

```c
void adcStartConversionI (ADCDriver * adcp, ADCConversionGroup * grpp, adcsample_t * samples, size_t depth )
```

Starts an ADC conversion.

Starts an asynchronous conversion operation.

**Postcondition**

The callbacks associated to the conversion group will be invoked on buffer fill and error events.

**Note**

The buffer is organized as a matrix of \( M \times N \) elements where \( M \) is the channels number configured into the conversion group and \( N \) is the buffer depth. The samples are sequentially written into the buffer with no gaps.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adcp</code></td>
<td>ADCDriver *</td>
<td>pointer to the ADCDriver object</td>
</tr>
<tr>
<td><code>grpp</code></td>
<td>const ADCConversionGroup *</td>
<td>pointer to a ADCConversionGroup object</td>
</tr>
<tr>
<td><code>samples</code></td>
<td>adcsample_t *</td>
<td>pointer to the samples buffer</td>
</tr>
<tr>
<td><code>depth</code></td>
<td>size_t</td>
<td>buffer depth (matrix rows number). The buffer depth must be one or an even number.</td>
</tr>
</tbody>
</table>

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
Here is the call graph for this function:

```
adcStartConversionI adc_lld_start_conversion
```

7.2.7.7 adcStopConversion()

```c
void adcStopConversion ( 
    ADCDriver * adcp )
```

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the ADC_READY state. If there was no conversion being processed then the function does nothing.

**Parameters**

```
in adcp pointer to the ADCDriver object
```

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
adcStopConversion adc_lld_stop_conversion
osalSysLock
osalSysUnlock
```
7.2.7.8  adcStopConversionI()

void adcStopConversionI (  
    ADCDriver * adcp  
)  

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the ADC_READY state. If there was no conversion being processed then the function does nothing.

Parameters

| in  | adcp | pointer to the ADCDriver object |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

7.2.7.9  adcConvert()

msg_t adcConvert (  
    ADCDriver * adcp,  
    const ADCConversionGroup * grpp,  
    adcsample_t * samples,  
    size_t depth  
)  

Performs an ADC conversion.

Performs a synchronous conversion operation.

Note

The buffer is organized as a matrix of M×N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>adcp</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>grpp</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>samples</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>depth</strong></td>
</tr>
</tbody>
</table>

Returns

The operation result.

Return values

| MSG_OK | Conversion finished. |
| MSG_RESET | The conversion has been stopped using acdStopConversion() or acdStopConversionI(), the result buffer may contain incorrect data. |
| MSG_TIMEOUT | The conversion has been stopped because an hardware error. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.2.7.10  adcAcquireBus()

```c
void adcAcquireBus (ADCDriver *adcp)
```

Gains exclusive access to the ADC peripheral.

This function tries to gain ownership to the ADC bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option ADC_USE_MUTUAL_EXCLUSION must be enabled.
7.2 ADC Driver

Parameters

\[ \text{in \ } adcp \ \text{pointer to the ADCDriver object} \]

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
adcAcquireBus \rightarrow osalMutexLock
```

7.2.7.11 adcReleaseBus()

```c
void adcReleaseBus (ADCDriver * adcp)
```

Releases exclusive access to the ADC peripheral.

Precondition

In order to use this function the option \texttt{ADC\_USE\_MUTUAL\_EXCLUSION} must be enabled.

Parameters

\[ \text{in \ } adcp \ \text{pointer to the ADCDriver object} \]

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
adcReleaseBus \rightarrow osalMutexUnlock
```
7.2.7.12 adc_lld_init()

void adc_lld_init (  
   void  )

Low level ADC driver initialization.

Function Class:  
   Not an API, this function is for internal use only.

Here is the call graph for this function:

7.2.7.13 adc_lld_start()

void adc_lld_start (  
   ADCDriver * adcp  )

Configures and activates the ADC peripheral.

Parameters  
   
in  adcp  pointer to the ADCDriver object

Function Class:  
   Not an API, this function is for internal use only.

7.2.7.14 adc_lld_stop()

void adc_lld_stop (  
   ADCDriver * adcp  )

Deactivates the ADC peripheral.
7.2 ADC Driver

Parameters

\[
\text{in } adcp \quad \text{pointer to the ADCDriver object}
\]

Function Class:

Not an API, this function is for internal use only.

7.2.7.15 adc_lld_start_conversion()

void adc_lld_start_conversion (ADCDriver * adcp)

Starts an ADC conversion.

Parameters

\[
\text{in } adcp \quad \text{pointer to the ADCDriver object}
\]

Function Class:

Not an API, this function is for internal use only.

7.2.7.16 adc_lld_stop_conversion()

void adc_lld_stop_conversion (ADCDriver * adcp)

Stops an ongoing conversion.

Parameters

\[
\text{in } adcp \quad \text{pointer to the ADCDriver object}
\]

Function Class:

Not an API, this function is for internal use only.

7.2.8 Variable Documentation
7.2.8.1 ADCD1

**ADCDriver ADCD1**

ADC1 driver identifier.
7.3 Buffered SIO Driver

Buffered SIO Driver.

Buffered SIO Driver.

This module implements circular buffers and events on top of a SIO driver. The behavior of the legacy Serial driver is emulated.
7.4 I/O Buffers Queues

7.4.1 Detailed Description

Buffers Queues are used when there is the need to exchange fixed-length data buffers between ISRs and threads. On the ISR side data can be exchanged only using buffers, on the thread side data can be exchanged both using buffers and/or using an emulation of regular byte queues. There are several kind of buffers queues:

- **Input queue**, unidirectional queue where the writer is the ISR side and the reader is the thread side.
- **Output queue**, unidirectional queue where the writer is the thread side and the reader is the ISR side.
- **Full duplex queue**, bidirectional queue. Full duplex queues are implemented by pairing an input queue and an output queue together.

**Macros**

- 
  ```c
  #define BUFFERS_CHUNKS_SIZE 64
  Maximum size of blocks copied in critical sections.
  ```
- 
  ```c
  #define BQ_BUFFER_SIZE(n, size) (((size_t)(size) + sizeof (size_t)) * (size_t)(n))
  Computes the size of a buffers queue buffer size.
  ```

**Macro Functions**

- 
  ```c
  #define bqSizeX(bqp) ((bqp)->bn)
  Returns the queue's number of buffers.
  ```
- 
  ```c
  #define bqSpaceI(bqp) ((bqp)->bcounter)
  Return the ready buffers number.
  ```
- 
  ```c
  #define bqGetLinkX(bqp) ((bqp)->link)
  Returns the queue application-defined link.
  ```
- 
  ```c
  #define bqSetLinkX(bqp, lk) ((bqp)->link = lk)
  Sets the queue application-defined link.
  ```
- 
  ```c
  #define bqlsSuspendedX(bqp) ((bqp)->suspended)
  Return the suspended state of the queue.
  ```
- 
  ```c
  #define bqSuspendI(bqp)
  Puts the queue in suspended state.
  ```
- 
  ```c
  #define bqResumeX(bqp)
  Resumes normal queue operations.
  ```
- 
  ```c
  #define ibqIsEmptyI(ibqp) ((bool)(bqSpaceI(ibqp) == 0U))
  Evaluates to true if the specified input buffers queue is empty.
  ```
- 
  ```c
  #define ibqIsFullI(ibqp) ((bool)(bqSpaceI(ibqp) == 0U))
  Evaluates to true if the specified input buffers queue is full.
  ```
- 
  ```c
  #define obqIsEmptyI(obqp)
  Evaluates to true if the specified output buffers queue is empty.
  ```
- 
  ```c
  #define obqIsFullI(obqp) ((bool)(bqSpaceI(obqp) == 0U))
  Evaluates to true if the specified output buffers queue is full.
  ```
7.4 I/O Buffers Queues

Typedefs

- typedef struct io_buffers_queue io_buffers_queue_t
  Type of a generic queue of buffers.
- typedef void (∗ bnotify_t) (io_buffers_queue_t ∗ bqp)
  Double buffer notification callback type.
- typedef io_buffers_queue_t input_buffers_queue_t
  Type of an input buffers queue.
- typedef io_buffers_queue_t output_buffers_queue_t
  Type of an output buffers queue.

Data Structures

- struct io_buffers_queue
  Structure of a generic buffers queue.

Functions

- void ibqObjectInit (input_buffers_queue_t ∗ ibqp, bool suspended, uint8_t ∗ bp, size_t size, size_t n, bnotify_t infy, void ∗ link)
  Initializes an input buffers queue object.
- void ibqResetI (input_buffers_queue_t ∗ ibqp)
  Resets an input buffers queue.
- uint8_t ∗ ibqGetEmptyBufferI (input_buffers_queue_t ∗ ibqp)
  Gets the next empty buffer from the queue.
- void ibqPostFullBufferI (input_buffers_queue_t ∗ ibqp, size_t size)
  Posts a new filled buffer to the queue.
- msg_t ibqGetFullBufferTimeout (input_buffers_queue_t ∗ ibqp, sysinterval_t timeout)
  Gets the next filled buffer from the queue.
- msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t ∗ ibqp, sysinterval_t timeout)
  Gets the next filled buffer from the queue.
- void ibqReleaseEmptyBuffer (input_buffers_queue_t ∗ ibqp)
  Releases the buffer back in the queue.
- void ibqReleaseEmptyBufferS (input_buffers_queue_t ∗ ibqp)
  Releases the buffer back in the queue.
- msg_t ibqGetTimeout (input_buffers_queue_t ∗ ibqp, sysinterval_t timeout)
  Input queue read with timeout.
- size_t ibqReadTimeout (input_buffers_queue_t ∗ ibqp, uint8_t ∗ bp, size_t n, sysinterval_t timeout)
  Input queue read with timeout.
- void obqObjectInit (outputBuffers_queue_t ∗ obqp, bool suspended, uint8_t ∗ bp, size_t size, size_t n, bnotify_t infy, void ∗ link)
  Initializes an output buffers queue object.
- void obqResetI (outputBuffers_queue_t ∗ obqp)
  Resets an output buffers queue.
- uint8_t ∗ obqGetFullBufferI (outputBuffers_queue_t ∗ obqp, size_t ∗ sizep)
  Gets the next filled buffer from the queue.
- void obqReleaseEmptyBufferI (outputBuffers_queue_t ∗ obqp)
  Releases the next filled buffer back in the queue.
- msg_t obqGetEmptyBufferTimeout (outputBuffers_queue_t ∗ obqp, sysinterval_t timeout)
  Gets the next empty buffer from the queue.
• msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t *obqp, sysinterval_t timeout)
  Gets the next empty buffer from the queue.
• void obqPostFullBuffer (output_buffers_queue_t *obqp, size_t size)
  Posts a new filled buffer to the queue.
• void obqPostFullBufferS (output_buffers_queue_t *obqp, size_t size)
  Posts a new filled buffer to the queue.
• msg_t obqPutTimeout (output_buffers_queue_t *obqp, uint8_t b, sysinterval_t timeout)
  Output queue write with timeout.
• size_t obqWriteTimeout (output_buffers_queue_t *obqp, const uint8_t *bp, size_t n, sysinterval_t timeout)
  Output queue write with timeout.
• bool obqTryFlushI (output_buffers_queue_t *obqp)
  Flushes the current, partially filled, buffer to the queue.
• void obqFlush (output_buffers_queue_t *obqp)
  Flushes the current, partially filled, buffer to the queue.

7.4.2 Macro Definition Documentation

7.4.2.1 BUFFERS_CHUNKS_SIZE

#define BUFFERS_CHUNKS_SIZE 64

Maximum size of blocks copied in critical sections.

Note
Increasing this value increases performance at expense of IRQ servicing efficiency.
It must be a power of two.

7.4.2.2 BQ_BUFFER_SIZE

#define BQ_BUFFER_SIZE( 
  n, 
  size ) (((size_t)(size) + sizeof (size_t)) * (size_t)(n))

Computes the size of a buffers queue buffer size.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>n</th>
<th>number of buffers in the queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of the buffers</td>
</tr>
</tbody>
</table>
7.4.2.3 bqSizeX

#define bqSizeX(bqp) ((bqp)->bn)

Returns the queue's number of buffers.

Parameters

in bqp pointer to an io_buffers_queue_t structure

Returns

The number of buffers.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.4.2.4 bqSpaceI

#define bqSpaceI(bqp) ((bqp)->bcounter)

Return the ready buffers number.

Returns the number of filled buffers if used on an input queue or the number of empty buffers if used on an output queue.

Parameters

in bqp pointer to an io_buffers_queue_t structure

Returns

The number of ready buffers.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.4.2.5 bqGetLinkX

#define bqGetLinkX(bqp) ((bqp)->link)

Returns the queue application-defined link.
Parameters

| in  | bqp | pointer to an io_buffers_queue_t structure |

Returns

The application-defined link.

Function Class:

Special function, this function has special requirements see the notes.

### 7.4.2.6 bqSetLinkX

```c
#define bqSetLinkX(bqp, lk) ((bqp)->link = lk)
```

Sets the queue application-defined link.

Parameters

| in  | bqp | pointer to an io_buffers_queue_t structure |
| in  | lk  | The application-defined link. |

Function Class:

Special function, this function has special requirements see the notes.

### 7.4.2.7 bqIsSuspendedX

```c
#define bqIsSuspendedX(bqp) ((bqp)->suspended)
```

Return the suspended state of the queue.

Parameters

| in  | bqp | pointer to an io_buffers_queue_t structure |

Returns

The suspended state.
Return values

| **false** | if blocking access to the queue is enabled. |
| **true**  | if blocking access to the queue is suspended. |

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

### 7.4.2.8 bqSuspendI

```c
#define bqSuspendI(bqp)
```

**Value:**

```c
{ (bqp)->suspended = true; \
  osalThreadDequeueAll(&(bqp)->waiting, MSG_RESET); \
}
```

Puts the queue in suspended state.

When the queue is put in suspended state all waiting threads are woken with message MSG_RESET and subsequent attempt at waiting on the queue will result in an immediate return with MSG_RESET message.

**Note**

The content of the queue is not altered, queues can be accessed is suspended state until a blocking operation is met then a MSG_RESET occurs.

**Parameters**

- **in** `bqp` pointer to an `io_buffers_queue_t` structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.4.2.9 bqResumeX

```c
#define bqResumeX(bqp)
```

**Value:**

```c
{ (bqp)->suspended = false; 
}
```

Resumes normal queue operations.
Parameters

in  ibqp  pointer to an io_buffers_queue_t structure

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.4.2.10  ibqIsEmptyI

#define ibqIsEmptyI(  
  ibqp )  ((bool)(bqSpaceI(ibqp) == 0U))

Evaluates to true if the specified input buffers queue is empty.

Parameters

in  ibqp  pointer to an input_buffers_queue_t structure

Returns

The queue status.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if the queue is not empty.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if the queue is empty.</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.4.2.11  ibqIsFullI

#define ibqIsFullI(  
  ibqp )

Value:

/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/  
{(bool)(((ibqp)->bwrptr == (ibqp)->brdptr) && ((ibqp)->bcounter != 0U))}  
/*lint -restore*/

Evaluates to true if the specified input buffers queue is full.
Parameters

| in  | ibqp | pointer to an input_buffers_queue_t structure |

Returns

The queue status.

Return values

| false | if the queue is not full. |
| true  | if the queue is full. |

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.4.2.12 obqIsEmptyI

```c
#define obqIsEmptyI( obqp )
```

Value:

```c
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/
{(bool)((obqp)->bwrptr == (obqp)->brdptr) && ((obqp)->bcounter != 0U))
/*lint -restore*/
```

Evaluates to **true** if the specified output buffers queue is empty.

Parameters

| in  | obqp | pointer to an outputBuffers_queue_t structure |

Returns

The queue status.

Return values

| false | if the queue is not empty. |
| true  | if the queue is empty. |

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
### 7.4.2.13 obqIsFullI

```c
#define obqIsFullI( obqp ) ((bool)(bqSpaceI(obqp) == 0U))
```

Evaluates to `true` if the specified output buffers queue is full.

**Parameters**

```
in  obqp  pointer to an output_buffers_queue_t structure
```

**Returns**

The queue status.

**Return values**

- `false` if the queue is not full.
- `true` if the queue is full.

**Function Class:**

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.4.3 Typedef Documentation

#### 7.4.3.1 io_buffers_queue_t

```c
typedef struct io_buffers_queue io_buffers_queue_t
```

Type of a generic queue of buffers.

#### 7.4.3.2 bqnotify_t

```c
typedef void(* bqnotify_t)( io_buffers_queue_t *bqp)
```

Double buffer notification callback type.
7.4 I/O Buffers Queues

Parameters

| in  | iodbp            | the buffers queue pointer |

7.4.3.3 input_buffers_queue_t

typedef io_buffers_queue_t input_buffers_queue_t

Type of an input buffers queue.

7.4.3.4 output_buffers_queue_t

typedef io_buffers_queue_t output_buffers_queue_t

Type of an output buffers queue.

7.4.4 Function Documentation

7.4.4.1 ibqObjectInit()

```
void ibqObjectInit (  
    input_buffers_queue_t * ibqp,
    bool suspended,
    uint8_t * bp,
    size_t size,
    size_t n,
    bqnotify_t infy,
    void * link )
```

Initializes an input buffers queue object.

Parameters

| out | ibqp           | pointer to the input_buffers_queue_t object |
| in  | suspended      | initial state of the queue                 |
| in  | bp             | pointer to a memory area allocated for buffers |
| in  | size           | buffers size                                |
| in  | n              | number of buffers                           |
| in  | infy           | callback called when a buffer is returned to the queue |
| in  | link           | application defined pointer                 |
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
ibqObjectInit  osalThreadQueueObjectInit
```

### 7.4.4.2 ibqResetI()

```c
void ibqResetI (
    input_buffers_queue_t *ibqp )
```

Resets an input buffers queue.

All the data in the input buffers queue is erased and lost, any waiting thread is resumed with status MSG_RESET.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

Parameters

```
in  ibqp  pointer to the input_buffers_queue_t object
```

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
ibqResetI  osalThreadDequeueAll
```
7.4.4.3  ibqGetEmptyBufferI()

```c
uint8_t * ibqGetEmptyBufferI ( input_buffers_queue_t * ibqp )
```

Gets the next empty buffer from the queue.

**Note**

The function always returns the same buffer if called repeatedly.

**Parameters**

| in | ibqp | pointer to the input_buffers_queue_t object |

**Returns**

A pointer to the next buffer to be filled.

**Return values**

| NULL | if the queue is full. |

**Function Class:**

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.4.4.4  ibqPostFullBufferI()

```c
void ibqPostFullBufferI ( input_buffers_queue_t * ibqp, size_t size )
```

Posts a new filled buffer to the queue.

**Parameters**

| in | ibqp | pointer to the input_buffers_queue_t object |
| in | size | used size of the buffer, cannot be zero |

**Function Class:**

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
Here is the call graph for this function:

ibqPostFullBufferI → osalThreadDequeueNextI

7.4.4.5 ibqGetFullBufferTimeout()

```c
msg_t ibqGetFullBufferTimeout (input_buffers_queue_t * ibqp,
                                sysinterval_t timeout)
```

Gets the next filled buffer from the queue.

Note
The function always acquires the same buffer if called repeatedly.

Postcondition
After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ibqp</th>
<th>pointer to the <code>input_buffers_queue_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

| MSG_OK | if a buffer has been acquired. |
| MSG_TIMEOUT | if the specified time expired. |
| MSG_RESET | if the queue has been reset or has been put in suspended state. |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.4.4.6 `ibqGetFullBufferTimeoutS()`

```c
msg_t ibqGetFullBufferTimeoutS (
    input_buffers_queue_t *ibqp,
    sysinterval_t timeout )
```

Gets the next filled buffer from the queue.

**Note**

The function always acquires the same buffer if called repeatedly.

**Postcondition**

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>ibqp</code></th>
<th>pointer to the <code>input_buffers_queue_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
</tbody>
</table>

- `TIME_IMMEDIATE` immediate timeout.
- `TIME_INFINITE` no timeout.
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if a buffer has been acquired.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the specified time expired.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the queue has been reset or has been put in suspended state.</td>
</tr>
</tbody>
</table>

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

### 7.4.4.7 ibqReleaseEmptyBuffer()

```c
void ibqReleaseEmptyBuffer (input_buffers_queue_t * ibqp)
```

Releases the buffer back in the queue.

**Note**

The object callback is called after releasing the buffer.

**Parameters**

| in ibqp | pointer to the input_buffers_queue_t object |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.4.4.8  ibqReleaseEmptyBufferS()

```c
void ibqReleaseEmptyBufferS ( 
    inputBuffersQueue_t * ibqp )
```
Releases the buffer back in the queue.

Note
The object callback is called after releasing the buffer.

Parameters
- **in ibqp** pointer to the `inputBuffersQueue_t` object

Function Class:
This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

7.4.4.9  ibqGetTimeout()

```c
msg_t ibqGetTimeout ( 
    inputBuffersQueue_t * ibqp, 
    sysInterval_t timeout )
```
Input queue read with timeout.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a new buffer arrives in the queue or a timeout occurs.

Parameters
- **in ibqp** pointer to the `inputBuffersQueue_t` object
- **in timeout** the number of ticks before the operation timeouts, the following special values are allowed:
  - `TIME_IMMEDIATE` immediate timeout.
  - `TIME_INFINITE` no timeout.

Returns
A byte value from the queue.
Return values

| MSG_TIMEOUT  | if the specified time expired. |
| MSG_RESET    | if the queue has been reset or has been put in suspended state. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
ibqGetTimeout
  ↓                  ↓
  ibqGetFullBufferTimeoutS
  ↓                  ↓
osalSysLock
```

### 7.4.4.10 ibqReadTimeout()

```c
size_t ibqReadTimeout ( 
  input_buffers_queue_t * ibqp, 
  uint8_t * bp, 
  size_t n, 
  sysinterval_t timeout )
```

Input queue read with timeout.

The function reads data from an input queue into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ibqp</th>
<th>pointer to the <code>input_buffers_queue_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

Returns
The number of bytes effectively transferred.

Return values

| 0  | if a timeout occurred. |

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
ibqReadTimeout
ibqGetFullBufferTimeoutS
osalSysLock
```

### 7.4.4.11 obqObjectInit()

```c
void obqObjectInit (  
    output_buffers_queue_t * obqp,  
    bool suspended,  
    uint8_t * bp,  
    size_t size,  
    size_t n,  
    bqnotify_t onfy,  
    void * link )
```

Initializes an output buffers queue object.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>obqp</th>
<th>pointer to the output_buffers_queue_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>suspended</td>
<td>initial state of the queue</td>
</tr>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to a memory area allocated for buffers</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>buffers size</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of buffers</td>
</tr>
<tr>
<td>in</td>
<td>onfy</td>
<td>callback called when a buffer is posted in the queue</td>
</tr>
<tr>
<td>in</td>
<td>link</td>
<td>application defined pointer</td>
</tr>
</tbody>
</table>
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
obqObjectInit → osalThreadQueueObjectInit
```

### 7.4.4.12 obqResetI()

```c
void obqResetI ( 
    output_buffers_queue_t * obqp )
```

Resets an output buffers queue.

All the data in the output buffers queue is erased and lost, any waiting thread is resumed with status MSG_RESET.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

**Parameters**

```c
in    obqp    pointer to the output_buffers_queue_t object
```

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
obqResetI → osalThreadDequeueAll
```
7.4 I/O Buffers Queues

7.4.4.13 obqGetFullBuffer()

```c
uint8_t * obqGetFullBufferI ( outputBuffers_queue_t * obqp, size_t * sizep )
```

Gets the next filled buffer from the queue.

Note

The function always returns the same buffer if called repeatedly.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>obqp</th>
<th>pointer to the outputBuffers_queue_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sizep</td>
<td>pointer to the filled buffer size</td>
</tr>
</tbody>
</table>

Returns

A pointer to the filled buffer.

Return values

`NULL` if the queue is empty.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.4.4.14 obqReleaseEmptyBuffer()

```c
void obqReleaseEmptyBufferI ( outputBuffers_queue_t * obqp )
```

Releases the next filled buffer back in the queue.

Parameters

| in  | obqp | pointer to the outputBuffers_queue_t object |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
Here is the call graph for this function:

![Call Graph]

7.4.4.15  **obqGetEmptyBufferTimeout()**

```c
define msg_t obqGetEmptyBufferTimeout (  
    output_buffers_queue_t ∗ obqp,  
    sysinterval_t timeout  
)
```

Gets the next empty buffer from the queue.

**Note**

The function always acquires the same buffer if called repeatedly.

**Postcondition**

After calling the function the fields *ptr* and *top* are set at beginning and end of the buffer data or NULL if the queue is empty.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in obqp</td>
<td>pointer to the <a href="#">output_buffers_queue_t</a> object</td>
</tr>
<tr>
<td>in timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td><code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td><code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if a buffer has been acquired.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the specified time expired.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the queue has been reset or has been put in suspended state.</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.4.4.16  `obqGetEmptyBufferTimeoutS()`

```c
msg_t obqGetEmptyBufferTimeoutS ( 
    output_buffers_queue_t * obqp, 
    sysinterval_t timeout )
```

Gets the next empty buffer from the queue.

**Note**

The function always acquires the same buffer if called repeatedly.

**Postcondition**

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>obqp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if a buffer has been acquired.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the specified time expired.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the queue has been reset or has been put in suspended state.</td>
</tr>
</tbody>
</table>

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

### 7.4.4.17 obqPostFullBuffer()

```c
void obqPostFullBuffer (      
    output_buffers_queue_t * obqp,  
    size_t size )
```

Posts a new filled buffer to the queue.

**Note**

The object callback is called after releasing the buffer.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>obqp</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.4 I/O Buffers Queues

Here is the call graph for this function:

![Call Graph Diagram]

7.4.4.18 obqPostFullBufferS()

```c
void obqPostFullBufferS ( 
    output_buffers_queue_t * obqp, 
    size_t size )
```

Posts a new filled buffer to the queue.

**Note**
The object callback is called after releasing the buffer.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>obqp</th>
<th>pointer to the <code>output_buffers_queue_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>used size of the buffer, cannot be zero</td>
</tr>
</tbody>
</table>

**Function Class:**

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

7.4.4.19 obqPutTimeout()

```c
msg_t obqPutTimeout ( 
    output_buffers_queue_t * obqp, 
    uint8_t b, 
    sysinterval_t timeout )
```

Output queue write with timeout.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until a new buffer is freed in the queue or a timeout occurs.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>obqp</th>
<th>pointer to the <code>output_buffers_queue_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>byte value to be transferred</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

Returns

A byte value from the queue.

Return values

| MSG_TIMEOUT | if the specified time expired. |
| MSG_RESET   | if the queue has been reset or has been put in suspended state. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    obqPutTimeout
   /       
  |        |
  |        |
  |        |
osalSysLock   obqGetEmptyBufferTimeoutS
```

7.4.4.20 obqWriteTimeout()

```c
size_t obqWriteTimeout ( 
    output_buffers_queue_t * obqp, 
    const uint8_t * bp, 
    size_t n, 
    sysinterval_t timeout )
```

Output queue write with timeout.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>obqp</code></td>
<td>pointer to the <code>output_buffers_queue_t</code> object</td>
</tr>
<tr>
<td><code>bp</code></td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td><code>n</code></td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

### Returns

The number of bytes effectively transferred.

### Return values

- `0` if a timeout occurred.

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.4.4.21 `obqTryFlushI()`

```c
bool obqTryFlushI ( 
    output_buffers_queue_t * obqp )
```

Flushes the current, partially filled, buffer to the queue.

### Note

The notification callback is not invoked because the function is meant to be called from ISR context. An operation status is returned instead.
Parameters

| in | obqp | pointer to the output_buffers_queue_t object |

Returns

The operation status.

Return values

| false | if no new filled buffer has been posted to the queue. |
| true  | if a new filled buffer has been posted to the queue. |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.4.422 obqFlush()

void obqFlush (output_buffers_queue_t * obqp )

Flushes the current, partially filled, buffer to the queue.

Parameters

| in | obqp | pointer to the output_buffers_queue_t object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.5 CAN Driver

Generic CAN Driver.

7.5.1 Detailed Description

Generic CAN Driver.

This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

Precondition

In order to use the CAN driver the HAL_USE_CAN option must be enabled in halconf.h.

7.5.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

![State Machine Diagram]

Macros

- #define CAN_ANY_MAILBOX 0U
  Special mailbox identifier.
- #define CAN_TX_MAILBOXES 1
  Number of transmit mailboxes.
- #define CAN_RX_MAILBOXES 1
  Number of receive mailboxes.
CAN status flags

- #define CAN_LIMIT_WARNING 1U
  Errors rate warning.
- #define CAN_LIMIT_ERROR 2U
  Errors rate error.
- #define CAN_BUS_OFF_ERROR 4U
  Bus off condition reached.
- #define CAN_FRAMING_ERROR 8U
  Framing error of some kind on the CAN bus.
- #define CAN_OVERFLOW_ERROR 16U
  Overflow in receive queue.

CAN configuration options

- #define CAN_USE_SLEEP_MODE TRUE
  Sleep mode related APIs inclusion switch.
- #define CAN_ENFORCE_USE_CALLBACKS FALSE
  Enforces the driver to use direct callbacks rather than OSAL events.

Macro Functions

- #define CAN_MAILBOX_TO_MASK(mbx) (1U << ((mbx) - 1U))
  Converts a mailbox index to a bit mask.
- #define canTransmit(canp, mailbox, ctfp, timeout) canTransmitTimeout(canp, mailbox, ctfp, timeout)
  Legacy name for canTransmitTimeout().
- #define canReceive(canp, mailbox, crfp, timeout) canReceiveTimeout(canp, mailbox, crfp, timeout)
  Legacy name for canReceiveTimeout().

Low level driver helper macros

- #define _can_tx_empty_isr(canp, flags)
  TX mailbox empty event.
- #define _can_rx_full_isr(canp, flags)
  RX mailbox empty full event.
- #define _can_wakeup_isr(canp)
  Wakeup event.
- #define _can_error_isr(canp, flags)
  Error event.

PLATFORM configuration options

- #define PLATFORM_CAN_USE_CAN1 FALSE
  CAN1 driver enable switch.
7.5 CAN Driver

Typedefs

- typedef struct hal_can_driver CANDriver
  * Type of a structure representing an CAN driver.
- typedef uint32_t canmbx_t
  * Type of a transmission mailbox index.
- typedef void(* can_callback_t) (CANDriver *canp, uint32_t flags)
  * Type of a CAN notification callback.
- typedef struct hal_can_config CANConfig
  * Type of a CAN configuration structure.

Data Structures

- struct CANTxFrame
  * CAN transmission frame.
- struct CANRxFrame
  * CAN received frame.
- struct hal_can_config
  * Type of a CAN configuration structure.
- struct hal_can_driver
  * Structure representing an CAN driver.

Functions

- void canInit (void)
  * CAN Driver initialization.
- void canObjectInit (CANDriver *canp)
  * Initializes the standard part of a CANDriver structure.
- msg_t canStart (CANDriver *canp, const CANConfig *config)
  * Configures and activates the CAN peripheral.
- void canStop (CANDriver *canp)
  * Deactivates the CAN peripheral.
- bool canTryTransmitI (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
  * Can frame transmission attempt.
- bool canTryReceiveI (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
  * Can frame receive attempt.
- void canTryAbortX (CANDriver *canp, canmbx_t mailbox)
  * Tries to abort an ongoing transmission.
- msg_t canTransmitTimeout (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, sysinterval_t timeout)
  * Can frame transmission.
- msg_t canReceiveTimeout (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, sysinterval_t timeout)
  * Can frame receive.
- void canSleep (CANDriver *canp)
  * Enters the sleep mode.
- void canWakeup (CANDriver *canp)
  * Enforces leaving the sleep mode.
- void can_lld_init (void)
  * Low level CAN driver initialization.
- void can_lld_start (CANDriver *canp)

ChibiOS/HAL
Configures and activates the CAN peripheral.

- **void can_lld_stop (CANDriver *canp)**
  Deactivates the CAN peripheral.

- **bool can_lld_is_tx_empty (CANDriver *canp, canmbx_t mailbox)**
  Determines whether a frame can be transmitted.

- **void can_lld_transmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)**
  Inserts a frame into the transmit queue.

- **bool can_lld_is_rx_nonempty (CANDriver *canp, canmbx_t mailbox)**
  Determines whether a frame has been received.

- **void can_lld_receive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)**
  Receives a frame from the input queue.

- **void can_lld_abort (CANDriver *canp, canmbx_t mailbox)**
  Tries to abort an ongoing transmission.

- **void can_lld_sleep (CANDriver *canp)**
  Enters the sleep mode.

- **void can_lld_wakeup (CANDriver *canp)**
  Enforces leaving the sleep mode.

### Enumerations

- **enum canstate_t {**
  CAN_UNINIT = 0, CAN_STOP = 1, CAN_STARTING = 2, CAN_STOPPING = 3, CAN_READY = 4, CAN_SLEEP = 5
  **}**
  Driver state machine possible states.

### Variables

- **CANDriver CAN1**
  CAN1 driver identifier.

### 7.5.3 Macro Definition Documentation

#### 7.5.3.1 CAN_LIMIT_WARNING

`#define CAN_LIMIT_WARNING 1U`

Errors rate warning.

#### 7.5.3.2 CAN_LIMIT_ERROR

`#define CAN_LIMIT_ERROR 2U`

Errors rate error.
7.5.3.3  CAN_BUS_OFF_ERROR

#define CAN_BUS_OFF_ERROR 4U

Bus off condition reached.

7.5.3.4  CAN_FRAMING_ERROR

#define CAN_FRAMING_ERROR 8U

Framing error of some kind on the CAN bus.

7.5.3.5  CAN_OVERFLOW_ERROR

#define CAN_OVERFLOW_ERROR 16U

Overflow in receive queue.

7.5.3.6  CAN_ANY_MAILBOX

#define CAN_ANY_MAILBOX 0U

Special mailbox identifier.

7.5.3.7  CAN_USE_SLEEP_MODE

#define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

This option can only be enabled if the CAN implementation supports the sleep mode, see the macro CAN_SUPP← ORTS_SLEEP exported by the underlying implementation.

7.5.3.8  CAN_ENFORCE_USE_CALLBACKS

#define CAN_ENFORCE_USE_CALLBACKS FALSE

Enforces the driver to use direct callbacks rather than OSAL events.
7.5.3.9 CAN_MAILBOX_TO_MASK

#define CAN_MAILBOX_TO_MASK(
    mbx ) (1U << ((mbx) - 1U))

Converts a mailbox index to a bit mask.

7.5.3.10 canTransmit

#define canTransmit(
    canp, mailbox, ctfp, timeout )
    canTransmitTimeout(canp, mailbox, ctfp, timeout)

Legacy name for canTransmitTimeout().

Deprecated

7.5.3.11 canReceive

#define canReceive(
    canp, mailbox, crfp, timeout )
    canReceiveTimeout(canp, mailbox, crfp, timeout)

Legacy name for canReceiveTimeout().

Deprecated

7.5.3.12 _can_tx_empty_isr

#define _can_tx_empty_isr(
    canp, flags )

Value:

\osalSysLockFromISR();
osalThreadDequeueAll(&(canp)->txqueue, MSG_OK);
osalEventBroadcastFlags(&(canp)->txempty_event, flags);
osalSysUnlockFromISR();

TX mailbox empty event.
7.5.3.13 _can_rx_full_isr

#define _can_rx_full_isr(
    canp,
    flags )

Value:
{
    \
    \ osalSysLockFromISR();
    \ osalThreadDequeueAllI(&(canp)->rxqueue, MSG_OK);
    \ osalEventBroadcastFlagsI(&(canp)->rxfull_event, flags);
    \ osalSysUnlockFromISR();
}

RX mailbox empty full event.

7.5.3.14 _can_wakeup_isr

#define _can_wakeup_isr(
    canp )

Value:
{
    \
    osalSysLockFromISR();
    osalEventBroadcastFlagsI(&(canp)->wakeup_event, 0U);
    osalSysUnlockFromISR();
}

Wakeup event.

7.5.3.15 _can_error_isr

#define _can_error_isr(
    canp,
    flags )

Value:
{
    \
    osalSysLockFromISR();
    osalEventBroadcastFlagsI(&(canp)->error_event, flags);
    osalSysUnlockFromISR();
}

Error event.

7.5.3.16 CAN_TX_MAILBOXES

#define CAN_TX_MAILBOXES 1

Number of transmit mailboxes.
7.5.3.17 CAN_RX_MAILBOXES

#define CAN_RX_MAILBOXES 1

Number of receive mailboxes.

7.5.3.18 PLATFORM_CAN_USE_CAN1

#define PLATFORM_CAN_USE_CAN1 FALSE

CAN1 driver enable switch.

If set to TRUE the support for CAN1 is included.

Note

The default is FALSE.

7.5.4 Typedef Documentation

7.5.4.1 CANDriver

typedef struct hal_can_driver CANDriver

Type of a structure representing an CAN driver.

7.5.4.2 canmbx_t

typedef uint32_t canmbx_t

Type of a transmission mailbox index.

7.5.4.3 can_callback_t

typedef void(* can_callback_t) (CANDriver *canp, uint32_t flags)

Type of a CAN notification callback.
7.5 CAN Driver

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>canp</td>
<td>pointer to the CANDriver object triggering the callback</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>flags associated to the mailbox callback</td>
</tr>
</tbody>
</table>

7.5.4.4 CANConfig

typedef struct hal_can_config CANConfig

Type of a CAN configuration structure.

7.5.5 Enumeration Type Documentation

7.5.5.1 canstate_t

enum canstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>CAN_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>CAN.STARTING</td>
<td>Starting.</td>
</tr>
<tr>
<td>CAN_STOPPING</td>
<td>Stopping.</td>
</tr>
<tr>
<td>CAN_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>CAN_SLEEP</td>
<td>Sleep state.</td>
</tr>
</tbody>
</table>

7.5.6 Function Documentation

7.5.6.1 canInit()

void canInit ( void )

CAN Driver initialization.
Note
This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
canInit → can_lld_init → canObjectInit
```

### 7.5.6.2 canObjectInit()

```c
void canObjectInit (  
    CANDriver * canp )
```

Initializes the standard part of a `CANDriver` structure.

**Parameters**
- `canp` pointer to the `CANDriver` object

**Function Class:**
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
canObjectInit → osalEventObjectInit → osalThreadQueueObjectInit
```
### 7.5.6.3 canStart()

```c
msg_t canStart (  
    CANDriver * canp,  
    const CANConfig * config )
```

Configures and activates the CAN peripheral.

**Note**

Activating the CAN bus can be a slow operation.
Unlike other drivers it is not possible to restart the CAN driver without first stopping it using `canStop()`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the CANConfig object. Depending on the implementation the value can be NULL.</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    canStart
      `-- can_lld_start
        `- osalSysLock
```

### 7.5.6.4 canStop()

```c
void canStop (  
    CANDriver * canp )
```

Deactivates the CAN peripheral.
Parameters

| in | canp | pointer to the CANDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    can_lld_stop
     /  
canStop       osalSysLock
      /  
osalThreadDequeueAll
```

7.5.6.5 canTryTransmitI()

```c
bool canTryTransmitI (  
    CANDriver * canp,  
    canmbx_t mailbox,  
    const CANTxFrame * ctfp )
```

Can frame transmission attempt.

The specified frame is queued for transmission, if the hardware queue is full then the function fails.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
<tr>
<td>in</td>
<td>ctfp</td>
<td>pointer to the CAN frame to be transmitted</td>
</tr>
</tbody>
</table>

Returns

The operation result.
Return values

<table>
<thead>
<tr>
<th>false</th>
<th>Frame transmitted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Mailbox full.</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
canTryTransmit
  can_lld_is_tx_empty
  can_lld_transmit
```

7.5.6.6 canTryReceiveI()

```c
bool canTryReceiveI (
    CANDriver * canp,
    canmbx_t mailbox,
    CANRxFrame * crfp )
```

Can frame receive attempt.

The function tries to fetch a frame from a mailbox.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
<tr>
<td>out</td>
<td>crfp</td>
<td>pointer to the buffer where the CAN frame is copied</td>
</tr>
</tbody>
</table>

Returns

The operation result.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>Frame fetched.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Mailbox empty</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

![Call Graph]

7.5.6.7 canTryAbortX()

```c
void canTryAbortX ( 
    CANDriver * canp, 
    canmbx_t mailbox )
```

Tries to abort an ongoing transmission.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number</td>
</tr>
</tbody>
</table>

Function Class:

This is an X-Class API, this function can be invoked from any context.

Here is the call graph for this function:

![Call Graph]
Can frame transmission.

The specified frame is queued for transmission, if the hardware queue is full then the invoking thread is queued.

Note

Trying to transmit while in sleep mode simply enqueues the thread.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
<tr>
<td>in</td>
<td>ctfp</td>
<td>pointer to the CAN frame to be transmitted</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
</tbody>
</table>

- TIME_IMMEDIATE immediate timeout.
- TIME_INFINITE no timeout.

Returns

The operation result.

Return values

| MSG_OK           | the frame has been queued for transmission. |
| MSG_TIMEOUT      | The operation has timed out. |
| MSG_RESET        | The driver has been stopped while waiting. |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
canTransmitTimeout
    can_lld_is_tx_empty
    osalSysLock
    osalThreadEnqueueTimeoutS
```

7.5.6.9 canReceiveTimeout()

```c
msg_t canReceiveTimeout (  
    CANDriver ∗ canp,  
    canmbx_t mailbox,  
    CANRxFrame ∗ crfp,  
    sysinterval_t timeout )
```

Can frame receive.

The function waits until a frame is received.

Note

Trying to receive while in sleep mode simply enqueues the thread.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
<tr>
<td>out</td>
<td>crfp</td>
<td>pointer to the buffer where the CAN frame is copied</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout (useful in an event driven scenario where a thread never blocks for I/O).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITY no timeout.</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Returns

The operation result.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>a frame has been received and placed in the buffer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>The operation has timed out.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>The driver has been stopped while waiting.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
   can_lld_is_rx_nonempty
      canReceiveTimeout
       osalSysLock
       osalThreadEnqueueTimeoutS
```

7.5.6.10 canSleep()

```c
void canSleep (CANDriver * canp )
```

Enters the sleep mode.

This function puts the CAN driver in sleep mode and broadcasts the sleep_event event source.

Precondition

In order to use this function the option CAN_USE_SLEEP_MODE must be enabled and the CAN_SUPPOR-
TS_SLEEP mode must be supported by the low level driver.

Parameters

| in  | canp | pointer to the CANDriver object |

ChibiOS/HAL
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

7.5.6.11 canWakeup()

```c
void canWakeup (CANDriver * canp)
```

Enforces leaving the sleep mode.

Note

The sleep mode is supposed to be usually exited automatically by an hardware event.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
</table>

ChibiOS-HAL
Here is the call graph for this function:

7.5.6.12 can_lld_init()

```c
void can_lld_init (  
  void  )
```

Low level CAN driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:
7.5.6.13  can_lld_start()

```c
void can_lld_start ( CANDriver * canp )
```

Configures and activates the CAN peripheral.
7.5 CAN Driver

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>canp</td>
<td>pointer to the CANDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.5.6.14 can_lld_stop()

```c
void can_lld_stop (
    CANDriver * canp )
```

Deactivates the CAN peripheral.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>canp</td>
<td>pointer to the CANDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.5.6.15 can_lld_is_tx_empty()

```c
bool can_lld_is_tx_empty (  
    CANDriver * canp,
    canmbx_t mailbox )
```

Determines whether a frame can be transmitted.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>canp</td>
<td>pointer to the CANDriver object</td>
</tr>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
</tbody>
</table>

Returns

The queue space availability.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>no space in the transmit queue.</td>
</tr>
<tr>
<td>true</td>
<td>transmit slot available.</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Function Class:
Not an API, this function is for internal use only.

7.5.6.16 can_lld_transmit()

```c
void can_lld_transmit (  
    CANDriver * canp,  
    canmbx_t mailbox,  
    const CANTxFrame * ctfp  )
```

Inserts a frame into the transmit queue.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ctfp</td>
<td>pointer to the CAN frame to be transmitted</td>
</tr>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.5.6.17 can_lld_is_rx_nonempty()

```c
bool can_lld_is_rx_nonempty (  
    CANDriver * canp,  
    canmbx_t mailbox )
```

Determines whether a frame has been received.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
</tbody>
</table>

Returns

The queue space availability.

Return values

| false | no space in the transmit queue. |
| true  | transmit slot available. |
7.5 CAN Driver

Function Class:

Not an API, this function is for internal use only.

7.5.6.18 can_lld_receive()

void can_lld_receive (  
    CANDriver * canp,  
    canmbx_t mailbox,  
    CANRxFrame * crfp )

Receives a frame from the input queue.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
<tr>
<td>out</td>
<td>crfp</td>
<td>pointer to the buffer where the CAN frame is copied</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.5.6.19 can_lld_abort()

void can_lld_abort (  
    CANDriver * canp,  
    canmbx_t mailbox )

Tries to abort an ongoing transmission.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the CANDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mailbox</td>
<td>mailbox number</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.5.6.20 can_lld_sleep()

void can_lld_sleep (  
    CANDriver * canp )

ChibiOS/HAL
Enters the sleep mode.

Parameters

```
in  canp  pointer to the CANDriver object
```

Function Class:
Not an API, this function is for internal use only.

### 7.5.6.21 can_lld_wakeup()

```c
void can_lld_wakeup (
    CANDriver * canp )
```

Enforces leaving the sleep mode.

Parameters

```
in  canp  pointer to the CANDriver object
```

Function Class:
Not an API, this function is for internal use only.

### 7.5.7 Variable Documentation

#### 7.5.7.1 CAND1

```c
CANDriver CAND1
```

CAND1 driver identifier.
7.6 Crypto Driver

General Crypto Driver.

7.6.1 Detailed Description

General Crypto Driver.

This module implements a generic Cryptography driver.

Precondition

In order to use the crypto driver the HAL_USE_CRY option must be enabled in halconf.h.

Macros

• #define HAL_CRY_USE_FALLBACK FALSE
  Enables the SW fall-back of the cryptographic driver.
• #define HAL_CRY_USE_FALLBACK TRUE
  Enables the SW fall-back of the cryptographic driver.
• #define HAL_CRY_ENFORCE_FALLBACK FALSE
  Makes the driver forcibly use the fall-back implementations.

Driver capability switches

• #define CRY_LLD_SUPPORTS_AES TRUE
• #define CRY_LLD_SUPPORTS_AES_ECB TRUE
• #define CRY_LLD_SUPPORTS_AES_CBC TRUE
• #define CRY_LLD_SUPPORTS_AES_CFB TRUE
• #define CRY_LLD_SUPPORTS_AES_CTR TRUE
• #define CRY_LLD_SUPPORTS_AES_GCM TRUE
• #define CRY_LLD_SUPPORTS_DES TRUE
• #define CRY_LLD_SUPPORTS_DES_ECB TRUE
• #define CRY_LLD_SUPPORTS_DES_CBC TRUE
• #define CRY_LLD_SUPPORTS_SHA1 TRUE
• #define CRY_LLD_SUPPORTS_SHA256 TRUE
• #define CRY_LLD_SUPPORTS_HMAC_SHA256 TRUE
• #define CRY_LLD_SUPPORTS_HMAC_SHA512 TRUE

PLATFORM configuration options

• #define PLATFORM_CRY_USE_CRY1 FALSE
  CRY1 driver enable switch.
**Typedefs**

- typedef size_t bitsize_t  
  Size, in bits, of a crypto field or message.
- typedef uint32_t crykey_t  
  CRY key identifier type.
- typedef struct CRYDriver CRYDriver  
  Type of a structure representing an CRY driver.

**Data Structures**

- struct SHA1Context  
  Type of a SHA1 context.
- struct SHA256Context  
  Type of a SHA256 context.
- struct SHA512Context  
  Type of a SHA512 context.
- struct HMACSHA256Context  
  Type of a HMAC_SHA256 context.
- struct HMACSHA512Context  
  Type of a HMAC_SHA512 context.
- struct CRYConfig  
  Driver configuration structure.
- struct CRYDriver  
  Structure representing an CRY driver.

**Functions**

- void cryInit (void)  
  Cryptographic Driver initialization.
- void cryObjectInit (CRYDriver *cryp)  
  Initializes the standard part of a CRYDriver structure.
- msg_t cryStart (CRYDriver *cryp, const CRYConfig *config)  
  Configures and activates the cryptographic peripheral.
- void cryStop (CRYDriver *cryp)  
  Deactivates the cryptographic peripheral.
- cryerror_t cryLoadAESTransientKey (CRYDriver *cryp, size_t size, const uint8_t *keyp)  
  Initializes the AES transient key.
- cryerror_t cryEncryptAES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)  
  Encryption of a single block using AES.
- cryerror_t cryDecryptAES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)  
  Decryption of a single block using AES.
- cryerror_t cryEncryptAES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)  
  Encryption operation using AES-ECB.
- cryerror_t cryDecryptAES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)  
  Decryption operation using AES-ECB.
- cryerror_t cryEncryptAES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)  
  Encryption operation using AES-CBC.
Encryption operation using AES-CBC.

- cryerror_t cryDecryptAES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using AES-CBC.

- cryerror_t cryEncryptAES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Encryption operation using AES-CFB.

- cryerror_t cryDecryptAES_CFB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using AES-CFB.

- cryerror_t cryEncryptAES_CFB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Encryption operation using AES-CTR.

- cryerror_t cryDecryptAES_CTR (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using AES-CTR.

- cryerror_t cryEncryptAES_CTR (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Encryption operation using AES-GCM.

- cryerror_t cryDecryptAES_GCM (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv, size_t tag_size, uint8_t *tag_out)

Decryption operation using AES-GCM.

- cryerror_t cryLoadDESTransientKey (CRYDriver *cryp, size_t size, const uint8_t *keyp)

Initializes the DES transient key.

- cryerror_t cryEncryptDES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)

Encryption of a single block using (T)DES.

- cryerror_t cryDecryptDES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)

Decryption of a single block using (T)DES.

- cryerror_t cryEncryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)

Encryption operation using (T)DES-ECB.

- cryerror_t cryDecryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)

Decryption operation using (T)DES-ECB.

- cryerror_t cryEncryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Encryption operation using (T)DES-CBC.

- cryerror_t cryDecryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using (T)DES-CBC.

- cryerror_t crySHA1Init (CRYDriver *cryp, SHA1Context *sha1ctxp)

Hash initialization using SHA1.

- cryerror_t crySHA1Update (CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)

Hash update using SHA1.

- cryerror_t crySHA1Final (CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)

Hash finalization using SHA1.

- cryerror_t crySHA256Init (CRYDriver *cryp, SHA256Context *sha256ctxp)

Hash initialization using SHA256.

- cryerror_t crySHA256Update (CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)

Hash update using SHA256.

- cryerror_t crySHA256Final (CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)

Hash finalization using SHA256.
Hash finalization using SHA256.

- `cryerror_t crySHA512Init (CRYDriver *cryp, SHA512Context *sha512ctxp)`
  Hash initialization using SHA512.

- `cryerror_t crySHA512Update (CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)`
  Hash update using SHA512.

- `cryerror_t crySHA512Final (CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)`
  Hash finalization using SHA512.

- `cryerror_t cryLoadHMACTransientKey (CRYDriver *cryp, size_t size, const uint8_t *keyp)`
  Initializes the HMAC transient key.

- `cryerror_t cryHMACSHA256Init (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp)`
  Hash initialization using HMAC_SHA256.

- `cryerror_t cryHMACSHA256Update (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, size_t size, const uint8_t *in)`
  Hash update using HMAC.

- `cryerror_t cryHMACSHA256Final (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, uint8_t *out)`
  Hash finalization using HMAC.

- `cryerror_t cryHMACSHA512Init (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp)`
  Hash initialization using HMAC_SHA512.

- `cryerror_t cryHMACSHA512Update (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, size_t size, const uint8_t *in)`
  Hash update using HMAC.

- `cryerror_t cryHMACSHA512Final (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, uint8_t *out)`
  Hash finalization using HMAC.

- `void cry_lld_init (void)`
  Low level crypto driver initialization.

- `void cry_lld_start (CRYDriver *cryp)`
  Configures and activates the crypto peripheral.

- `void cry_lld_stop (CRYDriver *cryp)`
  Deactivates the crypto peripheral.

- `cryerror_t cry_lld_aes_loadkey (CRYDriver *cryp, size_t size, const uint8_t *keyp)`
  Initializes the AES transient key.

- `cryerror_t cry_lld_encrypt_AES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`
  Encryption of a single block using AES.

- `cryerror_t cry_lld_decrypt_AES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`
  Decryption of a single block using AES.

- `cryerror_t cry_lld_encrypt_AES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`
  Encryption operation using AES-ECB.

- `cryerror_t cry_lld_decrypt_AES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`
  Decryption operation using AES-ECB.

- `cryerror_t cry_lld_encrypt_AES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Encryption operation using AES-CBC.

- `cryerror_t cry_lld_decrypt_AES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Decryption operation using AES-CBC.

- `cryerror_t cry_lld_encrypt_AES_CFB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Encryption operation using AES-CFB.

- `cryerror_t cry_lld_decrypt_AES_CFB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Decryption operation using AES-CFB.
Crypto Driver 113

Decryption operation using AES-CFB.

- cryerror_t cry_lld_encrypt_AES_CTR (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Encryption operation using AES-CTR.

- cryerror_t cry_lld_decrypt_AES_CTR (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using AES-CTR.

- cryerror_t cry_lld_encrypt_AES_GCM (CRYDriver *cryp, crykey_t key_id, size_t auth_size, const uint8_t *auth_in, size_t text_size, const uint8_t *text_in, uint8_t *text_out, const uint8_t *iv, size_t tag_size, uint8_t *tag_out)

Encryption operation using AES-GCM.

- cryerror_t cry_lld_decrypt_AES_GCM (CRYDriver *cryp, crykey_t key_id, size_t auth_size, const uint8_t *auth_in, size_t text_size, const uint8_t *text_in, uint8_t *text_out, const uint8_t *iv, size_t tag_size, const uint8_t *tag_in)

Decryption operation using AES-GCM.

- cryerror_t cry_lld_des_loadkey (CRYDriver *cryp, size_t size, const uint8_t *keyp)

Initializes the DES transient key.

- cryerror_t cry_lld_encrypt_DES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)

Encryption of a single block using (T)DES.

- cryerror_t cry_lld_decrypt_DES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)

Decryption of a single block using (T)DES.

- cryerror_t cry_lld_encrypt_DES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)

Encryption operation using (T)DES-ECB.

- cryerror_t cry_lld_decrypt_DES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)

Decryption operation using (T)DES-ECB.

- cryerror_t cry_lld_encrypt_DES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Encryption operation using (T)DES-CBC.

- cryerror_t cry_lld_decrypt_DES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using (T)DES-CBC.

- cryerror_t cry_lld_SHA1_init (CRYDriver *cryp, SHA1Context *sha1ctxp)

Hash initialization using SHA1.

- cryerror_t cry_lld_SHA1_update (CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)

Hash update using SHA1.

- cryerror_t cry_lld_SHA1_final (CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)

Hash finalization using SHA1.

- cryerror_t cry_lld_SHA256_init (CRYDriver *cryp, SHA256Context *sha256ctxp)

Hash initialization using SHA256.

- cryerror_t cry_lld_SHA256_update (CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)

Hash update using SHA256.

- cryerror_t cry_lld_SHA256_final (CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)

Hash finalization using SHA256.

- cryerror_t cry_lld_SHA512_init (CRYDriver *cryp, SHA512Context *sha512ctxp)

Hash initialization using SHA512.

- cryerror_t cry_lld_SHA512_update (CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)

Hash update using SHA512.

- cryerror_t cry_lld_SHA512_final (CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)

Hash finalization using SHA512.
Hash finalization using SHA512.

- cryerror_t cry_lld_hmac_loadkey (CRYDriver *cryp, size_t size, const uint8_t *keyp)
  Initializes the HMAC transient key.
- cryerror_t cry_lld_HMACSHA256_init (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp)
  Hash initialization using HMAC_SHA256.
- cryerror_t cry_lld_HMACSHA256_update (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, size_t size, const uint8_t *in)
  Hash update using HMAC.
- cryerror_t cry_lld_HMACSHA256_final (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, uint8_t *out)
  Hash finalization using HMAC.
- cryerror_t cry_lld_HMACSHA512_init (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp)
  Hash initialization using HMAC_SHA512.
- cryerror_t cry_lld_HMACSHA512_update (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, size_t size, const uint8_t *in)
  Hash update using HMAC.
- cryerror_t cry_lld_HMACSHA512_final (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, uint8_t *out)
  Hash finalization using HMAC.

Enumerations

- enum crystate_t { CRY_UNINIT = 0, CRY_STOP = 1, CRY_READY = 2 }
  Driver state machine possible states.
- enum cryerror_t { CRY_NOERROR = 0, CRY_ERR_INV_ALGO = 1, CRY_ERR_INV_KEY_SIZE = 2, CRY_ERR_INV_KEY_TYPE = 3, CRY_ERR_INV_KEY_ID = 4, CRY_ERR_AUTH_FAILED = 5, CRY_ERR_OP_FAILURE = 6 }
  Driver error codes.
- enum cryalgorithm_t { cry_algo_aes, cry_algo_des, cry_algo_hmac }
  Type of an algorithm identifier.

Variables

- CRYDriver CRYD1
  CRY1 driver identifier.

7.6.2  Macro Definition Documentation

7.6.2.1  HAL_CRY_USE_FALLBACK [1/2]

#define HAL_CRY_USE_FALLBACK FALSE

Enables the SW fall-back of the cryptographic driver.

When enabled, this option, activates a fall-back software implementation for algorithms not supported by the under-
lying hardware.

Note

Fall-back implementations may not be present for all algorithms.
7.6.2.2 HAL_CRY_USE_FALLBACK [2/2]

#define HAL_CRY_USE_FALLBACK TRUE

Enables the SW fall-back of the cryptographic driver.

When enabled, this option, activates a fall-back software implementation for algorithms not supported by the underlying hardware.

Note

Fall-back implementations may not be present for all algorithms.

7.6.2.3 HAL_CRY_ENFORCE_FALLBACK

#define HAL_CRY_ENFORCE_FALLBACK FALSE

Makes the driver forcibly use the fall-back implementations.

Note

If enabled then the LLD driver is not included at all.

7.6.2.4 PLATFORM_CRY_USE_CRY1

#define PLATFORM_CRY_USE_CRY1 FALSE

CRY1 driver enable switch.

If set to TRUE the support for CRY1 is included.

Note

The default is FALSE.

7.6.3 Typedef Documentation

7.6.3.1 bitsize_t

typedef size_t bitsize_t

Size, in bits, of a crypto field or message.

Note

It is assumed, for simplicity, that this type is equivalent to a size_t.
7.6.3.2 crykey_t

typedef uint32_t crykey_t

CRY key identifier type.

7.6.3.3 CRYDriver

typedef struct CRYDriver CRYDriver

Type of a structure representing an CRY driver.

7.6.4 Enumeration Type Documentation

7.6.4.1 crystate_t

enum crystate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>CRY_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>CRY_READY</td>
<td>Ready.</td>
</tr>
</tbody>
</table>

7.6.4.2 cryerror_t

enum cryerror_t

Driver error codes.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>No error.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>Invalid cypher/mode.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>Invalid key size.</td>
</tr>
</tbody>
</table>
7.6 Crypto Driver

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYP_ERR_INV_KEY_TYPE</td>
<td>Invalid key type.</td>
</tr>
<tr>
<td>CRYP_ERR_INV_KEY_ID</td>
<td>Invalid key identifier.</td>
</tr>
<tr>
<td>CRYP_ERR_AUTH_FAILED</td>
<td>Failed authentication.</td>
</tr>
<tr>
<td>CRYP_ERR_OP_FAILURE</td>
<td>Failed operation.</td>
</tr>
</tbody>
</table>

7.6.4.3 cryalgorithm_t

eenum cryalgorithm_t

Type of an algorithm identifier.

Note

It is only used to determine the key required for operations.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cry_algo_aes</td>
<td>AES 128, 192, 256 bits.</td>
</tr>
<tr>
<td>cry_algo_des</td>
<td>DES 56, TDES 112, 168 bits.</td>
</tr>
<tr>
<td>cry_algo_hmac</td>
<td>HMAC variable size.</td>
</tr>
</tbody>
</table>

7.6.5 Function Documentation

7.6.5.1 cryInit()

```c
void cryInit()
{
    void
}
```

Cryptographic Driver initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
cryInit  →  cry_lld_init
```

7.6.5.2 cryObjectInit()

```c
void cryObjectInit ( 
    CRYDriver * cryp )
```

Initializes the standard part of a CRYDriver structure.

Parameters

| out | cryp | pointer to the CRYDriver object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.6.5.3 cryStart()

```c
msg_t cryStart ( 
    CRYDriver * cryp,
    const CRYConfig * config )
```

Configures and activates the cryptographic peripheral.

Parameters

| in  | cryp | pointer to the CRYDriver object |
| in  | config | pointer to the CRYConfig object. Depending on the implementation the value can be NULL. |
7.6 Crypto Driver

Returns

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryStart
    cry_lld_start
    osalSysLock
```

7.6.5.4 cryStop()

```c
void cryStop (CRYDriver * cryp )
```

Deactivates the cryptographic peripheral.

Parameters

- **in cryp** pointer to the CRYDriver object
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryStop
    cry_lld_stop
    osalSysLock
    osalSysUnlock
```

7.6.5.5 cryLoadAESTransientKey()

cryerror_t cryLoadAESTransientKey (
    CRYDriver * cryp,
    size_t size,
    const uint8_t * keyp )

Initializes the AES transient key.

Note

It is the underlying implementation to decide which key sizes are allowable.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>key size in bytes</td>
</tr>
<tr>
<td>in</td>
<td>keyp</td>
<td>pointer to the key data</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the algorithm is unsupported.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>if the specified key size is invalid for the specified algorithm.</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryLoadAESTransientKey  \rightarrow  cry_lld_aes_loadkey
```

### 7.6.5.6 cryEncryptAES()

```c
cryerror_t cryEncryptAES (  
    CRYDriver * cryp,  
    crykey_t key_id,  
    const uint8_t * in,  
    uint8_t * out )
```

Encryption of a single block using AES.

**Note**

The implementation of this function must guarantee that it can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

```
    cryEncryptAES
    cry_lld_encrypt_AES
```

### 7.6.5.7 cryDecryptAES()

```c
cryerror_t cryDecryptAES(
    CRYDriver * cryp,
    crykey_t key_id,
    const uint8_t * in,
    uint8_t * out
)
```

Decryption of a single block using AES.

**Note**

The implementation of this function must guarantee that it can be called from any context.

**Parameters**

- **in cryp** pointer to the CRYDriver object
- **in key_id** the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way
- **in in** buffer containing the input ciphertext
- **out out** buffer for the output plaintext

**Returns**

The operation status.

**Return values**

- **CRY_NOERROR** if the operation succeeded.
- **CRY_ERR_INV_ALGO** if the operation is unsupported on this device instance.
- **CRY_ERR_INV_KEY_TYPE** the selected key is invalid for this operation.
- **CRY_ERR_INV_KEY_ID** if the specified key identifier is invalid or refers to an empty key slot.
- **CRY_ERR_OP_FAILURE** if the operation failed, implementation dependent.
Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

![Call Graph]

7.6.5.8 cryEncryptAES_ECB()

cryerror_t cryEncryptAES_ECB (  
    CRYDriver ∗ cryp,  
    crykey_t key_id,  
    size_t size,  
    const uint8_t ∗ in,  
    uint8_t ∗ out  )

Encryption operation using AES-ECB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of both buffers, this number must be a multiple of 16</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryEncryptAES_ECB -> cry_lld_encrypt_AES_ECB
```

7.6.5.9 cryDecryptAES_ECB()

```c
cryerror_t cryDecryptAES_ECB {
    CRYDriver * cryp,
    crykey_t key_id,
    size_t size,
    const uint8_t * in,
    uint8_t * out
}
```

Decryption operation using AES-ECB.

**Note**

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of both buffers, this number must be a multiple of 16</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output plaintext</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryDecryptAES_ECB -> cry_lld_decrypt_AES_ECB
```

### 7.6.5.10 cryEncryptAES_CBC()

cryerror_t cryEncryptAES_CBC (cryp, key_id, size, in, out, iv)

Encryption operation using AES-CBC.

**Note**

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>size</td>
<td>size of both buffers, this number must be a multiple of 16</td>
</tr>
<tr>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td>iv</td>
<td>128 bits input vector</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryEncryptAES_CBC  ←  cry_decrypt_AES_CBC
```

7.6.5.11 cryDecryptAES_CBC()

cryerror_t cryDecryptAES_CBC (cryDriver * cryp,
crykey_t key_id,
size_t size,
const uint8_t * in,
uint8_t * out,
const uint8_t * iv )

Decryption operation using AES-CBC.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in cryp</td>
<td>pointer to the cryDriver object</td>
</tr>
<tr>
<td>in key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in size</td>
<td>size of both buffers, this number must be a multiple of 16</td>
</tr>
<tr>
<td>in in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out out</td>
<td>buffer for the output plaintext</td>
</tr>
<tr>
<td>in iv</td>
<td>128 bits input vector</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]

7.6.5.12 cryEncryptAES_CFB()

```c
error_t cryEncryptAES_CFB (cryDriver * cryp, crykey_t key_id, size_t size,
                          const uint8_t * in, uint8_t * out, const uint8_t * iv)
```

Encryption operation using AES-CFB.

Note

This is a stream cipher, there are no size restrictions.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>key_id the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size size of both buffers</td>
</tr>
<tr>
<td>in</td>
<td>in buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out buffer for the output ciphertext</td>
</tr>
<tr>
<td>in</td>
<td>iv 128 bits input vector</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    cryEncryptAES_CFB  ->  cry_lld_encrypt_AES_CFB
```

7.6.5.13 cryDecryptAES_CFB()

```c
cryerror_t cryDecryptAES_CFB (  
    CRYDriver  *  cryp,
    crykey_t  key_id,  
    size_t  size,  
    const uint8_t  *  in,  
    uint8_t  *  out,  
    const uint8_t  *  iv )
```

Decryption operation using AES-CFB.

Note

This is a stream cipher, there are no size restrictions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>size</td>
<td>size of both buffers</td>
</tr>
<tr>
<td>in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out</td>
<td>buffer for the output plaintext</td>
</tr>
<tr>
<td>iv</td>
<td>128 bits input vector</td>
</tr>
</tbody>
</table>
7.6 Crypto Driver

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]

7.6.5.14 cryEncryptAES_CTR()

```c
cryerror_t cryEncryptAES_CTR ( 
    CRYDriver * cryp, 
    crykey_t key_id,  
    size_t  size, 
    const uint8_t * in, 
    uint8_t * out, 
    const uint8_t * iv )
```

Encryption operation using AES-CTR.

Note

This is a stream cipher, there are no size restrictions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>key_id the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size size of both buffers</td>
</tr>
<tr>
<td>in</td>
<td>in buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out buffer for the output ciphertext</td>
</tr>
<tr>
<td>in</td>
<td>iv 128 bits input vector + counter, it contains a 96 bits IV and a 32 bits counter</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](cryEncryptAES_CTR cry_lld_encrypt_AES_CTR)

7.6.5.15 cryDecryptAES_CTR()

```c
cryerror_t cryDecryptAES_CTR ( 
    CRYDriver * cryp,
    crykey_t key_id,
    size_t size,
    const uint8_t * in,
    uint8_t * out,
    const uint8_t * iv )
```

Decryption operation using AES-CTR.

Note

This is a stream cipher, there are no size restrictions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in size</td>
<td>size of both buffers</td>
</tr>
<tr>
<td>in in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out out</td>
<td>buffer for the output plaintext</td>
</tr>
<tr>
<td>in iv</td>
<td>128 bits input vector + counter, it contains a 96 bits IV and a 32 bits counter</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
| cryDecryptAES_CTR | cry_lld_decrypt_AES_CTR |
```

7.6.5.16 cryEncryptAES_GCM()

```c
cryerror_t cryEncryptAES_GCM ( 
    CRYDriver * cryp,
    crykey_t key_id,
    size_t auth_size,
    const uint8_t * auth_in,
    size_t text_size,
    const uint8_t * text_in,
    uint8_t * text_out,
    const uint8_t * iv,
    size_t tag_size,
    uint8_t * tag_out )
```

Encryption operation using AES-GCM.

Note

This is a stream cipher, there are no size restrictions.
Parameters

| in  | cryp                      | pointer to the CRYDriver object |
| in  | key_id                   | the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way |
| in  | auth_size                | size of the data buffer to be authenticated |
| in  | auth_in                  | buffer containing the data to be authenticated |
| in  | text_size                | size of the text buffer |
| in  | text_in                  | buffer containing the input plaintext |
| out | text_out                 | buffer for the output ciphertext |
| in  | iv                       | 128 bits input vector |
| in  | tag_size                 | size of the authentication tag, this number must be between 1 and 16 |
| out | tag_out                  | buffer for the generated authentication tag |

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.6.5.17 cryDecryptAES_GCM()

cryerror_t cryDecryptAES_GCM (crykey_t key_id, CRYDriver * cryp, CRYDriver * cryp)
```c
size_t auth_size,
const uint8_t * auth_in,
size_t text_size,
const uint8_t * text_in,
uint8_t * text_out,
const uint8_t * iv,
size_t tag_size,
const uint8_t * tag_in)
```

Decryption operation using AES-GCM.

**Note**

This is a stream cipher, there are no size restrictions.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code> <code>cryp</code></td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td><code>in</code> <code>key_id</code></td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td><code>in</code> <code>auth_size</code></td>
<td>size of the data buffer to be authenticated</td>
</tr>
<tr>
<td><code>in</code> <code>auth_in</code></td>
<td>buffer containing the data to be authenticated</td>
</tr>
<tr>
<td><code>in</code> <code>text_size</code></td>
<td>size of the text buffer</td>
</tr>
<tr>
<td><code>in</code> <code>text_in</code></td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td><code>out</code> <code>text_out</code></td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td><code>in</code> <code>iv</code></td>
<td>128 bits input vector</td>
</tr>
<tr>
<td><code>in</code> <code>tag_size</code></td>
<td>size of the authentication tag, this number must be between 1 and 16</td>
</tr>
<tr>
<td><code>in</code> <code>tag_in</code></td>
<td>buffer for the generated authentication tag</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CRY_NOERROR</code></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_ALGO</code></td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_KEY_TYPE</code></td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_KEY_ID</code></td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td><code>CRY_ERR_AUTH_FAILED</code></td>
<td>authentication failed</td>
</tr>
<tr>
<td><code>CRY_ERR_OP_FAILURE</code></td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](cryDecryptAES_GCM -> cry_lld_decrypt_AES_GCM)

### 7.6.5.18 cryLoadDESTransientKey()

cryerror_t cryLoadDESTransientKey {
    CRYDriver * cryp,
    size_t size,
    const uint8_t * keyp
}

Initializes the DES transient key.

**Note**
It is the underlying implementation to decide which key sizes are allowable.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>key size in bytes</td>
</tr>
<tr>
<td>in</td>
<td>keyp</td>
<td>pointer to the key data</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the algorithm is unsupported.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>if the specified key size is invalid for the specified algorithm.</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

```
cryLoadDESTransientKeycry_lld_des_loadkey
```

### 7.6.5.19 cryEncryptDES()

```c
cryerror_t cryEncryptDES (
    CRYDriver * cryp,
    crykey_t key_id,
    const uint8_t * in,
    uint8_t * out )
```

Encryption of a single block using (T)DES.

**Note**

The implementation of this function must guarantee that it can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>crypt</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **CRY_NOERROR** if the operation succeeded.
- **CRY_ERR_INV_ALGO** if the operation is unsupported on this device instance.
- **CRY_ERR_INV_KEY_TYPE** if the selected key is invalid for this operation.
- **CRY_ERR_INV_KEY_ID** if the specified key identifier is invalid or refers to an empty key slot.
- **CRY_ERR_OP_FAILURE** if the operation failed, implementation dependent.
Function Class:
Special function, this function has special requirements see the notes.

Here is the call graph for this function:

```
cryEncryptDES cry_lld_encrypt_DES
```

### 7.6.5.20 cryDecryptDES()

```c
cryerror_t cryDecryptDES (    
    CRYDriver * cryp,    
    crykey_t key_id,    
    const uint8_t * in,    
    uint8_t * out )
```

Decryption of a single block using (T)DES.

**Note**

The implementation of this function must guarantee that it can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output plaintext</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

```
cryDecryptDES -> cry_lld_decrypt_DES
```

### 7.6.5.21 `cryEncryptDES_ECB()`

```c
cryerror_t cryEncryptDES_ECB (  
    CRYDriver * cryp,  
    crykey_t key_id,  
    size_t size,  
    const uint8_t * in,  
    uint8_t * out)
```

Encryption operation using (T)DES-ECB.

**Note**

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the <code>CRYDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of both buffers, this number must be a multiple of 8</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

| `CRY_NOERROR` | if the operation succeeded. |
| `CRY_ERR_INV_ALGO` | if the operation is unsupported on this device instance. |
Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

Here is the call graph for this function:

7.6.5.22 cryDecryptDES_ECB()

cryerror_t cryDecryptDES_ECB (cryp, cryp, crykey_t key_id, size_t size, const uint8_t * in, uint8_t * out )

Decryption operation using (T)DES-ECB.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>key←_id the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size size of both buffers, this number must be a multiple of 8</td>
</tr>
<tr>
<td>in</td>
<td>in buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out</td>
<td>out buffer for the output plaintext</td>
</tr>
</tbody>
</table>
7.6 Crypto Driver

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryDecryptDES_ECB ◄cry_lld_decrypt_DES_ECB
```

7.6.5.23 cryEncryptDES_CBC()

`cryerror_t cryEncryptDES_CBC ( `CRYDriver` * cryp, `crykey_t` key_id, `size_t` size, `const uint8_t` * in, `uint8_t` * out, `const uint8_t` * iv )`

Encryption operation using (T)DES-CBC.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp  pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>key_id → the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>Ch / DOMA</td>
<td>size</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
</tr>
<tr>
<td>in</td>
<td>iv</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
  cryEncryptDES_CBC  <--- cry_lld_encrypt_DES_CBC
```

7.6.5.24 cryDecryptDES_CBC()

```c
cryerror_t cryDecryptDES_CBC (  
    CRYDriver  * cryp,  
    crykey_t  key_id,  
    size_t size,  
    const uint8_t  * in,  
    uint8_t  * out,  
    const uint8_t  * iv )
```

Decryption operation using (T)DES-CBC.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in  cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in  key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in  size</td>
<td>size of both buffers, this number must be a multiple of 8</td>
</tr>
<tr>
<td>in  in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out  out</td>
<td>buffer for the output plaintext</td>
</tr>
<tr>
<td>in  iv</td>
<td>64 bits input vector</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](cryDecryptDES_CBCcry_lld_decrypt_DES_CBC)

7.6.5.25 crySHA1Init()

cryerror_t crySHA1Init {
    CRYDriver * cryp,
    SHA1Context * sha1ctxp
}

Hash initialization using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
</tr>
<tr>
<td>out</td>
<td>sha1ctxp</td>
</tr>
</tbody>
</table>

pointer to the CRYDriver object
pointer to a SHA1 context to be initialized
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>If the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>If the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>If the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
7.6.5.26 crySHA1Update()

cryerror_t crySHA1Update (          
    CRYDriver * cryp, 
    SHA1Context * sha1ctxp, 
    size_t size, 
    const uint8_t * in )
```

Hash update using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>sha1ctxp</td>
<td>pointer to a SHA1 context</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of input buffer</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input text</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>If the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>If the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>If the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
crySHA1Update cry_lld_SHA1_update
```

### 7.6.5.27 crySHA1Final()

```c
cryerror_t crySHA1Final (cryp, SHA1Context * sha1ctxp, out)
```

Hash finalization using SHA1.

**Note**

Use of this algorithm is not recommended because proven weak.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>sha1ctxp</td>
<td>pointer to a SHA1 context</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>160 bits output buffer</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.
Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
| crySHA1Final | cry_lld_SHA1_final |
```

7.6.5.28  crySHA256Init()

```c
cryerror_t crySHA256Init (  
    CRYDriver * cryp,  
    SHA256Context * sha256ctxp )
```

Hash initialization using SHA256.

Parameters

<table>
<thead>
<tr>
<th>in cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out sha256ctxp</td>
<td>pointer to a SHA256 context to be initialized</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
7.6 Crypto Driver

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.6.5.29 crySHA256Update()

cryerror_t crySHA256Update (  
    CRYDriver * cryp,  
    SHA256Context * sha256ctxp,  
    size_t size,  
    const uint8_t * in )

Hash update using SHA256.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sha256ctxp</td>
<td>pointer to a SHA256 context</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of input buffer</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input text</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

7.6.5.30 crySHA256Final()

```
cryerror_t crySHA256Final (  
    CRYDriver * cryp,  
    SHA256Context * sha256ctxp,  
    uint8_t * out )
```

Hash finalization using SHA256.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sha256ctxp</td>
<td>pointer to a SHA256 context</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>256 bits output buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
<td></td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
<td></td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
<td></td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
| crySHA256Final | cry_lld_SHA256_final |
```

7.6.5.31 crySHA512Init()

```c
cryerror_t crySHA512Init (  
    CRYDriver ∗ cryp,  
    SHA512Context ∗ sha512ctxp )
```

Hash initialization using SHA512.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sha512ctxp</td>
<td>pointer to a SHA512 context to be initialized</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
crySHA512Init  
   
cry_lld_SHA512_init
```

7.6.5.32 crySHA512Update()

cryerror_t crySHA512Update (  
   CRYDriver * cryp,  
   SHA512Context * sha512ctxp,  
   size_t size,  
   const uint8_t * in )

Hash update using SHA512.

Parameters

| in  | cryp        | pointer to the CRYDriver object |
| in  | sha512ctxp  | pointer to a SHA512 context     |
| in  | size        | size of input buffer            |
| in  | in          | buffer containing the input text|

Returns

The operation status.

Return values

| CRY_NOERROR | if the operation succeeded. |
| CRY_ERR_INV_ALGO | if the operation is unsupported on this device instance. |
| CRY_ERR_OP_FAILURE | if the operation failed, implementation dependent. |
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
crySHA512Update -> cry_lld_SHA512_update
```

### 7.6.5.33 crySHA512Final()

```c
cryerror_t crySHA512Final (  
    CRYDriver * cryp,  
    SHA512Context * sha512ctxp,  
    uint8_t * out )
```

Hash finalization using SHA512.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>sha512ctxp pointer to a SHA512 context</td>
</tr>
<tr>
<td>out</td>
<td>out 512 bits output buffer</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>If the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>If the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>If the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
  crySHA512Final  cry_lld_SHA512_final
```

7.6.5.34 cryLoadHMACTransientKey()

```c
cryerror_t cryLoadHMACTransientKey (CryDriver * cryp,
                                 size_t size,
                                 const uint8_t * keyp )
```

Initializes the HMAC transient key.

Note

It is the underlying implementation to decide which key sizes are allowable.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CryDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>key size in bytes</td>
</tr>
<tr>
<td>in</td>
<td>keyp</td>
<td>pointer to the key data</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the algorithm is unsupported.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>if the specified key size is invalid for the specified algorithm.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

```
cryLoadHMACTransientKey  
cry_lld_hmac_loadkey
```

### 7.6.5.35 cryHMACSHA256Init()

```c
cryerror_t cryHMACSHA256Init ( 
    CRYDriver * cryp, 
    HMACSHA256Context * hmacsha256ctxp )
```

Hash initialization using HMAC_SHA256.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>hmacsha256ctxp</td>
<td>pointer to a HMAC_SHA256 context to be initialized</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryHMACSHA256Init
  
cry_lld_HMACSHA256_init
```
7.6.5.36 cryHMACSHA256Update()

cryerror_t cryHMACSHA256Update (  
    CRYDriver * cryp,  
    HMACSHA256Context * hmacsha256ctxp,  
    size_t size,  
    const uint8_t * in )

Hash update using HMAC.

Parameters

| in  | cryp                        | pointer to the CRYDriver object |
| in  | hmacsha256ctxp              | pointer to a HMAC_SHA256 context |
| in  | size                        | size of input buffer            |
| in  | in                          | buffer containing the input text |

Returns

The operation status.

Return values

| CRY_NOERROR | if the operation succeeded. |
| CRY_ERR_INV_ALGO | if the operation is unsupported on this device instance. |
| CRY_ERR_OP_FAILURE | if the operation failed, implementation dependent. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.6.5.37 cryHMACSHA256Final()

cryerror_t cryHMACSHA256Final ( 
    CRYDriver * cryp, 
    HMACSHA256Context * hmacsha256ctxp, 
    uint8_t * out )

Hash finalization using HMAC.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>hmacsha256ctxp</td>
<td>pointer to a HMAC_SHA256 context</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>256 bits output buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![call graph]

7.6.5.38 cryHMACSHA512Init()

cryerror_t cryHMACSHA512Init ( 
    CRYDriver * cryp, 
    HMACSHA512Context * hmacsha512ctxp )

Hash initialization using HMAC_SHA512.
7.6.5.39 cryHMACSHA512Update()

```c
cryerror_t cryHMACSHA512Update (  
    CRYDriver * cryp,
    HMACSHA512Context * hmacsha512ctxp,
    size_t size,
    const uint8_t * in )
```

Hash update using HMAC.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>hmacsha512ctxp</td>
<td>pointer to a HMAC_SHA512 context</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of input buffer</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input text</td>
</tr>
</tbody>
</table>
Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
cryHMACSHA512Update
   cry_lld_HMACSHA512
      _update
```

7.6.5.40 cryHMACSHA512Final()

cryerror_t cryHMACSHA512Final ( 
    CRYDriver * cryp,
    HMACSHA512Context * hmacsha512ctxp,
    uint8_t * out )

Hash finalization using HMAC.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>hmacsha512ctxp pointer to a HMAC_SHA512 context</td>
</tr>
<tr>
<td>out</td>
<td>out 512 bits output buffer</td>
</tr>
</tbody>
</table>

Returns
The operation status.
Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>If the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>If the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>If the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
+-----------------+-----------------+
| cryHMACSHA512Final | cry_lld_HMACSHA512
| _final           | _final          |
```

### 7.6.5.41 cry_lld_init()

```c
void cry_lld_init (  
    void )
```

Low level crypto driver initialization.

Function Class:
Not an API, this function is for internal use only.

### 7.6.5.42 cry_lld_start()

```c
void cry_lld_start (  
    CRYDriver * cryp )
```

Configures and activates the crypto peripheral.

**Parameters**

- **in cryp** pointer to the CRYDriver object
7.6.5.43 cry_lld_stop()

```c
void cry_lld_stop ( 
    CRYDriver * cryp )
```

Deactivates the crypto peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in cryp</code></td>
<td>pointer to the CRYDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.44 cry_lld_aes_loadkey()

```c
cryerror_t cry_lld_aes_loadkey ( 
    CRYDriver * cryp, 
    size_t size, 
    const uint8_t * keyp )
```

Initializes the AES transient key.

**Note**

It is the underlying implementation to decide which key sizes are allowable.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in cryp</code></td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td><code>in size</code></td>
<td>key size in bytes</td>
</tr>
<tr>
<td><code>in keyp</code></td>
<td>pointer to the key data</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CRY_NOERROR</code></td>
<td>if the operation succeeded.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the algorithm is unsupported.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>if the specified key size is invalid for the specified algorithm.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.6.5.45 cry_lld_encrypt_AES()

```c
cryerror_t cry_lld_encrypt_AES ( 
    CRYDriver * cryp,
    crykey_t key_id,
    const uint8_t * in,
    uint8_t * out )
```

Encryption of a single block using AES.

**Note**

The implementation of this function must guarantee that it can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cryp</code></td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td><code>key_id</code></td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td><code>in</code></td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td><code>out</code></td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.6.5.46 cry_lld_decrypt_AES()

```c
cryerror_t cry_lld_decrypt_AES (  
    CRYDriver * cryp,  
    crykey_t key_id,  
    const uint8_t * in,  
    uint8_t * out )
```

Decryption of a single block using AES.

**Note**

The implementation of this function must guarantee that it can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td><code>key_id</code></td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td><code>in</code></td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td><code>out</code></td>
<td>buffer for the output plaintext</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CRY_NOERROR</code></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_ALGO</code></td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_KEY_TYPE</code></td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_KEY_ID</code></td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td><code>CRY_ERR_OP_FAILURE</code></td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

7.6.5.47 cry_lld_encrypt_AES_ECB()

```c
cryerror_t cry_lld_encrypt_AES_ECB (  
    CRYDriver * cryp,  
    crykey_t key_id,  
    size_t size,  
    const uint8_t * in,  
    uint8_t * out )
```

Encryption operation using AES-ECB.
Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cryp</code></td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td><code>key_id</code></td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td><code>size</code></td>
<td>size of both buffers, this number must be a multiple of 16</td>
</tr>
<tr>
<td><code>in</code></td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td><code>out</code></td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CRY_NOERROR</code></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_ALGO</code></td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_KEY_TYPE</code></td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td><code>CRY_ERR_INV_KEY_ID</code></td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td><code>CRY_ERR_OP_FAILURE</code></td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.48 cry_lld_decrypt_AES_ECB()

cryerror_t cry_lld_decrypt_AES_ECB (   
    CRYDriver * cryp,   
    crykey_t key_id,   
    size_t size,   
    const uint8_t * in,   
    uint8_t * out )

Decryption operation using AES-ECB.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of both buffers, this number must be a multiple of 16</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.49 cry_lld_encrypt_AES_CBC()

cryerror_t cry_lld_encrypt_AES_CBC (cryp, key, size, in, out, iv)

Encryption operation using AES-CBC.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of both buffers, this number must be a multiple of 16</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td>in</td>
<td>iv</td>
<td>128 bits initial vector</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.50 cry_lld_decrypt_AES_CBC()

```c
cryerror_t cry_lld_decrypt_AES_CBC (crip Driver * cryp,
    crykey_t key_id,
    size_t size,
    const uint8_t * in,
    uint8_t * out,
    const uint8_t * iv )
```

Decryption operation using AES-CBC.

Note

The function operates on data buffers whose length is a multiple of an AES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>size</td>
<td>size of both buffers, this number must be a multiple of 16</td>
</tr>
<tr>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td>iv</td>
<td>128 bits initial vector</td>
</tr>
</tbody>
</table>

Returns

The operation status.
7.6 Crypto Driver

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.51 cry_lld_encrypt_AES_CFB()

cryerror_t cry_lld_encrypt_AES_CFB (cryDriver * cryp,
crykey_t key_id,
size_t size,
const uint8_t * in,
uint8_t * out,
const uint8_t * iv )

Encryption operation using AES-CFB.

Note

This is a stream cipher, there are no size restrictions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in size</td>
<td>size of both buffers</td>
</tr>
<tr>
<td>in in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out out</td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td>in iv</td>
<td>128 bits initial vector</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
</tbody>
</table>
7.6.5.52 cry_lld_decrypt_AES_CFB()

cryerror_t cry_lld_decrypt_AES_CFB ( 
    CRYDriver * cryp, 
    crykey_t key_id, 
    size_t size, 
    const uint8_t * in, 
    uint8_t * out, 
    const uint8_t * iv )

Decryption operation using AES-CFB.

Note
This is a stream cipher, there are no size restrictions.

Parameters

| in  | key_id | the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way |
|     | size   | size of both buffers |
|     | in     | buffer containing the input plaintext |
|     | out    | buffer for the output ciphertext |
|     | iv     | 128 bits initial vector |

Returns
The operation status.

Return values

| CRY_NOERROR | if the operation succeeded. |
| CRY_ERR_INV_ALGO | if the operation is unsupported on this device instance. |
| CRY_ERR_INV_KEY_TYPE | the selected key is invalid for this operation. |
| CRY_ERR_INV_KEY_ID | if the specified key identifier is invalid or refers to an empty key slot. |
| CRY_ERR_OP_FAILURE | if the operation failed, implementation dependent. |
Function Class:

Not an API, this function is for internal use only.

### 7.6.5.53  cry_lld_encrypt_AES_CTR()

```c
cryerror_t cry_lld_encrypt_AES_CTR (  
    CRYDriver * cryp,  
    crykey_t key_id,  
    size_t size,  
    const uint8_t * in,  
    uint8_t * out,  
    const uint8_t * iv )
```

Encryption operation using AES-CTR.

**Note**

This is a stream cipher, there are no size restrictions.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of both buffers</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td>in</td>
<td>iv</td>
<td>128 bits initial vector + counter, it contains a 96 bits IV and a 32 bits counter</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.6.5.54 cry_lld_decrypt_AES_CTR()

```c
cryerror_t cry_lld_decrypt_AES_CTR ( 
    CRYDriver * cryp,
    crykey_t key_id,
    size_t size,
    const uint8_t * in,
    uint8_t * out,
    const uint8_t * iv )
```

Decryption operation using AES-CTR.

**Note**

This is a stream cipher, there are no size restrictions.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key←_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of both buffers</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output plaintext</td>
</tr>
<tr>
<td>in</td>
<td>iv</td>
<td>128 bits initial vector + counter, it contains a 96 bits IV and a 32 bits counter</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

7.6.5.55 cry_lld_encrypt_AES_GCM()

```c
cryerror_t cry_lld_encrypt_AES_GCM ( 
    CRYDriver * cryp,
    crykey_t key_id,
```

ChibiOS/HAL
Encryption operation using AES-GCM.

Note
This is a stream cipher, there are no size restrictions.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>auth_size</td>
<td>size of the data buffer to be authenticated</td>
</tr>
<tr>
<td>in</td>
<td>auth_in</td>
<td>buffer containing the data to be authenticated</td>
</tr>
<tr>
<td>in</td>
<td>text_size</td>
<td>size of the text buffer</td>
</tr>
<tr>
<td>in</td>
<td>text_in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>text_out</td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td>in</td>
<td>iv</td>
<td>128 bits input vector</td>
</tr>
<tr>
<td>in</td>
<td>tag_size</td>
<td>size of the authentication tag, this number must be between 1 and 16</td>
</tr>
<tr>
<td>out</td>
<td>tag_out</td>
<td>buffer for the generated authentication tag</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

| CRY_NOERROR | if the operation succeeded. |
| CRY_ERR_INV_ALGO | if the operation is unsupported on this device instance. |
| CRY_ERR_INV_KEY_TYPE | the selected key is invalid for this operation. |
| CRY_ERR_INV_KEY_ID | if the specified key identifier is invalid or refers to an empty key slot. |
| CRY_ERR_OP_FAILURE | if the operation failed, implementation dependent. |

Function Class:
Not an API, this function is for internal use only.

7.6.5.56 cry_lld_decrypt_AES_GCM()

cryerror_t cry_lld_decrypt_AES_GCM (  
    CRYDriver * cryp,
crykey_t key_id,
size_t auth_size,
const uint8_t * auth_in,
size_t text_size,
const uint8_t * text_in,
uint8_t * text_out,
const uint8_t * iv,
size_t tag_size,
const uint8_t * tag_in )

Decryption operation using AES-GCM.

Note
This is a stream cipher, there are no size restrictions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
</tr>
<tr>
<td>in</td>
<td>key_id</td>
</tr>
<tr>
<td>in</td>
<td>auth_size</td>
</tr>
<tr>
<td>in</td>
<td>auth_in</td>
</tr>
<tr>
<td>in</td>
<td>text_size</td>
</tr>
<tr>
<td>in</td>
<td>text_in</td>
</tr>
<tr>
<td>out</td>
<td>text_out</td>
</tr>
<tr>
<td>in</td>
<td>iv</td>
</tr>
<tr>
<td>in</td>
<td>tag_size</td>
</tr>
<tr>
<td>in</td>
<td>tag_in</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_AUTH_FAILED</td>
<td>authentication failed</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.
7.6.5.57  cry_lld_des_loadkey()

```c
cryerror_t cry_lld_des_loadkey (  
    CRYDriver * cryp,  
    size_t size,  
    const uint8_t * keyp )
```

Initializes the DES transient key.

**Note**

It is the underlying implementation to decide which key sizes are allowable.

**Parameters**

| in  | cryp | pointer to the CRYDriver object |
|     | size | key size in bytes               |
| in  | keyp | pointer to the key data         |

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the algorithm is unsupported.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>if the specified key size is invalid for the specified algorithm.</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

7.6.5.58  cry_lld_encrypt_DES()

```c
cryerror_t cry_lld_encrypt_DES (  
    CRYDriver * cryp,  
    crykey_t key_id,  
    const uint8_t * in,  
    uint8_t * out )
```

Encryption of a single block using (T)DES.

**Note**

The implementation of this function must guarantee that it can be called from any context.
7.6.59 cry_lld_decrypt_DES()

cryerror_t cry_lld_decrypt_DES (  
  CRYDriver * cryp,  
  crykey_t key_id,  
  const uint8_t * in,  
  uint8_t * out )

Decryption of a single block using (T)DES.

Note

The implementation of this function must guarantee that it can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.60 cry_lld_encrypt_DES_ECB()

cryerror_t cry_lld_encrypt_DES_ECB (cryDriver * cryp,
crykey_t key_id,
size_t size,
const uint8_t * in,
uint8_t * out )

Encryption operation using (T)DES-ECB.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in size</td>
<td>size of the plaintext buffer, this number must be a multiple of 8</td>
</tr>
<tr>
<td>in in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out out</td>
<td>buffer for the output ciphertext</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.61 cry_lld_decrypt_DES_ECB()

cryerror_t cry_lld_decrypt_DES_ECB (
  CRYDriver * cryp,
  crykey_t key_id,
  size_t size,
  const uint8_t * in,
  uint8_t * out )

Decryption operation using (T)DES-ECB.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in key_id</td>
<td>the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way</td>
</tr>
<tr>
<td>in size</td>
<td>size of the plaintext buffer, this number must be a multiple of 8</td>
</tr>
<tr>
<td>in in</td>
<td>buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out out</td>
<td>buffer for the output plaintext</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
7.6 Crypto Driver

Function Class:

Not an API, this function is for internal use only.

7.6.5.62 cry_lld_encrypt_DES_CBC()

cryerror_t cry_lld_encrypt_DES_CBC {
    CRYDriver * cryp,
    crykey_t key_id,
    size_t size,
    const uint8_t * in,
    uint8_t * out,
    const uint8_t * iv )

Encryption operation using (T)DES-CBC.

Note

The function operates on data buffers whose length is a multiple of an DES block, this means that padding
must be done by the caller.

Parameters

| in  | cryp      | pointer to the CRYDriver object |
| in  | key_id    | the key to be used for the operation, zero is the transient key, other values are keys stored in |
|     |           | an unspecified way               |
| in  | size      | size of the plaintext buffer, this number must be a multiple of 8 |
| in  | in        | buffer containing the input plaintext |
| out | out       | buffer for the output ciphertext |
| in  | iv        | 64 bits input vector            |

Returns

The operation status.

Return values

| CRY_NOERROR    | if the operation succeeded. |
| CRY_ERR_INV_ALGO | if the operation is unsupported on this device instance. |
| CRY_ERR_INV_KEY_TYPE | the selected key is invalid for this operation. |
| CRY_ERR_INV_KEY_ID | if the specified key identifier is invalid or refers to an empty key slot. |
| CRY_ERR_OP_FAILURE | if the operation failed, implementation dependent. |

Function Class:

Not an API, this function is for internal use only.
7.6.5.63  cry_lld_decrypt_DES_CBC()

cryerror_t cry_lld_decrypt_DES_CBC (  
    CRYDriver * cryp,  
    crykey_t key_id,  
    size_t size,  
    const uint8_t * in,  
    uint8_t * out,  
    const uint8_t * iv )

Decryption operation using (T)DES-CBC.

Note
The function operates on data buffers whose length is a multiple of an DES block, this means that padding must be done by the caller.

Parameters

| in  | cryp | pointer to the CRYDriver object |
| in  | key←_id | the key to be used for the operation, zero is the transient key, other values are keys stored in an unspecified way |
| in  | size | size of the plaintext buffer, this number must be a multiple of 8 |
| in  | in | buffer containing the input ciphertext |
| out | out | buffer for the output plaintext |
| in  | iv | 64 bits input vector |

Returns
The operation status.

Return values

| CRY_NOERROR | if the operation succeeded. |
| CRY_ERR_INV_ALGO | if the operation is unsupported on this device instance. |
| CRY_ERR_INV_KEY_TYPE | the selected key is invalid for this operation. |
| CRY_ERR_INV_KEY_ID | if the specified key identifier is invalid or refers to an empty key slot. |
| CRY_ERR_OP_FAILURE | if the operation failed, implementation dependent. |

Function Class:
Not an API, this function is for internal use only.

7.6.5.64  cry_lld_SHA1_init()

cryerror_t cry_lld_SHA1_init (  
    CRYDriver * cryp,  
    SHA1Context * sha1ctxp )

Hash initialization using SHA1.
7.6 Crypto Driver

Note

Use of this algorithm is not recommended because proven weak.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sha1ctxp</td>
<td>pointer to a SHA1 context to be initialized</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.65 cry_lld_SHA1_update()

cryerror_t cry_lld_SHA1_update (  
    CRYDriver * cryp,  
    SHA1Context * sha1ctxp,  
    size_t size,  
    const uint8_t * in )

Hash update using SHA1.

Note

Use of this algorithm is not recommended because proven weak.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sha1ctxp</td>
<td>pointer to a SHA1 context</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of input buffer</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input text</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
</table>
Return values

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.6.5.66 cry_lld_SHA1_final()

cryerror_t cry_lld_SHA1_final (  
   CRYDriver * cryp,  
   SHA1Context * sha1ctxp,  
   uint8_t * out )

Hash finalization using SHA1.

Note
Use of this algorithm is not recommended because proven weak.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>sha1ctxp pointer to a SHA1 context</td>
</tr>
<tr>
<td>out</td>
<td>out 160 bits output buffer</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.
7.6.5.67 cry_lld_SHA256_init()

cryerror_t cry_lld_SHA256_init (  
    CRYDriver * cryp,  
    SHA256Context * sha256ctxp )

Hash initialization using SHA256.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sha256ctxp</td>
<td>pointer to a SHA256 context to be initialized</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.68 cry_lld_SHA256_update()

cryerror_t cry_lld_SHA256_update (  
    CRYDriver * cryp,  
    SHA256Context * sha256ctxp,  
    size_t size,  
    const uint8_t * in )

Hash update using SHA256.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sha256ctxp</td>
<td>pointer to a SHA256 context</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of input buffer</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input text</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>If the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>If the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.6.5.69 cry_lld_SHA256_final()

cryerror_t cry_lld_SHA256_final (  
    CRYDriver * cryp,  
    SHA256Context * sha256ctxp,  
    uint8_t * out )

Hash finalization using SHA256.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in sha256ctxp</td>
<td>pointer to a SHA256 context</td>
</tr>
<tr>
<td>out out</td>
<td>256 bits output buffer</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>If the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>If the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>If the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.6.5.70 cry_lld_SHA512_init()

cryerror_t cry_lld_SHA512_init (  
    CRYDriver * cryp,  
    SHA512Context * sha512ctxp )

Hash initialization using SHA512.
7.6 Crypto Driver

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>crypt</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>out</td>
<td>sha512ctxp</td>
<td>pointer to a SHA512 context to be initialized</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>If the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>If the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>If the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.71 cry_lld_SHA512_update()

cryerror_t cry_lld_SHA512_update (  
  CRYDriver * cryp,  
  SHA512Context * sha512ctxp,  
  size_t size,  
  const uint8_t * in )

Hash update using SHA512.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>sha512ctxp</td>
<td>pointer to a SHA512 context</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of input buffer</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input text</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>If the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>If the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>If the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.6.5.72 cry_lld_SHA512_final()

```c
cryerror_t cry_lld_SHA512_final (  
    CRYDriver * cryp,
    SHA512Context * sha512ctxp,
    uint8_t * out )
```

Hash finalization using SHA512.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sha512ctxp</td>
<td>pointer to a SHA512 context</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>512 bits output buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.73 cry_lld_hmac_loadkey()

```c
cryerror_t cry_lld_hmac_loadkey (  
    CRYDriver * cryp,
    size_t size,
    const uint8_t * keep )
```

Initializes the HMAC transient key.

Note

It is the underlying implementation to decide which key sizes are allowable.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>key size in bytes</td>
</tr>
<tr>
<td>in</td>
<td>keep</td>
<td>pointer to the key data</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the algorithm is unsupported.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>if the specified key size is invalid for the specified algorithm.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.74 cry_lld_HMACSHA256_init()

cryerror_t cry_lld_HMACSHA256_init {
  CRYDriver * cryp,
  HMACSHA256Context * hmacsha256ctxp
}

Hash initialization using HMAC_SHA256.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp pointer to the CRYDriver object</td>
</tr>
<tr>
<td>out</td>
<td>hmacsha256ctxp pointer to a HMAC_SHA256 context to be initialized</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.75 cry_lld_HMACSHA256_update()

cryerror_t cry_lld_HMACSHA256_update {
  CRYDriver * cryp,
  HMACSHA256Context * hmacsha256ctxp
}
Hash update using HMAC.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>hmacsha256ctxp</td>
<td>pointer to a HMAC_SHA256 context</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of input buffer</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input text</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.6.5.76 cry_lld_HMACSHA256_final()

cryerror_t cry_lld_HMACSHA256_final (  
    CRYDriver * cryp,  
    HMACSHA256Context * hmacsha256ctxp,  
    uint8_t * out )

Hash finalization using HMAC.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>hmacsha256ctxp</td>
<td>pointer to a HMAC_SHA256 context</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>256 bits output buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.
Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.6.5.77 cry_lld_HMACSHA512_init()

cryerror_t cry_lld_HMACSHA512_init {
  CRYDriver * cryp,
  HMACSHA512Context * hmacsha512ctxp
}

Hash initialization using HMAC_SHA512.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>hmacsha512ctxp</td>
<td>pointer to a HMAC_SHA512 context to be initialized</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th>CRY_NOERROR</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.6.5.78 cry_lld_HMACSHA512_update()

cryerror_t cry_lld_HMACSHA512_update (  
  CRYDriver * cryp,  
  HMACSHA512Context * hmacsha512ctxp,  
  size_t size,  
  const uint8_t * in )

Hash update using HMAC.
### cry_lld_HMACSHA512_final()

```
cryerror_t cry_lld_HMACSHA512_final (  
  CRYDriver * cryp,  
  HMACSHA512Context * hmacsha512ctxp,  
  uint8_t * out )
```

Hash finalization using HMAC.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp: pointer to the CRYDriver object</td>
</tr>
<tr>
<td>in</td>
<td>hmacsha512ctxp: pointer to a HMAC_SHA512 context</td>
</tr>
<tr>
<td>out</td>
<td>out: 512 bits output buffer</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **CRY_NOERROR**: if the operation succeeded.
- **CRY_ERR_INV_ALGO**: if the operation is unsupported on this device instance.
- **CRY_ERR_OP_FAILURE**: if the operation failed, implementation dependent.

**Function Class:**

Not an API, this function is for internal use only.
7.6 Crypto Driver

Function Class:

Not an API, this function is for internal use only.

7.6.6 Variable Documentation

7.6.6.1 CRYD1

CRYDriver CRYD1

CRY1 driver identifier.
7.7 DAC Driver

Generic DAC Driver.

7.7.1 Detailed Description

Generic DAC Driver.

This module implements a generic DAC (Digital to Analog Converter) driver.

Precondition

In order to use the DAC driver the `HAL_USE_DAC` option must be enabled in `halconf.h`.

Macros

- `#define DAC_MAX_CHANNELS 2`
  Maximum number of DAC channels per unit.
- `#define dac_lld_driver_fields`
  Low level fields of the DAC driver structure.
- `#define dac_lld_config_fields`
  Low level fields of the DAC configuration structure.
- `#define dac_lld_conversion_group_fields`
  Low level fields of the DAC group configuration structure.

DAC configuration options

- `#define DAC_USE_WAIT TRUE`
  Enables synchronous APIs.
- `#define DAC_USE_MUTUAL_EXCLUSION TRUE`
  Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

Low level driver helper macros

- `#define dacIsBufferComplete(dacp) ((bool)((dacp)->state == DAC_COMPLETE))`
  Buffer state.
- `#define _dac_wait_s(dacp) osalThreadSuspendS(&(dacp)->thread)`
  Waits for operation completion.
- `#define _dac_reset_i(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)`
  Resumes a thread waiting for a conversion completion.
- `#define _dac_reset_s(dacp) osalThreadResumeS(&(dacp)->thread, MSG_RESET)`
  Resumes a thread waiting for a conversion completion.
- `#define _dac_wakeup_isr(dacp)`
  Wakes up the waiting thread.
- `#define _dac_timeout_isr(dacp)`
  Wakes up the waiting thread with a timeout message.
- `#define _dac_isr_half_code(dacp)`
  Common ISR code, half buffer event.
- `#define _dac_isr_full_code(dacp)`
  Common ISR code, full buffer event.
- `#define _dac_isr_error_code(dacp, err)`
  Common ISR code, error event.
Configuration options

- `#define PLATFORM_DAC_USE_DAC1 FALSE`
  
  DAC1 CH1 driver enable switch.

Typedefs

- `typedef struct hal_dac_driver DACDriver`
  
  Type of a structure representing an DAC driver.

- `typedef struct hal_dac_config DACConfig`
  
  Type of a structure representing an DAC driver configuration.

- `typedef struct hal_dac_conversion_group DACConversionGroup`
  
  Type of a DAC conversion group.

- `typedef void(* daccallback_t) (DACDriver *dacp)`
  
  DAC notification callback type.

- `typedef void(* dacerrorcallback_t) (DACDriver *dacp, dacerror_t err)`
  
  DAC error callback type.

- `typedef uint32_t dacchannel_t`
  
  Type of a DAC channel index.

- `typedef uint16_t dacsample_t`
  
  Type representing a DAC sample.

Data Structures

- `struct hal_dac_conversion_group`
  
  DAC Conversion group structure.

- `struct hal_dac_config`
  
  Driver configuration structure.

- `struct hal_dac_driver`
  
  Structure representing a DAC driver.

Functions

- `void dacInit (void)`
  
  DAC Driver initialization.

- `void dacObjectInit (DACDriver *dacp)`
  
  Initializes the standard part of a DACDriver structure.

- `msg_t dacStart (DACDriver *dacp, const DACConfig *config)`
  
  Configures and activates the DAC peripheral.

- `void dacStop (DACDriver *dacp)`
  
  Deactivates the DAC peripheral.

- `void dacPutChannelX (DACDriver *dacp, dacchannel_t channel, dacsample_t sample)`
  
  Outputs a value directly on a DAC channel.

- `void dacStartConversion (DACDriver *dacp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)`
  
  Starts a DAC conversion.

- `void dacStartConversionI (DACDriver *dacp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)`
  
  Starts a DAC conversion.
• void dacStopConversion (DACDriver *dacp)
  Stops an ongoing conversion.
• void dacStopConversionI (DACDriver *dacp)
  Stops an ongoing conversion.
• msg_t dacConvert (DACDriver *dacp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)
  Performs a DAC conversion.
• void dacAcquireBus (DACDriver *dacp)
  Gains exclusive access to the DAC bus.
• void dacReleaseBus (DACDriver *dacp)
  Releases exclusive access to the DAC bus.
• void dac_lld_init (void)
  Low level DAC driver initialization.
• void dac_lld_start (DACDriver *dacp)
  Configures and activates the DAC peripheral.
• void dac_lld_stop (DACDriver *dacp)
  Deactivates the DAC peripheral.
• void dac_lld_put_channel (DACDriver *dacp, dacchannel_t channel, dacsample_t sample)
  Outputs a value directly on a DAC channel.
• void dac_lld_start_conversion (DACDriver *dacp)
  Starts a DAC conversion.
• void dac_lld_stop_conversion (DACDriver *dacp)
  Stops an ongoing conversion.

Enumerations
• enum dacstate_t {
  DAC_UNINIT = 0, DAC_STOP = 1, DAC_READY = 2, DAC_ACTIVE = 3,
  DAC_COMPLETE = 4, DAC_ERROR = 5 }
  Driver state machine possible states.
• enum dacerror_t { DAC_ERR_DMAFAILURE = 0, DAC_ERR_UNDERFLOW = 1 }
  Possible DAC failure causes.

Variables
• DACDriver DACD1
  DAC1 driver identifier.

7.7.2 Macro Definition Documentation
7.7.2.1 DAC_USE_WAIT

#define DAC_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.
7.7 DAC Driver

7.7.2 DAC_USE_MUTUAL_EXCLUSION

#define DAC_USE_MUTUAL_EXCLUSION TRUE

Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

Note
Disabling this option saves both code and data space.

7.7.2.3 dacIsBufferComplete

#define dacIsBufferComplete(
    dscp
) (bool)(dscp)->state == DAC_COMPLETE)

Buffer state.

Note
This function is meant to be called from the DAC callback only.

Parameters

| in   | dscp | pointer to the DACDriver object |

Returns

The buffer state.

Return values

- false if the driver filled/sent the first half of the buffer.
- true if the driver filled/sent the second half of the buffer.

Function Class:

Special function, this function has special requirements see the notes.

7.7.2.4 _dac_wait_s

#define _dac_wait_s(
    dscp
) osalThreadSuspendS(&(dscp)->thread)

Waits for operation completion.

This function waits for the driver to complete the current operation.
Precondition

An operation must be running while the function is invoked.

Note

No more than one thread can wait on a DAC driver using this function.

Parameters

| in | dacp | pointer to the DACDriver object |

Function Class:

Not an API, this function is for internal use only.

7.7.2.5 _dac_reset_i

```c
#define _dac_reset_i( 
    dacp ) osalThreadResumeI(&(dacp)->thread, MSG_RESET)
```

Resumes a thread waiting for a conversion completion.

Parameters

| in | dacp | pointer to the DACDriver object |

Function Class:

Not an API, this function is for internal use only.

7.7.2.6 _dac_reset_s

```c
#define _dac_reset_s( 
    dacp ) osalThreadResumeS(&(dacp)->thread, MSG_RESET)
```

Resumes a thread waiting for a conversion completion.

Parameters

| in | dacp | pointer to the DACDriver object |
Function Class:
Not an API, this function is for internal use only.

7.7.2.7 _dac_wakeup_isr

#define _dac_wakeup_isr(
  dacp )

Value:
{ 
  osalSysLockFromISR();
  osalThreadResumeI(&(dacp)->thread, MSG_OK);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>dacp</th>
<th>pointer to the DACDriver object</th>
</tr>
</thead>
</table>

Function Class:
Not an API, this function is for internal use only.

7.7.2.8 _dac_timeout_isr

#define _dac_timeout_isr(
  dacp )

Value:
{ 
  osalSysLockFromISR();
  osalThreadResumeI(&(dacp)->thread, MSG_TIMEOUT);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread with a timeout message.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>dacp</th>
<th>pointer to the DACDriver object</th>
</tr>
</thead>
</table>

Function Class:
Not an API, this function is for internal use only.
7.7.2.9 _dac_isr_half_code

#define _dac_isr_half_code(
dacp)

Value:
{
  if ((dacp)->grpp->end_cb != NULL) {
    (dacp)->grpp->end_cb(dacp);
  }
}

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

Note
This macro is meant to be used in the low level drivers implementation only.

Parameters

| in  | dacp | pointer to the DACDriver object |

Function Class:
Not an API, this function is for internal use only.

7.7.2.10 _dac_isr_full_code

#define _dac_isr_full_code(
dacp)

Value:
{
  if ((dacp)->grpp->end_cb) {
    (dacp)->state = DAC_COMPLETE;
    (dacp)->grpp->end_cb(dacp);
    if ((dacp)->state == DAC_COMPLETE)
      (dacp)->state = DAC_ACTIVE;
  }
}

Common ISR code, full buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Driver state transitions.

Note
This macro is meant to be used in the low level drivers implementation only.
7.7 DAC Driver

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dcp</td>
<td>pointer to the DACDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.7.2.11 _dac_isr_error_code

#define _dac_isr_error_code(
    dcp,
    err
)

Value:

\[
\text{dac_lld_stop_conversion}(dcp);
\]
\[
\text{if } ((dcp)->grpp->error_cb != \text{NULL}) \{ \text{\quad} \\
    (dcp)->state = \text{DAC_ERROR};
    (dcp)->grpp->error_cb(dcp, err);
    \text{if } ((dcp)->state == \text{DAC_ERROR})
    (dcp)->state = \text{DAC_READY};
\} \\
(dcp)->grpp = \text{NULL};
\text{_dac_timeout_isr}(dcp);
\]

Common ISR code, error event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread timeout signaling, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dcp</td>
<td>pointer to the DACDriver object</td>
</tr>
<tr>
<td>in</td>
<td>err</td>
<td>platform dependent error code</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.7.2.12 DAC_MAX_CHANNELS

#define DAC_MAX_CHANNELS 2

Maximum number of DAC channels per unit.

7.7.2.13 PLATFORM_DAC_USE_DAC1

#define PLATFORM_DAC_USE_DAC1 FALSE

DAC1 CH1 driver enable switch. If set to TRUE the support for DAC1 channel 1 is included.

Note
The default is FALSE.

7.7.2.14 dac_lld_driver_fields

#define dac_lld_driver_fields

Value:
/* Dummy field, it is not needed.*/

uint32_t dummy

Low level fields of the DAC driver structure.

7.7.2.15 dac_lld_config_fields

#define dac_lld_config_fields

Value:
/* Dummy configuration, it is not needed.*/

uint32_t dummy

Low level fields of the DAC configuration structure.

7.7.2.16 dac_lld_conversion_group_fields

#define dac_lld_conversion_group_fields

Value:
/* Dummy configuration, it is not needed.*/

uint32_t dummy

Low level fields of the DAC group configuration structure.
7.7 DAC Driver

7.7.3 Typedef Documentation

7.7.3.1 DACDriver

typedef struct hal_dac_driver DACDriver

Type of a structure representing an DAC driver.

7.7.3.2 DACConfig

typedef struct hal_dac_config DACConfig

Type of a structure representing an DAC driver configuration.

7.7.3.3 DACConversionGroup

typedef struct hal_dac_conversion_group DACConversionGroup

Type of a DAC conversion group.

7.7.3.4 daccallback_t

typedef void(* daccallback_t) (DACDriver *dcp)

DAC notification callback type.

Parameters

| in | dcp | pointer to the DACDriver object triggering the |

7.7.3.5 dacerrorcallback_t

typedef void(* dacerrorcallback_t) (DACDriver *dcp, dacerror_t err)

DAC error callback type.
Parameters

| in  | dacp | pointer to the DACDriver object triggering the callback |
| in  | err  | DAC error code |

### 7.7.3.6 dacchannel_t

typedef uint32_t dacchannel_t

Type of a DAC channel index.

### 7.7.3.7 dacsample_t

typedef uint16_t dacsample_t

Type representing a DAC sample.

### 7.7.4 Enumeration Type Documentation

#### 7.7.4.1 dacstate_t

enum dacstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>DAC_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>DAC_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>DAC_ACTIVE</td>
<td>Exchanging data.</td>
</tr>
<tr>
<td>DAC_COMPLETE</td>
<td>Asynchronous operation complete.</td>
</tr>
<tr>
<td>DAC_ERROR</td>
<td>Error.</td>
</tr>
</tbody>
</table>
7.7 DAC Driver

7.7.4.2 dacerror_t

enum dacerror_t

Possible DAC failure causes.

Note

Error codes are architecture dependent and should not relied upon.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_ERR_DMAFAILURE</td>
<td>DMA operations failure.</td>
</tr>
<tr>
<td>DAC_ERR_UNDERFLOW</td>
<td>DAC overflow condition.</td>
</tr>
</tbody>
</table>

7.7.5 Function Documentation

7.7.5.1 dacInit()

void dacInit ( )

DAC Driver initialization.

Note

This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.7.5.2 dacObjectInit()

void dacObjectInit ( DACDriver ∗ dcp )

Initializes the standard part of a DACDriver structure.
Parameters

| out | dacp | pointer to the DACDriver object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call graph diagram]

7.7.5.3 dacStart()

```c
msg_t dacStart (  
    DACDriver * dacp,
    const DACConfig * config)
```

Configures and activates the DAC peripheral.

Parameters

| in | dacp | pointer to the DACDriver object |
| in | config | pointer to the DACConfig object, it can be NULL if the low level driver implementation supports a default configuration |

Returns

The operation status.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
dacStart
    dac_lld_start
    osalSysLock
```

### 7.7.5.4 dacStop()

```c
void dacStop (DACDriver *dcp)
```

Deactivates the DAC peripheral.

**Note**

Deactivating the peripheral also enforces a release of the slave select line.

**Parameters**

- `in dcp` pointer to the DACDriver object
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    dacStop
       |
    +---- dac_lld_stop
         |
    +---- osalSysLock
         |
    +---- osalSysUnlock
```

7.7.5.5 dacPutChannelX()

```c
void dacPutChannelX ( 
    DACDriver * dacp,
    dacchannel_t channel,
    dacsample_t sample )
```

Outputs a value directly on a DAC channel.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dacp</code></td>
<td>pointer to the DACDriver object</td>
</tr>
<tr>
<td><code>channel</code></td>
<td>DAC channel number</td>
</tr>
<tr>
<td><code>sample</code></td>
<td>value to be output</td>
</tr>
</tbody>
</table>

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:

```
    dacPutChannelX
       |
    +---- dac_lld_put_channel
```
7.7.5.6 dacStartConversion()

void dacStartConversion (  
    DACDriver * dacp,  
    const DACConversionGroup * grpp,  
    dacsample_t * samples,  
    size_t depth  )

Starts a DAC conversion.

Starts an asynchronous conversion operation.

Note

The buffer is organized as a matrix of $M \times N$ elements where $M$ is the channels number configured into the conversion group and $N$ is the buffer depth. The samples are sequentially written into the buffer with no gaps.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dacp</th>
<th>pointer to the DACDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>grpp</td>
<td>pointer to a DACConversionGroup object</td>
</tr>
<tr>
<td>in</td>
<td>samples</td>
<td>pointer to the samples buffer</td>
</tr>
<tr>
<td>in</td>
<td>depth</td>
<td>buffer depth (matrix rows number). The buffer depth must be one or an even number.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
7.7.5.7 dacStartConversionI()

void dacStartConversionI (  
    DACDriver * dacp,  

ChibiOS/HAL
const DACConversionGroup * grpp,
dacsample_t * samples,
size_t depth )

Starts a DAC conversion.

Starts an asynchronous conversion operation.

Postcondition

The callbacks associated to the conversion group will be invoked on buffer fill and error events.

Note

The buffer is organized as a matrix of M×N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in dacp</td>
<td>pointer to the DACDriver object</td>
</tr>
<tr>
<td>in grpp</td>
<td>pointer to a DACConversionGroup object</td>
</tr>
<tr>
<td>in samples</td>
<td>pointer to the samples buffer</td>
</tr>
<tr>
<td>in depth</td>
<td>buffer depth (matrix rows number). The buffer depth must be one or an even number.</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```plaintext
| dacStartConversionI | dac_lld_start_conversion |
```

7.7.5.8 dacStopConversion()

void dacStopConversion (DACDriver * dacp )

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the DAC_READY state. If there was no conversion being processed then the function does nothing.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in dacp</strong></td>
<td>pointer to the DACDriver object</td>
</tr>
</tbody>
</table>

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.7.5.9 dacStopConversionI()

```c
void dacStopConversionI ( DACDriver * dacp )
```

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the DAC READY state. If there was no conversion being processed then the function does nothing.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in dacp</strong></td>
<td>pointer to the DACDriver object</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
+-------------------+-------------------+
| dacStopConversionI| dac_lld_stop_conversion|
```

7.7.5.10 dacConvert()

```
msg_t dacConvert (
    DACDriver * dcp,
    const DACConversionGroup * grpp,
    dacsample_t * samples,
    size_t depth )
```

Performs a DAC conversion.
Performs a synchronous conversion operation.

Note

The buffer is organized as a matrix of M×N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

Parameters

| in  | dacp    | pointer to the DACDriver object |
| in  | grpp    | pointer to a DACConversionGroup object |
| out | samples | pointer to the samples buffer |
| in  | depth   | buffer depth (matrix rows number). The buffer depth must be one or an even number. |

Returns

The operation result.

Return values

| MSG_OK   | Conversion finished. |
| MSG_RESET | The conversion has been stopped using acdStopConversion() or acdStopConversionI(), the result buffer may contain incorrect data. |
| MSG_TIMEOUT | The conversion has been stopped because an hardware error. |
7.7 DAC Driver

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.7.5.11 dacAcquireBus()

```c
void dacAcquireBus (  
    DACDriver * dacp 
)
```

Gains exclusive access to the DAC bus.

This function tries to gain ownership to the DAC bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option DAC_USE_MUTUAL_EXCLUSION must be enabled.

Parameters

| in | dacp | pointer to the DACDriver object |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.7.5.12 dacReleaseBus()

```c
void dacReleaseBus (DACDriver * dcp)
```

Releases exclusive access to the DAC bus.

**Precondition**

In order to use this function the option `DAC_USE_MUTUAL_EXCLUSION` must be enabled.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>dcp</code></td>
<td>pointer to the DACDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)
7.7 DAC Driver

7.7.5.13 dac_lld_init()

```c
void dac_lld_init (  
    void  )
```

Low level DAC driver initialization.

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
+----------------+----------------+----------------+
| dac_lld_init   | dacObjectInit  | osalMutexObjectInit |
+----------------+----------------+----------------+
```

7.7.5.14 dac_lld_start()

```c
void dac_lld_start (  
    DACDriver  * dacp  )
```

Configures and activates the DAC peripheral.

**Parameters**

| in  | dacom | pointer to the DACDriver object |

Function Class:
Not an API, this function is for internal use only.

7.7.5.15 dac_lld_stop()

```c
void dac_lld_stop (  
    DACDriver  * dacp  )
```

Deactivates the DAC peripheral.

ChibiOS/HAL
### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dacp</td>
<td>pointer to the DACDriver object</td>
</tr>
</tbody>
</table>

### Function Class:

Not an API, this function is for internal use only.

#### 7.7.5.16 dac_lld_put_channel()

```c
void dac_lld_put_channel (  
    DACDriver * dacp,  
    dacchannel_t channel,  
    dacsample_t sample )
```

Outputs a value directly on a DAC channel.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dacp</td>
<td>pointer to the DACDriver object</td>
</tr>
<tr>
<td>in</td>
<td>channel</td>
<td>DAC channel number</td>
</tr>
<tr>
<td>in</td>
<td>sample</td>
<td>value to be output</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

#### 7.7.5.17 dac_lld_start_conversion()

```c
void dac_lld_start_conversion (  
    DACDriver * dacp )
```

Starts a DAC conversion.

Starts an asynchronous conversion operation.

### Note

In DAC_DHRM_8BIT_RIGHT mode the parameters passed to the callback are wrong because two samples are packed in a single dacsample_t element. This will not be corrected, do not rely on those parameters.

In DAC_DHRM_8BIT_RIGHT_DUAL mode two samples are treated as a single 16 bits sample and packed into a single dacsample_t element. The num_channels must be set to one in the group conversion configuration structure.
7.7 DAC Driver

Parameters

| in  | dcp   | pointer to the DACDriver object |

Function Class:

Not an API, this function is for internal use only.

7.7.5.18 dac_lld_stop_conversion()

```c
void dac_lld_stop_conversion (  
    DACDriver * dcp  )
```

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the DAC READY state. If there was no conversion being processed then the function does nothing.

Parameters

| in  | dcp   | pointer to the DACDriver object |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.7.6 Variable Documentation

7.7.6.1 DACD1

```c
DACDriver DACD1
```

DAC1 driver identifier.
7.8 EFL Driver

Generic Embedded Flash Driver.

7.8.1 Detailed Description

Generic Embedded Flash Driver.

This module implements a generic embedded flash driver.

Precondition

In order to use the EFL driver the HAL_USE_EFL option must be enabled in halconf.h.

Macros

- \#define EFL_USE_MUTUAL_EXCLUSION TRUE
  Enables the eflAcquireUse() and eflReleaseUse() APIs.
- \#define _efl_flash_methods_alone
  EFlashDriver specific methods.
- \#define _efl_flash_methods
  EFlashDriver specific methods with inherited ones.
- \#define _efl_driver_data
  EFlashDriver specific data.
- \#define efl_lld_driver_fields
  Low level fields of the embedded flash driver structure.
- \#define efl_lld_config_fields
  Low level fields of the embedded flash configuration structure.

PLATFORM configuration options

- \#define PLATFORM_EFL_USE_EFL1 FALSE
  EFL1 driver enable switch.

Typedefs

- typedef struct hal_efl_driver EFlashDriver
  Type of external flash driver class.
- typedef struct hal_efl_config EFlashConfig
  Type of a structure representing a flash driver configuration.

Data Structures

- struct EFlashDriverVMT
  EFlash virtual methods table.
- struct hal_efl_config
  Type of a structure representing a flash driver configuration.
- struct hal_efl_driver
  Structure representing an embedded flash driver.
Functions

- void eflInit (void)
  
  Embedded Flash Driver initialization.

- void eflObjectInit (EFlashDriver *eflp)

  Initializes a generic EFlashDriver object.

- msg_t eflStart (EFlashDriver *eflp, const EFlashConfig *config)

  Configures and starts the driver.

- void eflStop (EFlashDriver *eflp)

  Stops the driver.

- void efl_lld_init (void)

  Low level Embedded Flash driver initialization.

- void efl_lld_start (EFlashDriver *eflp)

  Configures and activates the Embedded Flash peripheral.

- void efl_lld_stop (EFlashDriver *eflp)

  Deactivates the Embedded Flash peripheral.

- const flash_descriptor_t *efl_lld_get_descriptor (void *instance)

  Gets the flash descriptor structure.

- flash_error_t efl_lld_read (void *instance, flash_offset_t offset, size_t n, uint8_t *rp)

  Read operation.

- flash_error_t efl_lld_program (void *instance, flash_offset_t offset, size_t n, const uint8_t *pp)

  Program operation.

- flash_error_t efl_lld_start_erase_all (void *instance)

  Starts a whole-device erase operation.

- flash_error_t efl_lld_start_erase_sector (void *instance, flash_sector_t sector)

  Starts an sector erase operation.

- flash_error_t efl_lld_query_erase (void *instance, uint32_t *msec)

  Queries the driver for erase operation progress.

- flash_error_t efl_lld_verify_erase (void *instance, flash_sector_t sector)

  Returns the erase state of a sector.

Variables

- EFlashDriver EFLD1

  EFL1 driver identifier.

7.8.2  Macro Definition Documentation

7.8.2.1  EFL_USE_MUTUAL_EXCLUSION

#define EFL_USE_MUTUAL_EXCLUSION TRUE

Enables the eflAcquireUse() and eflReleaseUse() APIs.

Note

Disabling this option saves both code and data space.
7.8.2.2  _efl_flash_methods_alone

#define _efl_flash_methods_alone

EFlashDriver specific methods.

7.8.2.3  _efl_flash_methods

#define _efl_flash_methods

Value:

_base_flash_methods
_efl_flash_methods_alone

EFlashDriver specific methods with inherited ones.

7.8.2.4  _efl_driver_data

#define _efl_driver_data

Value:

_base_flash_data
/* Current configuration data.*/
const EFlashConfig *config;

EFlashDriver specific data.

7.8.2.5  PLATFORM_EFL_USE_EFL1

#define PLATFORM_EFL_USE_EFL1 FALSE

EFL1 driver enable switch.

If set to TRUE the support for EFL1 is included.

Note

The default is FALSE.
7.8 EFL Driver

7.8.2.6 efl_lld_driver_fields

#define efl_lld_driver_fields

Value:
/** Dummy field, it is not needed. */
uint32_t dummy

Low level fields of the embedded flash driver structure.

7.8.2.7 efl_lld_config_fields

#define efl_lld_config_fields

Value:
/** Dummy configuration, it is not needed. */
uint32_t dummy

Low level fields of the embedded flash configuration structure.

7.8.3 Typedef Documentation

7.8.3.1 EFlashDriver

typedef struct hal_efl_driver EFlashDriver

Type of external flash driver class.

7.8.3.2 EFlashConfig

typedef struct hal_efl_config EFlashConfig

Type of a structure representing a flash driver configuration.

7.8.4 Function Documentation
7.8.4.1 eflInit()

void eflInit (  
    void )

Embedded Flash Driver initialization.

Note
This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.

Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
    eflInit  →  efl_lld_init  →  eflObjectInit
```

7.8.4.2 eflObjectInit()

void eflObjectInit (  
    EFlashDriver * eflp )

Initializes a generic EFlashDriver object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out eflp</td>
<td>pointer to a EFlashDriver structure</td>
</tr>
</tbody>
</table>

Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.8.4.3 eflStart()

msg_t eflStart (  
    EFlashDriver * eflp,  
    const EFlashConfig * config )
Configures and starts the driver.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>eflp</th>
<th>pointer to a EFlashDriver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to a configuration structure. If this parameter is set to NULL then a default configuration is used.</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    eflStart —> osalSysLock
```

7.8.4.4 eflStop()

```c
void eflStop (EFlashDriver * eflp )
```

Stops the driver.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>eflp</th>
<th>pointer to a EFlashDriver structure</th>
</tr>
</thead>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

7.8.4.5 efl_lld_init()

```c
void efl_lld_init ( )
```

Low level Embedded Flash driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

![Call Graph](image)

7.8.4.6 efl_lld_start()

```c
void efl_lld_start ( EFlashDriver * ep )
```

Configures and activates the Embedded Flash peripheral.
Parameters

| in  | eflp | pointer to a EFlashDriver structure |

Function Class:

Not an API, this function is for internal use only.

### 7.8.4.7 efl_lld_stop()

```c
void efl_lld_stop ( 
    EFlashDriver ∗ eflp
)
```

Deactivates the Embedded Flash peripheral.

Parameters

| in  | eflp | pointer to a EFlashDriver structure |

Function Class:

Not an API, this function is for internal use only.

### 7.8.4.8 efl_lld_get_descriptor()

```c
const flash_descriptor_t ∗ efl_lld_get_descriptor ( 
    void ∗ instance
)
```

Gets the flash descriptor structure.

Parameters

| in  | instance | pointer to a EFlashDriver instance |

Returns

A flash device descriptor.

Function Class:

Not an API, this function is for internal use only.
### 7.8.4.9 efl_lld_read()

```c
flash_error_t efl_lld_read (  
    void * instance,  
    flash_offset_t offset,  
    size_t n,  
    uint8_t * rp )
```

Read operation.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>instance</th>
<th>pointer to a EFlashDriver instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>offset</td>
<td>flash offset</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to be read</td>
</tr>
<tr>
<td>out</td>
<td>rp</td>
<td>pointer to the data buffer</td>
</tr>
</tbody>
</table>

**Returns**

An error code.

**Return values**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_READ</td>
<td>if the read operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

### 7.8.4.10 efl_lld_program()

```c
flash_error_t efl_lld_program (  
    void * instance,  
    flash_offset_t offset,  
    size_t n,  
    const uint8_t * pp )
```

Program operation.

**Note**

The device supports ECC, it is only possible to write erased pages once except when writing all zeroes.
7.8 EFL Driver

Parameters

| in  | instance   | pointer to a EFlashDriver instance |
| in  | offset     | flash offset                      |
| in  | n          | number of bytes to be programmed  |
| in  | pp         | pointer to the data buffer        |

Returns

An error code.

Return values

| FLASH_NO_ERROR | if there is no erase operation in progress. |
| FLASH_BUSY_ERASING | if there is an erase operation in progress. |
| FLASH_ERROR_PROGRAM | if the program operation failed. |
| FLASH_ERROR_HW_FAILURE | if access to the memory failed. |

Function Class:

Not an API, this function is for internal use only.

7.8.4.11 efl_lld_start_erase_all()

flash_error_t efl_lld_start_erase_all ( void * instance )

Starts a whole-device erase operation.

Note

This function only erases bank 2 if it is present. Bank 1 is not touched because it is where the program is running on. Pages on bank 1 can be individually erased.

Parameters

| in  | instance   | pointer to a EFlashDriver instance |

Returns

An error code.

Return values

| FLASH_NO_ERROR | if there is no erase operation in progress. |
| FLASH_BUSY_ERASING | if there is an erase operation in progress. |
| FLASH_ERROR_HW_FAILURE | if access to the memory failed. |
Function Class:
Not an API, this function is for internal use only.

7.8.4.12 efl_lld_start_erase_sector()

```c
flash_error_t efl_lld_start_erase_sector (  
    void * instance,  
    flash_sector_t sector  
)
```

Starts an sector erase operation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>instance</th>
<th>pointer to a EFlashDriver instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sector</td>
<td>sector to be erased</td>
</tr>
</tbody>
</table>

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.8.4.13 efl_lld_query_erase()

```c
flash_error_t efl_lld_query_erase {  
    void * instance,  
    uint32_t * msec  
}
```

Queries the driver for erase operation progress.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>instance</th>
<th>pointer to a EFlashDriver instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msec</td>
<td>recommended time, in milliseconds, that should be spent before calling this function again, can be NULL</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.
Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_ERASE</td>
<td>if the erase operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.8.4.14  efl_lld_verify_erase()

flash_error_t efl_lld_verify_erase ( void * instance, flash_sector_t sector )

Returns the erase state of a sector.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in instance</td>
<td>pointer to a EFlashDriver instance</td>
</tr>
<tr>
<td>in sector</td>
<td>sector to be verified</td>
</tr>
</tbody>
</table>

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if the sector is erased.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_VERIFY</td>
<td>if the verify operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.8.5  Variable Documentation
7.8.5.1  EFLD1

EFlashDriver EFLD1

EFL1 driver identifier.
7.9 Generic Flash Interface

HAL Generic Flash Driver Interface.

7.9.1 Detailed Description

HAL Generic Flash Driver Interface.

7.9.2 Driver State Machine

The flash driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

![State Machine Diagram]

Macros

- **#define _base_flash_methods_alone**
  
  BaseFlash specific methods.

- **#define _base_flash_methods**
  
  BaseFlash specific methods with inherited ones.

- **#define _base_flash_data**
  
  BaseFlash specific data.
Flash attributes

- `#define FLASH_ATTR_ERASED_IS_ONE 0x00000001U`
  Defines one as the erased bit state.
- `#define FLASH_ATTR_MEMORY_MAPPED 0x00000002U`
  The memory is accessible in a memory mapped mode.
- `#define FLASH_ATTR_REWRITABLE 0x00000004U`
  Programmed pages can be programmed again.
- `#define FLASH_ATTR_ECC_CAPABLE 0x00000008U`
  The memory is protected by an ECC mechanism.
- `#define FLASH_ATTR_ECC_ZERO_LINE_CAPABLE 0x00000010U`
  The device is able to overwrite zero to a line.
- `#define FLASH_ATTR_SUSPEND_ERASE_CAPABLE 0x00000020U`
  The device is able to suspend erase operations.

Macro Functions (BaseFlash)

- `#define getBaseFlash(ip) ((BaseFlash *) (ip)->vmt)`
  Instance getter.
- `#define flashGetDescriptor(ip) (ip)->vmt->get_descriptor(ip)`
  Gets the flash descriptor structure.
- `#define flashRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)`
  Read operation.
- `#define flashProgram(ip, offset, n, pp) (ip)->vmt->program(ip, offset, n, pp)`
  Program operation.
- `#define flashStartEraseAll(ip) (ip)->vmt->start_erase_all(ip)`
  Starts a whole-device erase operation.
- `#define flashStartEraseSector(ip, sector) (ip)->vmt->start_erase_sector(ip, sector)`
  Starts an sector erase operation.
- `#define flashQueryErase(ip, msec) (ip)->vmt->query_erase(ip, msec)`
  Queries the driver for erase operation progress.
- `#define flashVerifyErase(ip, sector) (ip)->vmt->verify_erase(ip, sector)`
  Returns the erase state of a sector.
- `#define flashAcquireExclusive(ip) (ip)->vmt->acquire_exclusive(ip)`
  Acquires exclusive access to flash.
- `#define flashReleaseExclusive(ip) (ip)->vmt->release_exclusive(ip)`
  Releases exclusive access to flash.

Typedefs

- `typedef uint32_t flash_offset_t`
  Type of a flash offset.
- `typedef uint32_t flash_sector_t`
  Type of a flash sector number.
Data Structures

- struct flash_sector_descriptor_t
  
  Flash sector descriptor.

- struct flash_descriptor_t
  
  Type of a flash device descriptor.

- struct BaseFlashVMT
  
  BaseFlash virtual methods table.

- struct BaseFlash
  
  Base flash class.

Functions

- flash_error_t flashWaitErase (BaseFlash *devp)
  
  Waits until the current erase operation is finished.

- flash_offset_t flashGetSectorOffset (BaseFlash *devp, flash_sector_t sector)
  
  Returns the offset of a sector.

- uint32_t flashGetSectorSize (BaseFlash *devp, flash_sector_t sector)
  
  Returns the size of a sector.

- flash_sector_t flashGetOffsetSector (BaseFlash *devp, flash_offset_t offset)
  
  Returns the sector of an offset.

- void * flashGetOffsetAddress (BaseFlash *devp, flash_offset_t offset)
  
  Get absolute address from offset.

- flash_offset_t flashGetAddressOffset (BaseFlash *devp, void *addr)
  
  Get offset from absolute address.

Enumerations

- enum flash_state_t
  
  Driver state machine possible states.

- enum flash_error_t
  
  Type of a flash error code.

7.9.3  Macro Definition Documentation

7.9.3.1  FLASH_ATTR_ERASED_IS_ONE

#define FLASH_ATTR_ERASED_IS_ONE 0x00000001U

Defines one as the erased bit state.
7.9.3.2  FLASH_ATTR_MEMORY_MAPPED

#define FLASH_ATTR_MEMORY_MAPPED 0x00000002U

The memory is accessible in a memory mapped mode.

7.9.3.3  FLASH_ATTR_REWRITABLE

#define FLASH_ATTR_REWRITABLE 0x00000004U

Programmed pages can be programmed again.

Note

This is incompatible and alternative to FLASH_ATTR_ECC_CAPABLE.

7.9.3.4  FLASH_ATTR_ECC_CAPABLE

#define FLASH_ATTR_ECC_CAPABLE 0x00000008U

The memory is protected by an ECC mechanism.

Note

This usually puts restrictions on the program operations.

- Program operations can only happen at offsets aligned to write page boundaries.
- The programmed data size must be a multiple of the write page size.
- Programmed pages cannot be re-programmed.

7.9.3.5  FLASH_ATTR_ECC_ZERO_LINE_CAPABLE

#define FLASH_ATTR_ECC_ZERO_LINE_CAPABLE 0x00000010U

The device is able to overwrite zero to a line.

Note

This attribute is only meaningful for those devices that support ECC, so also FLASH_ATTR_ECC_CAPABLE must be specified.
7.9.3.6  FLASH_ATTR_SUSPEND_ERASE_CAPABLE

#define FLASH_ATTR_SUSPEND_ERASE_CAPABLE 0x00000020U

The device is able to suspend erase operations.

7.9.3.7 _base_flash_methods_alone

#define _base_flash_methods_alone

Value:

/\* Get flash device attributes. */
const flash_descriptor_t * (*get_descriptor)(void *instance);

/\* Read operation. */
flash_error_t (*read)(void *instance, flash_offset_t offset,
size_t n, uint8_t *rp);

/\* Program operation. */
flash_error_t (*program)(void *instance, flash_offset_t offset,
size_t n, const uint8_t *pp);

/\* Erase whole flash device. */
flash_error_t (*start_erase_all)(void *instance);

/\* Erase single sector. */
flash_error_t (*start_erase_sector)(void *instance,
flash_sector_t sector);

/\* Verify erase single sector. */
flash_error_t (*verify_erase)(void *instance, flash_sector_t sector);

/\* Acquire exclusive use of flash. */
flash_error_t (*acquire_exclusive)(void *instance);

/\* Release exclusive use of flash. */
flash_error_t (*release_exclusive)(void *instance);

BaseFlash specific methods.

7.9.3.8 _base_flash_methods

#define _base_flash_methods

Value:

_base_object_methods
_base_flash_methods_alone

BaseFlash specific methods with inherited ones.

7.9.3.9 _base_flash_data

#define _base_flash_data

Value:

_base_object_data

/\* Driver state. */
flash_state_t state;

BaseFlash specific data.
7.9.3.10 getBaseFlash

```c
#define getBaseFlash(ip) ((BaseFlash *)ip)->vmt
```

Instance getter.

This special method is used to get the instance of this class object from a derived class.

7.9.3.11 flashGetDescriptor

```c
#define flashGetDescriptor(ip) (ip)->vmt->get_descriptor(ip)
```

Gets the flash descriptor structure.

**Parameters**

| in | ip | pointer to a BaseFlash or derived class |

**Returns**

A flash device descriptor.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.3.12 flashRead

```c
#define flashRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)
```

Read operation.

**Parameters**

| in | ip | pointer to a BaseFlash or derived class |
| in | offset | flash offset |
| in | n | number of bytes to be read |
| out | rp | pointer to the data buffer |
Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_READ</td>
<td>if the read operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.3.13 flashProgram

```c
#define flashProgram(ip, offset, n, pp) (ip)->vmt->program(ip, offset, n, pp)
```

Program operation.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ip</td>
<td>pointer to a BaseFlash or derived class</td>
</tr>
<tr>
<td>in offset</td>
<td>flash offset</td>
</tr>
<tr>
<td>in n</td>
<td>number of bytes to be programmed</td>
</tr>
<tr>
<td>in pp</td>
<td>pointer to the data buffer</td>
</tr>
</tbody>
</table>

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_PROGRAM</td>
<td>if the program operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.9.3.14 flashStartEraseAll

#define flashStartEraseAll(ip) (ip)->vmt->start_erase_all(ip)

Starts a whole-device erase operation.

Parameters

| in  | ip     | pointer to a BaseFlash or derived class |

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>FLASH_NO_ERROR</th>
<th>if there is no erase operation in progress.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.3.15 flashStartEraseSector

#define flashStartEraseSector(ip, sector) (ip)->vmt->start_erase_sector(ip, sector)

Starts an sector erase operation.

Parameters

| in  | ip     | pointer to a BaseFlash or derived class |
| in  | sector | sector to be erased                     |

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>FLASH_NO_ERROR</th>
<th>if there is no erase operation in progress.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>
7.9 Generic Flash Interface

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.3.16 flashQueryErase

#define flashQueryErase(ip, msec) (ip)->vmt->query_erase(ip, msec)

Queries the driver for erase operation progress.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseFlash or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msec</td>
<td>recommended time, in milliseconds, that should be spent before calling this function again, can be NULL</td>
</tr>
</tbody>
</table>

Returns
An error code.

Return values

<table>
<thead>
<tr>
<th>FLASH_NO_ERROR</th>
<th>if there is no erase operation in progress.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_ERASE</td>
<td>if the erase operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.3.17 flashVerifyErase

#define flashVerifyErase(ip, sector) (ip)->vmt->verify_erase(ip, sector)

Returns the erase state of a sector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseFlash or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sector</td>
<td>sector to be verified</td>
</tr>
</tbody>
</table>
Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if the sector is erased.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_VERIFY</td>
<td>if the verify operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.3.18 flashAcquireExclusive

```c
#define flashAcquireExclusive(
    ip ) (ip)->vmt->acquire_exclusive(ip)
```

Acquires exclusive access to flash.

Parameters

| in | ip | pointer to a BaseFlash or derived class |

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if the access is obtained.</td>
</tr>
<tr>
<td>FLASH_ERROR_UNIMPLEMENTED</td>
<td>if exclusive access not enabled</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.3.19 flashReleaseExclusive

```c
#define flashReleaseExclusive(
    ip ) (ip)->vmt->release_exclusive(ip)
```

Releases exclusive access to flash.
7.9 Generic Flash Interface

Parameters

| in | ip | pointer to a BaseFlash or derived class |

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>FLASH_NO_ERROR</th>
<th>if the access is released.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_ERROR_UNIMPLEMENTED</td>
<td>if exclusive access not enabled</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.4 Typedef Documentation

7.9.4.1 flash_offset_t

typedef uint32_t flash_offset_t

Type of a flash offset.

7.9.4.2 flash_sector_t

typedef uint32_t flash_sector_t

Type of a flash sector number.

7.9.5 Enumeration Type Documentation

7.9.5.1 flash_state_t

enum flash_state_t

Driver state machine possible states.
7.9.5.2  flash_error_t

enum flash_error_t  

Type of a flash error code.

7.9.6  Function Documentation

7.9.6.1  flashWaitErase()

flash_error_t flashWaitErase (  
    BaseFlash * devp  )

Waits until the current erase operation is finished.

Parameters

| in  | devp | pointer to a BaseFlash object |

Returns

An error code.

Return values

| FLASH_NO_ERROR | if there is no erase operation in progress. |
|FLASH_ERROR_ERASE | if the erase operation failed. |
FLASH_ERROR_HW_FAILURE | if access to the memory failed. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.6.2  flashGetSectorOffset()

flash_offset_t flashGetSectorOffset (  
    BaseFlash * devp,  
    flash_sector_t sector  )

Returns the offset of a sector.
7.9.6.3 flashGetSectorSize()

uint32_t flashGetSectorSize (  
    BaseFlash * devp,  
    flash_sector_t sector  
)

Returns the size of a sector.

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>pointer to a BaseFlash object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>devp</td>
<td>pointer to a BaseFlash object</td>
</tr>
<tr>
<td>in</td>
<td>sector</td>
<td>flash sector number</td>
</tr>
</tbody>
</table>

Returns

Sector size.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.6.4 flashGetOffsetSector()

flash_sector_t flashGetOffsetSector (  
    BaseFlash * devp,  
    flash_offset_t offset  
)

Returns the sector of an offset.

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>pointer to a BaseFlash object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>devp</td>
<td>pointer to a BaseFlash object</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>flash offset</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>flash offset</td>
</tr>
</tbody>
</table>
Returns

Flash sector.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.6.5 flashGetOffsetAddress()

void * flashGetOffsetAddress ( 
    BaseFlash * devp, 
    flash_offset_t offset )

Get absolute address from offset.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>devp</th>
<th>pointer to a BaseFlash object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>offset</td>
<td>flash offset</td>
</tr>
</tbody>
</table>

Returns

A pointer to the offset.

Deprecated

7.9.6.6 flashGetAddressOffset()

flash_offset_t flashGetAddressOffset ( 
    BaseFlash * devp, 
    void * addr )

Get offset from absolute address.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>devp</th>
<th>pointer to a BaseFlash object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>addr</td>
<td>pointer</td>
</tr>
</tbody>
</table>

Returns

flash offset

Deprecated
7.10 GPT Driver

Generic GPT Driver.

7.10.1 Detailed Description

Generic GPT Driver.

This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

Precondition

In order to use the GPT driver the `HAL_USE_GPT` option must be enabled in `halconf.h`.

7.10.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).
7.10.3  GPT Operations.

This driver abstracts a generic timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. A callback is invoked when this happens.

The timer can operate in three different modes:

- **Continuous Mode**, a periodic callback is invoked until the driver is explicitly stopped.
- **One Shot Mode**, a callback is invoked after the programmed period and then the timer automatically stops.
- **Delay Mode**, the timer is used for inserting a brief delay into the execution flow, no callback is invoked in this mode.

### Macros

- `#define gptChangeInterval(gptp, interval)`  
  Changes the interval of GPT peripheral.
- `#define gptGetIntervalX(gptp) gpt_lld_get_interval(gptp)`  
  Returns the interval of GPT peripheral.
- `#define gptGetCounterX(gptp) gpt_lld_get_counter(gptp)`  
  Returns the counter value of GPT peripheral.
- `#define _gpt_isr_invoke_cb(gptp)`  
  Common ISR code, GPT period event.
- `#define gpt_lld_change_interval(gptp, interval)`  
  Changes the interval of GPT peripheral.

### PLATFORM configuration options

- `#define PLATFORM_GPT_USE_GPT1 FALSE`  
  GPTD1 driver enable switch.

### Typedefs

- `typedef struct GPTDriver GPTDriver`  
  Type of a structure representing a GPT driver.
- `typedef void (*gptcallback_t)(GPTDriver *gptp)`  
  GPT notification callback type.
- `typedef uint32_t gptfreq_t`  
  GPT frequency type.
- `typedef uint16_t gptcnt_t`  
  GPT counter type.
Data Structures

- `struct GPTConfig`
  
  Driver configuration structure.

- `struct GPTDriver`
  
  Structure representing a GPT driver.

Functions

- `void gptInit (void)`
  
  GPT Driver initialization.

- `void gptObjectInit (GPTDriver *gptp)`
  
  Initializes the standard part of a `GPTDriver` structure.

- `msg_t gptStart (GPTDriver *gptp, const GPTConfig *config)`
  
  Configures and activates the GPT peripheral.

- `void gptStop (GPTDriver *gptp)`
  
  Deactivates the GPT peripheral.

- `void gptChangeInterval (GPTDriver *gptp, gptcnt_t interval)`
  
  Changes the interval of GPT peripheral.

- `void gptStartContinuous (GPTDriver *gptp, gptcnt_t interval)`
  
  Starts the timer in continuous mode.

- `void gptStartContinuousI (GPTDriver *gptp, gptcnt_t interval)`
  
  Starts the timer in continuous mode.

- `void gptStartOneShot (GPTDriver *gptp, gptcnt_t interval)`
  
  Starts the timer in one shot mode.

- `void gptStartOneShotI (GPTDriver *gptp, gptcnt_t interval)`
  
  Starts the timer in one shot mode.

- `void gptStopTimer (GPTDriver *gptp)`
  
  Stops the timer.

- `void gptStopTimerI (GPTDriver *gptp)`
  
  Stops the timer.

- `void gptPolledDelay (GPTDriver *gptp, gptcnt_t interval)`
  
  Starts the timer in one shot mode and waits for completion.

- `void gpt_lld_init (void)`
  
  Low level GPT driver initialization.

- `void gpt_lld_start (GPTDriver *gptp)`
  
  Configures and activates the GPT peripheral.

- `void gpt_lld_stop (GPTDriver *gptp)`
  
  Deactivates the GPT peripheral.

- `void gpt_lld_start_timer (GPTDriver *gptp, gptcnt_t interval)`
  
  Starts the timer in continuous mode.

- `void gpt_lld_stop_timer (GPTDriver *gptp)`
  
  Stops the timer.

- `void gpt_lld_polled_delay (GPTDriver *gptp, gptcnt_t interval)`
  
  Starts the timer in one shot mode and waits for completion.

Enumerations

- `enum gptstate_t {
    GPT_UNINIT = 0, GPT_STOP = 1, GPT_READY = 2, GPT_CONTINUOUS = 3,
    GPT_ONESHOT = 4
  }
  
  Driver state machine possible states.
Variables

- **GPTDriver GPTD1**
  
  GPTD1 driver identifier.

### 7.10.4 Macro Definition Documentation

#### 7.10.4.1 gptChangeIntervalI

```c
#define gptChangeIntervalI(
    gptp,
    interval
)
```

**Value:**

```c
\{ 
  gpt_lld_change_interval(gptp, interval);
\}
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

**Precondition**

The GPT unit must be running in continuous mode.

**Postcondition**

The GPT unit interval is changed to the new value.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>gptp</strong> pointer to a <strong>GPTDriver</strong> object</td>
</tr>
<tr>
<td>in</td>
<td><strong>interval</strong> new cycle time in timer ticks</td>
</tr>
</tbody>
</table>

**Function Class:**

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

#### 7.10.4.2 gptGetIntervalX

```c
#define gptGetIntervalX(
    gptp
)
```

Returns the interval of GPT peripheral.
Precondition
The GPT unit must be running in continuous mode.

Parameters

\[
\text{in} \quad \text{gptp} \quad \text{pointer to a GPTDriver object}
\]

Returns
The current interval.

Function Class:
This is an X-Class API, this function can be invoked from any context.

7.10.4.3 gptGetCounterX

#define gptGetCounterX(
    gptp
) gpt_lld_get_counter(gptp)

Returns the counter value of GPT peripheral.

Precondition
The GPT unit must be running in continuous mode.

Note
The nature of the counter is not defined, it may count upward or downward, it could be continuously running or not.

Parameters

\[
\text{in} \quad \text{gptp} \quad \text{pointer to a GPTDriver object}
\]

Returns
The current counter value.

Function Class:
This is an X-Class API, this function can be invoked from any context.
7.10.4.4 _gpt_isr_invoke_cb

#define _gpt_isr_invoke_cb(
    gptp )

Value:

do {
    \
    if ((gptp)->state == GPT_ONESHOT) {
        (gptp)->state = GPT_READY;
        gpt_lld_stop_timer(gptp);
    }
    \
    if ((gptp)->config->callback != NULL) {
        (gptp)->config->callback(gptp);
    }
    \
} while (0)

Common ISR code, GPT period event.

Parameters

| in  | gptp | pointer to the GPTDriver object |

Function Class:

Not an API, this function is for internal use only.

7.10.4.5 PLATFORM_GPT_USE_GPT1

#define PLATFORM_GPT_USE_GPT1 FALSE

GPTD1 driver enable switch.

If set to TRUE the support for GPTD1 is included.

Note

The default is FALSE.

7.10.4.6 gpt_lld_change_interval

#define gpt_lld_change_interval(
    gptp, 
    interval )

Value:

{ 
    (void)gptp;
    (void)interval;
}

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.
Precondition

The GPT unit must have been activated using `gptStart()`. The GPT unit must have been running in continuous mode using `gptStartContinuous()`.

Postcondition

The GPT unit interval is changed to the new value.

Note

The function has effect at the next cycle start.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>gptp</code></th>
<th>pointer to a <code>GPTDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>interval</code></td>
<td>new cycle time in timer ticks</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.10.5 Typedef Documentation

7.10.5.1 GPTDriver

typedef struct GPTDriver GPTDriver

Type of a structure representing a GPT driver.

7.10.5.2 gptcallback_t

typedef void(* gptcallback_t) (GPTDriver *p)

GPT notification callback type.

Parameters

| in  | `gptp` | pointer to a `GPTDriver` object |
7.10.5.3 gptfreq_t

typedef uint32_t gptfreq_t

GPT frequency type.

7.10.5.4 gptcnt_t

typedef uint16_t gptcnt_t

GPT counter type.

7.10.6 Enumeration Type Documentation

7.10.6.1 gptstate_t

def gptstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>GPT_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>GPT_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>GPT_CONTINUOUS</td>
<td>Active in continuous mode.</td>
</tr>
<tr>
<td>GPT_ONESHOT</td>
<td>Active in one shot mode.</td>
</tr>
</tbody>
</table>

7.10.7 Function Documentation

7.10.7.1 gptInit()

void gptInit ( void )

GPT Driver initialization.
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
gptInit  ->  gpt_lld_init  ->  gptObjectInit
```

### 7.10.7.2 gptObjectInit()

```
void gptObjectInit (  
    GPTDriver  * gptp  )
```

Initializes the standard part of a `GPTDriver` structure.

**Parameters**

- **out gptp**: pointer to the `GPTDriver` object

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.10.7.3 gptStart()

```
msg_t gptStart (  
    GPTDriver  * gptp,  
    const GPTConfig  * config  )
```

Configures and activates the GPT peripheral.

**Parameters**

- **in gptp**: pointer to the `GPTDriver` object
- **in config**: pointer to the `GPTConfig` object
Returns
The operation status.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]

7.10.7.4 gptStop()

```c
void gptStop ( GPTDriver * gptp )
```

Deactivates the GPT peripheral.

Parameters

| in  | gptp | pointer to the GPTDriver object |
7.10 GPT Driver

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
gptStop
    gpt_lld_stop
osalSysLock
osalSysUnlock
```

7.10.7.5 gptChangeInterval()

```c
void gptChangeInterval ( 
    GPTDriver * gptp, 
    gptcnt_t interval 
)
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

Precondition
The GPT unit must be running in continuous mode.

Postcondition
The GPT unit interval is changed to the new value.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>gptp</th>
<th>pointer to a GPTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>interval</td>
<td>new cycle time in timer ticks</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

![Call Graph](image)

### 7.10.7.6 `gptStartContinuous()`

```c
void gptStartContinuous (  
    GPTDriver * gptp,  
    gptcnt_t interval )
```

Starts the timer in continuous mode.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gptp</code></td>
<td>pointer to the <code>GPTDriver</code> object</td>
</tr>
<tr>
<td><code>interval</code></td>
<td>period in ticks</td>
</tr>
</tbody>
</table>

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.10 GPT Driver

7.10.7.7 `gptStartContinuousI()`

```c
void gptStartContinuousI ( 
    GPTDriver * gptp,
    gptcnt_t interval )
```

Starts the timer in continuous mode.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>gptp</code></td>
<td>pointer to the GPTDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>interval</code></td>
<td>period in ticks</td>
</tr>
</tbody>
</table>

**Function Class:**

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
gptStartContinuousI  
                        gpt_lld_start_timer
```

7.10.7.8 `gptStartOneShot()`

```c
void gptStartOneShot ( 
    GPTDriver * gptp,
    gptcnt_t interval )
```

Starts the timer in one shot mode.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>gptp</code></td>
<td>pointer to the GPTDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>interval</code></td>
<td>time interval in ticks</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.10.7.9 gptStartOneShotI()

```c
void gptStartOneShotI ( 
    GPTDriver * gptp, 
    gptcnt_t interval )
```

Starts the timer in one shot mode.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>gptp</th>
<th>pointer to the GPTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>interval</td>
<td>time interval in ticks</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)
7.10.7.10  gptStopTimer()

void gptStopTimer (  
    GPTDriver * gtp  )

Stops the timer.

Parameters

| in | gtp | pointer to the GPTDriver object |

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
gptStopTimerI
    gptStopTimer
        gptId_stop_timer
            osalSysLock
                osalSysUnlock
```

7.10.7.11  gptStopTimerI()

void gptStopTimerI (  
    GPTDriver * gtp  )

Stops the timer.

Parameters

| in | gtp | pointer to the GPTDriver object |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]

7.10.7.12  gptPolledDelay()

```c
void gptPolledDelay ( 
    GPTDriver * gptp, 
    gptcnt_t interval )
```

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

Note

The configured callback is not invoked when using this function.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>gptp</th>
<th>pointer to the GPTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>interval</td>
<td>time interval in ticks</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]
7.10.7.13  

**gpt_lld_init()**

```c
void gpt_lld_init (  
    void  )
```

Low level GPT driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

![Call Graph](image)

7.10.7.14  

**gpt_lld_start()**

```c
void gpt_lld_start (  
    GPTDriver * gptp  )
```

Configures and activates the GPT peripheral.

**Parameters**

```c
in  gptp  pointer to the GPTDriver object
```

**Function Class:**

Not an API, this function is for internal use only.

7.10.7.15  

**gpt_lld_stop()**

```c
void gpt_lld_stop (  
    GPTDriver * gptp  )
```

Deactivates the GPT peripheral.
Parameters

in gtp pointer to the GPTDriver object

Function Class:
Not an API, this function is for internal use only.

7.10.7.16 gpt_lld_start_timer()

void gpt_lld_start_timer (  
    GPTDriver * gtp,  
    gptcnt_t interval )

Starts the timer in continuous mode.

Parameters

in gtp pointer to the GPTDriver object  
in interval period in ticks

Function Class:
Not an API, this function is for internal use only.

7.10.7.17 gpt_lld_stop_timer()

void gpt_lld_stop_timer (  
    GPTDriver * gtp )

Stops the timer.

Parameters

in gtp pointer to the GPTDriver object

Function Class:
Not an API, this function is for internal use only.
7.10.7.18  gpt_lld_polled_delay()

void gpt_lld_polled_delay (  
    GPTDriver * gptp,  
    gptcnt_t interval )

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>gptp</th>
<th>pointer to the GPTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>interval</td>
<td>time interval in ticks</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.10.8  Variable Documentation

7.10.8.1  GPTD1

GPTDriver GPTD1

GPTD1 driver identifier.
7.11 I2C Driver

Generic I2C Driver.

7.11.1 Detailed Description

Generic I2C Driver.

This module implements a generic I2C (Inter-Integrated Circuit) driver.

Precondition

In order to use the I2C driver the HAL_USE_I2C option must be enabled in halconf.h.

7.11.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

The driver is not thread safe for performance reasons, if you need to access the I2C bus from multiple threads then use the i2cAcquireBus() and i2cReleaseBus() APIs in order to gain exclusive access.
7.11 I2C Driver

Macros

- #define I2C_USE_MUTUAL_EXCLUSION TRUE
  Enables the mutual exclusion APIs on the I2C bus.
- #define _i2c_wakeup_isr(i2cp)
  Wakes up the waiting thread notifying no errors.
- #define _i2c_wakeup_error_isr(i2cp)
  Wakes up the waiting thread notifying errors.
- #define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)
  Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.
- #define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))
  Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.
- #define i2c_lld_get_errors(i2cp) ((i2cp)->errors)
  Get errors from I2C driver.

I2C bus error conditions

- #define I2C_NO_ERROR 0x00
  No error.
- #define I2C_BUS_ERROR 0x01
  Bus Error.
- #define I2C_ARBITRATION_LOST 0x02
  Arbitration Lost.
- #define I2C_ACK_FAILURE 0x04
  Acknowledge Failure.
- #define I2C_OVERRUN 0x08
  Overrun/Underrun.
- #define I2C_PEC_ERROR 0x10
  PEC Error in reception.
- #define I2C_TIMEOUT 0x20
  Hardware timeout.
- #define I2C_SMB_ALERT 0x40
  SMBus Alert.

PLATFORM configuration options

- #define PLATFORM_I2C_USE_I2C1 FALSE
  I2C1 driver enable switch.
**Typedefs**

- typedef uint16_t i2caddr_t
  
  Type representing an I2C address.

- typedef uint32_t i2cflags_t
  
  Type of I2C Driver condition flags.

- typedef struct hal_i2c_config I2CConfig
  
  Type of a structure representing an I2C configuration.

- typedef struct hal_i2c_driver I2CDriver
  
  Type of a structure representing an I2C driver.

**Data Structures**

- struct hal_i2c_config
  
  I2C driver configuration structure.

- struct hal_i2c_driver
  
  Structure representing an I2C driver.

**Functions**

- void i2cInit (void)
  
  I2C Driver initialization.

- void i2cObjectInit (I2CDriver *i2cp)
  
  Initializes the standard part of a I2CDriver structure.

- msg_t i2cStart (I2CDriver *i2cp, const I2CConfig *config)
  
  Configures and activates the I2C peripheral.

- void i2cStop (I2CDriver *i2cp)
  
  Deactivates the I2C peripheral.

- i2cflags_t i2cGetErrors (I2CDriver *i2cp)
  
  Returns the errors mask associated to the previous operation.

- msg_t i2cMasterTransmitTimeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  
  Sends data via the I2C bus.

- msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  
  Receives data from the I2C bus.

- void i2cAcquireBus (I2CDriver *i2cp)
  
  Gains exclusive access to the I2C bus.

- void i2cReleaseBus (I2CDriver *i2cp)
  
  Releases exclusive access to the I2C bus.

- void i2c_lld_init (void)
  
  Low level I2C driver initialization.

- void i2c_lld_start (I2CDriver *i2cp)
  
  Configures and activates the I2C peripheral.

- void i2c_lld_stop (I2CDriver *i2cp)
  
  Deactivates the I2C peripheral.

- msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  
  Receives data via the I2C bus as master.

- msg_t i2c_lld_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  
  Transmits data via the I2C bus as master.
Enumerations

• enum i2cstate_t {
  I2C_UNINIT = 0, I2C_STOP = 1, I2C_READY = 2, I2C_ACTIVE_TX = 3,
  I2C_ACTIVE_RX = 4, I2C_LOCKED = 5 }

  Driver state machine possible states.

Variables

• I2CDriver I2CD1

  I2C1 driver identifier.

7.11.3 Macro Definition Documentation

7.11.3.1 I2C_NO_ERROR

#define I2C_NO_ERROR 0x00

No error.

7.11.3.2 I2C_BUS_ERROR

#define I2C_BUS_ERROR 0x01

Bus Error.

7.11.3.3 I2C_ARBITRATION_LOST

#define I2C_ARBITRATION_LOST 0x02

Arbitration Lost.

7.11.3.4 I2C_ACK_FAILURE

#define I2C_ACK_FAILURE 0x04

Acknowledge Failure.
7.11.3.5 I2C_OVERRUN

#define I2C_OVERRUN 0x08
Overrun/Underrun.

7.11.3.6 I2C_PEC_ERROR

#define I2C_PEC_ERROR 0x10
PEC Error in reception.

7.11.3.7 I2C_TIMEOUT

#define I2C_TIMEOUT 0x20
Hardware timeout.

7.11.3.8 I2C_SMB_ALERT

#define I2C_SMB_ALERT 0x40
SMBus Alert.

7.11.3.9 I2C_USE_MUTUAL_EXCLUSION

#define I2C_USE_MUTUAL_EXCLUSION TRUE
Enables the mutual exclusion APIs on the I2C bus.

7.11.3.10 _i2c_wakeup_isr

#define _i2c_wakeup_isr(i2cp)

Value:

do {   \ osalSysLockFromISR();   \ osalThreadResumeI(&(i2cp)->thread, MSG_OK);   \ osalSysUnlockFromISR();   } while (0)
Wakes up the waiting thread notifying no errors.
7.11 I2C Driver

Parameters

| in | i2cp | pointer to the I2C Driver object |

Function Class:

Not an API, this function is for internal use only.

7.11.3.11 _i2c_wakeup_error_isr

#define _i2c_wakeup_error_isr(
   i2cp )

Value:

\[
\text{do} \{
   \text{osalSysLockFromISR();}
   \text{osalThreadResumeI(&(i2cp)->thread, MSG_RESET);}
   \text{osalSysUnlockFromISR();}
\}\text{while (0)}
\]

Wakes up the waiting thread notifying errors.

Parameters

| in | i2cp | pointer to the I2C Driver object |

Function Class:

Not an API, this function is for internal use only.

7.11.3.12 i2cMasterTransmit

#define i2cMasterTransmit(
   i2cp, addr, txbuf, txbytes, rxbuf, rxbytes )

Value:

\text{\{i2cMasterTransmitTimeout(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes, TIME_INFINITE)\}}

Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.11.3.13  i2cMasterReceive

#define i2cMasterReceive(
i2cp,
addr,
rxbufl,
rxbytes ) (i2cMasterReceiveTimeout(i2cp, addr, rxbufl, rxbytes, TIME_INFINITE))

Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.3.14  PLATFORM_I2C_USE_I2C1

#define PLATFORM_I2C_USE_I2C1 FALSE

I2C1 driver enable switch.

If set to TRUE the support for I2C1 is included.

Note
The default is FALSE.

7.11.3.15  i2c_lld_get_errors

#define i2c_lld_get_errors(
i2cp)
((i2cp)->errors)

Get errors from I2C driver.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.11.4  Typedef Documentation
7.11 I2C Driver

7.11.4.1 i2caddr_t

typedef uint16_t i2caddr_t

Type representing an I2C address.

7.11.4.2 i2cflags_t

typedef uint32_t i2cflags_t

Type of I2C Driver condition flags.

7.11.4.3 I2CConfig

typedef struct hal_i2c_config I2CConfig

Type of a structure representing an I2C configuration.

7.11.4.4 I2CDriver

typedef struct hal_i2c_driver I2CDriver

Type of a structure representing an I2C driver.

7.11.5 Enumeration Type Documentation

7.11.5.1 i2cstate_t

enum i2cstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>I2C_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>I2C_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>I2C_ACTIVE_TX</td>
<td>Transmitting.</td>
</tr>
<tr>
<td>I2C_ACTIVE_RX</td>
<td>Receiving.</td>
</tr>
<tr>
<td>I2C_LOCKED</td>
<td>Bus locked.</td>
</tr>
</tbody>
</table>
7.11.6 Function Documentation

7.11.6.1 i2cInit()

void i2cInit (
   void )

I2C Driver initialization.

Note
This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.

Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
  i2cInit -> i2c_lld_init -> i2cObjectInit
```

7.11.6.2 i2cObjectInit()

void i2cObjectInit ( 
   I2CDriver * i2cp )

Initializes the standard part of a I2CDriver structure.

Parameters
- **out** i2cp pointer to the I2CDriver object
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
7.11.6.3 i2cStart()
```

```c
msg_t i2cStart (  
    I2CDriver * i2cp,  
    const I2CConfig * config )
```

Configures and activates the I2C peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>i2cp</th>
<th>pointer to the I2CDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the I2CConfig object</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.11.6.4 i2cStop()

```c
void i2cStop ( 
    I2CDriver * i2cp 
)
```

Deactivates the I2C peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
</tbody>
</table>

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    i2cStop
    |__________________________|
    |                           |
    |                           |
    |                           |
    |__________________________|
    |                           |
    |                           |
    |__________________________|
    |                           |
    |                           |
    |__________________________|
    |                           |
    |                           |
    |__________________________|
    |                           |
    |                           |
    |__________________________|
```

7.11.6.5 i2cGetErrors()

```c
i2cflags_t i2cGetErrors ( 
    I2CDriver * i2cp 
)
```

Returns the errors mask associated to the previous operation.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
</tbody>
</table>

**Returns**

The errors mask.
7.11 I2C Driver

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.6.6 i2cMasterTransmitTimeout()

```c
msg_t i2cMasterTransmitTimeout ( 
    I2CDriver * i2cp,
    i2caddr_t addr,
    const uint8_t * txbuf,
    size_t txbytes,
    uint8_t * rxbuf,
    size_t rxbytes,
    sysinterval_t timeout )
```

Sends data via the I2C bus.

Function designed to realize "read-through-write" transfer paradigm. If you want transmit data without any further read, than set `rxbytes` field to 0.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>i2cp</code></td>
<td>pointer to the I2CDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>addr</code></td>
<td>slave device address (7 bits) without R/W bit</td>
</tr>
<tr>
<td>in</td>
<td><code>txbuf</code></td>
<td>pointer to transmit buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>txbytes</code></td>
<td>number of bytes to be transmitted</td>
</tr>
<tr>
<td>out</td>
<td><code>rxbuf</code></td>
<td>pointer to receive buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>rxbytes</code></td>
<td>number of bytes to be received, set it to 0 if you want transmit only</td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_OK</code></td>
<td>if the function succeeded.</td>
</tr>
</tbody>
</table>
| `MSG_RESET` | if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.
| `MSG_TIMEOUT` | if a timeout occurred before operation end.                      |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.11.6.7  

**i2cMasterReceiveTimeout()**

```c
msg_t i2cMasterReceiveTimeout ( 
    I2CDriver * i2cp, 
    i2caddr_t addr, 
    uint8_t * rxbuf, 
    size_t rxbytes, 
    sysinterval_t timeout )
```

Receives data from the I2C bus.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>i2cp</code></td>
<td>pointer to the I2CDriver object</td>
</tr>
<tr>
<td><code>addr</code></td>
<td>slave device address (7 bits) without R/W bit</td>
</tr>
<tr>
<td><code>rxbuf</code></td>
<td>pointer to receive buffer</td>
</tr>
<tr>
<td><code>rxbytes</code></td>
<td>number of bytes to be received</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **MSG_OK** if the function succeeded.
- **MSG_RESET** if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.
- **MSG_TIMEOUT** if a timeout occurred before operation end.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

---

7.11.6.8  

**i2cAcquireBus()**

```c
void i2cAcquireBus ( 
    I2CDriver * i2cp )
```

Gains exclusive access to the I2C bus.

This function tries to gain ownership to the I2C bus, if the bus is already being used then the invoking thread is queued.

**Precondition**

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.
7.11 I2C Driver

Parameters

| in | i2cp | pointer to the I2CDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

[Call Graph Diagram]

7.11.6.9 i2cReleaseBus()

tvoid i2cReleaseBus (I2CDriver * i2cp)

Releases exclusive access to the I2C bus.

Precondition

In order to use this function the option I2C_USE_MUTUAL_EXCLUSION must be enabled.

Parameters

| in | i2cp | pointer to the I2CDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

[Call Graph Diagram]
7.11.6.10  i2c_lld_init()

void i2c_lld_init (  
    void  )  

Low level I2C driver initialization.

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
    i2c_lld_init  
    └── i2cObjectInit  
        └── osalMutexObjectInit
```

7.11.6.11  i2c_lld_start()

void i2c_lld_start (  
    I2CDriver * i2cp  )  

Configures and activates the I2C peripheral.

Parameters

```
in   i2cp   pointer to the I2CDriver object
```

Function Class:
Not an API, this function is for internal use only.

7.11.6.12  i2c_lld_stop()

void i2c_lld_stop (  
    I2CDriver * i2cp  )  

Deactivates the I2C peripheral.
Parameters

| in   | i2cp | pointer to the I2CDriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.11.6.13 i2c_lld_master_receive_timeout()

Receives data via the I2C bus as master.

Parameters

| in   | i2cp | pointer to the I2CDriver object |
| in   | addr | slave device address          |
| out  | rxbuf| pointer to the receive buffer  |
| in   | rxbytes | number of bytes to be received |
| in   | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|      |       | • TIME_INFINITE no timeout. |

Returns

The operation status.

Return values

- **MSG_OK** if the function succeeded.
- **MSG_RESET** if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.
- **MSG_TIMEOUT** if a timeout occurred before operation end. After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.

Function Class:

Not an API, this function is for internal use only.
7.11.6.14  i2c_lld_master_transmit_timeout()

```c
msg_t i2c_lld_master_transmit_timeout ( 
    I2CDriver * i2cp, 
    i2caddr_t addr, 
    const uint8_t * txbuf, 
    size_t txbytes, 
    uint8_t * rxbuf, 
    size_t rxbytes, 
    sysinterval_t timeout )
```

Transmits data via the I2C bus as master.

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp pointer to the I2CDriver object</td>
</tr>
<tr>
<td>in</td>
<td>addr slave device address</td>
</tr>
<tr>
<td>in</td>
<td>txbuf pointer to the transmit buffer</td>
</tr>
<tr>
<td>in</td>
<td>txbytes number of bytes to be transmitted</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf pointer to the receive buffer</td>
</tr>
<tr>
<td>in</td>
<td>rxbytes number of bytes to be received</td>
</tr>
<tr>
<td>in</td>
<td>timeout the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
</tbody>
</table>

- `TIME_INFINITE` no timeout.

### Returns

The operation status.

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if the function succeeded.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if one or more I2C errors occurred, the errors can be retrieved using <code>i2cGetErrors()</code>.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if a timeout occurred before operation end. After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.</td>
</tr>
</tbody>
</table>

### Function Class:

Not an API, this function is for internal use only.

7.11.7  Variable Documentation

7.11.7.1  I2CD1

I2CDriver I2CD1

I2C1 driver identifier.
7.12 I2S Driver

Generic I2S Driver.

7.12.1 Detailed Description

Generic I2S Driver.

This module implements a generic I2S driver.

Precondition

In order to use the I2S driver the HAL_USE_I2S option must be enabled in halconf.h.

7.12.2 Driver State Machine

Macros

- \#define i2s_lld_driver_fields
  
  Low level fields of the I2S driver structure.

- \#define i2s_lld_config_fields
  
  Low level fields of the I2S configuration structure.

I2S modes

- \#define I2S_MODE_SLAVE 0
- \#define I2S_MODE_MASTER 1

Macro Functions

- \#define i2sIsBufferComplete(i2sp) ((bool)((i2sp)->state == I2S_COMPLETE))
  
  Buffer state.

- \#define i2sStartExchange(i2sp)
  
  Starts a I2S data exchange.

- \#define i2sStopExchange(i2sp)
  
  Stops the ongoing data exchange.

- \#define _i2s_isr_half_code(i2sp)
  
  Common ISR code, half buffer event.

- \#define _i2s_isr_full_code(i2sp)
  
  Common ISR code.

PLATFORM configuration options

- \#define PLATFORM_I2S_USE_I2S1 FALSE
  
  I2SD1 driver enable switch.
Typedefs

- typedef struct hal_i2s_driver I2SDriver
  Type of a structure representing an I2S driver.
- typedef struct hal_i2s_config I2SConfig
  Type of a structure representing an I2S driver configuration.
- typedef void(*i2scallback_t)(I2SDriver *i2sp)
  I2S notification callback type.

Data Structures

- struct hal_i2s_driver
  Structure representing an I2S driver.
- struct hal_i2s_config
  Driver configuration structure.

Functions

- void i2sInit (void)
  I2S Driver initialization.
- void i2sObjectInit (I2SDriver *i2sp)
  Initializes the standard part of a I2SDriver structure.
- msg_t i2sStart (I2SDriver *i2sp, const I2SConfig *config)
  Configures and activates the I2S peripheral.
- void i2sStop (I2SDriver *i2sp)
  Deactivates the I2S peripheral.
- void i2sStartExchange (I2SDriver *i2sp)
  Starts a I2S data exchange.
- void i2sStopExchange (I2SDriver *i2sp)
  Stops the ongoing data exchange.
- void i2s_lld_init (void)
  Low level I2S driver initialization.
- void i2s_lld_start (I2SDriver *i2sp)
  Configures and activates the I2S peripheral.
- void i2s_lld_stop (I2SDriver *i2sp)
  Deactivates the I2S peripheral.
- void i2s_lld_start_exchange (I2SDriver *i2sp)
  Starts a I2S data exchange.
- void i2s_lld_stop_exchange (I2SDriver *i2sp)
  Stops the ongoing data exchange.

Enumerations

- enum i2sstate_t {
  I2S_UNINIT = 0, I2S_STOP = 1, I2S_READY = 2, I2S_ACTIVE = 3,
  I2S_COMPLETE = 4
}
  Driver state machine possible states.
7.12 I2S Driver

Variables

- **I2SDriver I2SD1**
  
  I2S2 driver identifier.

7.12.3 Macro Definition Documentation

7.12.3.1 i2sIsBufferComplete

```c
#define i2sIsBufferComplete( i2sp ) ((bool)(((i2sp)->state == I2S_COMPLETE))
```

Buffer state.

**Note**

This function is meant to be called from the SPI callback only.

**Parameters**

- `in i2sp` pointer to the I2SDriver object

**Returns**

The buffer state.

**Return values**

- `false` if the driver filled/sent the first half of the buffer.
- `true` if the driver filled/sent the second half of the buffer.

**Function Class:**

Special function, this function has special requirements see the notes.

7.12.3.2 i2sStartExchangeI

```c
#define i2sStartExchangeI( i2sp )
```

**Value:**

- `i2s_lld_start_exchange(i2sp);`
- `(i2sp)->state = I2S_ACTIVE;`

Starts a I2S data exchange.
Parameters

in i2sp pointer to the I2SDriver object

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.3.3 i2sStopExchangeI

#define i2sStopExchangeI(
    i2sp )

Value:

    i2s_lld_stop_exchange(i2sp);
    (i2sp)->state = I2S_READY;

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

Parameters

in i2sp pointer to the I2SDriver object

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.3.4 _i2s_isr_half_code

#define _i2s_isr_half_code(
    i2sp )

Value:

    if ((i2sp)->config->end_cb != NULL) {
        (i2sp)->config->end_cb(i2sp);
    }

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

• Callback invocation.

Note

This macro is meant to be used in the low level drivers implementation only.
Parameters

| in | i2sp | pointer to the I2CDriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.12.3.5 _i2s_isr_full_code

```c
#define _i2s_isr_full_code(i2sp)

Value:

```c
if ((i2sp)->config->end_cb) {
  (i2sp)->state = I2S_COMPLETE;
  (i2sp)->config->end_cb(i2sp);
  if ((i2sp)->state == I2S_COMPLETE) {
    (i2sp)->state = I2S_ACTIVE;
  }
}
```

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

| in | i2sp | pointer to the I2CDriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.12.3.6 PLATFORM_I2S_USE_I2S1

```c
#define PLATFORM_I2S_USE_I2S1 FALSE
```

I2SD1 driver enable switch.

If set to TRUE the support for I2S1 is included.
7.12.3.7 i2s_lld_driver_fields

#define i2s_lld_driver_fields

Value:
/* Dummy field, it is not needed. */
uint32_t dummy

Low level fields of the I2S driver structure.

7.12.3.8 i2s_lld_config_fields

#define i2s_lld_config_fields

Value:
/* Dummy configuration, it is not needed. */
uint32_t dummy

Low level fields of the I2S configuration structure.

7.12.4 Typedef Documentation

7.12.4.1 I2SDriver

typedef struct hal_i2s_driver I2SDriver

Type of a structure representing an I2S driver.

7.12.4.2 I2SConfig

typedef struct hal_i2s_config I2SConfig

Type of a structure representing an I2S driver configuration.

7.12.4.3 i2scallback_t

typedef void(* i2scallback_t) (I2SDriver *i2sp)

I2S notification callback type.
Parameters

in  `i2sp`  pointer to the I2SDriver object

### 7.12.5 Enumeration Type Documentation

#### 7.12.5.1 i2sstate_t

```c
enum i2sstate_t
```

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2S_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>I2S_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>I2S_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>I2S_ACTIVE</td>
<td>Active.</td>
</tr>
<tr>
<td>I2S_COMPLETE</td>
<td>Transmission complete.</td>
</tr>
</tbody>
</table>

### 7.12.6 Function Documentation

#### 7.12.6.1 i2sInit()

```c
void i2sInit (
    void )
```

I2S Driver initialization.

**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call Graph Image]

### 7.12.6.2 i2sObjectInit()

```c
void i2sObjectInit (  
    I2SDriver * i2sp )
```

Initializes the standard part of a I2SDriver structure.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>i2sp</td>
<td>pointer to the I2SDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.12.6.3 i2sStart()

```c
msg_t i2sStart (  
    I2SDriver * i2sp,  
    const I2SConfig * config )
```

Configures and activates the I2S peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2sp</td>
<td>pointer to the I2SDriver object</td>
</tr>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the I2SConfig object</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]

### 7.12.6.4 i2sStop()

```c
void i2sStop (  
    I2SDriver *i2sp )
```

Deactivates the I2S peripheral.

**Parameters**

| in | i2sp | pointer to the I2SDriver object |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
  i2sStop
    ↓
  osalSysLock
    ↓
  osalSysUnlock
```

### 7.12.6.5 i2sStartExchange()

```c
void i2sStartExchange (  
    I2SDriver * i2sp  
)
```

Starts a I2S data exchange.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>i2sp</code></td>
</tr>
</tbody>
</table>

pointer to the I2SDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
  i2sStartExchange
    ↓
  osalSysLock
    ↓
  osalSysUnlock
```
### 7.12.6.6 i2sStopExchange()

```c
void i2sStopExchange ( I2SDriver * i2sp )
```

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

**Parameters**

| in | i2sp | pointer to the I2SDriver object |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.12.6.7 i2s_lld_init()

```c
void i2s_lld_init ( void )
```

Low level I2S driver initialization.

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

![Call Graph](image)
7.12.6.8  i2s_lld_start()

```c
void i2s_lld_start (
    I2SDriver * i2sp )
```

Configures and activates the I2S peripheral.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>i2sp</strong></td>
<td>pointer to the I2SDriver object</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

7.12.6.9  i2s_lld_stop()

```c
void i2s_lld_stop (
    I2SDriver * i2sp )
```

Deactivates the I2S peripheral.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>i2sp</strong></td>
<td>pointer to the I2SDriver object</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

7.12.6.10  i2s_lld_start_exchange()

```c
void i2s_lld_start_exchange ( 
    I2SDriver * i2sp )
```

Starts a I2S data exchange.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>i2sp</strong></td>
<td>pointer to the I2SDriver object</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.
7.12.6.11  i2s_lld_stop_exchange()

void i2s_lld_stop_exchange (  
    I2SDriver * i2sp )

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

Parameters

|   | i2sp | pointer to the I2SDriver object |

Function Class:

Not an API, this function is for internal use only.

7.12.7  Variable Documentation

7.12.7.1  I2SD1

I2SDriver I2SD1

I2S2 driver identifier.
7.13 ICU Driver

Generic ICU Driver.

7.13.1 Detailed Description

Generic ICU Driver.

This module implements a generic ICU (Input Capture Unit) driver. The purpose of the driver is to measure period and duty cycle of an input digital signal (PWM input).

Precondition

In order to use the ICU driver the `HAL_USE_ICU` option must be enabled in `halconf.h`.

7.13.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

7.13.3 ICU Operations.

This driver abstracts a generic Input Capture Unit composed of:

- A clock prescaler.
- A main up counter.
• Two capture registers triggered by the rising and falling edges on the sampled input.

The ICU unit can be programmed to synchronize on the rising or falling edge of the sample input:

• ICU_INPUT_ACTIVE_HIGH, a rising edge is the start signal.
• ICU_INPUT_ACTIVE_LOW, a falling edge is the start signal.

Callbacks are optionally invoked when:

• On the PWM de-activation edge.
• On the PWM activation edge, measurements for the previous cycle are available from this callback and can be retrieved using icuGetPeriodX() and icuGetWidthX().

Macros

• #define icu_lld_get_width(icup) 0
  Returns the width of the latest pulse.
• #define icu_lld_get_period(icup) 0
  Returns the width of the latest cycle.
• #define icu_lld_are_notifications_enabled(icup) false
  Check on notifications status.

Macro Functions

• #define icuStartCapture(icup)
  Starts the input capture.
• #define icuStopCapture(icup)
  Stops the input capture.
• #define icuEnableNotifications(icup) icu_lld_enable_notifications(icup)
  Enables notifications.
• #define icuDisableNotifications(icup) icu_lld_disable_notifications(icup)
  Disables notifications.
• #define icuAreNotificationsEnabled(icup) icu_lld_are_notifications_enabled(icup)
  Check on notifications status.
• #define icuGetWidth(icup) icu_lld_get_width(icup)
  Returns the width of the latest pulse.
• #define icuGetPeriod(icup) icu_lld_get_period(icup)
  Returns the width of the latest cycle.

Low level driver helper macros

• #define _icu_isr_invoke_width_cb(icup)
  Common ISR code, ICU width event.
• #define _icu_isr_invoke_period_cb(icup)
  Common ISR code, ICU period event.
• #define _icu_isr_invoke_overflow_cb(icup)
  Common ISR code, ICU timer overflow event.
PLATFORM configuration options

- `#define PLATFORM_ICU_USE_ICU1 FALSE`
  
  ICU1 driver enable switch.

typedefs

- `typedef struct ICUDriver ICUDriver`
  
  Type of a structure representing an ICU driver.

- `typedef void (icucallback_t) (ICUDriver *icup)`
  
  ICU notification callback type.

- `typedef uint32_t icufreq_t`
  
  ICU frequency type.

- `typedef uint32_t icucnt_t`
  
  ICU counter type.

Data Structures

- `struct ICUConfig`
  
  Driver configuration structure.

- `struct ICUDriver`
  
  Structure representing an ICU driver.

Functions

- `void icuInit (void)`
  
  ICU Driver initialization.

- `void icuObjectInit (ICUDriver *icup)`
  
  Initializes the standard part of a ICUDriver structure.

- `msg_t icuStart (ICUDriver *icup, const ICUConfig *config)`
  
  Configures and activates the ICU peripheral.

- `void icuStop (ICUDriver *icup)`
  
  Deactivates the ICU peripheral.

- `void icuStartCapture (ICUDriver *icup)`
  
  Starts the input capture.

- `bool icuWaitCapture (ICUDriver *icup)`
  
  Waits for a completed capture.

- `void icuStopCapture (ICUDriver *icup)`
  
  Stops the input capture.

- `void icuEnableNotifications (ICUDriver *icup)`
  
  Enables notifications.

- `void icuDisableNotifications (ICUDriver *icup)`
  
  Disables notifications.

- `void icu_lld_init (void)`
  
  Low level ICU driver initialization.

- `void icu_lld_start (ICUDriver *icup)`
  
  Configures and activates the ICU peripheral.

- `void icu_lld_stop (ICUDriver *icup)`
  
  Deactivates the ICU peripheral.
7.13 ICU Driver

• void icu_lld_start_capture (ICUDriver *icup)
  Starts the input capture.
• bool icu_lld_wait_capture (ICUDriver *icup)
  Waits for a completed capture.
• void icu_lld_stop_capture (ICUDriver *icup)
  Stops the input capture.
• void icu_lld_enable_notifications (ICUDriver *icup)
  Enables notifications.
• void icu_lld_disable_notifications (ICUDriver *icup)
  Disables notifications.

Enumerations

• enum icustate_t {
  ICU_UNINIT = 0, ICU_STOP = 1, ICU_READY = 2, ICU_WAITING = 3,
  ICU_ACTIVE = 4 }
  Driver state machine possible states.
• enum icumode_t { ICU_INPUT_ACTIVE_HIGH = 0, ICU_INPUT_ACTIVE_LOW = 1 }
  ICU driver mode.

Variables

• ICUDriver ICUD1
  ICUD1 driver identifier.

7.13.4 Macro Definition Documentation

7.13.4.1 icuStartCaptureI

#define icuStartCaptureI(
  icup )

Value:

do {
  icu_lld_start_capture(icup);
  {icup}->state = ICU_WAITING;
} while (false)

Starts the input capture.

Parameters

  in  icup  pointer to the ICUDriver object
Function Class:
This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.13.4.2 icuStopCaptureI

```c
#define icuStopCaptureI(
    icup )
```

**Value:**
```c
do {
    icu_lld_stop_capture(icup);
    (icup)->state = ICU_READY;
} while (false)
```

Stops the input capture.

**Parameters**

| in | icup | pointer to the **ICUDriver** object |

Function Class:
This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.13.4.3 icuEnableNotificationsI

```c
#define icuEnableNotificationsI(
    icup ) icu_lld_enable_notifications(icup)
```

Enables notifications.

**Precondition**
The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

**Note**
If the notification is already enabled then the call has no effect.

**Parameters**

| in | icup | pointer to the **ICUDriver** object |
Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.13.4.4 icuDisableNotificationsI

```c
#define icuDisableNotificationsI(icup) icu_lld_disable_notifications(icup)
```

Disables notifications.

**Precondition**

The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

**Note**

If the notification is already disabled then the call has no effect.

**Parameters**

- `icup` pointer to the `ICUDriver` object

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.13.4.5 icuAreNotificationsEnabledX

```c
#define icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)
```

Check on notifications status.

**Parameters**

- `icup` pointer to the `ICUDriver` object

**Returns**

The notifications status.
Return values

| false | if notifications are not enabled. |
| true  | if notifications are enabled. |

Function Class:

Not an API, this function is for internal use only.

7.13.4.6 icuGetWidthX

#define icuGetWidthX(
    icup
) icu_lld_get_width(icup)

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

Note

This function is meant to be invoked from the width capture callback.

Parameters

| in icup | pointer to the ICUDriver object |

Returns

The number of ticks.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.13.4.7 icuGetPeriodX

#define icuGetPeriodX(
    icup
) icu_lld_get_period(icup)

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

Note

This function is meant to be invoked from the width capture callback.
Parameters

in icup pointer to the ICU Driver object

Returns

The number of ticks.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.13.4.8 _icu_isr_invoke_width_cb

#define _icu_isr_invoke_width_cb(
    icup )

Value:

do {
  if (((icup)->state == ICU_ACTIVE) &&
       ((icup)->config->width_cb != NULL)) {
    (icup)->config->width_cb(icup);
  }
} while (0)

Common ISR code, ICU width event.

Parameters

in icup pointer to the ICU Driver object

Function Class:

Not an API, this function is for internal use only.

7.13.4.9 _icu_isr_invoke_period_cb

#define _icu_isr_invoke_period_cb(
    icup )

Value:

do {
  if (((icup)->state == ICU_ACTIVE) &&
       ((icup)->config->period_cb != NULL)) {
    (icup)->config->period_cb(icup);
    (icup)->state = ICU_ACTIVE;
  }
} while (0)

Common ISR code, ICU period event.

Note

A period event brings the driver into the ICU_ACTIVE state.
Parameters

```c
in icup pointer to the ICUDriver object
```

Function Class:

Not an API, this function is for internal use only.

### 7.13.4.10  _icu_isr_invoke_overflow_cb

```c
#define _icu_isr_invoke_overflow_cb(
  icup )
```

**Value:**

```c
do {
  if ((icup)->config->overflow_cb != NULL && (icup)->state == ICU_ACTIVE)
    (icup)->config->overflow_cb(icup);
  else
    (icup)->state = ICU_WAITING;
} while (0)
```

Common ISR code, ICU timer overflow event.

**Note**

An overflow when in ICU_ACTIVE state with a callback set executes the callback code (supports overflow scaling). Otherwise overflow brings the driver back to the ICU_WAITING state.

Parameters

```c
in icup pointer to the ICUDriver object
```

Function Class:

Not an API, this function is for internal use only.

### 7.13.4.11  PLATFORM_ICU_USE_ICU1

```c
#define PLATFORM_ICU_USE_ICU1 FALSE
```

ICUD1 driver enable switch.

If set to TRUE the support for ICUD1 is included.

**Note**

The default is FALSE.
7.13 ICU Driver

7.13.4.12 icu_lld_get_width

#define icu_lld_get_width(
  icup ) 0

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

Parameters

  in icup pointer to the ICUDriver object

Returns

  The number of ticks.

Function Class:

  Not an API, this function is for internal use only.

7.13.4.13 icu_lld_get_period

#define icu_lld_get_period(
  icup ) 0

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

Parameters

  in icup pointer to the ICUDriver object

Returns

  The number of ticks.

Function Class:

  Not an API, this function is for internal use only.

7.13.4.14 icu_lld_are_notifications_enabled

#define icu_lld_are_notifications_enabled(
  icup ) false

Check on notifications status.
Parameters

| in | icup | pointer to the ICUDriver object |

Returns

The notifications status.

Return values

- **false** if notifications are not enabled.
- **true** if notifications are enabled.

Function Class:

Not an API, this function is for internal use only.

### 7.13.5 Typedef Documentation

#### 7.13.5.1 ICUDriver

typedef struct ICUDriver ICUDriver

Type of a structure representing an ICU driver.

#### 7.13.5.2 icucallback_t

typedef void(* icucallback_t) (ICUDriver *icup)

ICU notification callback type.

Parameters

| in | icup | pointer to a ICUDriver object |

#### 7.13.5.3 icufreq_t

typedef uint32_t icufreq_t

ICU frequency type.
7.13 ICU Driver

7.13.5.4 `icucnt_t`

typedef uint32_t icucnt_t

ICU counter type.

7.13.6 Enumeration Type Documentation

7.13.6.1 `icustate_t`

enum icustate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICU_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>ICU_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>ICU_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>ICU_WAITING</td>
<td>Waiting for first front.</td>
</tr>
<tr>
<td>ICU_ACTIVE</td>
<td>First front detected.</td>
</tr>
</tbody>
</table>

7.13.6.2 `icumode_t`

enum icumode_t

ICU driver mode.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICU_INPUT_ACTIVE_HIGH</td>
<td>Trigger on rising edge.</td>
</tr>
<tr>
<td>ICU_INPUT_ACTIVE_LOW</td>
<td>Trigger on falling edge.</td>
</tr>
</tbody>
</table>

7.13.7 Function Documentation
7.13.7.1 icuInit()

```c
void icuInit (  
    void  )
```

ICU Driver initialization.

**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call Graph]

7.13.7.2 icuObjectInit()

```c
void icuObjectInit (  
    ICUDriver * icup  )
```

Initializes the standard part of a `ICUDriver` structure.

**Parameters**

| out icup | pointer to the `ICUDriver` object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.13.7.3 icuStart()

```c
msg_t icuStart (  
    ICUDriver * icup,  
    const ICUConfig * config  )
```
Configures and activates the ICU peripheral.
7.13.7.4 icuStop()

Deactivates the ICU peripheral.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>icup</th>
<th>pointer to the ICUDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the ICUConfig object</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
   icuStop
   icu_lld_stop
   osalSysLock
   osalSysUnlock
```

### 7.13.7.5 icuStartCapture()

```c
void icuStartCapture ( ICUDriver * icup )
```

Starts the input capture.

Parameters

- **in icup**  pointer to the ICUDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
   icuStartCapture
   osalSysLock
   osalSysUnlock
```
7.13.7.6  icuWaitCapture()

```c
bool icuWaitCapture (ICUDriver * icup)
```

Waits for a completed capture.

Note

The operation could be performed in polled mode depending on.
In order to use this function notifications must be disabled.

Precondition

The driver must be in ICU_WAITING or ICU_ACTIVE states.

Postcondition

After the capture is available the driver is in ICU_ACTIVE state. If a capture fails then the driver is in ICU← _WAITING state.

Parameters

- **in icup** pointer to the ICUDriver object

Returns

The capture status.

Return values

- **false** if the capture is successful.
- **true** if a timer overflow occurred.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph Image]

7.13.7.7 icuStopCapture()

```c
void icuStopCapture ( ICUDriver * icup )
```

Stops the input capture.

Parameters

| in | icup | pointer to the ICUDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph Image]
### 7.13.7.8 icuEnableNotifications()

```c
void icuEnableNotifications ( ICUDriver * icup )
```

Enables notifications.

**Precondition**

The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

**Note**

If the notification is already enabled then the call has no effect.

**Parameters**

| in | icup | pointer to the ICUDriver object |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph for icuEnableNotifications](chart)

### 7.13.7.9 icuDisableNotifications()

```c
void icuDisableNotifications ( ICUDriver * icup )
```

Disables notifications.

**Precondition**

The ICU unit must have been activated using `icuStart()` and the capture started using `icuStartCapture()`.

**Note**

If the notification is already disabled then the call has no effect.
7.13 ICU Driver

Parameters

| in | icup | pointer to the ICUDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    icuDisableNotifications
    osalSysLock
    icu_lld_init
    icuObjectInit
```

7.13.7.10 icu_lld_init()

```c
void icu_lld_init ( void )
```

Low level ICU driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
    icu_lld_init
    icuObjectInit
```

7.13.7.11 icu_lld_start()

```c
void icu_lld_start ( ICUDriver * icup )
```

Configures and activates the ICU peripheral.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>icup</td>
<td>pointer to the ICUDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.13.7.12 icu_lld_stop()

```c
void icu_lld_stop ( 
    ICUDriver * icup )
```

Deactivates the ICU peripheral.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>icup</td>
<td>pointer to the ICUDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.13.7.13 icu_lld_start_capture()

```c
void icu_lld_start_capture ( 
    ICUDriver * icup )
```

Starts the input capture.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>icup</td>
<td>pointer to the ICUDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.13.7.14 icu_lld_wait_capture()

```c
bool icu_lld_wait_capture ( 
    ICUDriver * icup )
```

Waits for a completed capture.
7.13 ICU Driver

Note

The operation is performed in polled mode.
In order to use this function notifications must be disabled.

Parameters

| in  | icup | pointer to the ICUDriver object |

Returns

The capture status.

Return values

- false if the capture is successful.
- true if a timer overflow occurred.

Function Class:

Not an API, this function is for internal use only.

7.13.7.15 icu_lld_stop_capture()

```c
void icu_lld_stop_capture (  
  ICUDriver * icup )
```

Stops the input capture.

Parameters

| in  | icup | pointer to the ICUDriver object |

Function Class:

Not an API, this function is for internal use only.

7.13.7.16 icu_lld_enable_notifications()

```c
void icu_lld_enable_notifications (  
  ICUDriver * icup )
```

Enables notifications.
Precondition

The ICU unit must have been activated using \texttt{icuStart()} and the capture started using \texttt{icuStartCapture()}.  

Note

If the notification is already enabled then the call has no effect.

Parameters

\begin{verbatim}
in icup pointer to the ICUDriver object
\end{verbatim}

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

\subsection{7.13.7.17 icu_lld_disable_notifications()}

\begin{verbatim}
void icu_lld_disable_notifications ( ICUDriver * icup )
\end{verbatim}

Disables notifications.

Precondition

The ICU unit must have been activated using \texttt{icuStart()} and the capture started using \texttt{icuStartCapture()}.  

Note

If the notification is already disabled then the call has no effect.

Parameters

\begin{verbatim}
in icup pointer to the ICUDriver object
\end{verbatim}

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

\subsection{7.13.8 Variable Documentation}
7.13.8.1 ICUD1

ICUDriver ICUD1

ICUD1 driver identifier.

Note

The driver ICUD1 allocates the complex timer TIM1 when enabled.
7.14 Blocks

7.14.1 Detailed Description

Modules

- Abstract I/O Block Device
7.15 Streams

7.15.1 Detailed Description

Modules

- Abstract Streams Interface
- Abstract I/O Channel Interface
- Abstract Files Interface
- Abstract Persistent Storage Interface
- Memory Streams Class
- Null Streams Class
- Buffered Serial Class
- Buffered Streams Class
- Output Formatter Utility
- Input Formatter Utility
7.16 Abstract Streams Interface

7.16.1 Detailed Description

This module define an abstract interface for generic data streams. Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to data streams independent from the implementation logic. The stream interface can be used as base class for high level object types such as files, sockets, serial ports, pipes etc.

Macros

• #define _base_sequential_stream_methods
  BaseSequentialStream specific methods.
• #define _base_sequential_stream_data _base_object_data
  BaseSequentialStream specific data.
• #define _base_buffered_stream_methods
  BaseBufferedStream specific methods.
• #define _base_buffered_stream_data _base_sequential_stream_data
  BaseBufferedStream specific data.

Streams return codes

• #define STM_OK MSG_OK
• #define STM_TIMEOUT MSG_TIMEOUT
• #define STM_RESET MSG_RESET

Macro Functions (BaseSequentialStream)

• #define streamWrite(ip, bp, n) ((ip)->vmt->write(ip, bp, n))
  Sequential Stream write.
• #define streamRead(ip, bp, n) ((ip)->vmt->read(ip, bp, n))
  Sequential Stream read.
• #define streamPut(ip, b) ((ip)->vmt->put(ip, b))
  Sequential Stream blocking byte write.
• #define streamGet(ip) ((ip)->vmt->get(ip))
  Sequential Stream blocking byte read.

Macro Functions (BaseBufferedStream)

• #define streamUnget(ip, b) ((ip)->vmt->unget(ip, b))
  Buffered Stream unget.
Data Structures

- struct BaseSequentialStreamVMT
  
  BaseSequentialStream virtual methods table.
- struct BaseSequentialStream
  
  Base stream class.
- struct BaseBufferedStreamVMT
  
  BaseBufferedStream virtual methods table.
- struct BaseBufferedStream
  
  Buffered stream class.

7.16.2 Macro Definition Documentation

7.16.2.1 _base_sequential_stream_methods

#define _base_sequential_stream_methods

Value:

_value: _base_object_methods

BaseSequentialStream specific methods.

7.16.2.2 _base_sequential_stream_data

#define _base_sequential_stream_data _base_object_data

BaseSequentialStream specific data.

Note

It is empty because BaseSequentialStream is only an interface without implementation.

7.16.2.3 streamWrite

#define streamWrite(ip, bp, n) ((ip)->vmt->write(ip, bp, n))

Sequential Stream write.

The function writes data from a buffer to a stream.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseSequentialStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</tbody>
</table>

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.2.4 streamRead

```c
#define streamRead( 
    ip, 
    bp, 
    n ) ((ip)->vmt->read(ip, bp, n))
```

Sequential Stream read.

The function reads data from a stream into a buffer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseSequentialStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</tbody>
</table>

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.2.5 streamPut

```c
#define streamPut( 
    ip, 
    b   ) ((ip)->vmt->put(ip, b))
```
Sequential Stream blocking byte write.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

| in ip | pointer to a BaseChannel or derived class |
| in b  | the byte value to be written to the channel |

Returns

The operation status.

Return values

| STM_OK  | if the operation succeeded. |
| STM_RESET | if an end-of-file condition has been met. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.2.6 streamGet

```
#define streamGet(
    ip ) ((ip)->vmt->get(ip))
```

Sequential Stream blocking byte read.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

Parameters

| in ip | pointer to a BaseChannel or derived class |

Returns

A byte value from the queue.

Return values

| STM_RESET | if an end-of-file condition has been met. |
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.2.7 _base_buffered_stream_methods

#define _base_buffered_stream_methods

Value:
_base_sequential_stream_methods
/* Channel unget method */
msg_t (*unget)(void *instance, uint8_t b);

BaseBufferedStream specific methods.

7.16.2.8 _base_buffered_stream_data

#define _base_buffered_stream_data _base_sequential_stream_data

BaseBufferedStream specific data.

Note
It is empty because BaseBufferedStream is only an interface without implementation.

7.16.2.9 streamUnget

#define streamUnget(ip, b) ((ip)->vmt->unget(ip, b))

Buffered Stream unget.
This function replaces a byte value to a stream. streamUnget only guarantees a single byte can be replaced, and multiple calls without intervening calls to streamGet or streamRead may fail.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseBufferedStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written to the channel</td>
</tr>
</tbody>
</table>

Postcondition
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>STM_OK</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM_RESET</td>
<td>if the operation failed</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.17 Abstract I/O Channel Interface

7.17.1 Detailed Description

This module defines an abstract interface for I/O channels by extending the `BaseSequentialStream` interface. Note that no code is present, I/O channels are just abstract interface like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). Specific device drivers can use/extend the interface and implement them.

This system has the advantage to make the access to channels independent from the implementation logic.

Macros

- `#define _base_channel_methods`  
  BaseChannel specific methods.
- `#define _base_channel_data _base_sequential_stream_data`  
  BaseChannel specific data.
- `#define _base_asynchronous_channel_methods _base_channel_methods`  
  BaseAsynchronousChannel specific methods.
- `#define _base_asynchronous_channel_data`  
  BaseAsynchronousChannel specific data.

Default control operation codes.

- `#define CHN_CTL_INVALID 0`  
  Invalid operation code.
- `#define CHN_CTL_NOP 1`  
  Does nothing.
- `#define CHN_CTL_TX_WAIT 2`  
  Wait for TX completion.

Macro Functions (BaseChannel)

- `#define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))`  
  Channel blocking byte write with timeout.
- `#define chnGetTimeout(ip, time) ((ip)->vmt->gett(ip, time))`  
  Channel blocking byte read with timeout.
- `#define chnWrite(ip, bp, n) streamWrite(ip, bp, n)`  
  Channel blocking write.
- `#define chnWriteTimeout(ip, bp, n, time) ((ip)->vmt->writet(ip, bp, n, time))`  
  Channel blocking write with timeout.
- `#define chnRead(ip, bp, n) streamRead(ip, bp, n)`  
  Channel blocking read.
- `#define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))`  
  Channel blocking read with timeout.
- `#define chnControl(ip, operation, arg) ((ip)->vmt->ctl(ip, operation, arg))`  
  Control operation on a channel.
I/O status flags added to the event listener

- `#define CHN_NO_ERROR (eventflags_t)0`
  No pending conditions.
- `#define CHN_CONNECTED (eventflags_t)1`
  Connection happened.
- `#define CHN_DISCONNECTED (eventflags_t)2`
  Disconnection happened.
- `#define CHN_INPUT_AVAILABLE (eventflags_t)4`
  Data available in the input queue.
- `#define CHN_OUTPUT_EMPTY (eventflags_t)8`
  Output queue empty.
- `#define CHN_TRANSMISSION_END (eventflags_t)16`
  Transmission end.
- `#define CHN_PARITY_ERROR (eventflags_t)32`
  Parity error.
- `#define CHN_FRAMING_ERROR (eventflags_t)64`
  Framing error.
- `#define CHN_NOISE_ERROR (eventflags_t)128`
  Line noise error.
- `#define CHN_OVERRUN_ERROR (eventflags_t)256`
  Overflow error.
- `#define CHN_IDLE_DETECTED (eventflags_t)512`
  RX line idle.
- `#define CHN_BREAK_DETECTED (eventflags_t)1024`
  LIN Break.
- `#define CHN_BUFFER_FULL_ERROR (eventflags_t)2048`

Macro Functions (BaseAsynchronousChannel)

- `#define chnGetEventSource(ip) (&((ip)->event))`
  Returns the I/O condition event source.
- `#define chnAddFlags(ip, flags)`
  Adds status flags to the listener's flags mask.

Data Structures

- `struct BaseChannelVMT`
  BaseChannel virtual methods table.
- `struct BaseChannel`
  Base channel class.
- `struct BaseAsynchronousChannelVMT`
  BaseAsynchronousChannel virtual methods table.
- `struct BaseAsynchronousChannel`
  Base asynchronous channel class.

7.17.2 Macro Definition Documentation
7.17.2.1 CHN_CTL_INVALID

#define CHN_CTL_INVALID 0

Invalid operation code.

7.17.2.2 CHN_CTL_NOP

#define CHN_CTL_NOP 1

Does nothing.

7.17.2.3 CHN_CTL_TX_WAIT

#define CHN_CTL_TX_WAIT 2

Wait for TX completion.

7.17.2.4 _base_channel_methods

#define _base_channel_methods

Value:

_base_sequential_stream_methods
/-- Channel put method with timeout specification.*/
msg_t (*putt)(void *instance, uint8_t b, sysinterval_t time);
/-- Channel get method with timeout specification.*/
msg_t (*gett)(void *instance, sysinterval_t time);
/-- Channel write method with timeout specification.*/
size_t (*writet)(void *instance, const uint8_t *bp,
 size_t n, sysinterval_t time);
/-- Channel read method with timeout specification.*/
size_t (*readt)(void *instance, uint8_t *bp, size_t n,
 sysinterval_t time);
/-- Channel control method.*/
msg_t (*ctl)(void *instance, unsigned int operation, void *arg);

BaseChannel specific methods.
### 7.17.2.5 _base_channel_data

```c
#define _base_channel_data _base_sequential_stream_data
```

Diese Funktion schreibt einen Byte-Wert an eine Channel. Wenn der Channel nicht bereit ist, um Daten anzunehmen, wird der aufrufende Thread suspendiert.

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>pointer to a <code>BaseChannel</code> or derived class</td>
</tr>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written to the channel</td>
</tr>
<tr>
<td>in</td>
<td>time</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

### Returns

The operation status.

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>STM_OK</code></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><code>STM_TIMEOUT</code></td>
<td>if the specified time expired.</td>
</tr>
<tr>
<td><code>STM_RESET</code></td>
<td>if the channel associated queue (if any) was reset.</td>
</tr>
</tbody>
</table>

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.17.2.7 chnGetTimeout

#define chnGetTimeout(
ip,
    time ) ((ip)->vmt->gett(ip, time))

Channel blocking byte read with timeout.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>pointer to a BaseChannel or derived class</td>
</tr>
<tr>
<td>in</td>
<td>time</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

A byte value from the queue.

Return values

| STM_TIMEOUT | if the specified time expired. |
| STM_RESET | if the channel associated queue (if any) has been reset. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.8 chnWrite

#define chnWrite(
ip,
    bp,
    n ) streamWrite(ip, bp, n)

Channel blocking write.

The function writes data from a buffer to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>pointer to a BaseChannel or derived class</td>
</tr>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</tbody>
</table>
Returns

The number of bytes transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.9 chnWriteTimeout

#define chnWriteTimeout({
    \(ip\),
    \(bp\),
    \(n\),
    \(time\) (\(ip\)->vmt->writet(ip, bp, n, time))

Channel blocking write with timeout.

The function writes data from a buffer to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseChannel or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
<tr>
<td>in</td>
<td>time</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.10 chnRead

#define chnRead({
    \(ip\),
    \(bp\),
    \(n\) streamRead(ip, bp, n)

Channel blocking read.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseChannel or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</tbody>
</table>

Returns

The number of bytes transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.11 chnReadTimeout

#define chnReadTimeout(
    ip,  
    bp,  
    n,   
    time) ((ip)->vmt->read(ip, bp, n, time))

Channel blocking read with timeout.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseChannel or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
<tr>
<td>in</td>
<td>time</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.17 Abstract I/O Channel Interface

7.17.2.12 chnControl


```c
#define chnControl(
    ip,
    operation,
    arg ) ((ip)->vmt->ctl(ip, operation, arg))
```

Control operation on a channel.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>ip</code> pointer to a BaseChannel or derived class</td>
</tr>
<tr>
<td>in</td>
<td><code>operation</code> control operation code</td>
</tr>
<tr>
<td>in, out</td>
<td><code>arg</code> operation argument</td>
</tr>
</tbody>
</table>

Returns

The control operation status.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>in case of success.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>in case of operation timeout.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>in case of operation reset.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.13 CHN_NO_ERROR


```c
#define CHN_NO_ERROR (eventflags_t)0
```

No pending conditions.

7.17.2.14 CHN_CONNECTED


```c
#define CHN_CONNECTED (eventflags_t)1
```

Connection happened.
7.17.2.15 CHN_DISCONNECTED

#define CHN_DISCONNECTED (eventflags_t)2

Disconnection happened.

7.17.2.16 CHN_INPUT_AVAILABLE

#define CHN_INPUT_AVAILABLE (eventflags_t)4

Data available in the input queue.

7.17.2.17 CHN_OUTPUT_EMPTY

#define CHN_OUTPUT_EMPTY (eventflags_t)8

Output queue empty.

7.17.2.18 CHN_TRANSMISSION_END

#define CHN_TRANSMISSION_END (eventflags_t)16

Transmission end.

7.17.2.19 CHN_PARITY_ERROR

#define CHN_PARITY_ERROR (eventflags_t)32

Parity error.

7.17.2.20 CHN_FRAMING_ERROR

#define CHN_FRAMING_ERROR (eventflags_t)64

Framing error.
7.17 Abstract I/O Channel Interface

7.17.21 CHN_NOISE_ERROR

#define CHN_NOISE_ERROR (eventflags_t)128
Line noise error.

7.17.22 CHN_OVERRUN_ERROR

#define CHN_OVERRUN_ERROR (eventflags_t)256
Overflow error.

7.17.23 CHN_IDLE_DETECTED

#define CHN_IDLE_DETECTED (eventflags_t)512
RX line idle.

7.17.24 CHN_BREAK_DETECTED

#define CHN_BREAK_DETECTED (eventflags_t)1024
LIN Break.
RX buffer full.

7.17.25 _base_asynchronous_channel_methods

#define _base_asynchronous_channel_methods _base_channel_methods \BaseAsynchronousChannel specific methods.

7.17.26 _base_asynchronous_channel_data

#define _base_asynchronous_channel_data

Value:

_base_channel_data
// I/O condition event source.*/
evento_t event;
BaseAsynchronousChannel specific data.

7.17.27 chnGetEventSource

#define chnGetEventSource(ip) &((ip)->event)
Returns the I/O condition event source.
The event source is broadcasted when an I/O condition happens.
Parameters

| in  | ip  | pointer to a **BaseAsynchronousChannel** or derived class |

Returns

A pointer to an **EventSource** object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.28  **chnAddFlagsI**

```c
#define chnAddFlagsI(
   ip,
   flags )
```

Value:

```c
{ osalEventBroadcastFlagsI(&(ip)->event, flags); }
```

Adds status flags to the listeners's flags mask.

This function is usually called from the I/O ISRs in order to notify I/O conditions such as data events, errors, signal changes etc.

Parameters

| in  | ip  | pointer to a **BaseAsynchronousChannel** or derived class |
| in  | flags | condition flags to be added to the listener flags mask |

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
7.18 Abstract Files Interface

7.18.1 Detailed Description

This module defines an abstract interface for generic data files by extending the `BaseSequentialStream` interface. Note that no code is present, data files are just abstract interface-like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). This system has the advantage to make the access to streams independent from the implementation logic.

The data files interface can be used as base class for high level object types such as an API for a File System implementation.

Macros

- `#define _file_stream_methods`  
  `FileStream` specific methods.
- `#define _file_stream_data_base_sequential_stream_data`  
  `FileStream` specific data.

Files return codes

- `#define FILE_OK STM_OK`  
  No error return code.
- `#define FILE_ERROR STM_TIMEOUT`  
  Error code from the file stream methods.
- `#define FILE_EOF STM_RESET`  
  End-of-file condition for file get/put methods.

Macro Functions (FileStream)

- `#define fileStreamWrite(ip, bp, n) streamWrite(ip, bp, n)`  
  File stream write.
- `#define fileStreamRead(ip, bp, n) streamRead(ip, bp, n)`  
  File stream read.
- `#define fileStreamPut(ip, b) streamPut(ip, b)`  
  File stream blocking byte write.
- `#define fileStreamGet(ip) streamGet(ip)`  
  File stream blocking byte read.
- `#define fileStreamClose(ip) ((ip)->vmt->close(ip))`  
  File Stream close.
- `#define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))`  
  Returns an implementation dependent error code.
- `#define fileStreamGetSize(ip, offset) ((ip)->vmt->getsize(ip, offset))`  
  Returns the current file size.
- `#define fileStreamGetPosition(ip, offset) ((ip)->vmt->getposition(ip, offset))`  
  Returns the current file pointer position.
- `#define fileStreamSetPosition(ip, offset) ((ip)->vmt->setposition(ip, offset))`  
  Moves the file current pointer to an absolute position.
Typedefs

- typedef uint32_t fileoffset_t
  
  *File offset type.*

Data Structures

- struct FileStreamVMT
  
  *FileStream virtual methods table.*

- struct FileStream
  
  *Base file stream class.*

7.18.2 Macro Definition Documentation

7.18.2.1 FILE_OK

#define FILE_OK STM_OK

No error return code.

7.18.2.2 FILE_ERROR

#define FILE_ERROR STM_TIMEOUT

Error code from the file stream methods.

7.18.2.3 FILE_EOF

#define FILE_EOF STM_RESET

End-of-file condition for file get/put methods.


7.18.2.4 _file_stream_methods

#define _file_stream_methods

Value:

```c
#define _base_sequential_stream_methods
/* File close method.*/
msg_t (*close)(void *instance);
/* Get last error code method.*/
msg_t (*geterror)(void *instance);
/* File get size method.*/
msg_t (*getsize)(void *instance, fileoffset_t *offset);
/* File get current position method.*/
msg_t (*getposition)(void *instance, fileoffset_t *offset);
/* File set current position method.*/
msg_t (*setposition)(void *instance, fileoffset_t offset);
```

FileStream specific methods.

7.18.2.5 _file_stream_data

#define _file_stream_data _base_sequential_stream_data

FileStream specific data.

Note

It is empty because FileStream is only an interface without implementation.

7.18.2.6 fileStreamWrite

#define fileStreamWrite(
    ip,
    bp,
    n
) streamWrite(ip, bp, n)

FileStream write.

The function writes data from a buffer to a file stream.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</tbody>
</table>

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.
Return values

| FILE_ERROR | operation failed |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.18.2.7 fileStreamRead

```c
#define fileStreamRead(
    ip,
    bp,
    n ) streamRead(ip, bp, n)
```

File stream read.

The function reads data from a file stream into a buffer.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</tbody>
</table>

**Returns**

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

**Return values**

| FILE_ERROR | operation failed |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.18.2.8 fileStreamPut

```c
#define fileStreamPut(
    ip,
    b ) streamPut(ip, b)
```

File stream blocking byte write.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

ChibiOS/HAL
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written to the channel</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>FILE_OK</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_ERROR</td>
<td>operation failed.</td>
</tr>
<tr>
<td>FILE_EOF</td>
<td>if an end-of-file condition has been met.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.18.2.9 fileStreamGet

#define fileStreamGet(
    ip ) streamGet(ip)

File stream blocking byte read.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
</table>

Returns

A byte value from the queue.

Return values

<table>
<thead>
<tr>
<th>FILE_ERROR</th>
<th>operation failed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_EOF</td>
<td>if an end-of-file condition has been met.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.18.2.10 fileStreamClose

#define fileStreamClose(ip) ((ip)->vmt->close(ip))

File Stream close.

The function closes a file stream.

Parameters

| in | ip | pointer to a FileStream or derived class |

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>FILE_OK</th>
<th>no error.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_ERROR</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.18.2.11 fileStreamGetError

#define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))

Returns an implementation dependent error code.

Precondition

The previously called function must have returned FILE_ERROR.

Parameters

| in | ip | pointer to a FileStream or derived class |

Returns

Implementation dependent error code.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.18.2.12 fileStreamGetSize

```c
#define fileStreamGetSize(ip, offset) ((ip)->vmt->getsize(ip, offset))
```

Returns the current file size.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>offset</td>
<td>current size of the file</td>
</tr>
</tbody>
</table>

Returns

The file size.

**Return values**

- `FILE_ERROR` operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.18.2.13 fileStreamGetPosition

```c
#define fileStreamGetPosition(ip, offset) ((ip)->vmt->getposition(ip, offset))
```

Returns the current file pointer position.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>offset</td>
<td>current position in the file</td>
</tr>
</tbody>
</table>
Returns

The current position inside the file.

Return values

<table>
<thead>
<tr>
<th>FILE_ERROR</th>
<th>operation failed</th>
</tr>
</thead>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.18.2.14 fileStreamSetPosition

```c
#define fileStreamSetPosition(ip, offset) ((ip)->vmt->setposition(ip, offset))
```

Moves the file current pointer to an absolute position.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>offset</td>
<td>new absolute position</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>FILE_OK</th>
<th>no error.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_ERROR</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.18.3 Typedef Documentation

#### 7.18.3.1 fileoffset_t

```c
typedef uint32_t fileoffset_t
```

File offset type.
7.19 Abstract Persistent Storage Interface

7.19.1 Detailed Description

This module define an abstract interface for generic persistent storage. Such storage has a fixed size and can be read and written.

Macros

- `#define _base_pers_storage_methods_alone`
  `BasePersistentStorage` specific methods.
- `#define _base_pers_storage_methods`
  `BasePersistentStorage` specific methods with inherited ones.
- `#define _base_persistent_storage_data _base_object_data`
  `BasePersistentStorage` specific data.

Macro Functions (BasePersistentStorage)

- `#define getBasePersistentStorage(ip) ((BasePersistentStorage *)(&ip)->vmt)`
  Instance getter.
- `#define psGetStorageSize(ip) (ip)->vmt->getsize(ip)`
  Get storage size.
- `#define psRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)`
  Read operation.
- `#define psWrite(ip, offset, n, wp) (ip)->vmt->write(ip, offset, n, wp)`
  Write operation.

Typedefs

- `typedef uint32_t ps_offset_t`
  Type of a persistent storage offset.

Data Structures

- `struct BasePersistentStorageVMT`
  `BasePersistentStorage` virtual methods table.
- `struct BasePersistentStorage`
  Base persistent storage class.

Enumerations

- `enum ps_error_t`
  Type of a persistent storage error code.
7.19.2  Macro Definition Documentation

7.19.2.1  _base_pers_storage_methods_alone

#define _base_pers_storage_methods_alone

Value:

/* Storage size. */
size_t (*getsize)(void *instance);

/* Read operation. */
ps_error_t (*read)(void *instance, ps_offset_t offset,
size_t n, uint8_t *rp);

/* Write operation. */
ps_error_t (*write)(void *instance, ps_offset_t offset,
size_t n, const uint8_t *wp);

_base_persistent_storage_data specific methods.

7.19.2.2  _base_pers_storage_methods

#define _base_pers_storage_methods

Value:

_base_object_methods
_base_pers_storage_methods_alone

_base_persistent_storage_data specific methods with inherited ones.

7.19.2.3  _base_persistent_storage_data

#define _base_persistent_storage_data _base_object_data

_base_persistent_storage_data specific data.

7.19.2.4  getBasePersistentStorage

#define getBasePersistentStorage(ip) ((BasePersistentStorage *)&(ip)->vmt)

Instance getter.

This special method is used to get the instance of this class object from a derived class.

7.19.2.5  psGetStorageSize

#define psGetStorageSize(ip) (ip)->vmt->getsize(ip)

Get storage size.
7.19 Abstract Persistent Storage Interface

Parameters

| in  | ip | pointer to a BasePersistentStorage or derived class |

Returns

The storage size in bytes.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.19.2.6 psRead

#define psRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)

Read operation.

Parameters

| in  | ip          | pointer to a BasePersistentStorage or derived class |
|     | offset      | persistent storage offset                               |
| in  | n           | number of bytes to be read                               |
| out | rp          | pointer to the data buffer                              |

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>PS_NO_ERROR</th>
<th>if there is no erase operation in progress.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_ERROR_READ</td>
<td>if the read operation failed.</td>
</tr>
<tr>
<td>PS_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.19.2.7 psWrite

```c
#define psWrite(ip, offset, n, wp) (ip)->vmt->write(ip, offset, n, wp)
```

Write operation.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>ip</code></td>
<td>pointer to a <code>BasePersistentStorage</code> or derived class</td>
</tr>
<tr>
<td>in <code>offset</code></td>
<td>persistent storage offset</td>
</tr>
<tr>
<td>in <code>n</code></td>
<td>number of bytes to be written</td>
</tr>
<tr>
<td>in <code>wp</code></td>
<td>pointer to the data buffer</td>
</tr>
</tbody>
</table>

### Returns

An error code.

### Return values

- `PS_NO_ERROR` if there is no erase operation in progress.
- `PS_ERROR_WRITE` if the write operation failed.
- `PS_ERROR_HW_FAILURE` if access to the memory failed.

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.19.3 Typedef Documentation

7.19.3.1 `ps_offset_t`

```c
typedef uint32_t ps_offset_t
```

Type of a persistent storage offset.

7.19.4 Enumeration Type Documentation
7.19 Abstract Persistent Storage Interface

7.19.4.1 ps_error_t

enum ps_error_t

Type of a persistent storage error code.

Note

Code values are kept equal to the equivalent codes in the flash interface, this is intentional.
7.20 Memory Streams Class

7.20.1 Detailed Description

Memory buffers handled as streams.

Macros

- \#define _memory_stream_data
  MemStream specific data.

Data Structures

- struct MemStreamVMT
  MemStream virtual methods table.
- struct MemoryStream
  Memory stream object.

Functions

- void msObjectInit (MemoryStream *msp, uint8_t *buffer, size_t size, size_t eos)
  Memory stream object initialization.

7.20.2 Macro Definition Documentation

7.20.2.1 _memory_stream_data

#define _memory_stream_data

Value:

_base_buffered_stream_data
  /* Pointer to the stream buffer.*/
  uint8_t *buffer;
  /* Size of the stream.*/
  size_t size;
  /* Current end of stream.*/
  size_t eos;
  /* Current read offset.*/
  size_t offset;

MemStream specific data.

7.20.3 Function Documentation

7.20.3.1 msObjectInit()

void msObjectInit (MemoryStream *msp, uint8_t *buffer, size_t size, size_t eos)

Memory stream object initialization.
### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msp</td>
<td>pointer to the <code>MemoryStream</code> object to be initialized</td>
</tr>
<tr>
<td>in</td>
<td>buffer</td>
<td>pointer to the memory buffer for the memory stream</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>total size of the memory stream buffer</td>
</tr>
<tr>
<td>in</td>
<td>eos</td>
<td>Initial End Of Stream offset. Normally you need to put this to zero for RAM buffers or equal to <code>size</code> for ROM streams.</td>
</tr>
</tbody>
</table>
7.21 Null Streams Class

7.21.1 Detailed Description

A null streams.

Macros

- \#define _null_stream_data _base_buffered_stream_data
  
  NullStream specific data.

Data Structures

- struct NullStreamVMT
  
  NullStream virtual methods table.
- struct NullStream
  
  Null stream object.

Functions

- void nullObjectInit (NullStream *nsp)
  
  Null stream object initialization.

7.21.2 Macro Definition Documentation

7.21.2.1 _null_stream_data

\#define _null_stream_data _base_buffered_stream_data

NullStream specific data.

7.21.3 Function Documentation

7.21.3.1 nullObjectInit()

void nullObjectInit (  
  NullStream * nsp )

Null stream object initialization.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>nsp</code></td>
<td>pointer to the <code>NullStream</code> object to be initialized</td>
</tr>
</tbody>
</table>
### 7.22 Buffered Serial Class

#### 7.22.1 Detailed Description

**Macros**

- `#define __buffered_serial_methods _base_asynchronous_channel_methods`  
  BufferedSerial specific methods.
- `#define __buffered_serial_data`  
  BufferedSerial specific data.

**Serial status flags (legacy)**

- `#define SD_PARITY_ERROR CHN_PARITY_ERROR`
- `#define SD_FRAMING_ERROR CHN_FRAMING_ERROR`
- `#define SD_OVERRUN_ERROR CHN_OVERRUN_ERROR`
- `#define SD_NOISE_ERROR CHN_NOISE_ERROR`
- `#define SD_BREAK_DETECTED CHN_BREAK_DETECTED`
- `#define SD_QUEUE_FULL_ERROR CHN_BUFFER_FULL_ERROR`

**Methods implementations**

- static `CC_FORCE_INLINE void __buffered_serial_objinit_impl (void *ip, const void *vmt, uint8_t *ib, size_t ibsize, qnotify_t inotify, void *iarg, uint8_t *ob, size_t obsize, qnotify_t onotify, void *oarg)`  
  Object initialization implementation.
- static `CC_FORCE_INLINE size_t __buffered_serial_write_impl (void *ip, const uint8_t *bp, size_t n)`  
  ```
  • static CC_FORCE_INLINE size_t __buffered_serial_write_impl (void *ip, const uint8_t *bp, size_t n)  
  • static CC_FORCE_INLINE size_t __buffered_serial_read_impl (void *ip, uint8_t *bp, size_t n)  
  • static CC_FORCE_INLINE msg_t __buffered_serial_put_impl (void *ip, uint8_t b)  
  • static CC_FORCE_INLINE msg_t __buffered_serial_get_impl (void *ip)  
  • static CC_FORCE_INLINE msg_t __buffered_serial_put_timeout_impl (void *ip, uint8_t b, sysinterval_t timeout)  
  • static CC_FORCE_INLINE msg_t __buffered_serial_get_timeout_impl (void *ip, sysinterval_t timeout)  
  • static CC_FORCE_INLINE size_t __buffered_serial_write_timeout_impl (void *ip, const uint8_t *bp, size_t n, sysinterval_t timeout)  
  • static CC_FORCE_INLINE size_t __buffered_serial_read_timeout_impl (void *ip, uint8_t *bp, size_t n, sysinterval_t timeout)  
  • static CC_FORCE_INLINE msg_t __buffered_serial_ctl_impl (void *ip, unsigned int operation, void *arg)`

**Typedefs**

- typedef struct hal_buffered_serial BufferedSerial  
  Structure representing a buffered serial class.

**Data Structures**

- struct BufferedSerialVMT  
  BufferedSerial virtual methods table.
- struct hal_buffered_serial  
  Buffered serial channel class.
7.22 Buffered Serial Class

Functions

- void bsIncomingData (BufferedSerial *bsp, uint8_t b)
  - Handles incoming data.
- msg_t bsRequestData (BufferedSerial *bsp)
  - Handles outgoing data.

Enumerations

- enum bsstate_t { BS_UNINIT = 0, BS_STOP = 1, BS_READY = 2 }
  - BufferedSerial state machine states.

7.22.2 Macro Definition Documentation

7.22.2.1 __buffered_serial_methods

#define __buffered_serial_methods _base_asynchronous_channel_methods

BufferedSerial specific methods.

7.22.2.2 __buffered_serial_data

#define __buffered_serial_data

Value:

__base_asynchronous_channel_data

/* Driver state.*/ bsstate_t state;
/* Input queue.*/ input_queue_t iqqueue;
/* Output queue.*/ output_queue_t oqueue;

BufferedSerial specific data.

7.22.3 Typedef Documentation

7.22.3.1 BufferedSerial

typedef struct hal_buffered_serial BufferedSerial

Structure representing a buffered serial class.

7.22.4 Enumeration Type Documentation

7.22.4.1 bsstate_t

enum bsstate_t

BufferedSerial state machine states.
### 7.22.5 Function Documentation

#### 7.22.5.1 bsIncomingDataI()

```c
void bsIncomingDataI ( BufferedSerial * bsp, uint8_t b )
```

Handles incoming data.

This function must be called from the input interrupt service routine in order to enqueue incoming data and generate the related events.

**Note**

The incoming data event is only generated when the input queue becomes non-empty.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>bsp</th>
<th>pointer to a BufferedSerial structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte to be written in the driver’s Input Queue</td>
</tr>
</tbody>
</table>

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
bsIncomingDataI -> iqPutI -> osalThreadDequeueNextI
```
### 7.22.5.2 bsRequestDataI()

```c
msg_t bsRequestDataI ( BufferedSerial * bsp )
```

Handles outgoing data.

Must be called from the output interrupt service routine in order to get the next byte to be transmitted.

**Note**

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

**Parameters**

| in  | bsp | pointer to a BufferedSerial structure |

**Returns**

The byte value read from the driver’s output queue.

**Return values**

- `MSG_TIMEOUT` if the queue is empty.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.22.5.3 __buffered_serial_objinit_impl()

```c
static CC_FORCE_INLINE void __buffered_serial_objinit_impl ( void * ip,
                                                             const void * vmt,
                                                             const void * vmf,
                                                             void * io,
                                                             void * nvs )
```

ChibiOS/HAL
void initBufferedSerial(
        uint8_t *ib,
        size_t ibsize,
        qnotify_t inotify,
        void *iarg,
        uint8_t *ob,
        size_t obsize,
        qnotify_t onotify,
        void *oarg ) [inline], [static]

Object initialization implementation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>Pointer to a BufferedSerial structure to be initialized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vmt</td>
<td>VMT pointer for the new object.</td>
</tr>
<tr>
<td>in</td>
<td>ib</td>
<td>pointer to the input buffer</td>
</tr>
<tr>
<td>in</td>
<td>ibsize</td>
<td>size of the input buffer</td>
</tr>
<tr>
<td>in</td>
<td>inotify</td>
<td>pointer to a callback function that is invoked when some data is read from the input queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value can be NULL.</td>
</tr>
<tr>
<td>in</td>
<td>iarg</td>
<td>parameter for the input notification callback</td>
</tr>
<tr>
<td>in</td>
<td>ob</td>
<td>pointer to the output buffer</td>
</tr>
<tr>
<td>in</td>
<td>obsize</td>
<td>size of the output buffer</td>
</tr>
<tr>
<td>in</td>
<td>onotify</td>
<td>pointer to a callback function that is invoked when some data is written in the output queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value can be NULL.</td>
</tr>
<tr>
<td>in</td>
<td>oarg</td>
<td>parameter for the output notification callback</td>
</tr>
</tbody>
</table>

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.
7.23 Buffered Streams Class

7.23.1 Detailed Description

Wrapper for BaseSequentialStreams that allows some ungetting.

Macros

- `#define BUFSTREAM_BUFFER_SIZE 1`
  Buffer size for unget.
- `#define _buffered_stream_adapter_data`
  BufferedStreamAdapter specific data.

Data Structures

- `struct BufferedStreamAdapterVMT`
  BufferedStreamAdapter virtual methods table.
- `struct BufferedStreamAdapter`
  Buffered stream adapter object.

Functions

- `void bsaObjectInit (BufferedStreamAdapter *bsap, BaseSequentialStream *bssp)`
  Buffered stream adapter object initialization.

7.23.2 Macro Definition Documentation

7.23.2.1 BUFSTREAM_BUFFER_SIZE

`#define BUFSTREAM_BUFFER_SIZE 1`

Buffer size for unget.

7.23.2.2 _buffered_stream_adapter_data

`#define _buffered_stream_adapter_data`

Value:

```
_base_buffered_stream_data
  // Pointer to a wrapped BaseSequentialStream object *bss;
  BaseSequentialStream* bss;
  // Stream buffer */
  uint8_t buffer[BUFSTREAM_BUFFER_SIZE];
  // Index in the stream buffer */
  size_t ndx;
```

BufferedStreamAdapter specific data.
7.23.3 Function Documentation

7.23.3.1 bsaObjectInit()

```c
void bsaObjectInit (
    BufferedStreamAdapter * bsap,
    BaseSequentialStream * bssp )
```

Buffered stream adapter object initialization.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bsap</td>
<td>pointer to the BufferedStream object to be initialized</td>
</tr>
<tr>
<td>in</td>
<td>bssp</td>
<td>pointer to a BaseSequentialStream fulfilling object to be wrapped</td>
</tr>
</tbody>
</table>
7.24 Output Formatter Utility

7.24.1 Detailed Description

Mini printf-like functionality.

Macros

- \#define CHPRINTF_USE_FLOAT FALSE
  Float type support.

Functions

- int chvprintf (BaseSequentialStream *chp, const char *fmt, va_list ap)
  System formatted output function.
- int chprintf (BaseSequentialStream *chp, const char *fmt, ...)
  System formatted output function.
- int chsnprintf (char *str, size_t size, const char *fmt, ...)
  System formatted output function.
- int chvsnprintf (char *str, size_t size, const char *fmt, va_list ap)
  System formatted output function.

7.24.2 Macro Definition Documentation

7.24.2.1 CHPRINTF_USE_FLOAT

\#define CHPRINTF_USE_FLOAT FALSE

Float type support.

7.24.3 Function Documentation
7.24.3.1 chvprintf()

```c
int chvprintf (    
    BaseSequentialStream * chp,  
    const char * fmt,  
    va_list ap )
```

System formatted output function.

This function implements a minimal vprintf()-like functionality with output on a `BaseSequentialStream`. The general parameters format is: `%[-][width][.precision][lL]p`. The following parameter types (p) are supported:

- x hexadecimal integer.
- X hexadecimal long.
- o octal integer.
- O octal long.
- d decimal signed integer.
- D decimal signed long.
- u decimal unsigned integer.
- U decimal unsigned long.
- c character.
- s string.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>chp</td>
</tr>
<tr>
<td>in</td>
<td>fmt</td>
</tr>
<tr>
<td>in</td>
<td>ap</td>
</tr>
</tbody>
</table>

**Returns**

The number of bytes that would have been written to `chp` if no stream error occurs

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.24.3.2 chprintf()

```c
int chprintf (    
    BaseSequentialStream * chp,
```
System formatted output function.

This function implements a minimal printf() like functionality with output on a BaseSequentialStream. The general parameters format is: %[-][width][.precision][][L][L][p]. The following parameter types (p) are supported:

- **x** hexadecimal integer.
- **X** hexadecimal long.
- **o** octal integer.
- **O** octal long.
- **d** decimal signed integer.
- **D** decimal signed long.
- **u** decimal unsigned integer.
- **U** decimal unsigned long.
- **c** character.
- **s** string.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in chp</td>
<td>pointer to a BaseSequentialStream implementing object</td>
</tr>
<tr>
<td>in fmt</td>
<td>formatting string</td>
</tr>
</tbody>
</table>

The number of bytes that would have been written to chp if no stream error occurs

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.24.3.3  chsnprintf()

```c
int chsnprintf(
    char * str,
    size_t size,
    const char * fmt,
    ...)
```

System formatted output function.

This function implements a minimal `snprintf()`-like functionality. The general parameters format is: `%[[-][width][.]precision][l|L]p`. The following parameter types (p) are supported:

- `x` hexadecimal integer.
- `X` hexadecimal long.
- `o` octal integer.
- `O` octal long.
- `d` decimal signed integer.
- `D` decimal signed long.
- `u` decimal unsigned integer.
- `U` decimal unsigned long.
- `c` character.
- `s` string.

**Postcondition**

`str` is NUL-terminated, unless `size` is 0.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>str</th>
<th>pointer to a buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>maximum size of the buffer</td>
</tr>
<tr>
<td>in</td>
<td>fmt</td>
<td>formatting string</td>
</tr>
</tbody>
</table>

**Returns**

The number of characters (excluding the terminating NUL byte) that would have been stored in `str` if there was room.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]

### 7.24.3.4 chvsnprintf()

```c
int chvsnprintf ( 
    char * str, 
    size_t size, 
    const char * fmt, 
    va_list ap )
```

System formatted output function.

This function implements a minimal vsnprintf()-like functionality. The general parameters format is: %[\[-]][width][\].precision][\|L]p. The following parameter types (p) are supported:

- x hexadecimal integer.
- X hexadecimal long.
- o octal integer.
- O octal long.
- d decimal signed integer.
- D decimal signed long.
- u decimal unsigned integer.
- U decimal unsigned long.
- c character.
- s string.

Postcondition

str is NULL-terminated, unless size is 0.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>str</th>
<th>pointer to a buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>maximum size of the buffer</td>
</tr>
<tr>
<td>in</td>
<td>fmt</td>
<td>formatting string</td>
</tr>
<tr>
<td>in</td>
<td>ap</td>
<td>list of parameters</td>
</tr>
</tbody>
</table>

Returns

The number of characters (excluding the terminating NUL byte) that would have been stored in str if there was room.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
chvsnprintf
  
  chvprintf

  msObjectInit
```
7.25 Input Formatter Utility

7.25.1 Detailed Description

Mini `scanf`-like functionality.

Macros

- `#define CHSCANF_USE_FLOAT FALSE`
  
  Float type support.

Functions

- `int chvscanf (BaseBufferedStream *chp, const char *fmt, va_list ap)`
  
  System formatted input function.
- `int chscanf (BaseBufferedStream *chp, const char *fmt,...)`
  
  System formatted input function.
- `int chsnscanf (char *str, size_t size, const char *fmt,...)`
  
  System formatted input function.
- `int chvsnscanf (char *str, size_t size, const char *fmt, va_list ap)`
  
  System formatted input function.

7.25.2 Macro Definition Documentation

7.25.2.1 CHSCANF_USE_FLOAT

`#define CHSCANF_USE_FLOAT FALSE`

Float type support.

7.25.3 Function Documentation
7.25.3.1 chvscanf()

int chvscanf (  
    BaseBufferedStream * chp,  
    const char * fmt,  
    va_list ap )

System formatted input function.

This function implements a minimal vscanf()-like functionality with input on a BaseSequentialStream. The general parameters format is: \%[+][-][width][lL]p The following parameter types (p) are supported:

- x hexadecimal integer.
- X hexadecimal long.
- o octal integer.
- O octal long.
- d decimal signed integer.
- D decimal signed long.
- u decimal unsigned integer.
- U decimal unsigned long.
- c character.
- s string.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>chp</th>
<th>pointer to a BufferedStream implementing object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fmt</td>
<td>formatting string</td>
</tr>
<tr>
<td>in</td>
<td>ap</td>
<td>list of parameters</td>
</tr>
</tbody>
</table>

Returns

The number parameters in ap that have been successfully filled. This does not conform to the standard in that if a failure (either matching or input) occurs before any parameters are assigned, the function will return 0.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.25.3.2 chscanf()

int chscanf (  
    BaseBufferedStream * chp,  
    ChibiOS/HAL
const char * fmt, 
...)

System formatted input function.

This function implements a minimal `scanf()` like functionality with input from a `BufferedStream`. The general parameters format is: `%\[\[width\][I|L]p` The following parameter types (p) are supported:

- `x` hexadecimal integer.
- `X` hexadecimal long.
- `o` octal integer.
- `O` octal long.
- `d` decimal signed integer.
- `D` decimal signed long.
- `u` decimal unsigned integer.
- `U` decimal unsigned long.
- `c` character.
- `s` string.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>chp</th>
<th>pointer to a <code>BufferedStream</code> implementing object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fmt</td>
<td>formatting string</td>
</tr>
</tbody>
</table>

**Returns**

The number parameters in ap that have been successfully filled. This does not conform to the standard in that if a failure (either matching or input) occurs before any parameters are assigned, the function will return 0.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
  chscanf -> chvscanf
```
7.25.3.3 chsnscanf()

```
int chsnscanf (  
    char * str,  
    size_t size,  
    const char * fmt,  
    ...  )
```

System formatted input function.

This function implements a minimal `snscanf()`-like functionality. The general parameters format is:

```
%[[+][width][l|L]]p
```

The following parameter types (p) are supported:

- `x` hexadecimal integer.
- `X` hexadecimal long.
- `o` octal integer.
- `O` octal long.
- `d` decimal signed integer.
- `D` decimal signed long.
- `u` decimal unsigned integer.
- `U` decimal unsigned long.
- `c` character.
- `s` string.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>str</code></th>
<th>pointer to a buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>size</code></td>
<td>size of the buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>fmt</code></td>
<td>formatting string</td>
</tr>
</tbody>
</table>

**Returns**

The number parameters in `ap` that have been successfully filled. This does not conform to the standard in that if a failure (either matching or input) occurs before any parameters are assigned, the function will return 0.
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](chsnscanf chvsnscanf chvscanf msObjectInit)

### 7.25.3.4 chvsnscanf()

```c
int chvsnscanf (  
    char * str,  
    size_t size,  
    const char * fmt,  
    va_list ap )
```

System formatted input function.

This function implements a minimal `vsnscanf()`-like functionality. The general parameters format is: `%[+][width][l|L]p` The following parameter types (p) are supported:

- `x` hexadecimal integer.
- `X` hexadecimal long.
- `o` octal integer.
- `O` octal long.
- `d` decimal signed integer.
- `D` decimal signed long.
- `u` decimal unsigned integer.
- `U` decimal unsigned long.
- `c` character.
- `s` string.

#### Parameters

<table>
<thead>
<tr>
<th></th>
<th>str</th>
<th>size</th>
<th>fmt</th>
<th>ap</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pointer to a buffer</td>
<td>size of the buffer</td>
<td>formatting string</td>
<td>list of parameters</td>
</tr>
</tbody>
</table>
Returns

The number parameters in ap that have been successfully filled. This does not conform to the standard in that if a failure (either matching or input) occurs before any parameters are assigned, the function will return 0.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.26 Abstract I/O Block Device

7.26.1 Detailed Description

7.26.2 Driver State Machine

The drivers implementing this interface shall implement the following state machine internally. Not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

This module defines an abstract interface for accessing generic block devices. Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to block devices independent from the implementation logic.

Macros

- #define _base_block_device_methods
  
  *BaseBlockDevice* specific methods.

- #define _base_block_device_data
  
  *BaseBlockDevice* specific data.

Macro Functions (BaseBlockDevice)

- #define blkGetDriverState(ip) ((ip)->state)
  
  Returns the driver state.

- #define blkIsTransferring(ip)
  
  Determines if the device is transferring data.

- #define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))
  
  Returns the media insertion status.

- #define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))
  
  Returns the media write protection status.
• `#define blkConnect(ip) ((ip)->vmt->connect(ip))`
  Performs the initialization procedure on the block device.
• `#define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))`
  Terminates operations on the block device.
• `#define blkRead(ip, startblk, buf, n) ((ip)->vmt->read(ip, startblk, buf, n))`
  Reads one or more blocks.
• `#define blkWrite(ip, startblk, buf, n) ((ip)->vmt->write(ip, startblk, buf, n))`
  Writes one or more blocks.
• `#define blkSync(ip) ((ip)->vmt->sync(ip))`
  Ensures write synchronization.
• `#define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))`
  Returns a media information structure.

**Data Structures**

• `struct BlockDeviceInfo`
  Block device info.
• `struct BaseBlockDeviceVMT`
  BaseBlockDevice virtual methods table.
• `struct BaseBlockDevice`
  Base block device class.

**Enumerations**

• `enum blkstate_t {
  BLK_UNINIT = 0, BLK_STOP = 1, BLK_ACTIVE = 2, BLK_CONNECTING = 3,
  BLK_DISCONNECTING = 4, BLK_READY = 5, BLK_READING = 6, BLK_WRITING = 7,
  BLK_SYNCING = 8 }
  Driver state machine possible states.

### 7.26.3 Macro Definition Documentation

#### 7.26.3.1 `_base_block_device_methods`

```c
#define _base_block_device_methods

value:
  _base_object_methods
  /* Removable media detection. */
  bool (*is_inserted)(void *instance);
  /* Removable write protection detection. */
  bool (*is_protected)(void *instance);
  /* Connection to the block device. */
  bool (*connect)(void *instance);
  /* Disconnection from the block device. */
  bool (*disconnect)(void *instance);
  /* Reads one or more blocks. */
  bool (*read)(void *instance, uint32_t startblk,
               uint8_t *buffer, uint32_t n);
  /* Writes one or more blocks. */
  bool (*write)(void *instance, uint32_t startblk,
                const uint8_t *buffer, uint32_t n);
  /* Write operations synchronization. */
  bool (*sync)(void *instance);
  /* Obtains info about the media. */
  bool (*get_info)(void *instance, BlockDeviceInfo *bdip);

BaseBlockDevice specific methods.
```

ChibiOS/HAL
7.26 Abstract I/O Block Device

7.26.3.2 _base_block_device_data

#define _base_block_device_data

Value:

_base_object_data
/* Driver state. */
blkstate_t state;

BaseBlockDevice specific data.

7.26.3.3 blkGetDriverState

#define blkGetDriverState(ip) ((ip)->state)

Returns the driver state.

Note
Can be called in ISR context.

Parameters

| in | ip | pointer to a BaseBlockDevice or derived class |

Returns
The driver state.

Function Class:
Special function, this function has special requirements see the notes.

7.26.3.4 blkIsTransferring

#define blkIsTransferring(ip)

Value:

(((ip)->state) == BLK_CONNECTING) ||
(((ip)->state) == BLK_DISCONNECTING) ||
(((ip)->state) == BLK_READING) ||
(((ip)->state) == BLK_WRITING)

Determines if the device is transferring data.

Note
Can be called in ISR context.
Parameters

| in  | __ip__ | pointer to a `BaseBlockDevice` or derived class |

Returns

The driver state.

Return values

| false | the device is not transferring data. |
| true  | the device not transferring data. |

Function Class:

Special function, this function has special requirements see the notes.

7.26.3.5 blkIsInserted

```c
#define blkIsInserted(
    __ip__)
    {(__ip__)->vmt->is_inserted(__ip__)}
```

Returns the media insertion status.

Note

On some implementations this function can only be called if the device is not transferring data. The function `blkIsTransferring()` should be used before calling this function.

Parameters

| in  | __ip__ | pointer to a `BaseBlockDevice` or derived class |

Returns

The media state.

Return values

| false | media not inserted. |
| true  | media inserted. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.26 Abstract I/O Block Device

7.26.3.6 blkIsWriteProtected

#define blkIsWriteProtected(ip) 
    ((ip)->vmt->is_protected(ip))

Returns the media write protection status.

Parameters

| in | ip | pointer to a BaseBlockDevice or derived class |

Returns

The media state.

Return values

| false | writable media. |
| true  | non writable media. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.26.3.7 blkConnect

#define blkConnect(ip) 
    ((ip)->vmt->connect(ip))

Performs the initialization procedure on the block device.

This function should be performed before I/O operations can be attempted on the block device and after insertion has been confirmed using blkIsInserted().

Parameters

| in | ip | pointer to a BaseBlockDevice or derived class |

Returns

The operation status.

Return values

| HAL_SUCCESS | operation succeeded. |
| HAL_FAILED  | operation failed. |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.26.3.8 blkDisconnect

```c
#define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))
```

Terminates operations on the block device.

This operation safely terminates operations on the block device.

**Parameters**

| in | ip | pointer to a `BaseBlockDevice` or derived class |

**Returns**

The operation status.

**Return values**

| `HAL_SUCCESS` | operation succeeded. |
| `HAL_FAILED`  | operation failed. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.26.3.9 blkRead

```c
#define blkRead(ip, startblk, buf, n) ((ip)->vmt->read(ip, startblk, buf, n))
```

Reads one or more blocks.

**Parameters**

| in | ip | pointer to a `BaseBlockDevice` or derived class |
| in | startblk | first block to read |
| out | buf | pointer to the read buffer |
| in | n | number of blocks to read |
Returns

The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.26.3.10 blkWrite

```c
#define blkWrite(ip, startblk, buf, n) ((ip)->vmt->write(ip, startblk, buf, n))
```

Writes one or more blocks.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
</tr>
<tr>
<td>in</td>
<td>startblk</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.26.3.11 blkSync

```c
#define blkSync(ip) ((ip)->vmt->sync(ip))
```

Ensures write synchronization.

**Parameters**

- **in** `ip` pointer to a `BaseBlockDevice` or derived class

**Returns**

The operation status.

**Return values**

- `HAL_SUCCESS` operation succeeded.
- `HAL_FAILED` operation failed.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.26.3.12 blkGetInfo

```c
#define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))
```

Returns a media information structure.

**Parameters**

- **in** `ip` pointer to a `BaseBlockDevice` or derived class
- **out** `bdip` pointer to a `BlockDeviceInfo` structure

**Returns**

The operation status.

**Return values**

- `HAL_SUCCESS` operation succeeded.
- `HAL_FAILED` operation failed.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.26.4 Enumeration Type Documentation

7.26.4.1 blkstate_t

```c
enum blkstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLK_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>BLK_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>BLK_ACTIVE</td>
<td>Interface active.</td>
</tr>
<tr>
<td>BLK_CONNECTING</td>
<td>Connection in progress.</td>
</tr>
<tr>
<td>BLK_DISCONNECTING</td>
<td>Disconnection in progress.</td>
</tr>
<tr>
<td>BLK_READY</td>
<td>Device ready.</td>
</tr>
<tr>
<td>BLK_READING</td>
<td>Read operation in progress.</td>
</tr>
<tr>
<td>BLK_WRITING</td>
<td>Write operation in progress.</td>
</tr>
<tr>
<td>BLK_SYNCING</td>
<td>Sync. operation in progress.</td>
</tr>
</tbody>
</table>
```

ChibiOS/HAL
7.27 MAC Driver

Generic MAC Driver.

7.27.1 Detailed Description

Generic MAC Driver.

This module implements a generic MAC (Media Access Control) driver for Ethernet controllers.

Precondition

In order to use the MAC driver the HAL_USE_MAC option must be enabled in halconf.h.

Macros

- `#define MAC_SUPPORTS_ZERO_COPY TRUE`
  
  This implementation supports the zero-copy mode API.

- `#define mac_lld_driver_fields`
  
  Low level fields of the MAC driver structure.

- `#define mac_lld_config_fields`
  
  Low level fields of the MAC configuration structure.

- `#define mac_lld_transmit_descriptor_fields`
  
  Low level fields of the MAC transmit descriptor structure.

- `#define mac_lld_receive_descriptor_fields`
  
  Low level fields of the MAC receive descriptor structure.

MAC event flags

- `#define MAC_FLAGS_TX (1U << 0)`
- `#define MAC_FLAGS_RX (1U << 1)`

MAC configuration options

- `#define MAC_USE_ZERO_COPY FALSE`
  
  Enables an event sources for incoming packets.

- `#define MAC_USE_EVENTS TRUE`
  
  Enables an event sources for incoming packets.

Low level driver helper macros

- `#define __mac_callback(macp)`
  
  MAC callback.

- `#define __mac_tx_wakeup(macp)`
  
  MAC TX wakeup and event.

- `#define __mac_rx_wakeup(macp)`
  
  MAC RX wakeup and event.
Macro Functions

- \#define macSetCallbackX(macp, f) (siop) -> cb = f
  Associates a callback to the MAC instance.
- \#define macGetEventSource(macp) (&(macp) -> es)
  Returns the driver events source.
- \#define macGetTransmitDescriptorX(macp, tdp) mac_lld_get_transmit_descriptor(macp, tdp)
  Returns a transmission descriptor.
- \#define macReleaseTransmitDescriptorX(tdp) mac_lld_release_transmit_descriptor(tdp)
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- \#define macGetReceiveDescriptorX(macp, rdp)
  Returns a receive descriptor.
- \#define macReleaseReceiveDescriptorX(rdp) mac_lld_release_receive_descriptor(rdp)
  Releases a receive descriptor.
- \#define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)
  Writes to a transmit descriptor's stream.
- \#define macReadReceiveDescriptor(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)
  Reads from a receive descriptor's stream.
- \#define macGetNextTransmitBuffer(tdp, size, sizep)
  Returns a pointer to the next transmit buffer in the descriptor chain.
- \#define macGetNextReceiveBuffer(rdp, sizep)
  Returns a pointer to the next receive buffer in the descriptor chain.

PLATFORM configuration options

- \#define PLATFORM_MAC_USE_MAC1 FALSE
  MAC driver enable switch.

Typedefs

- typedef struct hal_mac_driver MACDriver
  Type of a structure representing a MAC driver.
- typedef struct hal_mac_config MACConfig
  Type of structure representing a MAC configuration.
- typedef struct hal_mac_transmit_descriptor MACTransmitDescriptor
  Type of structure representing a MAC transmit descriptor.
- typedef struct hal_mac_receive_descriptor MACReceiveDescriptor
  Type of structure representing a MAC receive descriptor.
- typedef void (maccb_t) (MACDriver *macp)
  Generic ETH notification callback type.

Data Structures

- struct hal_mac_config
  Driver configuration structure.
- struct hal_mac_driver
  Structure representing a MAC driver.
- struct hal_mac_transmit_descriptor
  Structure representing a MAC transmit descriptor.
- struct hal_mac_receive_descriptor
  Structure representing a MAC receive descriptor.
Functions

- **void** macInit (void)
  
  MAC Driver initialization.

- **void** macObjectInit (MACDriver *macp)
  
  Initialize the standard part of a MACDriver structure.

- **msg_t** macStart (MACDriver *macp, const MACConfig *config)
  
  Configures and activates the MAC peripheral.

- **void** macStop (MACDriver *macp)
  
  Deactivates the MAC peripheral.

- **eventflags_t** macGetAndClearEventsI (MACDriver *macp)
  
  Get and clears SIO event flags.

- **msg_t** macWaitTransmitDescriptor (MACDriver *macp, MACTransmitDescriptor *tdp, sysinterval_t timeout)
  
  Allocates a transmission descriptor.

- **msg_t** macWaitReceiveDescriptor (MACDriver *macp, MACReceiveDescriptor *rdp, sysinterval_t timeout)
  
  Waits for a received frame.

- **bool** macPollLinkStatus (MACDriver *macp)
  
  Updates and returns the link status.

- **void** mac_lld_init (void)
  
  Low level MAC initialization.

- **void** mac_lld_start (MACDriver *macp)
  
  Configures and activates the MAC peripheral.

- **void** mac_lld_stop (MACDriver *macp)
  
  Deactivates the MAC peripheral.

- **msg_t** mac_lld_get_transmit_descriptor (MACDriver *macp, MACTransmitDescriptor *tdp)
  
  Returns a transmission descriptor.

- **void** mac_lld_release_transmit_descriptor (MACTransmitDescriptor *tdp)
  
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

- **msg_t** mac_lld_get_receive_descriptor (MACDriver *macp, MACReceiveDescriptor *rdp)
  
  Returns a receive descriptor.

- **void** mac_lld_release_receive_descriptor (MACReceiveDescriptor *rdp)
  
  Releases a receive descriptor.

- **bool** mac_lld_poll_link_status (MACDriver *macp)
  
  Updates and returns the link status.

- **size_t** mac_lld_write_transmit_descriptor (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)
  
  Writes to a transmit descriptor's stream.

- **size_t** mac_lld_read_receive_descriptor (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)
  
  Reads from a receive descriptor's stream.

- **uint8_t** *mac_lld_get_next_transmit_buffer (MACTransmitDescriptor *tdp, size_t size_t, size_t *sizep)
  
  Returns a pointer to the next transmit buffer in the descriptor chain.

- **const uint8_t** *mac_lld_get_next_receive_buffer (MACReceiveDescriptor *rdp, size_t *sizep)
  
  Returns a pointer to the next receive buffer in the descriptor chain.

Enumerations

- **enum** macstate_t { MAC_UNINIT = 0, MAC_STOP = 1, MAC_ACTIVE = 2 }
  
  Driver state machine possible states.
Variables

- **MACDriver ETHD1**
  
  MAC1 driver identifier.

### 7.27.2 Macro Definition Documentation

#### 7.27.2.1 MAC_USE_ZERO_COPY

```c
#define MAC_USE_ZERO_COPY FALSE
```

Enables an event sources for incoming packets.

#### 7.27.2.2 MAC_USE_EVENTS

```c
#define MAC_USE_EVENTS TRUE
```

Enables an event sources for incoming packets.

#### 7.27.2.3 __mac_callback

```c
#define __mac_callback(
    macp )

Value:

```c
do { 
    if ((macp)->cb != NULL) { 
        (macp)->cb(macp);
    } 
} while (false)
```

MAC callback.

**Parameters**

| in  | macp | pointer to the MACDriver object |

**Function Class:**

Not an API, this function is for internal use only.
7.27.2.4 __mac_tx_wakeup

#define __mac_tx_wakeup(
    macp )

Value:
    do {
        osalSysLockFromISR();
        osalThreadDequeueAllI(&(macp)->tdqueue, MSG_OK);
        (macp)->flags |= MAC_FLAGS_TX;
        osalEventBroadcastFlagsI(&(macp)->es, MAC_FLAGS_TX);
        osalSysUnlockFromISR();
    } while (false)

MAC TX wakeup and event.

Parameters
    in  macp  pointer to the MACDriver object

Function Class:
    Not an API, this function is for internal use only.

7.27.2.5 __mac_rx_wakeup

#define __mac_rx_wakeup(
    macp )

Value:
    do {
        osalSysLockFromISR();
        osalThreadDequeueAllI(&(macp)->rdqueue, MSG_OK);
        (macp)->flags |= MAC_FLAGS_RX;
        osalEventBroadcastFlagsI(&(macp)->es, MAC_FLAGS_RX);
        osalSysUnlockFromISR();
    } while (false)

MAC RX wakeup and event.

Parameters
    in  macp  pointer to the MACDriver object

Function Class:
    Not an API, this function is for internal use only.

7.27.2.6 macSetCallbackX

#define macSetCallbackX(
    macp,  
    f ) (siop)>
    cb = (f)

ChibiOS/HAL
Associates a callback to the MAC instance.

Parameters

| in | macp | pointer to the MACDriver object |
| in | f    | callback to be associated       |

### 7.27.2.7 macGetEventSource

```c
#define macGetEventSource( 
    macp ) (&(macp)->es)
```

Returns the driver events source.

Parameters

| in | macp | pointer to the MACDriver object |

Returns

*The pointer to the EventSource structure.*

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.27.2.8 macGetTransmitDescriptorX

```c
#define macGetTransmitDescriptorX( 
    macp, 
    tdp ) mac_lld_get_transmit_descriptor(macp, tdp)
```

Returns a transmission descriptor.

One of the available transmission descriptors is locked and returned.

Parameters

| in  | macp     | pointer to the MACDriver object |
| out | tdp      | pointer to a MACTransmitDescriptor structure |

Returns

*The operation status.*
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>the descriptor has been obtained.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>descriptor not available.</td>
</tr>
</tbody>
</table>

Function Class:
This is an **X-Class** API, this function can be invoked from any context.

7.27.2.9 *macReleaseTransmitDescriptorX*

```c
#define macReleaseTransmitDescriptorX(tdp) mac_lld_release_transmit_descriptor(tdp)
```

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

Parameters

- **in tdp** the pointer to the `MACTransmitDescriptor` structure

Function Class:
Not an API, this function is for internal use only.

7.27.2.10 *macGetReceiveDescriptorX*

```c
#define macGetReceiveDescriptorX(macp, rdp) mac_lld_get_receive_descriptor(macp, rdp)
```

Returns a receive descriptor.

Parameters

- **in macp** pointer to the `MACDriver` object
- **out rdp** pointer to a `MACReceiveDescriptor` structure

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>the descriptor has been obtained.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>descriptor not available.</td>
</tr>
</tbody>
</table>
7.27 MAC Driver

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.27.2.11 macReleaseReceiveDescriptorX

#define macReleaseReceiveDescriptorX(
    rdp ) mac_lld_release_receive_descriptor(rdp)

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

Parameters

| in  | rdp    | the pointer to the MACReceiveDescriptor structure |

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.27.2.12 macWriteTransmitDescriptor

#define macWriteTransmitDescriptor(
    tdp, 
    buf, 
    size ) mac_lld_write_transmit_descriptor(tdp, buf, size)

Writes to a transmit descriptor’s stream.

Parameters

| in  | tdp    | pointer to a MACTransmitDescriptor structure |
| in  | buf    | pointer to the buffer containing the data to be written |
| in  | size   | number of bytes to be written |

Returns

The number of bytes written into the descriptor’s stream, this value can be less than the amount specified in the parameter size if the maximum frame size is reached.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.27.2.13 macReadReceiveDescriptor

```c
#define macReadReceiveDescriptor(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)
```

Reads from a receive descriptor's stream.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>rdp</code> pointer to a MACReceiveDescriptor structure</td>
</tr>
<tr>
<td>in</td>
<td><code>buf</code> pointer to the buffer that will receive the read data</td>
</tr>
<tr>
<td>in</td>
<td><code>size</code> number of bytes to be read</td>
</tr>
</tbody>
</table>

**Returns**

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter `size` if there are no more bytes to read.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.27.2.14 macGetNextTransmitBuffer

```c
#define macGetNextTransmitBuffer(tdp, size, sizep) mac_lld_get_next_transmit_buffer(tdp, size, sizep)
```

Returns a pointer to the next transmit buffer in the descriptor chain.

**Note**

The API guarantees that enough buffers can be requested to fill a whole frame.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>tdp</code> pointer to a MACTransmitDescriptor structure</td>
</tr>
<tr>
<td>in</td>
<td><code>size</code> size of the requested buffer. Specify the frame size on the first call then scale the value down subtracting the amount of data already copied into the previous buffers.</td>
</tr>
<tr>
<td>out</td>
<td><code>sizep</code> pointer to variable receiving the real buffer size. The returned value can be less than the amount requested, this means that more buffers must be requested in order to fill the frame data entirely.</td>
</tr>
</tbody>
</table>
7.27 MAC Driver

Returns

Pointer to the returned buffer.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.27.2.15 macGetNextReceiveBuffer

#define macGetNextReceiveBuffer(
    rdp,
    sizep
) mac_lld_get_next_receive_buffer(rdp, sizep)

Returns a pointer to the next receive buffer in the descriptor chain.

Note

The API guarantees that the descriptor chain contains a whole frame.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>rdp</th>
<th>pointer to a MACReceiveDescriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sizep</td>
<td>pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.</td>
</tr>
</tbody>
</table>

Returns

Pointer to the returned buffer.

Return values

| NULL | if the buffer chain has been entirely scanned. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.27.2.16 MAC_SUPPORTS_ZERO_COPY

#define MAC_SUPPORTS_ZERO_COPY TRUE

This implementation supports the zero-copy mode API.
7.27.2.17  PLATFORM_MAC_USE_MAC1

#define PLATFORM_MAC_USE_MAC1 FALSE

MAC driver enable switch.

If set to **TRUE** the support for MAC1 is included.

**Note**

The default is **FALSE**.

7.27.2.18  mac_lld_driver_fields

#define mac_lld_driver_fields

Low level fields of the MAC driver structure.

7.27.2.19  mac_lld_config_fields

#define mac_lld_config_fields

Low level fields of the MAC configuration structure.

7.27.2.20  mac_lld_transmit_descriptor_fields

#define mac_lld_transmit_descriptor_fields

Low level fields of the MAC transmit descriptor structure.

7.27.2.21  mac_lld_receive_descriptor_fields

#define mac_lld_receive_descriptor_fields

Low level fields of the MAC receive descriptor structure.
7.27 MAC Driver

7.27.3.1 MACDriver

typedef struct hal_mac_driver MACDriver

Type of a structure representing a MAC driver.

7.27.3.2 MACConfig

typedef struct hal_mac_config MACConfig

Type of structure representing a MAC configuration.

7.27.3.3 MACTransmitDescriptor

typedef struct hal_mac_transmit_descriptor MACTransmitDescriptor

Type of structure representing a MAC transmit descriptor.

7.27.3.4 MACReceiveDescriptor

typedef struct hal_mac_receive_descriptor MACReceiveDescriptor

Type of structure representing a MAC receive descriptor.

7.27.3.5 maccb_t

typedef void(* maccb_t) (MACDriver *macp)

Generic ETH notification callback type.

Parameters
- `ethp` pointer to the ETHDriver object

7.27.4 Enumeration Type Documentation
7.27.4.1 macstate_t

enum macstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>MAC_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>MAC_ACTIVE</td>
<td>Active.</td>
</tr>
</tbody>
</table>

7.27.5 Function Documentation

7.27.5.1 macInit()

```c
void macInit (
    void )
```

MAC Driver initialization.

**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.27.5.2 macObjectInit()

```c
void macObjectInit (  
    MACDriver * macp  )
```

Initialize the standard part of a MACDriver structure.
Parameters

| out | macp | pointer to the MACDriver object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call graph for macObjectInit function]

7.27.5.3 macStart()

```c
msg_t macStart (
    MACDriver * macp,
    const MACConfig * config )
```

Configures and activates the MAC peripheral.

Parameters

| in | macp | pointer to the MACDriver object |
| in | config | pointer to the MACConfig object |

Returns

The operation status.
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
macStart
\rightarrow mac_lld_start
\rightarrow osalSysLock
```

### 7.27.5.4 macStop()

```c
void macStop (MACDriver * macp)
```

Deactivates the MAC peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>macp</code></td>
<td>pointer to the MACDriver object</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
macStop
\rightarrow mac_lld_stop
\rightarrow osalSysLock
\rightarrow osalSysUnlock
```
7.27.5.5  macGetAndClearEventsI()

```c
eventflags_t macGetAndClearEventsI ( 
    MACDriver * macp
)
```

Get and clears SIO event flags.

**Parameters**

| in  | macp  | pointer to the MACDriver object |

**Returns**

The pending event flags.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.27.5.6  macWaitTransmitDescriptor()

```c
msg_t macWaitTransmitDescriptor ( 
    MACDriver * macp, 
    MACTransmitDescriptor * tdp, 
    sysinterval_t timeout
)
```

Allocates a transmission descriptor.

One of the available transmission descriptors is locked and returned. If a descriptor is not currently available then the invoking thread is queued until one is freed.

**Parameters**

| in  | macp  | pointer to the MACDriver object |
| out | tdp   | pointer to a MACTransmitDescriptor structure |
| in  | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|     |        | • TIME_IMMEDIATE immediate timeout. |
|     |        | • TIME_INFINITE no timeout. |

**Returns**

The operation status.
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>the descriptor was obtained.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>the operation timed out, descriptor not initialized.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
macWaitTransmitDescriptor  
                         osalSysLock
```

7.27.5.7  macWaitReceiveDescriptor()

```
msg_t macWaitReceiveDescriptor (  
    MACDriver * macp,  
    MACReceiveDescriptor * rdp,  
    sysinterval_t timeout )
```

Waits for a received frame.

 stops until a frame is received and buffered. If a frame is not immediately available then the invoking thread is queued until one is received.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>macp</th>
<th>pointer to the MACDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>rdp</td>
<td>pointer to a MACReceiveDescriptor structure</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

The operation status.
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>the descriptor was obtained.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>the operation timed out, descriptor not initialized.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
macWaitReceiveDescriptor    osalSysLock
```

7.27.5.8 `macPollLinkStatus()`

```c
bool macPollLinkStatus (MACDriver * macp)
```

Updates and returns the link status.

Parameters

| in macp | pointer to the MACDriver object |

Returns

The link status.

Return values

| true | if the link is active. |
| false | if the link is down. |
Function Class:
   Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
macPollLinkStatus -> mac_lld_poll_link_status
```

7.27.5.9  mac_lld_init()

```c
void mac_lld_init (
    void
)
```

Low level MAC initialization.

Function Class:
   Not an API, this function is for internal use only.

Here is the call graph for this function:

```
mac_lld_init -> macObjectInit
    |            | osalEventObjectInit
    |            | osalThreadQueueObjectInit
```

7.27.5.10  mac_lld_start()

```c
void mac_lld_start (  
    MACDriver * macp )
```

Configures and activates the MAC peripheral.
7.27 MAC Driver

Parameters

| in | macp | pointer to the MACDriver object |

Function Class:

Not an API, this function is for internal use only.

7.27.5.11 mac_lld_stop()

void mac_lld_stop (  
    MACDriver ∗ macp )

Deactivates the MAC peripheral.

Parameters

| in | macp | pointer to the MACDriver object |

Function Class:

Not an API, this function is for internal use only.

7.27.5.12 mac_lld_get_transmit_descriptor()

msg_t mac_lld_get_transmit_descriptor (  
    MACDriver ∗ macp,  
    MACTransmitDescriptor ∗ tdp )

Returns a transmission descriptor.

One of the available transmission descriptors is locked and returned.

Parameters

| in | macp | pointer to the MACDriver object |
| out | tdp | pointer to a MACTransmitDescriptor structure |

Returns

The operation status.
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>the descriptor has been obtained.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>descriptor not available.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.27.5.13  mac_lld_release_transmit_descriptor()

```c
void mac_lld_release_transmit_descriptor ( 
    MACTransmitDescriptor * tdp )
```

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

Parameters

| in tdp the pointer to the MACTransmitDescriptor structure |

Function Class:

Not an API, this function is for internal use only.

### 7.27.5.14  mac_lld_get_receive_descriptor()

```c
msg_t mac_lld_get_receive_descriptor ( 
    MACDriver * macp, 
    MACReceiveDescriptor * rdp )
```

Returns a receive descriptor.

Parameters

| in macp pointer to the MACDriver object |
| out rdp pointer to a MACReceiveDescriptor structure |

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>the descriptor has been obtained.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>descriptor not available.</td>
</tr>
</tbody>
</table>
7.27 MAC Driver

Function Class:
Not an API, this function is for internal use only.

7.27.5.15  mac_lld_release_receive_descriptor()

```c
void mac_lld_release_receive_descriptor (MACReceiveDescriptor * rdp )
```

Releases a receive descriptor.
The descriptor and its buffer are made available for more incoming frames.

Parameters

| **in**  | **rdp** | the pointer to the MACReceiveDescriptor structure |

Function Class:
Not an API, this function is for internal use only.

7.27.5.16  mac_lld_poll_link_status()

```c
bool mac_lld_poll_link_status (MACDriver * macp )
```

Updates and returns the link status.

Parameters

| **in**  | **macp** | pointer to the MACDriver object |

Returns
The link status.

Return values

| **true** | if the link is active. |
| **false** | if the link is down. |

Function Class:
Not an API, this function is for internal use only.
7.27.5.17  mac_lld_write_transmit_descriptor()

size_t mac_lld_write_transmit_descriptor (  
    MACTransmitDescriptor  ∗ tdp,  
    uint8_t  ∗ buf,  
    size_t  size  )

Writes to a transmit descriptor’s stream.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tdp</td>
</tr>
<tr>
<td>in</td>
<td>buf</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
</tr>
</tbody>
</table>

Returns

The number of bytes written into the descriptor’s stream, this value can be less than the amount specified in the parameter size if the maximum frame size is reached.

Function Class:

Not an API, this function is for internal use only.

7.27.5.18  mac_lld_read_receive_descriptor()

size_t mac_lld_read_receive_descriptor (  
    MACReceiveDescriptor  ∗ rdp,  
    uint8_t  ∗ buf,  
    size_t  size  )

Reads from a receive descriptor’s stream.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rdp</td>
</tr>
<tr>
<td>in</td>
<td>buf</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
</tr>
</tbody>
</table>

Returns

The number of bytes read from the descriptor’s stream, this value can be less than the amount specified in the parameter size if there are no more bytes to read.

Function Class:

Not an API, this function is for internal use only.
7.27.5.19  

mac_lld_get_next_transmit_buffer()

```c
uint8_t * mac_lld_get_next_transmit_buffer ( 
    MACTransmitDescriptor * tdp,
    size_t size,
    size_t * sizep 
)
```

Returns a pointer to the next transmit buffer in the descriptor chain.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tdp</code></td>
<td>pointer to a MACTransmitDescriptor structure</td>
</tr>
<tr>
<td><code>size</code></td>
<td>size of the requested buffer. Specify the frame size on the first call then scale the value down subtracting the amount of data already copied into the previous buffers.</td>
</tr>
<tr>
<td><code>sizep</code></td>
<td>pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned. Note that a returned size lower than the amount requested means that more buffers must be requested in order to fill the frame data entirely.</td>
</tr>
</tbody>
</table>

**Returns**

Pointer to the returned buffer.

**Return values**

- `NULL` if the buffer chain has been entirely scanned.

**Function Class:**

Not an API, this function is for internal use only.

7.27.5.20  

mac_lld_get_next_receive_buffer()

```c
const uint8_t * mac_lld_get_next_receive_buffer ( 
    MACReceiveDescriptor * rdp,
    size_t * sizep 
)
```

Returns a pointer to the next receive buffer in the descriptor chain.

**Note**

The API guarantees that the descriptor chain contains a whole frame.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>rdp</th>
<th>pointer to a MACReceiveDescriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sizep</td>
<td>pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.</td>
</tr>
</tbody>
</table>

Returns

Pointer to the returned buffer.

Return values

| NULL | if the buffer chain has been entirely scanned. |

Function Class:

Not an API, this function is for internal use only.

7.27.6 Variable Documentation

7.27.6.1 ETHD1

**MACDriver ETHD1**

MAC1 driver identifier.
7.28 Managed Flash Storage Driver

Managed Flash Storage Driver.

7.28.1 Detailed Description

Managed Flash Storage Driver.

This module implements a managed flash storage able to store a finite number of variable-size records. Records are retrieved by their index number. The driver is automatically performs:

- Wear leveling.
- Auto repair after power loss.
- Garbage collection in order to remove erased data.

Macros

- `#define ALIGNED_REC_SIZE(n) (flash_offset_t)MFS_ALIGN_NEXT(sizeof (mfs_data_header_t) + (size_t)(n))`
  Data record size aligned.
- `#define ALIGNED_DHDR_SIZE ALIGNED_REC_SIZE(0)`
  Data record header size aligned.
- `#define ALIGNED_SIZEOF(t) (((sizeof (t) - 1U) | MFS_ALIGN_MASK) + 1U)`
  Aligned size of a type.
- `#define PAIR(a, b) (((unsigned)(a)<<2U) | (unsigned)(b))`
  Combines two values (0..3) in one (0..15).
- `#define RET_ON_ERROR(err)`
  Error check helper.

Configuration options

- `#define MFS_CFG_MAX_RECORDS 32`
  Maximum number of indexed records in the managed storage.
- `#define MFS_CFG_MAX_REPAIR_ATTEMPTS 3`
  Maximum number of repair attempts on partition mount.
- `#define MFS_CFG_WRITE_VERIFY TRUE`
  Verify written data.
- `#define MFS_CFG_STRONG_CHECKING TRUE`
  Enables a stronger and slower check procedure on mount.
- `#define MFS_CFG_BUFFER_SIZE 32`
  Size of the buffer used for data copying.
- `#define MFS_CFG_MEMORY_ALIGNMENT 2`
  Enforced memory alignment.
- `#define MFS_CFG_TRANSACTION_MAX 16`
  Maximum number of objects writable in a single transaction.
Error codes handling macros

- `#define MFS_IS_ERROR(err) ((err) < MFS_NO_ERROR)`
- `#define MFS_IS_WARNING(err) ((err) > MFS_NO_ERROR)`

Alignment macros

- `#define MFS_ALIGN_MASK ((uint32_t)MFS_CFG_MEMORY_ALIGNMENT - 1U)`
- `#define MFS_IS_ALIGNED(v) (((uint32_t)(v) & MFS_ALIGN_MASK) == 0U)`
- `#define MFS_ALIGN_PREV(v) ((uint32_t)(v) & ~MFS_ALIGN_MASK)`
- `#define MFS_ALIGN_NEXT(v)`

Typedefs

- `typedef uint32_t mfs_id_t
  Type of a record identifier.
- typedef union mfs_nocache_buffer mfs_nocache_buffer_t
  Type of a non-cacheable MFS buffer.`

Data Structures

- `union mfs_bank_header_t
  Type of a bank header.
- union mfs_data_header_t
  Type of a data block header.
- struct MFSConfig
  Type of a MFS configuration structure.
- struct mfs_transaction_op_t
  Type of a buffered write/erase operation within a transaction.
- union mfs_nocache_buffer
  Type of a non-cacheable MFS buffer.
- struct MFSDriver
  Type of an MFS instance.`

Functions

- `static mfs_error_t mfs_flash_read (MFSDriver *mfsp, flash_offset_t offset, size_t n, uint8_t *rp)`
  Flash read.
- `static mfs_error_t mfs_flash_write (MFSDriver *mfsp, flash_offset_t offset, size_t n, const uint8_t *wp)`
  Flash write.
- `static mfs_error_t mfs_flash_copy (MFSDriver *mfsp, flash_offset_t doffset, flash_offset_t soffset, uint32_t n)`
  Flash copy.
- `static mfs_error_t mfs_bank_erase (MFSDriver *mfsp, mfs_bank_t bank)`
  Erases and verifies all sectors belonging to a bank.
- `static mfs_error_t mfs_bank_verify_erase (MFSDriver *mfsp, mfs_bank_t bank)`
  Erases and verifies all sectors belonging to a bank.
- `static mfs_error_t mfs_bank_write_header (MFSDriver *mfsp, mfs_bank_t bank, uint32_t cnt)`
  Writes the validation header in a bank.
- `static mfs_bank_state_t mfs_bank_check_header (MFSDriver *mfsp)`
7.28 Managed Flash Storage Driver

Checks integrity of the header in the shared buffer.

- static mfs_error_t mfs_bank_scan_records (MFSDriver ∗mfsp, mfs_bank_t bank, bool ∗wflagp)
  Scans blocks searching for records.

- static mfs_error_t mfs_bank_get_state (MFSDriver ∗mfsp, mfs_bank_t bank, mfs_bank_state_t ∗statep, uint32_t ∗cntp)
  Determines the state of a bank.

- static mfs_error_t mfs_garbage_collect (MFSDriver ∗mfsp)
  Enforces a garbage collection.

- static mfs_error_t mfs_try_mount (MFSDriver ∗mfsp)
  Performs a flash partition mount attempt.

- mfs_error_t mfs_mount (MFSDriver ∗mfsp)
  Configures and activates a MFS driver.

- mfs_error_t mfsErase (MFSDriver ∗mfsp)
  Destroys the state of the managed storage by erasing the flash.

- mfs_error_t mfsReadRecord (MFSDriver ∗mfsp, mfs_id_t id, size_t ∗np, uint8_t ∗buffer)
  Retrieves and reads a data record.

- mfs_error_t mfsWriteRecord (MFSDriver ∗mfsp, mfs_id_t id, size_t n, const uint8_t ∗buffer)
  Creates or updates a data record.

- mfs_error_t mfsEraseRecord (MFSDriver ∗mfsp, mfs_id_t id)
  Erases a data record.

- mfs_error_t mfsPerformGarbageCollection (MFSDriver ∗mfsp)
  Enforces a garbage collection operation.

- mfs_error_t mfsStartTransaction (MFSDriver ∗mfsp, size_t size)
  Puts the driver in transaction mode.

- mfs_error_t mfsCommitTransaction (MFSDriver ∗mfsp)
  A transaction is committed and finalized atomically.

- mfs_error_t mfsRollbackTransaction (MFSDriver ∗mfsp)
  A transaction is rolled back atomically.

Enumerations

- enum mfs_bank_t
  Type of a flash bank.

- enum mfs_state_t
  Type of driver state machine states.

- enum mfs_error_t
  Type of an MFS error code.

- enum mfs_bank_state_t
  Type of a bank state assessment.

7.28.2 Macro Definition Documentation
7.28.2.1 ALIGNED_REC_SIZE

#define ALIGNED_REC_SIZE(
            n) (flash_offset_t)MFS_ALIGN_NEXT(sizeof (mfs_data_header_t) + (size_t)(n))

Data record size aligned.

7.28.2.2 ALIGNED_DHDR_SIZE

#define ALIGNED_DHDR_SIZE ALIGNED_REC_SIZE(0)

Data record header size aligned.

7.28.2.3 ALIGNED_SIZEOF

#define ALIGNED_SIZEOF(
            t) (((sizeof (t) - 1U) | MFS_ALIGN_MASK) + 1U)

Aligned size of a type.

7.28.2.4 PAIR

#define PAIR(
            a,
            b) (((unsigned)(a) << 2U) | (unsigned)(b))

Combines two values (0..3) in one (0..15).

7.28.2.5 RET_ON_ERROR

#define RET_ON_ERROR(
            err)

Value:

   do { 
       mfs_error_t e = (err);
       if (e != MFS_NO_ERROR) { 
           return e;
       }
   } while (false)

Error check helper.
7.28.2.6 MFS_CFG_MAX_RECORDS

#define MFS_CFG_MAX_RECORDS 32

Maximum number of indexed records in the managed storage.

Note

Record indexes go from 1 to MFS_CFG_MAX_RECORDS.

7.28.2.7 MFS_CFG_MAX_REPAIR_ATTEMPTS

#define MFS_CFG_MAX_REPAIR_ATTEMPTS 3

Maximum number of repair attempts on partition mount.

7.28.2.8 MFS_CFG_WRITE_VERIFY

#define MFS_CFG_WRITE_VERIFY TRUE

Verify written data.

7.28.2.9 MFS_CFG_STRONG_CHECKING

#define MFS_CFG_STRONG_CHECKING TRUE

Enables a stronger and slower check procedure on mount.

Strong checking requires reading of the whole written data and this can be slow, normal checking only checks integrity of metadata, data errors would be detected on read.

7.28.2.10 MFS_CFG_BUFFER_SIZE

#define MFS_CFG_BUFFER_SIZE 32

Size of the buffer used for data copying.

Note

The buffer size must be a power of two and not smaller than 16 bytes.
Larger buffers improve performance, buffers with size multiple of the flash program page size work better.
7.28.2.11 MFS_CFG_MEMORY_ALIGNMENT

#define MFS_CFG_MEMORY_ALIGNMENT 2

Enforced memory alignment.

This value must be a power of two, it enforces a memory alignment for records in the flash array. This is required when alignment constraints exist, for example when using a DTR mode on OSPI devices.

7.28.2.12 MFS_CFG_TRANSACTION_MAX

#define MFS_CFG_TRANSACTION_MAX 16

Maximum number of objects writable in a single transaction.

7.28.3 Typedef Documentation

7.28.3.1 mfs_id_t

typedef uint32_t mfs_id_t

Type of a record identifier.

7.28.3.2 mfs_nocache_buffer_t

typedef union mfs_nocache_buffer mfs_nocache_buffer_t

Type of an non-cacheable MFS buffer.

7.28.4 Enumeration Type Documentation

7.28.4.1 mfs_bank_t

enum mfs_bank_t

Type of a flash bank.
7.28.4.2  

enum mfs_state_t

Type of driver state machine states.

7.28.4.3  

enum mfs_error_t

Type of an MFS error code.

Note

Errors are negative integers, informative warnings are positive integers.

7.28.4.4  

enum mfs_bank_state_t

Type of a bank state assessment.

7.28.5  

Function Documentation

7.28.5.1  

mfs_flash_read()

static mfs_error_t mfs_flash_read ( 
    MFSDriver * mfsp,
    flash_offset_t offset,
    size_t n,
    uint8_t * rp ) [static]

Flash read.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mfsp</td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td>in offset</td>
<td>flash offset</td>
</tr>
<tr>
<td>in n</td>
<td>number of bytes to be read</td>
</tr>
<tr>
<td>out rp</td>
<td>pointer to the data buffer</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.

### 7.28.5.2 mfs_flash_write()

```c
static mfs_error_t mfs_flash_write ( 
  MFSDriver ∗ mfsp, 
  flash_offset_t offset, 
  size_t n, 
  const uint8_t ∗ wp ) [static]
```

Flash write.

Note

If the option MFS_CFG_WRITE_VERIFY is enabled then the flash is also read back for verification.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mfsp</td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td>in offset</td>
<td>flash offset</td>
</tr>
<tr>
<td>in n</td>
<td>number of bytes to be written</td>
</tr>
<tr>
<td>in wp</td>
<td>pointer to the data buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.

### 7.28.5.3 mfs_flash_copy()

```c
static mfs_error_t mfs_flash_copy ( 
  MFSDriver ∗ mfsp, 
  flash_offset_t doffset, 
  flash_offset_t soffset, 
  uint32_t n ) [static]
```

Flash copy.

Note

If the option MFS_CFG_WRITE_VERIFY is enabled then the flash is also read back for verification.
Parameters

| in  | mfsp       | pointer to the MFSDriver object |
| in  | offset     | destination flash offset       |
| in  | soffset    | source flash offset            |
| in  | n          | number of bytes to be copied   |

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
    mfs_flash_copy
       |
       |       mfs_flash_read
       |       
       |       mfs_flash_write
```

7.28.5.4 mfs_bank erase()

```c
static mfs_error_t mfs_bank_erase (  
    MFSDriver * mfsp,  
    mfs_bank_t bank ) [static]
```

Erases and verifies all sectors belonging to a bank.

Parameters

| in  | mfsp       | pointer to the MFSDriver object |
| in  | bank       | bank to be erased               |

Returns

The operation status.
7.28.5.5  mfs_bank_verify_erase()

static mfs_error_t mfs_bank_verify_erase (  
    MFSDriver * mfsp,  
    mfs_bank_t bank ) [static]

Erases and verifies all sectors belonging to a bank.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bank</td>
<td>bank to be verified</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.

7.28.5.6  mfs_bank_write_header()

static mfs_error_t mfs_bank_write_header (  
    MFSDriver * mfsp,  
    mfs_bank_t bank,  
    uint32_t cnt ) [static]

Writes the validation header in a bank.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bank</td>
<td>bank to be validated</td>
</tr>
<tr>
<td>in</td>
<td>cnt</td>
<td>value for the flash usage counter</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.
Function Class:
Not an API, this function is for internal use only.

### 7.28.5.7 mfs_bank_check_header()

`static mfs_bank_state_t mfs_bank_check_header (MFSDriver * mfsp)` [static]

Checks integrity of the header in the shared buffer.

**Parameters**

| in  | `mfsp` pointer to the MFSDriver object |

**Returns**

The header state.

Function Class:
Not an API, this function is for internal use only.

### 7.28.5.8 mfs_bank_scan_records()

`static mfs_error_t mfs_bank_scan_records (MFSDriver * mfsp, mfs_bank_t bank, bool * wflagp)` [static]

Scans blocks searching for records.

**Note**

The block integrity is strongly checked.

**Parameters**

| in  | `mfsp` pointer to the MFSDriver object |
| in  | `bank` the bank identifier |
| out | `wflagp` warning flag on anomalies |
Returns
The operation status.

Function Class:
Not an API, this function is for internal use only.

7.28.5.9 mfs_bank_get_state()

```c
static mfs_error_t mfs_bank_get_state ( 
    MFSDriver * mfsp,
    mfs_bank_t bank,
    mfs_bank_state_t * statep,
    uint32_t * cntp ) [static]
```

Determines the state of a bank.

Note
This function does not test the bank integrity by scanning the data area, it just checks the header.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bank</td>
<td>bank to be checked</td>
</tr>
<tr>
<td>out</td>
<td>statep</td>
<td>bank state, it can be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MFS_BANK_ERASED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MFS_BANK_GARBAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MFS_BANK_OK</td>
</tr>
<tr>
<td>out</td>
<td>cntp</td>
<td>bank counter</td>
</tr>
</tbody>
</table>

Returns
The operation status.
Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
mfs_bank_get_state -> mfs_flash_read
```

### 7.28.5.10 mfs_garbage_collect()

```c
static mfs_error_t mfs_garbage_collect (MFSDriver *mfsp) [static]
```

Enforces a garbage collection.

Storage data is compacted into a single bank.

**Parameters**

- `out mfsp` pointer to the MFSDriver object

**Returns**

The operation status.

**Function Class:**
Not an API, this function is for internal use only.

### 7.28.5.11 mfs_try_mount()

```c
static mfs_error_t mfs_try_mount (MFSDriver *mfsp) [static]
```

Performs a flash partition mount attempt.
Parameters

| in  | mfsp | pointer to the MFSDriver object |

Returns

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.12  mfs_mount()

```c
mfs_error_t mfs_mount (MFSDriver * mfsp )
```

Configures and activates a MFS driver.

Parameters

| in  | mfsp | pointer to the MFSDriver object |

Returns

The operation status.

Return values

| MFS_NO_ERROR | if the operation has been successfully completed. |
| MFS_WARN_GC  | if the operation triggered a garbage collection. |
| MFS_ERR_FLASH_FAILURE | if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state. |
| MFS_ERR_INTERNAL | if an internal logic failure is detected. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.13  mfsObjectInit()

```c
void mfsObjectInit (MFSDriver * mfsp, mfs_nocache_buffer_t * ncbuf )
```

Initializes an instance.
7.28 Managed Flash Storage Driver

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ncbuf</td>
<td>pointer to an mfs_nocache_buffer_t buffer structure</td>
</tr>
</tbody>
</table>

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

**7.28.5.14 mfsStart()**

```c
mfs_error_t mfsStart (  
    MFSDriver * mfsp,  
    const MFSConfig * config  
)
```

Configures and activates a MFS driver.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the configuration</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_NO_ERROR</td>
<td>if the operation has been completed.</td>
</tr>
<tr>
<td>MFS_WARN_GC</td>
<td>if the operation triggered a garbage collection.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

**7.28.5.15 mfsStop()**

```c
void mfsStop (  
    MFSDriver * mfsp  
)
```

Deactivates a MFS driver.
7.28.5.16  
mfsErase()

\[ \text{mfs\_error\_t mfsErase (}} \]
\[ \quad \text{MFSDriver * mfsp) } \]

Destroys the state of the managed storage by erasing the flash.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{msp}</td>
<td>pointer to the \text{MFSDriver} object</td>
</tr>
</tbody>
</table>

### Returns

The operation status.

### Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{MFS_ERR_INV_STATE}</td>
<td>if the driver is in not in \text{MFS_READY} state.</td>
</tr>
<tr>
<td>\text{MFS_NO_ERROR}</td>
<td>if the operation has been successfully completed.</td>
</tr>
<tr>
<td>\text{MFS_ERR_FLASH_FAILURE}</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the \text{MFS_ERROR} state.</td>
</tr>
<tr>
<td>\text{MFS_ERR_INTERNAL}</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.17  
mfsReadRecord()

\[ \text{mfs\_error\_t mfsReadRecord (}} \]
\[ \quad \text{MFSDriver * mfsp,} \]
\[ \quad \text{mfs\_id\_t id,} \]
\[ \quad \text{size\_t * np,} \]
\[ \quad \text{uint8\_t * buffer) } \]

Retrieves and reads a data record.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>mfsp</td>
<td>Pointer to the MFSDriver object</td>
</tr>
<tr>
<td>In</td>
<td>id</td>
<td>Record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS</td>
</tr>
<tr>
<td>In, Out</td>
<td>np</td>
<td>On input is the maximum buffer size, on return it is the size of the data copied into the buffer</td>
</tr>
<tr>
<td>Out</td>
<td>buffer</td>
<td>Pointer to a buffer for record data</td>
</tr>
</tbody>
</table>

Returns

- The operation status.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_NO_ERROR</td>
<td>If the operation has been successfully completed.</td>
</tr>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>If the driver is not in MFS_READY state.</td>
</tr>
<tr>
<td>MFS_ERR_INV_SIZE</td>
<td>If the passed buffer is not large enough to contain the record data.</td>
</tr>
<tr>
<td>MFS_ERR_NOT_FOUND</td>
<td>If the specified id does not exist.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>If the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>If an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

- Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.18 mfsWriteRecord()

mfs_error_t mfsWriteRecord (  
  MFSDriver * mfsp,  
  mfs_id_t id,  
  size_t n,  
  const uint8_t * buffer )

Creates or updates a data record.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>mfsp</td>
<td>Pointer to the MFSDriver object</td>
</tr>
<tr>
<td>In</td>
<td>id</td>
<td>Record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS</td>
</tr>
<tr>
<td>In</td>
<td>n</td>
<td>Size of data to be written, it cannot be zero</td>
</tr>
<tr>
<td>In</td>
<td>buffer</td>
<td>Pointer to a buffer for record data</td>
</tr>
</tbody>
</table>

Returns

- The operation status.
Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_NO_ERROR</td>
<td>if the operation has been successfully completed.</td>
</tr>
<tr>
<td>MFS_WARN_GC</td>
<td>if the operation triggered a garbage collection.</td>
</tr>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>if the driver is in not in MFS_READY state.</td>
</tr>
<tr>
<td>MFS_ERR_OUT_OF_MEM</td>
<td>if there is not enough flash space for the operation.</td>
</tr>
<tr>
<td>MFS_ERR_TRANSACTION_NUM</td>
<td>if the transaction operations buffer space has been exceeded.</td>
</tr>
<tr>
<td>MFS_ERR_TRANSACTION_SIZE</td>
<td>if the transaction allocated space has been exceeded.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.19 mfsEraseRecord()

mfs_error_t mfsEraseRecord ( MFSDriver * mfsp, mfs_id_t id )

Erases a data record.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in msp</td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td>in id</td>
<td>record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_NO_ERROR</td>
<td>if the operation has been successfully completed.</td>
</tr>
<tr>
<td>MFS_WARN_GC</td>
<td>if the operation triggered a garbage collection.</td>
</tr>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>if the driver is in not in MFS_READY state.</td>
</tr>
<tr>
<td>MFS_ERR_OUT_OF_MEM</td>
<td>if there is not enough flash space for the operation.</td>
</tr>
<tr>
<td>MFS_ERR_TRANSACTION_NUM</td>
<td>if the transaction operations buffer space has been exceeded.</td>
</tr>
<tr>
<td>MFS_ERR_TRANSACTION_SIZE</td>
<td>if the transaction allocated space has been exceeded.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.20 mfsPerformGarbageCollection()

```c
mfs_error_t mfsPerformGarbageCollection (MFSDriver * mfsp)
```

Enforces a garbage collection operation.

Garbage collection involves: integrity check, optionally repairs, obsolete data removal, data compaction and a flash bank swap.

Parameters

| in | mfsp | pointer to the MFSDriver object |

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MFS_NO_ERROR</th>
<th>if the operation has been successfully completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>if the driver is in not in MFS_READY state.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.21 mfsStartTransaction()

```c
mfs_error_t mfsStartTransaction (MFSDriver * mfsp, size_t size)
```

Puts the driver in transaction mode.
The parameters n and size are used to make an estimation of the space required for the transaction to succeed. Note that the estimated size must include also the extra space required by alignment enforcement option. If the estimated size is wrong (by defect) what could happen is that there is a failure in the middle of a transaction and a roll-back would be required.

The conditions for starting a transaction are:

- The driver must be started.
- There must be enough compacted storage to accommodate the whole transaction. If the required space is available but it is not compacted then a garbage collect operation is performed.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mfsp</td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td>in size</td>
<td>estimated total size of written records in transaction, this includes, data, headers and alignment gaps</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_NO_ERROR</td>
<td>if the operation has been successfully completed.</td>
</tr>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>if the driver is in not in MFS_READY state.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.22 mfsCommitTransaction()

mfs_error_t mfsCommitTransaction ( MFSDriver * mfsp )

A transaction is committed and finalized atomically.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mfsp</td>
<td>pointer to the MFSDriver object</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MFS_NO_ERROR</th>
<th>if the operation has been successfully completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>if the driver is in not in MFS_TRANSACTION state.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.5.23 mfsRollbackTransaction()


\[
\text{mfsError_t mfsRollbackTransaction} (\text{MFSDriver } \ast \text{mfsp})
\]

A transaction is rolled back atomically.

This function performs a garbage collection in order to discard all written data that has not been finalized.

Parameters

- **mfsp**: pointer to the MFSDriver object

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MFS_NO_ERROR</th>
<th>if the operation has been successfully completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>if the driver is in not in MFS_TRANSACTION state.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.29  MII/RMII Header

MII/RMII Support Header.

7.29.1  Detailed Description

MII/RMII Support Header.

This header contains definitions and types related to MII/RMII.

Generic MII registers

- #define MII_BMCR 0x00
- #define MII_BMSR 0x01
- #define MII_PHYSID1 0x02
- #define MII_PHYSID2 0x03
- #define MII_ADVERTISE 0x04
- #define MII_LPA 0x05
- #define MII_EXPANSION 0x06
- #define MII_ANPNT 0x07
- #define MII_CTRL1000 0x09
- #define MII_STAT1000 0x0a
- #define MII_ESTATUS 0x0f
- #define MII_PHYSTS 0x10
- #define MII_MICR 0x11
- #define MII_DCOUNTER 0x12
- #define MII_FCSCOUNTER 0x13
- #define MII_NWAYTEST 0x14
- #define MII_RERRCOUNTER 0x15
- #define MII_SREVISION 0x16
- #define MII_RESV1 0x17
- #define MII_LBRERROR 0x18
- #define MII_PHYADDR 0x19
- #define MII_RESV2 0x1a
- #define MII_TPISTATUS 0x1b
- #define MII_NCONFIG 0x1c

Basic mode control register

- #define BMCR_RESV 0x007f
- #define BMCR_CTST 0x0080
- #define BMCR_FULLDPLX 0x0100
- #define BMCR_ANRESTART 0x0200
- #define BMCR_ISOLATE 0x0400
- #define BMCR_PDWN 0x0800
- #define BMCR_ANENABLE 0x1000
- #define BMCR_SPEED100 0x2000
- #define BMCR_LOOPBACK 0x4000
- #define BMCR_RESET 0x8000
Basic mode status register

- \#define BMSR_ERCAP 0x0001
- \#define BMSR_JCD 0x0002
- \#define BMSR_LSTATUS 0x0004
- \#define BMSR_ANEGCAPABLE 0x0008
- \#define BMSR_RFault 0x0010
- \#define BMSR_ANEGCOMPLETE 0x0020
- \#define BMSR_MFPRESUPPCAP 0x0040
- \#define BMSR_RESV 0x0780
- \#define BMSR_10HALF 0x0800
- \#define BMSR_10FULL 0x1000
- \#define BMSR_100HALF 0x2000
- \#define BMSR_100FULL 0x4000
- \#define BMSR_100BASE4 0x8000

Advertisement control register

- \#define ADVERTISE_SLCT 0x001f
- \#define ADVERTISE_CSMA 0x0001
- \#define ADVERTISE_10HALF 0x0020
- \#define ADVERTISE_10FULL 0x0040
- \#define ADVERTISE_100HALF 0x0080
- \#define ADVERTISE_100FULL 0x0100
- \#define ADVERTISE_100BASE4 0x0200
- \#define ADVERTISE_PAUSE_CAP 0x0400
- \#define ADVERTISE_PAUSE_ASYM 0x0800
- \#define ADVERTISE_RESV 0x1000
- \#define ADVERTISE_RFault 0x2000
- \#define ADVERTISE_LPACK 0x4000
- \#define ADVERTISE_NPAGE 0x8000
- \#define ADVERTISE_FULL
- \#define ADVERTISE_ALL

Link partner ability register

- \#define LPA_SLCT 0x001f
- \#define LPA_10HALF 0x0020
- \#define LPA_10FULL 0x0040
- \#define LPA_100HALF 0x0080
- \#define LPA_100FULL 0x0100
- \#define LPA_100BASE4 0x0200
- \#define LPA_PAUSE_CAP 0x0400
- \#define LPA_PAUSE_ASYM 0x0800
- \#define LPA_RESV 0x1000
- \#define LPA_RFault 0x2000
- \#define LPA_LPACK 0x4000
- \#define LPA_NPAGE 0x8000
- \#define LPA_DUPLEX (LPA_10FULL | LPA_100FULL)
- \#define LPA_100 (LPA_100FULL | LPA_100HALF | LPA_100BASE4)
Expansion register for auto-negotiation

- #define EXPANSION_NWAY 0x0001
- #define EXPANSION_LCWP 0x0002
- #define EXPANSION_ENABLENPAGE 0x0004
- #define EXPANSION_NPCAPABLE 0x0008
- #define EXPANSION_MFAULTS 0x0010
- #define EXPANSION_RESV 0xffe0

N-way test register

- #define NWAYTEST_RESV1 0x00ff
- #define NWAYTEST_LOOPBACK 0x0100
- #define NWAYTEST_RESV2 0xfe00

PHY identifiers

- #define MII_DM9161_ID 0x0181b8a0
- #define MII_AM79C875_ID 0x00225540
- #define MII_KSZ8081_ID 0x00221560
- #define MII_KS8721_ID 0x00221610
- #define MII_STE101P_ID 0x00061C50
- #define MII_DP83848I_ID 0x20005C90
- #define MII_LAN8710A_ID 0x0007C0F1
- #define MII_LAN8720_ID 0x0007C0F0
- #define MII_LAN8742A_ID 0x0007C130
- #define MII_LAN8740A_ID 0x0007C110

7.29.2 Macro Definition Documentation

7.29.2.1 MII_BMCR

#define MII_BMCR 0x00

Basic mode control register.

7.29.2.2 MII_BMSR

#define MII_BMSR 0x01

Basic mode status register.
7.29.2.3 MII_PHYSID1

#define MII_PHYSID1 0x02

PHYS ID 1.

7.29.2.4 MII_PHYSID2

#define MII_PHYSID2 0x03

PHYS ID 2.

7.29.2.5 MII_ADVERTISE

#define MII_ADVERTISE 0x04

Advertisement control reg.

7.29.2.6 MII_LPA

#define MII_LPA 0x05

Link partner ability reg.

7.29.2.7 MII_EXPANSION

#define MII_EXPANSION 0x06

Expansion register.

7.29.2.8 MII_ANNPTR

#define MII_ANNPTR 0x07

1000BASE-T control.

7.29.2.9 MII_CTRL1000

#define MII_CTRL1000 0x09

1000BASE-T control.
7.29.2.10   MII_STAT1000

#define MII_STAT1000 0x0a

1000BASE-T status.

7.29.2.11   MII_ESTATUS

#define MII_ESTATUS 0x0f

Extended Status.

7.29.2.12   MII_PHYSTS

#define MII_PHYSTS 0x10

PHY Status register.

7.29.2.13   MII_MICR

#define MII_MICR 0x11

MII Interrupt ctrl register.

7.29.2.14   MII_DCOUNTER

#define MII_DCOUNTER 0x12

Disconnect counter.

7.29.2.15   MII_FCSCOUNTER

#define MII_FCSCOUNTER 0x13

False carrier counter.

7.29.2.16   MII_NWAYTEST

#define MII_NWAYTEST 0x14

N-way auto-neg test reg.
7.29.2.17  MII_RERRCOUNTER

#define MII_RERRCOUNTER 0x15
Receive error counter.

7.29.2.18  MII_SREVISION

#define MII_SREVISION 0x16
Silicon revision.

7.29.2.19  MII_RESV1

#define MII_RESV1 0x17
Reserved.

7.29.2.20  MII_LBRERROR

#define MII_LBRERROR 0x18
Lpback, rx, bypass error.

7.29.2.21  MII_PHYADDR

#define MII_PHYADDR 0x19
PHY address.

7.29.2.22  MII_RESV2

#define MII_RESV2 0x1a
Reserved.

7.29.2.23  MII_TPISTATUS

#define MII_TPISTATUS 0x1b
TPI status for 10Mbps.
7.29.2.24  MII_NCONFIG

#define MII_NCONFIG 0x1c

Network interface config.

7.29.2.25  BMCR_RESV

#define BMCR_RESV 0x007f

Unused.

7.29.2.26  BMCR_CTST

#define BMCR_CTST 0x0080

Collision test.

7.29.2.27  BMCR_FULLDPLX

#define BMCR_FULLDPLX 0x0100

Full duplex.

7.29.2.28  BMCR_ANRESTART

#define BMCR_ANRESTART 0x0200

Auto negotiation restart.

7.29.2.29  BMCR_ISOLATE

#define BMCR_ISOLATE 0x0400

Disconnect DP83840 from MII.

7.29.2.30  BMCR_PDOWN

#define BMCR_PDOWN 0x0800

Powerdown.
7.29.2.31 **BMCR_ANENABLE**

`#define BMCR_ANENABLE 0x1000`

Enable auto negotiation.

7.29.2.32 **BMCR_SPEED100**

`#define BMCR_SPEED100 0x2000`

Select 100Mbps.

7.29.2.33 **BMCR_LOOPBACK**

`#define BMCR_LOOPBACK 0x4000`

TXD loopback bit.

7.29.2.34 **BMCR_RESET**

`#define BMCR_RESET 0x8000`

Reset.

7.29.2.35 **BMSR_ERCAP**

`#define BMSR_ERCAP 0x0001`

Ext-reg capability.

7.29.2.36 **BMSR_JCD**

`#define BMSR_JCD 0x0002`

Jabber detected.

7.29.2.37 **BMSR_LSTATUS**

`#define BMSR_LSTATUS 0x0004`

Link status.
7.29.2.38 BMSR_ANEGCAPABLE

#define BMSR_ANEGCAPABLE 0x0008

Able to do auto-negotiation.

7.29.2.39 BMSR_RFAULT

#define BMSR_RFAULT 0x0010

Remote fault detected.

7.29.2.40 BMSR_ANEGCOMPLETE

#define BMSR_ANEGCOMPLETE 0x0020

Auto-negotiation complete.

7.29.2.41 BMSR_MFPRESUPPCAP

#define BMSR_MFPRESUPPCAP 0x0040

Able to suppress preamble.

7.29.2.42 BMSR_RESV

#define BMSR_RESV 0x0780

Unused.

7.29.2.43 BMSR_10HALF

#define BMSR_10HALF 0x0800

Can do 10mbps, half-duplex.

7.29.2.44 BMSR_10FULL

#define BMSR_10FULL 0x1000

Can do 10mbps, full-duplex.
7.29.2.45 BMSR_100HALF

#define BMSR_100HALF 0x2000
Can do 100mbps, half-duplex.

7.29.2.46 BMSR_100FULL

#define BMSR_100FULL 0x4000
Can do 100mbps, full-duplex.

7.29.2.47 BMSR_100BASE4

#define BMSR_100BASE4 0x8000
Can do 100mbps, 4k packets.

7.29.2.48 ADVERTISE_SLCT

#define ADVERTISE_SLCT 0x001f
Selector bits.

7.29.2.49 ADVERTISE_CSMA

#define ADVERTISE_CSMA 0x0001
Only selector supported.

7.29.2.50 ADVERTISE_10HALF

#define ADVERTISE_10HALF 0x0020
Try for 10mbps half-duplex.

7.29.2.51 ADVERTISE_10FULL

#define ADVERTISE_10FULL 0x0040
Try for 10mbps full-duplex.
7.29.2.52 ADVERTISE_100HALF

#define ADVERTISE_100HALF 0x0080

Try for 100mbps half-duplex.

7.29.2.53 ADVERTISE_100FULL

#define ADVERTISE_100FULL 0x0100

Try for 100mbps full-duplex.

7.29.2.54 ADVERTISE_100BASE4

#define ADVERTISE_100BASE4 0x0200

Try for 100mbps 4k packets.

7.29.2.55 ADVERTISE_PAUSE_CAP

#define ADVERTISE_PAUSE_CAP 0x0400

Try for pause.

7.29.2.56 ADVERTISE_PAUSEASYM

#define ADVERTISE_PAUSEASYM 0x0800

Try for asymmetric pause.

7.29.2.57 ADVERTISE_RESV

#define ADVERTISE_RESV 0x1000

Unused.

7.29.2.58 ADVERTISE_RFAULT

#define ADVERTISE_RFAULT 0x2000

Say we can detect faults.
7.29.2.59 ADVERTISE_LPACK

#define ADVERTISE_LPACK 0x4000

Ack link partners response.

7.29.2.60 ADVERTISE_NPAGE

#define ADVERTISE_NPAGE 0x8000

Next page bit.

7.29.2.61 LPA_SLCT

#define LPA_SLCT 0x001f

Same as advertise selector.

7.29.2.62 LPA_10HALF

#define LPA_10HALF 0x0020

Can do 10mbps half-duplex.

7.29.2.63 LPA_10FULL

#define LPA_10FULL 0x0040

Can do 10mbps full-duplex.

7.29.2.64 LPA_100HALF

#define LPA_100HALF 0x0080

Can do 100mbps half-duplex.

7.29.2.65 LPA_100FULL

#define LPA_100FULL 0x0100

Can do 100mbps full-duplex.
7.29.2.66  LPA_100BASE4

#define LPA_100BASE4 0x0200
Can do 100mbps 4k packets.

7.29.2.67  LPA_PAUSE_CAP

#define LPA_PAUSE_CAP 0x0400
Can pause.

7.29.2.68  LPA_PAUSE_ASYM

#define LPA_PAUSE_ASYM 0x0800
Can pause asymmetrically.

7.29.2.69  LPA_RESV

#define LPA_RESV 0x1000
Unused.

7.29.2.70  LPA_RFAULT

#define LPA_RFAULT 0x2000
Link partner faulted.

7.29.2.71  LPA_LPACK

#define LPA_LPACK 0x4000
Link partner acked us.

7.29.2.72  LPA_NPAGE

#define LPA_NPAGE 0x8000
Next page bit.
7.29.2.73  EXPANSION_NWAY

#define EXPANSION_NWAY 0x0001

Can do N-way auto-nego.

7.29.2.74  EXPANSION_LCWP

#define EXPANSION_LCWP 0x0002

Got new RX page code word.

7.29.2.75  EXPANSION_ENABLENPAGE

#define EXPANSION_ENABLENPAGE 0x0004

This enables npage words.

7.29.2.76  EXPANSION_NPCAPABLE

#define EXPANSION_NPCAPABLE 0x0008

Link partner supports npage.

7.29.2.77  EXPANSION_MFAULTS

#define EXPANSION_MFAULTS 0x0010

Multiple faults detected.

7.29.2.78  EXPANSION_RESV

#define EXPANSION_RESV 0xffe0

Unused.

7.29.2.79  NWAYTEST_RESV1

#define NWAYTEST_RESV1 0x00ff

Unused.
7.29.2.80  NWAYTEST_LOOPBACK

#define NWAYTEST_LOOPBACK 0x0100

Enable loopback for N-way.

7.29.2.81  NWAYTEST_RESV2

#define NWAYTEST_RESV2 0xfe00

Unused.
7.30 MMC over SPI Driver

Generic MMC driver.

7.30.1 Detailed Description

Generic MMC driver.

This module implements a portable MMC/SD driver that uses a SPI driver as physical layer. Hot plugging and removal are supported through kernel events.

Precondition

In order to use the MMC_SPI driver the HAL_USE_MMC_SPI and HAL_USE_SPI options must be enabled in halconf.h.

7.30.2 Driver State Machine

This driver implements a state machine internally, see the Abstract I/O Block Device module documentation for details.

7.30.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Macros

- `#define MMC_BUFFER_SIZE 16U`
  
  Size of the buffer to be supplied to the driver.

- `#define __mmc_driver_methods _mmcsd_block_device_methods`
  
  MMCDriver specific methods.

MMC_SPI configuration options

- `#define MMC_IDLE_TIMEOUT_MS 1000`
  
  Timeout before assuming a failure while waiting for card idle.

- `#define MMC_USE_MUTUAL_EXCLUSION TRUE`
  
  Mutual exclusion on the SPI bus.

Macro Functions

- `#define mmcIsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)`
  
  Returns the card insertion status.

- `#define mmcIsWriteProtected(mmcp) mmc_lld_is_write_protected(mmcp)`
  
  Returns the write protect status.
Typedefs

- typedef mmc_spi_config_t MMCConfig
  Legacy name for compatibility.
- typedef mmc_spi_driver_t MMCDriver
  Legacy name for compatibility.

Data Structures

- struct mmc_spi_config_t
  Type of a MMC/SD over SPI driver configuration structure.
- struct mmc_spi_driver_vmt
  MMCDriver virtual methods table.
- struct mmc_spi_driver_t
  Structure representing a MMC/SD over SPI driver.

Functions

- static uint8_t mmc_crc7 (uint8_t crc, const uint8_t ∗buffer, size_t len)
  Calculate the MMC standard CRC-7 based on a lookup table.
- static bool mmc_wait_idle (MMCDriver ∗mmcp)
  Waits an idle condition.
- static bool mmc_send_hdr (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg)
  Sends a command header.
- static bool mmc_recvr1 (MMCDriver ∗mmcp, uint8_t ∗r1p)
  Receives a single byte response.
- static bool mmc_recvr3 (MMCDriver ∗mmcp, uint8_t ∗r1p)
  Receives a three byte response.
- static bool mmc_send_command_R1 (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg, uint8_t ∗r1p)
  Sends a command which returns a single byte response.
- static bool mmc_send_command_R3 (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg, uint8_t ∗r1p)
  Sends a command which returns a five bytes response (R3).
- static bool mmc_read_CxD (MMCDriver ∗mmcp, uint8_t cmd, uint32_t cxd[4])
  Reads the CSD.
- void mmcInit (void)
  MMC over SPI driver initialization.
- void mmcObjectInit (MMCDriver ∗mmcp, uint8_t ∗buffer)
  Initializes an instance.
- msg_t mmcStart (MMCDriver ∗mmcp, const MMCConfig ∗config)
  Configures and activates the MMC peripheral.
- void mmcStop (MMCDriver ∗mmcp)
  Disables the MMC peripheral.
- bool mmcConnect (MMCDriver ∗mmcp)
  Performs the initialization procedure on the inserted card.
- bool mmcDisconnect (MMCDriver ∗mmcp)
  Brings the driver in a state safe for card removal.
- bool mmcStartSequentialRead (MMCDriver ∗mmcp, uint32_t startblk)
  Starts a sequential read.
- bool mmcSequentialRead (MMCDriver ∗mmcp, uint8_t ∗buffer)
Reads a block within a sequential read operation.

- bool mmcStopSequentialRead (MMCDriver *mmcp)
  Stops a sequential read gracefully.

- bool mmcStartSequentialWrite (MMCDriver *mmcp, uint32_t startblk)
  Starts a sequential write.

- bool mmcSequentialWrite (MMCDriver *mmcp, const uint8_t *buffer)
  Writes a block within a sequential write operation.

- bool mmcStopSequentialWrite (MMCDriver *mmcp)
  Stops a sequential write gracefully.

- bool mmcSync (MMCDriver *mmcp)
  Waits for card idle condition.

- bool mmcGetInfo (MMCDriver *mmcp, BlockDeviceInfo *bdip)
  Returns the media info.

- bool mmcErase (MMCDriver *mmcp, uint32_t startblk, uint32_t endblk)
  Erases blocks.

Variables

- static const struct mmc_spi_driver_vmt mmc_vmt
  Virtual methods table.

- static const uint8_t mmc_crc7_lookup_table[256]
  Lookup table for CRC-7 (based on polynomial \(x^7 + x^3 + 1\)).

### 7.30.4 Macro Definition Documentation

#### 7.30.4.1 MMC_BUFFER_SIZE

#define MMC_BUFFER_SIZE 16U

Size of the buffer to be supplied to the driver.

**Note**

The buffer is meant to be non-cacheable on platforms with data cache.

#### 7.30.4.2 MMC_IDLE_TIMEOUT_MS

#define MMC_IDLE_TIMEOUT_MS 1000

Timeout before assuming a failure while waiting for card idle.

**Note**

Time is in milliseconds.
7.30.4.3  MMC_USE_MUTUAL_EXCLUSION

#define MMC_USE_MUTUAL_EXCLUSION TRUE

Mutual exclusion on the SPI bus.

7.30.4.4  __mmc_driver_methods

#define __mmc_driver_methods _mmcsd_block_device_methods

MMCDriver specific methods.

7.30.4.5  mmcIsCardInserted

#define mmcIsCardInserted(mmc) mmc_lld_is_card_inserted(mmc)

Returns the card insertion status.

Note

This macro wraps a low level function named sdc_lld_is_card_inserted(), this function must be provided by the application because it is not part of the SDC driver.

Parameters

| in  | mmc   | pointer to the MMCDriver object |

Returns

The card state.

Return values

| false | card not inserted. |
| true  | card inserted. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.30 MMC over SPI Driver

7.30.4.6 mmcIsWriteProtected

#define mmcIsWriteProtected(
    mmcp ) mmc_lld_is_write_protected(mmcp)

Returns the write protect status.

Parameters

| in | mmcp | pointer to the MMCDriver object |

Returns

The card state.

Return values

| false | card not inserted. |
| true  | card inserted.     |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.5 Typedef Documentation

7.30.5.1 MMCConfig

typedef mmc_spi_config_t MMCConfig

Legacy name for compatibility.

Deprecated

7.30.5.2 MMCDriver

typedef mmc_spi_driver_t MMCDriver

Legacy name for compatibility.

Deprecated
### 7.30.6 Function Documentation

#### 7.30.6.1 mmc_crc7()

```c
static uint8_t mmc_crc7 (  
    uint8_t crc,  
    const uint8_t * buffer,  
    size_t len  ) [static]
```

Calculate the MMC standard CRC-7 based on a lookup table.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>crc</td>
<td>start value for CRC</td>
</tr>
<tr>
<td>in</td>
<td>buffer</td>
<td>pointer to data buffer</td>
</tr>
<tr>
<td>in</td>
<td>len</td>
<td>length of data</td>
</tr>
</tbody>
</table>

**Returns**

Calculated CRC

#### 7.30.6.2 mmc_wait_idle()

```c
static bool mmc_wait_idle (  
    MMCDriver * mmcp ) [static]
```

Waits an idle condition.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **HAL_SUCCESS** if the operation succeeded.
- **HAL_FAILED** if the operation failed.
Function Class:
Not an API, this function is for internal use only.

### 7.30.6.3 mmc_send_hdr()

```c
static bool mmc_send_hdr ( 
    MMCDriver * mmcp,
    uint8_t cmd,
    uint32_t arg ) [static]
```

Sends a command header.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>the command id</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>the command argument</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- `HAL_SUCCESS` if the operation succeeded.
- `HAL_FAILED` if the operation failed.

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:
### 7.30.6.4 mmc_recvr1()

```c
static bool mmc_recvr1 (MMCDriver * mmcp,
                        uint8_t * r1p) [static]
```

Receives a single byte response.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
</tr>
<tr>
<td>out</td>
<td>r1p</td>
</tr>
</tbody>
</table>

**Returns**

Operation status.

**Return values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>if the operation failed.</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

### 7.30.6.5 mmc_recvr3()

```c
static bool mmc_recvr3 (MMCDriver * mmcp,
                        uint8_t * r1p) [static]
```

Receives a three byte response.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
</tr>
<tr>
<td>out</td>
<td>r1p</td>
</tr>
</tbody>
</table>

**Returns**

Operation status.

**Return values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>if the operation failed.</td>
</tr>
</tbody>
</table>
7.30 MMC over SPI Driver

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
mccc_recvr3  ->  mccc_recvr1
```

7.30.6.6  mmc_send_command_R1()

```c
static bool mmc_send_command_R1 (  
    MMCDriver * mmcp,
    uint8_t cmd,
    uint32_t arg,
    uint8_t * r1p )  [static]
```

Sends a command an returns a single byte response.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmd</td>
<td>the command id</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>the command argument</td>
</tr>
<tr>
<td>out</td>
<td>r1p</td>
<td>pointer to buffer for R1 answer</td>
</tr>
</tbody>
</table>

**Returns**

Operation status.

**Return values**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>if the operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.30.6.7 mmc_send_command_R3()

```c
static bool mmc_send_command_R3 ( 
    MMCDriver * mmcp,
    uint8_t cmd,
    uint32_t arg,
    uint8_t * rlp ) [static]
```

Sends a command which returns a five bytes response (R3).

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmd</td>
<td>the command id</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>the command argument</td>
</tr>
<tr>
<td>out</td>
<td>rlp</td>
<td>pointer to buffer for R1 answer</td>
</tr>
</tbody>
</table>

**Returns**

The first byte of the response (R1) as an `uint8_t` value.

**Return values**

- `0xFF` timed out.

**Function Class:**

Not an API, this function is for internal use only.

7.30.6.8 mmc_read_CxD()

```c
static bool mmc_read_CxD ( 
    MMCDriver * mmcp,
    uint8_t cmd,
    uint32_t cxd[4] ) [static]
```

Reads the CSD.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
<tr>
<td>out</td>
<td>cmd</td>
<td>command</td>
</tr>
<tr>
<td>out</td>
<td>cxd</td>
<td>pointer to the CSD/CID buffer</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.
7.30 MMC over SPI Driver

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>if the operation failed.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.30.6.9 mmcInit()

void mmcInit (  
    void )

MMC over SPI driver initialization.

Note
This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.

Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.30.6.10 mmcObjectInit()

void mmcObjectInit (  
    MMCDriver * mmcp,  
    uint8_t * buffer )

Initializes an instance.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
<tr>
<td>in buffer</td>
<td>operations buffer</td>
</tr>
</tbody>
</table>

Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.
7.30.6.11 mmcStart()

```c
msg_t mmcStart (MMCDriver * mmcp, const MMCConfig * config)
```

Configures and activates the MMC peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the MMCConfig object</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.6.12 mmcStop()

```c
void mmcStop (MMCDriver * mmcp)
```

Disables the MMC peripheral.

**Parameters**

| in   | mmcp | pointer to the MMCDriver object |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.6.13 mmcConnect()

```c
bool mmcConnect (MMCDriver * mmcp)
```

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the MMC_READY state where it is possible to perform read and write operations.

**Note**

It is possible to invoke this function from the insertion event handler.
**Parameters**

| in | mmcp | pointer to the MMCDriver object |

**Returns**

The operation status.

**Return values**

| HAL_SUCCESS | if the operation succeeded. |
| HAL_FAILED | if the operation failed. |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.30.6.14 mmcDisconnect()

```c
bool mmcDisconnect ( 
    MMCDriver * mmcp 
)
```

Brings the driver in a state safe for card removal.

**Parameters**

| in | mmcp | pointer to the MMCDriver object |

**Returns**

The operation status.

**Return values**

| HAL_SUCCESS | if the operation succeeded. |
| HAL_FAILED | if the operation failed. |
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
mmcDisconnect
osalSysLock
osalSysUnlock
```

### 7.30.6.15 mmcStartSequentialRead()

```c
bool mmcStartSequentialRead (  
    MMCDriver * mmcp,  
    uint32_t startblk )
```

Starts a sequential read.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to read</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>HAL_FAILED</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>if the operation succeeded.</td>
<td>if the operation failed.</td>
<td></td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.30.6.16 mmcSequentialRead()

bool mmcSequentialRead (  
    MMCDriver * mmcp,  
    uint8_t * buffer )

Reads a block within a sequential read operation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>buffer</td>
<td>pointer to the read buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

- **HAL_SUCCESS** if the operation succeeded.
- **HAL_FAILED** if the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.6.17 mmcStopSequentialRead()

bool mmcStopSequentialRead (  
    MMCDriver * mmcp )

 Stops a sequential read gracefully.

Parameters

| in | mmcp | pointer to the MMCDriver object |

Returns

The operation status.

Return values

- **HAL_SUCCESS** if the operation succeeded.
- **HAL_FAILED** if the operation failed.
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.30.6.18 mmcStartSequentialWrite()

```c
bool mmcStartSequentialWrite (
    MMCDriver ∗ mmcp,
    uint32_t startblk )
```

Starts a sequential write.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to write</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **HAL_SUCCESS** if the operation succeeded.
- **HAL_FAILED** if the operation failed.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.30.6.19 mmcSequentialWrite()

```c
bool mmcSequentialWrite ( 
    MMCDriver ∗ mmcp,
    const uint8_t ∗ buffer )
```

Writes a block within a sequential write operation.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>buffer</td>
<td>pointer to the write buffer</td>
</tr>
</tbody>
</table>
7.30 MMC over SPI Driver

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>if the operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.6.20 mmcStopSequentialWrite()

bool mmcStopSequentialWrite (MMCDriver * mmcp)

Stops a sequential write gracefully.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>if the operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.6.21 mmcSync()

bool mmcSync (MMCDriver * mmcp)

Waits for card idle condition.
Parameters

\[\text{in} \quad \text{mmcp} \quad \text{pointer to the MMCDriver object}\]

Returns

The operation status.

Return values

| \textit{HAL_SUCCESS} | if the operation succeeded. |
| \textit{HAL_FAILED}  | if the operation failed.   |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.6.22 \texttt{mmcGetInfo()}

bool mmcGetInfo (  
    MMCDriver * mmcp,  
    BlockDeviceInfo * bdip )

Returns the media info.

Parameters

\[\begin{array}{|c|c|}
\hline
\text{in} & \text{mmcp} & \text{pointer to the MMCDriver object} \\
\hline
\text{out} & \text{bdip} & \text{pointer to a BlockDeviceInfo structure} \\
\hline
\end{array}\]

Returns

The operation status.

Return values

| \textit{HAL_SUCCESS} | if the operation succeeded. |
| \textit{HAL_FAILED}  | if the operation failed.   |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.30.6.23  mmcErase()

```c
bool mmcErase (  
    MMCDriver ∗ mmcp,  
    uint32_t startblk,  
    uint32_t endblk );
```

Erases blocks.

Parameters

| in | mmcp | pointer to the MMCDriver object |
| in | startblk | starting block number |
| in | endblk | ending block number |

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>if the operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]

7.30.7  Variable Documentation

7.30.7.1  mmc_vmt

```c
const struct mmc_spi_driver_vmt mmc_vmt  [static]
```

Initial value:

ChibiOS/HAL
Virtual methods table.

7.30.7.2 mmc_crc7_lookup_table

const uint8_t mmc_crc7_lookup_table[256] [static]

Initial value:

= {
  0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53,
  0x6c, 0x65, 0x7e, 0x77, 0x19, 0x10, 0x0b, 0x02, 0x3d, 0x34, 0x2f, 0x26,
  0x51, 0x58, 0x43, 0x4a, 0x75, 0x7c, 0x67, 0x6e, 0x32, 0x3b, 0x20, 0x29,
  0x16, 0x1f, 0x04, 0x0d, 0x7a, 0x73, 0x68, 0x61, 0x5e, 0x57, 0x4c, 0x45,
  0x2b, 0x22, 0x39, 0x30, 0x0f, 0x06, 0x1d, 0x14, 0x33, 0x32, 0x29, 0x26,
  0x54, 0x51, 0x44, 0x41, 0x78, 0x75, 0x66, 0x63, 0x30, 0x31, 0x2a, 0x24,
  0x1b, 0x10, 0x09, 0x02, 0x4b, 0x42, 0x33, 0x3a, 0x2f, 0x26, 0x19, 0x12,
  0x1b, 0x1a, 0x15, 0x18, 0x29, 0x2c, 0x33, 0x36, 0x23, 0x26, 0x19, 0x12,
  0x1b, 0x1a, 0x15, 0x18, 0x29, 0x2c, 0x33, 0x36, 0x23, 0x26, 0x19, 0x12,
  0x1b, 0x1a, 0x15, 0x18, 0x29, 0x2c, 0x33, 0x36, 0x23, 0x26, 0x19, 0x12,
  0x1b, 0x1a, 0x15, 0x18, 0x29, 0x2c, 0x33, 0x36, 0x23, 0x26, 0x19, 0x12,

Lookup table for CRC-7 (based on polynomial $x^7 + x^3 + 1$).
7.31 MMC/SD Block Device

7.31.1 Detailed Description

This module implements a common ancestor for all device drivers accessing MMC or SD cards. This interface inherits the state machine and the interface from the Abstract I/O Block Device module.

Macros

- `#define MMCSD_BLOCK_SIZE 512U`
  Fixed block size for MMC/SD block devices.
- `#define MMCSD_R1_ERROR_MASK 0xFDFFE008U`
  Mask of error bits in R1 responses.
- `#define MMCSD_CMD8_PATTERN 0x000001AAU`
  Fixed pattern for CMD8.
- `#define _mmcsd_block_device_methods _base_block_device_methods`
  MMCSDBlockDevice specific methods.
- `#define _mmcsd_block_device_data`
  MMCSDBlockDevice specific data.

SD/MMC status conditions

- `#define MMCSD_STS_IDLE 0U`
- `#define MMCSD_STS_READY 1U`
- `#define MMCSD_STS_IDENT 2U`
- `#define MMCSD_STS_STBY 3U`
- `#define MMCSD_STS_TRAN 4U`
- `#define MMCSD_STS_DATA 5U`
- `#define MMCSD_STS_RCV 6U`
- `#define MMCSD_STS_PRG 7U`
- `#define MMCSD_STS_DIS 8U`

SD/MMC commands

- `#define MMCSD_CMD_GO_IDLE_STATE 0U`
- `#define MMCSD_CMD_INIT 1U`
- `#define MMCSD_CMD_ALL_SEND_CID 2U`
- `#define MMCSD_CMD_SEND_RELATIVE_ADDR 3U`
- `#define MMCSD_CMD_SET_RELATIVE_ADDR MMCSD_CMD_SEND_RELATIVE_ADDR`
- `#define MMCSD_CMD_SET_BLOCKLEN 16U`
- `#define MMCSD_CMD_READ_SINGLE_BLOCK 17U`
• #define MMCSD_CMD_READ_MULTIPLE_BLOCK 18U
• #define MMCSD_CMD_SET_BLOCK_COUNT 23U
• #define MMCSD_CMD_WRITE_BLOCK 24U
• #define MMCSD_CMD_WRITE_MULTIPLE_BLOCK 25U
• #define MMCSD_CMD_ERASE_RW_BLK_START 32U
• #define MMCSD_CMD_ERASE_RW_BLK_END 33U
• #define MMCSD_CMD_ERASE 38U
• #define MMCSD_CMD_APP_OP_COND 41U
• #define MMCSD_CMD_LOCK_UNLOCK 42U
• #define MMCSD_CMD_APP_CMD 55U
• #define MMCSD_CMD_READ_OCR 58U

CSD record offsets

• #define MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE 127U, 126U
• #define MMCSD_CSD_MMC_SPEC_VERS_SLICE 125U, 122U
• #define MMCSD_CSD_MMC_TAAC_SLICE 119U, 112U
• #define MMCSD_CSD_MMC_NSAC_SLICE 111U, 104U
• #define MMCSD_CSD_MMC_TRAN_SPEED_SLICE 103U, 96U
• #define MMCSD_CSD_MMC_CCC_SLICE 95U, 84U
• #define MMCSD_CSD_MMC_READ_BLK_LEN_SLICE 83U, 80U
• #define MMCSD_CSD_MMC_READ_BLK_PARTIAL_SLICE 79U, 79U
• #define MMCSD_CSD_MMC_WRITE_BLK_MISALIGN_SLICE 78U, 78U
• #define MMCSD_CSD_MMC_READ_BLK_MISALIGN_SLICE 77U, 77U
• #define MMCSD_CSD_MMC_DSR_IMP_SLICE 76U, 76U
• #define MMCSD_CSD_MMC_C_SIZE_SLICE 73U, 62U
• #define MMCSD_CSD_MMC_VDD_R_CURR_MIN_SLICE 61U, 59U
• #define MMCSD_CSD_MMC_VDD_R_CURR_MAX_SLICE 58U, 56U
• #define MMCSD_CSD_MMC_VDD_W_CURR_MIN_SLICE 55U, 53U
• #define MMCSD_CSD_MMC_VDD_W_CURR_MAX_SLICE 52U, 50U
• #define MMCSD_CSD_MMC_C_SIZE_MULT_SLICE 49U, 47U
• #define MMCSD_CSD_MMC_ERASE_GRP_SIZE_SLICE 46U, 42U
• #define MMCSD_CSD_MMC_ERASE_GRP_MULT_SLICE 41U, 37U
• #define MMCSD_CSD_MMC_WP_GRP_SIZE_SLICE 36U, 32U
• #define MMCSD_CSD_MMC_WP_GRP_ENABLE_SLICE 31U, 31U
• #define MMCSD_CSD_MMC_DEFAULT_ECC_SLICE 30U, 29U
• #define MMCSD_CSD_MMC_R2W_FACTOR_SLICE 28U, 26U
• #define MMCSD_CSD_MMC_WRITE_BL_PARTIAL_SLICE 21U, 21U
• #define MMCSD_CSD_MMC_ECC_SLICE 9U, 8U
• #define MMCSD_CSD_MMC_CRC_SLICE 7U, 1U
• #define MMCSD_CSD_20_CRC_SLICE 7U, 1U
• #define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 15U, 13U
• #define MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE 13U, 13U
• #define MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE 12U, 12U
• #define MMCSD_CSD_20_COPY_SLICE 11U, 10U
• #define MMCSD_CSD_20_CRC_SLICE 7U, 1U
• #define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 11U, 10U
• #define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 15U, 15U
• #define MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE 13U, 13U
• #define MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE 12U, 12U
• #define MMCSD_CSD_20_COPY_SLICE 11U, 10U
• #define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 15U, 15U
• #define MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE 21U, 21U
<table>
<thead>
<tr>
<th>Definition</th>
<th>MMCSD_CSD_20_WRITE_BL_LEN_SLICE</th>
<th>25U, 12U</th>
</tr>
</thead>
<tbody>
<tr>
<td>define</td>
<td>MMCSD_CSD_20_R2W_FACTOR_SLICE</td>
<td>28U, 26U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_WP_GRP_ENABLE_SLICE</td>
<td>31U, 31U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_WP_GRP_SIZE_SLICE</td>
<td>38U, 32U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE</td>
<td>45U, 39U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_ERASE_BLK_EN_SLICE</td>
<td>46U, 46U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_C_SIZE_SLICE</td>
<td>69U, 48U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_DSR_IMP_SLICE</td>
<td>76U, 76U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE</td>
<td>77U, 77U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE</td>
<td>78U, 78U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_READ_BL_PARTIAL_SLICE</td>
<td>79U, 79U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_READ_BL_LEN_SLICE</td>
<td>83U, 80U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_CCC_SLICE</td>
<td>95U, 84U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_TRANS_SPEED_SLICE</td>
<td>103U, 96U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_NSAC_SLICE</td>
<td>111U, 104U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_TAAC_SLICE</td>
<td>119U, 112U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_CSD_STRUCTURE_SLICE</td>
<td>127U, 126U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_20_CRC_SLICE</td>
<td>MMCSD_CSD_20_CRC_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_FILE_FORMAT_SLICE</td>
<td>MMCSD_CSD_20_FILE_FORMAT_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE</td>
<td>MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE</td>
<td>MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_COPY_SLICE</td>
<td>MMCSD_CSD_20_COPY_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE</td>
<td>MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE</td>
<td>MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_WRITE_BL_LEN_SLICE</td>
<td>MMCSD_CSD_20_WRITE_BL_LEN_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_WP_GRP_ENABLE_SLICE</td>
<td>MMCSD_CSD_20_WP_GRP_ENABLE_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_WP_GRP_SIZE_SLICE</td>
<td>MMCSD_CSD_20_WP_GRP_SIZE_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE</td>
<td>MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_ERASE_BLK_EN_SLICE</td>
<td>MMCSD_CSD_20_ERASE_BLK_EN_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_C_SIZE_MULT_SLICE</td>
<td>49U, 47U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_VDD_W_CURR_MAX_SLICE</td>
<td>52U, 50U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_VDD_W_CURR_MIN_SLICE</td>
<td>55U, 53U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_VDD_R_CURR_MAX_SLICE</td>
<td>58U, 56U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_VDD_R_CURR_MIX_SLICE</td>
<td>61U, 59U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_C_SIZE_SLICE</td>
<td>73U, 62U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_DSR_IMP_SLICE</td>
<td>MMCSD_CSD_20_DSR_IMP_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_READ_BLK_MISALIGN_SLICE</td>
<td>MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_READ_BL_PARTIAL_SLICE</td>
<td>MMCSD_CSD_20_READ_BL_PARTIAL_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_READ_BL_LEN_SLICE</td>
<td>83U, 80U</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_CCC_SLICE</td>
<td>MMCSD_CSD_20_CCC_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_TRANS_SPEED_SLICE</td>
<td>MMCSD_CSD_20_TRANS_SPEED_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_NSAC_SLICE</td>
<td>MMCSD_CSD_20_NSAC_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_TAAC_SLICE</td>
<td>MMCSD_CSD_20_TAAC_SLICE</td>
</tr>
<tr>
<td>#define</td>
<td>MMCSD_CSD_10_CSD_STRUCTURE_SLICE</td>
<td>MMCSD_CSD_20_CSD_STRUCTURE_SLICE</td>
</tr>
</tbody>
</table>
CID record offsets

- #define MMCSD_CID_SDC_CRC_SLICE 7U, 1U
- #define MMCSD_CID_SDC_MDT_M_SLICE 11U, 8U
- #define MMCSD_CID_SDC_MDT_Y_SLICE 19U, 12U
- #define MMCSD_CID_SDC_PSN_SLICE 55U, 24U
- #define MMCSD_CID_SDC_PRV_M_SLICE 59U, 56U
- #define MMCSD_CID_SDC_PRV_N_SLICE 63U, 60U
- #define MMCSD_CID_SDC_PNM0_SLICE 71U, 64U
- #define MMCSD_CID_SDC_PNM1_SLICE 79U, 72U
- #define MMCSD_CID_SDC_PNM2_SLICE 87U, 80U
- #define MMCSD_CID_SDC_PNM3_SLICE 95U, 88U
- #define MMCSD_CID_SDC_PNM4_SLICE 103U, 96U
- #define MMCSD_CID_SDC_OID_SLICE 119U, 104U
- #define MMCSD_CID_MMC_CRC_SLICE 7U, 1U
- #define MMCSD_CID_MMC_MDT_Y_SLICE 11U, 8U
- #define MMCSD_CID_MMC_MDT_M_SLICE 15U, 12U
- #define MMCSD_CID_MMC_PSN_SLICE 47U, 16U
- #define MMCSD_CID_MMC_PRV_M_SLICE 51U, 48U
- #define MMCSD_CID_MMC_PRV_N_SLICE 55U, 52U
- #define MMCSD_CID_MMC_PNM0_SLICE 63U, 56U
- #define MMCSD_CID_MMC_PNM1_SLICE 71U, 64U
- #define MMCSD_CID_MMC_PNM2_SLICE 79U, 72U
- #define MMCSD_CID_MMC_PNM3_SLICE 87U, 80U
- #define MMCSD_CID_MMC_PNM4_SLICE 95U, 88U
- #define MMCSD_CID_MMC_PNM5_SLICE 103U, 96U
- #define MMCSD_CID_MMC_OID_SLICE 119U, 104U
- #define MMCSD_CID_MMC_MID_SLICE 127U, 120U

R1 response utilities

- #define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)
  
  Evaluates to true if the R1 response contains error flags.

- #define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)
  
  Returns the status field of an R1 response.

- #define MMCSD_R1_IS_CARD_LOCKED(r1) ((((r1) >> 21U) & 1U) != 0U)
  
  Evaluates to true if the R1 response indicates a locked card.

Macro Functions

- #define mmcsdGetCardCapacity(ip) ((ip)-capacity)
  
  Returns the card capacity in blocks.
Data Structures

- struct MMCSDBlockDeviceVMT
  \textit{MMCSDBlockDevice} virtual methods table.
- struct MMCSDBlockDevice
  MCC/SD block device class.
- struct unpacked_sdc_cid_t
  Unpacked CID register from SDC.
- struct unpacked_mmc_cid_t
  Unpacked CID register from MMC.
- struct unpacked_sdc_csd_10_t
  Unpacked CSD v1.0 register from SDC.
- struct unpacked_sdc_csd_20_t
  Unpacked CSD v2.0 register from SDC.
- struct unpacked_mmc_csd_t
  Unpacked CSD register from MMC.

Functions

- uint32_t \_mmcsd\_get\_slice (const uint32_t ∗data, uint32_t end, uint32_t start)
  Gets a bit field from a words array.
- uint32_t \_mmcsd\_get\_capacity (const uint32_t ∗csd)
  Extract card capacity from a CSD.
- uint32_t \_mmcsd\_get\_capacity\_ext (const uint8_t ∗ext\_csd)
  Extract MMC card capacity from EXT_CSD.
- void \_mmcsd\_unpack\_sdc\_cid (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_cid_t ∗cidsdc)
  Unpacks SDC CID array in structure.
- void \_mmcsd\_unpack\_mmc\_cid (const MMCSDBlockDevice ∗sdcp, unpacked_mmc_cid_t ∗cid mmc)
  Unpacks MMC CID array in structure.
- void \_mmcsd\_unpack\_csd\_mmc (const MMCSDBlockDevice ∗sdcp, unpacked_mmc_csd_t ∗csd mmc)
  Unpacks MMC CSD array in structure.
- void \_mmcsd\_unpack\_csd\_v10 (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_csd_10_t ∗csd10)
  Unpacks SDC CSD v1.0 array in structure.
- void \_mmcsd\_unpack\_csd\_v20 (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_csd_20_t ∗csd20)
  Unpacks SDC CSD v2.0 array in structure.

7.31.2 Macro Definition Documentation

7.31.2.1 MMCSD\_BLOCK\_SIZE

#define MMCSD\_BLOCK\_SIZE 512U

Fixed block size for MMC/SD block devices.
7.31.2.2 MMCSD_R1_ERROR_MASK

#define MMCSD_R1_ERROR_MASK 0xFDFFE008U

Mask of error bits in R1 responses.

7.31.2.3 MMCSD_CMD8_PATTERN

#define MMCSD_CMD8_PATTERN 0x000001AAU

Fixed pattern for CMD8.

7.31.2.4 _mmcsd_block_device_methods

#define _mmcsd_block_device_methods _base_block_device_methods

MMCSDBlockDevice specific methods.

7.31.2.5 _mmcsd_block_device_data

#define _mmcsd_block_device_data

Value:

_value_block_device_data
/* Card CID.*/
uint32_t cid[4];
/* Card CSD.*/
uint32_t csd[4];
/* Total number of blocks in card.*/
uint32_t capacity;

MMCSDBlockDevice specific data.

Note

It is empty because MMCSDBlockDevice is only an interface without implementation.

7.31.2.6 MMCSD_R1_ERROR

#define MMCSD_R1_ERROR(
    r1 ) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)

Evaluates to true if the R1 response contains error flags.
7.31 MMC/SD Block Device

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in r1</td>
<td></td>
<td>the r1 response</td>
</tr>
</tbody>
</table>

7.31.2.7 MMCSD_R1_STS

```c
#define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)
```

Returns the status field of an R1 response.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in r1</td>
<td></td>
<td>the r1 response</td>
</tr>
</tbody>
</table>

7.31.2.8 MMCSD_R1_IS_CARD_LOCKED

```c
#define MMCSD_R1_IS_CARD_LOCKED(r1) ((((r1) >> 21U) & 1U) != 0U)
```

Evaluates to true if the R1 response indicates a locked card.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in r1</td>
<td></td>
<td>the r1 response</td>
</tr>
</tbody>
</table>

7.31.2.9 mmcsdGetCardCapacity

```c
#define mmcsdGetCardCapacity(ip) ((ip)->capacity)
```

Returns the card capacity in blocks.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ip</td>
<td></td>
<td>pointer to a MMCSDBlockDevice or derived class</td>
</tr>
</tbody>
</table>

Returns

The card capacity.

ChibiOS/HAL
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.31.3 Function Documentation

#### 7.31.3.1 _mmcsd_get_slice()

```c
uint32_t _mmcsd_get_slice (  
    const uint32_t *data,  
    uint32_t end,  
    uint32_t start )
```

Gets a bit field from a words array.

**Note**

The bit zero is the LSb of the first word.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>data</th>
<th>pointer to the words array</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>end</td>
<td>bit offset of the last bit of the field, inclusive</td>
</tr>
<tr>
<td>in</td>
<td>start</td>
<td>bit offset of the first bit of the field, inclusive</td>
</tr>
</tbody>
</table>

**Returns**

The bits field value, left aligned.

Function Class:

Not an API, this function is for internal use only.

#### 7.31.3.2 _mmcsd_get_capacity()

```c
uint32_t _mmcsd_get_capacity (  
    const uint32_t *csd )
```

Extract card capacity from a CSD.

The capacity is returned as number of available blocks.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>csd</th>
<th>the CSD record</th>
</tr>
</thead>
</table>
Returns

The card capacity.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CSD format error</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
/mmcsd_get_capacity_ext
/mmcsd_get_capacity
/mmcsd_get_slice
```

### 7.31.3.3 _mmcsd_get_capacity_ext()

```
uint32_t _mmcsd_get_capacity_ext (const uint8_t *ext_csd)
```

Extract MMC card capacity from EXT_CSD.

The capacity is returned as number of available blocks.

Parameters

- **in ext_csd** : the extended CSD record

Returns

The card capacity.

Function Class:

Not an API, this function is for internal use only.
7.31.3.4 _mmcsd_unpack_sdc_cid()

void _mmcsd_unpack_sdc_cid (  
    const MMCSDBlockDevice * sdcp,  
    unpacked_sdc_cid_t * cidsdc )

Unpacks SDC CID array in structure.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the MMCSDBlockDevice object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>cidsdc</td>
<td>pointer to the unpacked_sdc_cid_t object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

[Call graph image]

7.31.3.5 _mmcsd_unpack_mmc_cid()

void _mmcsd_unpack_mmc_cid (  
    const MMCSDBlockDevice * sdcp,  
    unpacked_mmc_cid_t * cidmmc )

Unpacks MMC CID array in structure.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the MMCSDBlockDevice object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>cidmmc</td>
<td>pointer to the unpacked_mmc_cid_t object</td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
| _mmcsd_unpack_mmc_cid | _mmcsd_get_slice |
```

### 7.31.3.6 _mmcsd_unpack_csd_mmc()

```c
void _mmcsd_unpack_csd_mmc (  
    const MMCSDBlockDevice * sdp,  
    unpacked_mmc_csd_t * csdmcc )
```

Unpacks MMC CSD array in structure.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to the MMCSDBlockDevice object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>csdmcc</td>
<td>pointer to the unpacked_mmc_csd_t object</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
| _mmcsd_unpack_csd_mmc | _mmcsd_get_slice |
```
7.31.3.7 _mmcsd_unpack_csd_v10()

```c
void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice * sdcp,
                          unpacked_sdc_csd_10_t * csd10 )
```

Unpacks SDC CSD v1.0 array in structure.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the MMCSDBlockDevice object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>csd10</td>
<td>pointer to the unpacked_sdc_csd_10_t object</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

Here is the call graph for this function:

![Call Graph](#)

7.31.3.8 _mmcsd_unpack_csd_v20()

```c
void _mmcsd_unpack_csd_v20 (const MMCSDBlockDevice * sdcp,
                          unpacked_sdc_csd_20_t * csd20 )
```

Unpacks SDC CSD v2.0 array in structure.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the MMCSDBlockDevice object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>csd20</td>
<td>pointer to the unpacked_sdc_csd_20_t object</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:
7.32 Serial NOR Flash Driver

Serial NOR Flash driver.

7.32.1 Detailed Description

Serial NOR Flash driver.
This module implements a generic driver for serial NOR Flash devices.

7.32.2 Driver State Machine

The flash driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

Macros

• #define SNOR_BUFFER_SIZE 32
  Size of the buffer for internal operations.
• #define _snor_flash_methods_alone
  SNORDriver specific methods.
• #define _snor_flash_methods
  SNORDriver specific methods with inherited ones.
Bus interface modes.

- `#define SNOR_BUS_DRIVER_SPI 0U`
- `#define SNOR_BUS_DRIVER_WSPI 1U`

Configuration options

- `#define SNOR_BUS_DRIVER SNOR_BUS_DRIVER_WSPI`
  
  Physical transport interface.

- `#define SNOR_SHARED_BUS TRUE`
  
  Shared bus switch.

- `#define SNOR_USE_MUTUAL_EXCLUSION TRUE`
  
  Exclusive access control.

- `#define SNOR_SPI_4BYTES_ADDRESS FALSE`
  
  SPI 4-bytes address switch.

Data Structures

- `struct SNORConfig`
  
  Type of a SNOR configuration structure.

- `struct SNORDriverVMT`
  
  SNOR virtual methods table.

- `struct SNORDriver`
  
  Type of SNOR flash class.

Functions

- `static const flash_descriptor_t * snor_get_descriptor (void *instance)`
  
  Returns a pointer to the device descriptor.

- `void bus_acquire (BUSDriver *busp, const BUSConfig *config)`
  
  Bus acquisition and lock.

- `void bus_release (BUSDriver *busp)`
  
  Bus release.

- `void bus_stop (BUSDriver *busp)`
  
  Stops the underlying bus driver.

- `void bus_cmd (BUSDriver *busp, uint32_t cmd)`
  
  Sends a naked command.

- `void bus_cmd_send (BUSDriver *busp, uint32_t cmd, size_t n, const uint8_t *p)`
  
  Sends a command followed by a data transmit phase.

- `void bus_cmd_receive (BUSDriver *busp, uint32_t cmd, size_t n, uint8_t *p)`
  
  Sends a command followed by a data receive phase.

- `void bus_cmd_addr (BUSDriver *busp, uint32_t cmd, flash_offset_t offset)`
  
  Sends a command followed by a flash address.

- `void bus_cmd_addr_send (BUSDriver *busp, uint32_t cmd, flash_offset_t offset, size_t n, const uint8_t *p)`
  
  Sends a command followed by a flash address and a data transmit phase.

- `void bus_cmd_addr_receive (BUSDriver *busp, uint32_t cmd, flash_offset_t offset, size_t n, uint8_t *p)`
  
  Sends a command followed by a flash address and a data receive phase.

- `void bus_cmd_dummy_receive (BUSDriver *busp, uint32_t cmd, uint32_t dummy, size_t n, uint8_t *p)`
  
  Sends a command followed by dummy cycles and a data receive phase.
Module Documentation

- **void** `bus_cmd_addr_dummy_receive`(BUSDriver *busp, uint32_t cmd, flash_offset_t offset, uint32_t dummy, size_t n, uint8_t *p)
  
  Sends a command followed by a flash address, dummy cycles and a data receive phase.

- **void** `snorObjectInit`(SNORDriver *devp, snor_nocache_buffer_t *nocache)
  
  Initializes an instance.

- **void** `snorStart`(SNORDriver *devp, const SNORConfig *config)
  
  Configures and activates SNOR driver.

- **void** `snorStop`(SNORDriver *devp)
  
  Deactivates the SNOR driver.

- **void** `snorMemoryMap`(SNORDriver *devp, uint8_t **addrp)
  
  Enters the memory Mapping mode.

- **void** `snorMemoryUnmap`(SNORDriver *devp)
  
  Leaves the memory Mapping mode.

**Variables**

- static const struct SNORDriverVMT snor_vmt
  
  Virtual methods table.

### 7.32.3 Macro Definition Documentation

#### 7.32.3.1 SNOR_BUFFER_SIZE

```
#define SNOR_BUFFER_SIZE 32
```

Size of the buffer for internal operations.

#### 7.32.3.2 SNOR_BUS_DRIVER

```
#define SNOR_BUS_DRIVER SNOR_BUS_DRIVER_WSPI
```

Physical transport interface.

#### 7.32.3.3 SNOR_SHARED_BUS

```
#define SNOR_SHARED_BUS TRUE
```

Shared bus switch.

If set to TRUE the device acquires bus ownership on each transaction.

Note

Requires `SPI_USE_MUTUAL_EXCLUSION` or `WSPI_USE_MUTUAL_EXCLUSION` depending on mode selected with `SNOR_BUS_MODE`.
7.32.3.4 SNOR_USE_MUTUAL_EXCLUSION

#define SNOR_USE_MUTUAL_EXCLUSION TRUE

Exclusive access control.

Note

Disabling this option saves both code and data space.

7.32.3.5 SNOR_SPI_4BYTES_ADDRESS

#define SNOR_SPI_4BYTES_ADDRESS FALSE

SPI 4-bytes address switch.

If set to TRUE the device will use 4-bytes address in SPI bus, only relevant if SPI is used.

7.32.3.6 _snor_flash_methods_alone

#define _snor_flash_methods_alone

Value:

/* Read SFDP. */
flash_error_t (*read_sfdp)(void *instance,
flash_offset_t offset,
size_t n,
uint8_t *rp);

SNORDriver specific methods.

7.32.3.7 _snor_flash_methods

#define _snor_flash_methods

Value:

_base_flash_methods
_snor_flash_methods_alone

SNORDriver specific methods with inherited ones.

7.32.4 Function Documentation

7.32.4.1 snor_get_descriptor()

static const flash_descriptor_t * snor_get_descriptor ( void * instance ) [static]

Returns a pointer to the device descriptor.
Parameters

| in  | instance | instance pointer |

Returns

Pointer to a static descriptor structure.

### 7.32.4.2 bus_acquire()

```c
void bus_acquire (
    BUSDriver * busp,
    const BUSConfig * config )
```

Bus acquirement and lock.

Parameters

| in  | busp    | pointer to the bus driver |
| in  | config  | bus configuration         |

Function Class:

Not an API, this function is for internal use only.

### 7.32.4.3 bus_release()

```c
void bus_release ( 
    BUSDriver * busp )
```

Bus release.

Parameters

| in  | busp    | pointer to the bus driver |

Function Class:

Not an API, this function is for internal use only.
7.32.4.4  bus_stop()

void bus_stop (  
    BUSDriver ∗ busp )

Stops the underlying bus driver.

Parameters

| in | busp | pointer to the bus driver |

Function Class:

Not an API, this function is for internal use only.

7.32.4.5  bus_cmd()

void bus_cmd (  
    BUSDriver ∗ busp,  
    uint32_t cmd )

Sends a naked command.

Parameters

| in | busp | pointer to the bus driver |
| in | cmd  | instruction code          |

Function Class:

Not an API, this function is for internal use only.

7.32.4.6  bus_cmd_send()

void bus_cmd_send (  
    BUSDriver ∗ busp,  
    uint32_t cmd,  
    size_t n,  
    const uint8_t ∗ p )

Sends a command followed by a data transmit phase.

Parameters

| in | busp | pointer to the bus driver |
| in | cmd  | instruction code          |
| in | n    | number of bytes to receive|
| in | p    | data buffer                |
Function Class:
Not an API, this function is for internal use only.

7.32.4.7 bus_cmd_receive()

void bus_cmd_receive (  
    BUSDriver * busp,  
    uint32_t cmd,  
    size_t n,  
    uint8_t * p )

Sends a command followed by a data receive phase.

Parameters

| in  | busp | pointer to the bus driver |
| in  | cmd  | instruction code          |
| in  | n    | number of bytes to receive |
| out | p    | data buffer                |

Function Class:
Not an API, this function is for internal use only.

7.32.4.8 bus_cmd_addr()

void bus_cmd_addr (  
    BUSDriver * busp,  
    uint32_t cmd,  
    flash_offset_t offset )

Sends a command followed by a flash address.

Parameters

| in  | busp   | pointer to the bus driver |
| in  | cmd    | instruction code          |
| in  | offset | flash offset               |

Function Class:
Not an API, this function is for internal use only.
### 7.32.4.9 bus_cmd_addr_send()

```c
void bus_cmd_addr_send ( 
    BUSDriver ∗ busp, 
    uint32_t cmd, 
    flash_offset_t offset, 
    size_t n, 
    const uint8_t ∗ p )
```

Sends a command followed by a flash address and a data transmit phase.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>busp</code> pointer to the bus driver</td>
</tr>
<tr>
<td>in</td>
<td><code>cmd</code> instruction code</td>
</tr>
<tr>
<td>in</td>
<td><code>offset</code> flash offset</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code> number of bytes to receive</td>
</tr>
<tr>
<td>in</td>
<td><code>p</code> data buffer</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

### 7.32.4.10 bus_cmd_addr_receive()

```c
void bus_cmd_addr_receive ( 
    BUSDriver ∗ busp, 
    uint32_t cmd, 
    flash_offset_t offset, 
    size_t n, 
    uint8_t ∗ p )
```

Sends a command followed by a flash address and a data receive phase.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>busp</code> pointer to the bus driver</td>
</tr>
<tr>
<td>in</td>
<td><code>cmd</code> instruction code</td>
</tr>
<tr>
<td>in</td>
<td><code>offset</code> flash offset</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code> number of bytes to receive</td>
</tr>
<tr>
<td>out</td>
<td><code>p</code> data buffer</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.
7.32.4.11  bus_cmd_dummy_receive()

void bus_cmd_dummy_receive (  
    BUSDriver * busp,  
    uint32_t cmd,  
    uint32_t dummy,  
    size_t n,  
    uint8_t * p )

Sends a command followed by dummy cycles and a data receive phase.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>busp</td>
</tr>
<tr>
<td>in</td>
<td>cmd</td>
</tr>
<tr>
<td>in</td>
<td>dummy</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
<tr>
<td>out</td>
<td>p</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.32.4.12  bus_cmd_addr_dummy_receive()

void bus_cmd_addr_dummy_receive (  
    BUSDriver * busp,  
    uint32_t cmd,  
    flash_offset_t offset,  
    uint32_t dummy,  
    size_t n,  
    uint8_t * p )

Sends a command followed by a flash address, dummy cycles and a data receive phase.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>busp</td>
</tr>
<tr>
<td>in</td>
<td>cmd</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
</tr>
<tr>
<td>in</td>
<td>dummy</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
<tr>
<td>out</td>
<td>p</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.32.4.13  snorObjectInit()

```c
void snorObjectInit ( 
    SNORDriver * devp, 
    snor_nocache_buffer_t * nocache )
```

Initializes an instance.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>devp</th>
<th>pointer to the <strong>SNORDriver</strong> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>nocache</td>
<td>pointer to the non-cacheable buffers</td>
</tr>
</tbody>
</table>

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.32.4.14  snorStart()

```c
void snorStart ( 
    SNORDriver * devp, 
    const SNORConfig * config )
```

Configures and activates SNOR driver.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>devp</th>
<th>pointer to the <strong>SNORDriver</strong> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the configuration</td>
</tr>
</tbody>
</table>

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.32.4.15  snorStop()

```c
void snorStop ( 
    SNORDriver * devp )
```

Deactivates the SNOR driver.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>devp</th>
<th>pointer to the <strong>SNORDriver</strong> object</th>
</tr>
</thead>
</table>

ChibiOS/HAL
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.32.4.16 snorMemoryMap()

void snorMemoryMap (  
   SNORDriver * devp,  
   uint8_t ** addrp )

Enters the memory Mapping mode.

The memory mapping mode is only available when the WSPI mode is selected and the underlying WSPI controller supports the feature.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in devp</td>
<td>pointer to the SNORDriver object</td>
</tr>
<tr>
<td>out addrp</td>
<td>pointer to the memory start address of the mapped flash or NULL</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
bus_acquire
  snorMemoryMap
    bus_release
      wspiMapFlash
```

7.32.4.17 snorMemoryUnmap()

void snorMemoryUnmap (  
   SNORDriver * devp )

Leaves the memory Mapping mode.
7.32 Serial NOR Flash Driver

Parameters

| in | devp | pointer to the SNORDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call graph image]

7.32.5 Variable Documentation

7.32.5.1 snor_vmt

const struct SNORDriverVMT snor_vmt [static]

Initial value:

```c
= {
  (size_t)0,
  snor_get_descriptor, snor_read, snor_program,
  snor_start_erase_all, snor_start_erase_sector,
  snor_query_erase, snor_verify_erase, snor_acquire_exclusive,
  snor_release_exclusive, snor_read_sfdp,
}
```

Virtual methods table.
7.33  Base Object

7.33.1 Detailed Description

HAL uses concepts of Object Oriented Programming even if it is written in C. Things like simple inheritance, multiple inheritance and interfaces are used through the system. This module defines a "base object" that is the ancestor of all classes in the system.

Macros

- #define _base_object_methods
  BaseObject specific methods.
- #define _base_object_data
  BaseObject specific data.

Macro Functions (BaseObject)

- #define objGetInstance(type, ip) ((type)(((size_t)(ip)) - (ip)->vmt->instance_offset))
  Returns the instance pointer starting from an interface pointer.

Data Structures

- struct BaseObjectVMT
  BaseObject virtual methods table.
- struct BaseObject
  Base object class.

7.33.2 Macro Definition Documentation

7.33.2.1 _base_object_methods

#define _base_object_methods

Value:
/* instance offset, used for multiple inheritance, normally zero. It represents the offset between the current object and the container object*/
size_t instance_offset;

BaseObject specific methods.

Note
This object defines no methods.
7.33.2 _base_object_data

#define _base_object_data

*BaseObject* specific data.

Note

This object defines no data.

7.33.3 objGetInstance

#define objGetInstance(
   type,
   ip ) (type)((size_t)(ip)) - (ip)->vmt->instance_offset)

Returns the instance pointer starting from an interface pointer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>type</th>
<th>the type of the instance pointer, it is used for casting</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>the interface pointer</td>
</tr>
</tbody>
</table>

Returns

A pointer to the object implementing the interface
7.34 PAL Driver

I/O Ports Abstraction Layer.

7.34.1 Detailed Description

I/O Ports Abstraction Layer.

This module defines an abstract interface for digital I/O ports. Note that most I/O ports functions are just macros. The macros have default software implementations that can be redefined in a PAL Low Level Driver if the target hardware supports special features like, for example, atomic bit set/reset/masking. Please refer to the ports specific documentation for details.

The PAL Driver driver has the advantage to make the access to the I/O ports platform independent and still be optimized for the specific architectures.

Note that the PAL Low Level Driver may also offer non standard macro and functions in order to support specific features but, of course, the use of such interfaces would not be portable. Such interfaces shall be marked with the architecture name inside the function names.

Precondition

In order to use the PAL driver the HAL_USE_PAL option must be enabled in halconf.h.

7.34.2 Implementation Rules

In implementing a PAL Low Level Driver there are some rules/behaviors that should be respected.

7.34.2.1 Writing on input pads

The behavior is not specified but there are implementations better than others, this is the list of possible implementations, preferred options are on top:

1. The written value is not actually output but latched, should the pads be reprogrammed as outputs the value would be in effect.
2. The write operation is ignored.
3. The write operation has side effects, as example disabling/enabling pull up/down resistors or changing the pad direction. This scenario is discouraged, please try to avoid this scenario.

7.34.2.2 Reading from output pads

The behavior is not specified but there are implementations better than others, this is the list of possible implementations, preferred options are on top:

1. The actual pads states are read (not the output latch).
2. The output latch value is read (regardless of the actual pads states).
3. Unspecified, please try to avoid this scenario.
7.34.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

7.34.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

7.34.2.5 Reading or writing on pins associated to other functionalities

The behavior is not specified.

 Macros

- `#define PAL_PORT_BIT(n) ((ioportmask_t)(1U << (n)))`
  Port bit helper macro.
- `#define PAL_GROUP_MASK(width) ((ioportmask_t)(1U << (width)) - 1U)`
  Bits group mask helper.
- `#define _IOBUS_DATA(name, port, width, offset) (port, PAL_GROUP_MASK(width), offset)`
  Data part of a static I/O bus initializer.
- `#define IOBUS_DECL(name, port, width, offset) _IOBUS_DATA(name, port, width, offset)`
  Static I/O bus initializer.
- `#define IOPORT1 0`
  First I/O port identifier.
- `#define pal_lld_init() _pal_lld_init()` 
  Low level PAL subsystem initialization.
- `#define pal_lld_readport(port) 0U`
  Reads the physical I/O port states.
- `#define pal_lld_readlatch(port) 0U`
  Reads the output latch.
- `#define pal_lld_writeport(port, bits)`
  Writes a bits mask on a I/O port.
- `#define pal_lld_setport(port, bits)`
  Sets a bits mask on a I/O port.
- `#define pal_lld_clearport(port, bits)`
  Clears a bits mask on a I/O port.
- `#define pal_lld_toggleport(port, bits)`
  Toggles a bits mask on a I/O port.
- `#define pal_lld_readgroup(port, mask, offset) 0U`
  Reads a group of bits.
- `#define pal_lld_writegroup(port, mask, offset, bits)`
  Writes a group of bits.
- `#define pal_lld_setgroupmode(port, mask, offset, mode) _pal_lld_setgroupmode(port, mask << offset, mode)`
  Pads group mode setup.
- `#define pal_lld_readpad(port, pad) PAL_LOW`
  Reads a logical state from an I/O pad.
- `#define pal_lld_writepad(port, pad, bit)`
Writes a logical state on an output pad.

- `#define pal_lld_setpad(port, pad)`
  Sets a pad logical state to \texttt{PAL\_HIGH}.

- `#define pal_lld_clearpad(port, pad)`
  Clears a pad logical state to \texttt{PAL\_LOW}.

- `#define pal_lld_togglepad(port, pad)`
  Toggles a pad logical state.

- `#define pal_lld_setpadmode(port, pad, mode)`
  Pad mode setup.

- `#define pal_lld_get_pad_event(port, pad) \_pal\_events\[0\]; (void)(port); (void)pad`
  Returns a PAL event structure associated to a pad.

- `#define pal_lld_get_line_event(line) \_pal\_events\[0\]; (void)line`
  Returns a PAL event structure associated to a line.

**Pads mode constants**

- `#define PAL\_MODE\_RESET 0U`
  After reset state.

- `#define PAL\_MODE\_UNCONNECTED 1U`
  Safe state for \texttt{unconnected} pads.

- `#define PAL\_MODE\_INPUT 2U`
  Regular input high-Z pad.

- `#define PAL\_MODE\_INPUT\_PULLUP 3U`
  Input pad with weak pull up resistor.

- `#define PAL\_MODE\_INPUT\_PULLDOWN 4U`
  Input pad with weak pull down resistor.

- `#define PAL\_MODE\_INPUT\_ANALOG 5U`
  Analog input mode.

- `#define PAL\_MODE\_OUTPUT\_PUSHPULL 6U`
  Push-pull output pad.

- `#define PAL\_MODE\_OUTPUT\_OPENDRAIN 7U`
  Open-drain output pad.

**Logic level constants**

- `#define PAL\_LOW 0U`
  Logical low state.

- `#define PAL\_HIGH 1U`
  Logical high state.
PAL event modes

- #define PAL_EVENT_MODE_EDGES_MASK 3U
  Mask of edges field.
- #define PAL_EVENT_MODE_DISABLED 0U
  Channel disabled.
- #define PAL_EVENT_MODE_RISING_EDGE 1U
  Rising edge callback.
- #define PAL_EVENT_MODE_FALLING_EDGE 2U
  Falling edge callback.
- #define PAL_EVENT_MODE_BOTH_EDGES 3U
  Both edges callback.

PAL configuration options

- #define PAL_USE_CALLBACKS TRUE
  Enables synchronous APIs.
- #define PAL_USE_WAIT TRUE
  Enables synchronous APIs.

Low level driver helper macros

- #define _pal_init_event(e)
  Initializes a PAL event object.
- #define _pal_init_event(e)
  Initializes a PAL event object.
- #define _pal_init_event(e)
  Initializes a PAL event object.
- #define _pal_clear_event(e)
  Clears a PAL event object.
- #define _pal_clear_event(e)
  Clears a PAL event object.
- #define _pal_clear_event(e)
  Clears a PAL event object.
- #define _pal_isr_code(e)
  Common ISR code.
- #define _pal_isr_code(e)
  Common ISR code.
- #define _pal_isr_code(e)
  Common ISR code.
Macro Functions

- `#define palInit() pal_lld_init()`
  PAL subsystem initialization.
- `#define palReadPort(port) ((void)(port), 0U)`
  Reads the physical I/O port states.
- `#define palReadLatch(port) ((void)(port), 0U)`
  Reads the output latch.
- `#define palWritePort(port, bits) ((void)(port), (void)(bits))`
  Writes a bits mask on a I/O port.
- `#define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))`
  Sets a bits mask on a I/O port.
- `#define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ~ (bits))`
  Clears a bits mask on a I/O port.
- `#define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))`
  Toggles a bits mask on a I/O port.
- `#define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))`
  Reads a group of bits.
- `#define palReadGroupLatch(port, mask, offset) ((palReadLatch(port) >> (offset)) & (mask))`
  Reads the group latch.
- `#define palWriteGroup(port, mask, offset, bits)`
  Writes a group of bits.
- `#define palSetGroupMode(port, mask, offset, mode)`
  Pads group mode setup.
- `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)`
  Reads an input pad logic state.
- `#define palWritePad(port, pad, bit)`
  Writes a logic state on an output pad.
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`
  Sets a pad logic state to PAL_HIGH.
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`
  Clears a pad logic state to PAL_LOW.
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`
  Toggles a pad logic state.
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)`
  Pad mode setup.
- `#define palReadLine(line) palReadPad(PAL_PORT(line), PAL_PAD(line))`
  Reads an input line logic state.
- `#define palWriteLine(line, bit) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)`
  Writes a logic state on an output line.
- `#define palSetLine(line) palSetPad(PAL_PORT(line), PAL_PAD(line))`
  Sets a line logic state to PAL_HIGH.
- `#define palClearLine(line) palClearPad(PAL_PORT(line), PAL_PAD(line))`
  Clears a line logic state to PAL_LOW.
- `#define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))`
  Toggles a line logic state.
- `#define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)`
  Line mode setup.
- `#define palEnablePadEventI(port, pad, mode)`
  Pad event enable.
- `#define palDisablePadEventI(port, pad)`
  Pad event disable.
Pad event disable.
• #define palEnablePadEvent(port, pad, mode)
  Pad event enable.
• #define palDisablePadEvent(port, pad)
  Pad event disable.
• #define palEnableLineEventI(line, mode) palEnablePadEventI(PAL_PORT(line), PAL_PAD(line), mode)
  Line event enable.
• #define palDisableLineEventI(line) palDisablePadEventI(PAL_PORT(line), PAL_PAD(line))
  Line event disable.
• #define palEnableLineEvent(line, mode)
  Line event enable.
• #define palDisableLineEvent(line)
  Line event disable.
• #define palIsPadEventEnabledX(port, pad) false
  Pad event enable check.
• #define palIsLineEventEnabledX(line) pal_lld_ispadeventenabled(PAL_PORT(line), PAL_PAD(line))
  Line event enable check.
• #define palSetPadCallback(port, pad, cb, arg)
  Associates a callback to a pad.
• #define palSetLineCallback(line, cb, arg)
  Associates a callback to a line.

Port related definitions
• #define PAL_IOPORTS_WIDTH 16U
  Width, in bits, of an I/O port.
• #define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)
  Whole port mask.

Line handling macros
• #define PAL_LINE(port, pad) (((ioline_t)((uint32_t)(port)) | (uint32_t)(pad)))
  Forms a line identifier.
• #define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFFFU))
  Decodes a port identifier from a line identifier.
• #define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))
  Decodes a pad identifier from a line identifier.
• #define PAL_NOLINE 0U
  Value identifying an invalid line.

Typedefs
• typedef void (*palcallback_t)(void *arg)
  Type of a PAL event callback.
• typedef uint32_t ioportmask_t
  Digital I/O port sized unsigned type.
• typedef uint32_t iomode_t
  Digital I/O modes.
• typedef uint32_t ioline_t
  Type of an I/O line.
• typedef uint32_t ioportid_t
  Port Identifier.
• typedef uint32_t iopadid_t
  Type of an pad identifier.
Data Structures

- struct palevent_t
  Type of a PAL event record.
- struct IOBus
  I/O bus descriptor.
- struct PALConfig
  Generic I/O ports static initializer.

Functions

- ioportmask_t palReadBus (const IOBus *bus)
  Read from an I/O bus.
- void palWriteBus (const IOBus *bus, ioportmask_t bits)
  Write to an I/O bus.
- void palSetBusMode (const IOBus *bus, iomode_t mode)
  Programs a bus with the specified mode.
- void palSetPadCallbackI (ioportid_t port, iopadid_t pad, palcallback_t cb, void *arg)
  Associates a callback to a port/pad.
- void palSetLineCallbackI (ioline_t line, palcallback_t cb, void *arg)
  Associates a callback to a line.
- msg_t palWaitPadTimeoutS (ioportid_t port, iopadid_t pad, sysinterval_t timeout)
  Waits for an edge on the specified port/pad.
- msg_t palWaitPadTimeout (ioportid_t port, iopadid_t pad, sysinterval_t timeout)
  Waits for an edge on the specified port/pad.
- msg_t palWaitLineTimeoutS (ioline_t line, sysinterval_t timeout)
  Waits for an edge on the specified line.
- msg_t palWaitLineTimeout (ioline_t line, sysinterval_t timeout)
  Waits for an edge on the specified line.
- void _pal_lld_init (void)
  STM32 I/O ports configuration.
- void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)
  Pads mode setup.

7.34.3 Macro Definition Documentation

7.34.3.1 PAL_MODE_RESET

#define PAL_MODE_RESET 0U

After reset state.

The state itself is not specified and is architecture dependent, it is guaranteed to be equal to the after-reset state. It is usually an input state.
7.34.3.2 PAL_MODE_UNCONNECTED

#define PAL_MODE_UNCONNECTED 1U

Safe state for unconnected pads.

The state itself is not specified and is architecture dependent, it may be mapped on PAL_MODE_INPUT_PULLUP, PAL_MODE_INPUT_PULLDOWN or PAL_MODE_OUTPUT_PUSHPULL for example.

7.34.3.3 PAL_MODE_INPUT

#define PAL_MODE_INPUT 2U

Regular input high-Z pad.

7.34.3.4 PAL_MODE_INPUT_PULLUP

#define PAL_MODE_INPUT_PULLUP 3U

Input pad with weak pull up resistor.

7.34.3.5 PAL_MODE_INPUT_PULLDOWN

#define PAL_MODE_INPUT_PULLDOWN 4U

Input pad with weak pull down resistor.

7.34.3.6 PAL_MODE_INPUT_ANALOG

#define PAL_MODE_INPUT_ANALOG 5U

Analog input mode.

7.34.3.7 PAL_MODE_OUTPUT_PUSHPULL

#define PAL_MODE_OUTPUT_PUSHPULL 6U

Push-pull output pad.
7.34.3.8 PAL_MODE_OUTPUT_OPENDRAIN

#define PAL_MODE_OUTPUT_OPENDRAIN 7U
Open-drain output pad.

7.34.3.9 PAL_LOW

#define PAL_LOW 0U
Logical low state.

7.34.3.10 PAL_HIGH

#define PAL_HIGH 1U
Logical high state.

7.34.3.11 PAL_EVENT_MODE_EDGES_MASK

#define PAL_EVENT_MODE_EDGES_MASK 3U
Mask of edges field.

7.34.3.12 PAL_EVENT_MODE_DISABLED

#define PAL_EVENT_MODE_DISABLED 0U
Channel disabled.

7.34.3.13 PAL_EVENT_MODE_RISING_EDGE

#define PAL_EVENT_MODE_RISING_EDGE 1U
Rising edge callback.
### 7.34.3.14 PAL_EVENT_MODE_FALLING_EDGE

```c
#define PAL_EVENT_MODE_FALLING_EDGE 2U
```

Falling edge callback.

### 7.34.3.15 PAL_EVENT_MODE_BOTH_EDGES

```c
#define PAL_EVENT_MODE_BOTH_EDGES 3U
```

Both edges callback.

### 7.34.3.16 PAL_USE_CALLBACKS

```c
#define PAL_USE_CALLBACKS TRUE
```

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

### 7.34.3.17 PAL_USE_WAIT

```c
#define PAL_USE_WAIT TRUE
```

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

### 7.34.3.18 PAL_PORT_BIT

```c
#define PAL_PORT_BIT(n) ((ioportmask_t)(1U << (n)))
```

Port bit helper macro.

This macro calculates the mask of a bit within a port.
Parameters

| in  | n  | bit position within the port |

Returns

The bit mask.

### 7.34.3.19 PAL_GROUP_MASK

```c
#define PAL_GROUP_MASK(width) ((ioportmask_t)(1U << (width)) - 1U)
```

Bits group mask helper.

This macro calculates the mask of a bits group.

Parameters

| in  | width | group width |

Returns

The group mask.

### 7.34.3.20 _IOBUS_DATA

```c
#define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}
```

Data part of a static I/O bus initializer.

This macro should be used when statically initializing an I/O bus that is part of a bigger structure.

Parameters

| in  | name | name of the IOBus variable |
| in  | port | I/O port descriptor |
| in  | width | bus width in bits |
| in  | offset | bus bit offset within the port |
7.34.3.21 IOBUS DECL

#define IOBUS_DECL(
    name,
    port,
    width,
    offset ) IObus name = _IOBUS_DATA(name, port, width, offset)

Static I/O bus initializer.

Parameters

| in  | name   | name of the IObus variable |
| in  | port   | I/O port descriptor        |
| in  | width  | bus width in bits          |
| in  | offset | bus bit offset within the port |

7.34.3.22 _pal_init_event [1/3]

#define _pal_init_event(
    e )

Value:

do { 
  osalThreadQueueObjectInit(&_pal_events[e].threads);
  _pal_events[e].cb = NULL;
  _pal_events[e].arg = NULL;
} while (false)

Initializes a PAL event object.

Parameters

| in  | e     | event index |

Function Class:

   Not an API, this function is for internal use only.

7.34.3.23 _pal_init_event [2/3]

#define _pal_init_event(
    e )

Value:
Initialize a PAL event object.

Parameters

| in | e | event index |

Function Class:

Not an API, this function is for internal use only.

7.34.3.24 _pal_init_event [3/3]

#define _pal_init_event{
  e }

Value:

do {
  osalThreadQueueObjectInit(&_pal_events[e].threads);
} while (false)

Initialize a PAL event object.

Parameters

| in | e | event index |

Function Class:

Not an API, this function is for internal use only.

7.34.3.25 _pal_clear_event [1/3]

#define _pal_clear_event{
  e }

Value:

do {
  osalThreadDequeueAll(&_pal_events[e].threads, MSG_RESET);
  _pal_events[e].cb = NULL;
  _pal_events[e].arg = NULL;
} while (false)

Clears a PAL event object.
Parameters

```c
#define _pal_clear_event(
    e)
```

Value:

```c
do {
    _pal_events[e].cb = NULL;
    _pal_events[e].arg = NULL;
} while (false)
```

Clears a PAL event object.

Function Class:

Not an API, this function is for internal use only.

---

7.34.3.27 _pal_clear_event [3/3]

Parameters

```c
#define _pal_clear_event(
    e)
```

Value:

```c
do {
    osalThreadDequeueAllI(&_pal_events[e].threads, MSG_RESET);
} while (false)
```

Clears a PAL event object.

Function Class:

Not an API, this function is for internal use only.
Function Class:

Not an API, this function is for internal use only.

7.34.3.28 _pal_isr_code [1/3]

#define _pal_isr_code(e)

Value:

do {
    if (_pal_events[e].cb != NULL) {
        _pal_events[e].cb(_pal_events[e].arg);
    }
    osalSysLockFromISR();
    osalThreadDequeueAll(&_pal_events[e].threads, MSG_OK);
    osalSysUnlockFromISR();
} while (false)

Common ISR code.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>event index</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.34.3.29 _pal_isr_code [2/3]

#define _pal_isr_code(e)

Value:

do {
    if (_pal_events[e].cb != NULL) {
        _pal_events[e].cb(_pal_events[e].arg);
    }
} while (false)

Common ISR code.

Note

This macro is meant to be used in the low level drivers implementation only.
Parameters

\[ \text{in} \; \mathbf{e} \; \text{event index} \]

Function Class:

Not an API, this function is for internal use only.

### 7.34.3.30 \_pal_isr_code [3/3]

```c
#define _pal_isr_code(
   e )
```

**Value:**

```c
do {
   osalSysLockFromISR();
   osalThreadDequeueAll(&_pal_events[e].threads, MSG_OK);
   osalSysUnlockFromISR();
} while (false)
```

Common ISR code.

**Note**

This macro is meant to be used in the low level drivers implementation only.

Parameters

\[ \text{in} \; \mathbf{e} \; \text{event index} \]

Function Class:

Not an API, this function is for internal use only.

### 7.34.3.31 palInit

```c
#define palInit( ) pal_lld_init()
```

PAL subsystem initialization.

**Note**

This function is implicitly invoked by \texttt{halInit()}, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.
7.34.3.32 palReadPort

#define palReadPort(
    port ) ((void){port}, 0U)

Reads the physical I/O port states.

Note

The function can be called from any context.

Parameters

| in  | port | port identifier |

Returns

The port logic states.

Function Class:

Special function, this function has special requirements see the notes.

7.34.3.33 palReadLatch

#define palReadLatch(
    port ) ((void){port}, 0U)

Reads the output latch.

The purpose of this function is to read back the latched output value.

Note

The function can be called from any context.

Parameters

| in  | port | port identifier |

Returns

The latched logic states.

Function Class:

Special function, this function has special requirements see the notes.
### 7.34.3.34 palWritePort

```c
#define palWritePort(
    port,
    bits ) ((void)(port), (void)(bits))
```

Writes a bits mask on a I/O port.

**Note**

The function can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be written on the specified port</td>
</tr>
</tbody>
</table>

**Function Class:**

Special function, this function has special requirements see the notes.

### 7.34.3.35 palSetPort

```c
#define palSetPort(
    port,
    bits ) palWritePort(port, palReadLatch(port) | (bits))
```

Sets a bits mask on a I/O port.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The function can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be ORed on the specified port</td>
</tr>
</tbody>
</table>

**Function Class:**

Special function, this function has special requirements see the notes.
7.34.3.36  palClearPort

#define palClearPort(
    port,
    bits ) palWritePort(port, palReadLatch(port) & ∼{bits})

Clears a bits mask on a I/O port.

Note
The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons
you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().
The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be cleared on the specified port</td>
</tr>
</tbody>
</table>

Function Class:
Special function, this function has special requirements see the notes.

7.34.3.37  palTogglePort

#define palTogglePort(
    port,
    bits ) palWritePort(port, palReadLatch(port) ^ (bits))

Toggles a bits mask on a I/O port.

Note
The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons
you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().
The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be XORed on the specified port</td>
</tr>
</tbody>
</table>

Function Class:
Special function, this function has special requirements see the notes.
7.34.3.38   **palReadGroup**

```c
#define palReadGroup(
    port,
    mask,
    offset ) ((palReadPort(port) >> (offset)) & (mask))
```

Reads a group of bits.

**Note**

The function can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>type</th>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>port</td>
<td>port identifier</td>
</tr>
<tr>
<td>in</td>
<td>mask</td>
<td>group mask, a logic AND is performed on the input data</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>group bit offset within the port</td>
</tr>
</tbody>
</table>

**Returns**

The group logic states.

**Function Class:**

Special function, this function has special requirements see the notes.

---

7.34.3.39   **palReadGroupLatch**

```c
#define palReadGroupLatch(
    port,
    mask,
    offset ) ((palReadLatch(port) >> (offset)) & (mask))
```

Reads the group latch.

**Note**

The function can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>type</th>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>port</td>
<td>port identifier</td>
</tr>
<tr>
<td>in</td>
<td>mask</td>
<td>group mask, a logic AND is performed on the input data</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>group bit offset within the port</td>
</tr>
</tbody>
</table>
Returns

The group logic states.

Function Class:

Special function, this function has special requirements see the notes.

### 7.34.3.40 palWriteGroup

```c
#define palWriteGroup(
    port,
    mask,
    offset,
    bits )
```

**Value:**

```c
palWritePort(port, (palReadLatch(port) & ~((mask) << (offset))) |
            ((bits) & (mask)) << (offset))
```

 Writes a group of bits.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The function can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mask</td>
<td>group mask, a logic AND is performed on the output data</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>group bit offset within the port</td>
</tr>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be written. Values exceeding the group width are masked.</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

### 7.34.3.41 palSetGroupMode

```c
#define palSetGroupMode(
    port,
    mask,
    offset,
    mode )
```
Value:
```c
    do {
        (void)(port);
        (void)(mask);
        (void)(offset);
        (void)(port);
    } while (mode)
```

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

Parameters

- **in port** port identifier
- **in mask** group mask
- **in offset** group bit offset within the port
- **in mode** group mode

Function Class:

Special function, this function has special requirements see the notes.

7.34.3.42 palReadPad

```c
#define palReadPad(
    port,
    pad ) ((palReadPort(port) >> (pad)) & 1U)
```

Reads an input pad logic state.

Note

The function can be called from any context.

Parameters

- **in port** port identifier
- **in pad** pad number within the port
Returns

The logic state.

Return values

<table>
<thead>
<tr>
<th>PAL_LOW</th>
<th>low logic state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL_HIGH</td>
<td>high logic state.</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

7.34.3.43 palWritePad

```c
#define palWritePad(
    port,
    pad,
    bit)
```

Value:

```c
palWritePort(port, (palReadLatch(port) & ~PAL_PORT_BIT(pad)) | ((bit) & 1U) << pad))
```

Writes a logic state on an output pad.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>bit</td>
<td>logic value, the value must be PAL_LOW or PAL_HIGH</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

7.34.3.44 palSetPad

```c
#define palSetPad(
    port,
    pad)
palSetPort(port, PAL_PORT_BIT(pad))
```

Sets a pad logic state to PAL_HIGH.
Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`. The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

### 7.34.3.45 palClearPad

```c
#define palClearPad( 
    port, 
    pad ) palClearPort(port, PAL_PORT_BIT(pad))
```

Clears a pad logic state to `PAL_LOW`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`. The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

### 7.34.3.46 palTogglePad

```c
#define palTogglePad( 
    port, 
    pad ) palTogglePort(port, PAL_PORT_BIT(pad))
```

Toggles a pad logic state.
Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock(). The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

7.34.3.47 palSetPadMode

#define palSetPadMode(  
    port,  
    pad,  
    mode ) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)

Pad mode setup.

This function programs a pad with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>pad mode</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.
7.34.3.48  palReadLine

#define palReadLine(line) palReadPad(PAL_PORT(line), PAL_PAD(line))

Reads an input line logic state.

Note
The function can be called from any context.

Parameters

in  line  line identifier

Returns
The logic state.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL_LOW</td>
<td>low logic state.</td>
</tr>
<tr>
<td>PAL_HIGH</td>
<td>high logic state.</td>
</tr>
</tbody>
</table>

Function Class:
Special function, this function has special requirements see the notes.

7.34.3.49  palWriteLine

#define palWriteLine(line, bit) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)

Writes a logic state on an output line.

Note
The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock(). The function can be called from any context.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>line</td>
<td>line identifier</td>
</tr>
<tr>
<td>in</td>
<td>bit</td>
<td>logic value, the value must be PAL_LOW or PAL_HIGH</td>
</tr>
</tbody>
</table>
Function Class:

Special function, this function has special requirements see the notes.

### 7.34.3.50 palSetLine

#define palSetLine(
    line ) palSetPad(PAL_PORT(line), PAL_PAD(line))

Sets a line logic state to PAL_HIGH.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The function can be called from any context.

**Parameters**

| in | line | line identifier |

Function Class:

Special function, this function has special requirements see the notes.

### 7.34.3.51 palClearLine

#define palClearLine(
    line ) palClearPad(PAL_PORT(line), PAL_PAD(line))

Clears a line logic state to PAL_LOW.

**Note**

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.

The function can be called from any context.

**Parameters**

| in | line | line identifier |
Function Class:
   Special function, this function has special requirements see the notes.

7.34.3.52 palToggleLine

#define palToggleLine(
   line ) palTogglePad(PAL_PORT(line), PAL_PAD(line))

Toggles a line logic state.

Note
   The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().
   The function can be called from any context.

Parameters
   in line   line identifier

Function Class:
   Special function, this function has special requirements see the notes.

7.34.3.53 palSetLineMode

#define palSetLineMode(
   line,   
   mode ) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)

Line mode setup.

Note
   The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().
   The function can be called from any context.

Parameters
   in line   line identifier
   in mode   pad mode
Function Class:

Special function, this function has special requirements see the notes.

### 7.34.3.54 `palEnablePadEventI`

```c
#define palEnablePadEventI(
    port,
    pad,
    mode )
```

Pad event enable.

**Note**

Programming an unknown or unsupported mode is silently ignored.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>pad event mode</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.34.3.55 `palDisablePadEventI`

```c
#define palDisablePadEventI(
    port,
    pad )
```

Pad event disable.

This function also disables previously programmed event callbacks.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.34.3.56 palEnablePadEvent

#define palEnablePadEvent(
   port,
   pad,
   mode )

Value:

do {
   osalSysLock();
   palEnablePadEventI(port, pad, mode);
   osalSysUnlock();
} while (false)

Pad event enable.

Note
Programming an unknown or unsupported mode is silently ignored.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>pad event mode</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.57 palDisablePadEvent

#define palDisablePadEvent(
   port,
   pad )

Value:

do {
   osalSysLock();
   palDisablePadEventI(port, pad);
   osalSysUnlock();
} while (false)

Pad event disable.

This function also disables previously programmed event callbacks.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.58 palEnableLineEventI

#define palEnableLineEventI(
    line,
    mode ) palEnablePadEventI(PAL_PORT(line), PAL_PAD(line), mode)

Line event enable.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>line event mode</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.34.3.59 palDisableLineEventI

#define palDisableLineEventI(
    line ) palDisablePadEventI(PAL_PORT(line), PAL_PAD(line))

Line event disable.

This function also disables previously programmed event callbacks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.34.3.60 palEnableLineEvent

```c
#define palEnableLineEvent( 
    line, 
    mode )
```

**Value:**

```c
do { 
    osalSysLock(); \ 
    palEnableLineEventI(line, mode); \ 
    osalSysUnlock(); \ 
} while (false)
```

Line event enable.

**Note**

Programming an unknown or unsupported mode is silently ignored.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mode</td>
<td>line event mode</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.34.3.61 palDisableLineEvent

```c
#define palDisableLineEvent( 
    line )
```

**Value:**

```c
do { 
    osalSysLock(); \ 
    palDisableLineEventI(line); \ 
    osalSysUnlock(); \ 
} while (false)
```

Line event disable.

This function also disables previously programmed event callbacks.
Parameters

| in  | line  | line identifier |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.34.3.62 palIsPadEventEnabledX

```c
#define palIsPadEventEnabledX( 
    port, 
    pad ) false
```

Pad event enable check.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

**Returns**

Pad event status.

**Return values**

- **false** if the pad event is disabled.
- **true** if the pad event is enabled.

Function Class:

This is an X-Class API, this function can be invoked from any context.

### 7.34.3.63 palIsLineEventEnabledX

```c
#define palIsLineEventEnabledX( 
    line ) pal_lld_ispadeventenabled(PAL_PORT(line), PAL_PAD(line))
```

Line event enable check.

**Parameters**

| in  | line  | line identifier |

ChibiOS/HAL
Returns

Line event status.

Return values

<table>
<thead>
<tr>
<th>true</th>
<th>if the line event is enabled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the line event is disabled.</td>
</tr>
</tbody>
</table>

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.34.3.64 palSetPadCallback

#define palSetPadCallback(
    port,
    pad,
    cb,
    arg
)

Value:

do {
   osalSysLock();
    palSetPadCallback(port, pad, cb, arg);
    osalSysUnlock();
} while (false)

Associates a callback to a pad.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>cb</td>
<td>event callback function</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>callback argument</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.65 palSetLineCallback

#define palSetLineCallback(
    line,
    cb,
    arg
)
Value:
```c
do {
    osalSysLock();
    palSetLineCallbackI(line, cb, arg);
    osalSysUnlock();
} while (false)
```

Associates a callback to a line.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>line</td>
<td>line identifier</td>
</tr>
<tr>
<td>in</td>
<td>cb</td>
<td>event callback function</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>callback argument</td>
</tr>
</tbody>
</table>

**Function Class:**
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.34.3.66 PAL_IOPORTS_WIDTH

```c
#define PAL_IOPORTS_WIDTH 16U
```

Width, in bits, of an I/O port.

### 7.34.3.67 PAL_WHOLE_PORT

```c
#define PAL_WHOLE_PORT ((ioparmask_t)0xFFFFU)
```

Whole port mask.

This macro specifies all the valid bits into a port.

### 7.34.3.68 PAL_LINE

```c
#define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad)))
```

Forms a line identifier.

A port/pad pair are encoded into an `ioline_t` type. The encoding of this type is platform-dependent.

### 7.34.3.69 PAL_PORT

```c
#define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFFFU))
```

Decodes a port identifier from a line identifier.
### 7.34.3.70 PAL_PAD

```c
#define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))
```

Decodes a pad identifier from a line identifier.

### 7.34.3.71 PAL_NOLINE

```c
#define PAL_NOLINE 0U
```

Value identifying an invalid line.

### 7.34.3.72 IOPORT1

```c
#define IOPORT1 0
```

First I/O port identifier.

Low level drivers can define multiple ports, it is suggested to use this naming convention.

### 7.34.3.73 pal_lld_init

```c
#define pal_lld_init() _pal_lld_init()
```

Low level PAL subsystem initialization.

Function Class:

Not an API, this function is for internal use only.

### 7.34.3.74 pal_lld_readport

```c
#define pal_lld_readport(port) 0U
```

Reads the physical I/O port states.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
</table>
Returns

The port bits.

Function Class:

Not an API, this function is for internal use only.

### 7.34.3.75 pal_lld_readlatch

```c
#define pal_lld_readlatch( port ) 0U
```

Reads the output latch.

The purpose of this function is to read back the latched output value.

Parameters

| in  | port  | port identifier |

Returns

The latched logical states.

Function Class:

Not an API, this function is for internal use only.

### 7.34.3.76 pal_lld_writeport

```c
#define pal_lld_writeport( port, bits )
```

Value:

```c
do {
    (void)port;
    (void)bits;
} while (false)
```

Writes a bits mask on a I/O port.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be written on the specified port</td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

### 7.34.3.77  **pal_lld_setport**

#define pal_lld_setport(
    port,
    bits)

**Value:**

```c
    do {
        (void)port;
        (void)bits;
    } while (false)
```

Sets a bits mask on a I/O port.

**Note**

The **PAL Driver** provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>port</td>
<td>port identifier</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>bits</td>
<td>bits to be ORed on the specified port</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

### 7.34.3.78  **pal_lld_clearport**

#define pal_lld_clearport(
    port,
    bits)

**Value:**

```c
    do {
        (void)port;
        (void)bits;
    } while (false)
```

Clears a bits mask on a I/O port.

**Note**

The **PAL Driver** provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be cleared on the specified port</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.34.3.79 pal_lld_toggleport

```c
#define pal_lld_toggleport(
    port,
    bits)
```

**Value:**

```
    do {
        (void)port;  
        (void)bits;  
    } while (false)
```

Toggles a bits mask on a I/O port.

**Note**

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be XORed on the specified port</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.34.3.80 pal_lld_readgroup

```c
#define pal_lld_readgroup(
    port,
    mask,
    offset ) 0U
```

Reads a group of bits.

**Note**

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>port</td>
<td>port identifier</td>
</tr>
<tr>
<td>in</td>
<td>mask</td>
<td>group mask</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>group bit offset within the port</td>
</tr>
</tbody>
</table>

Returns

The group logical states.

Function Class:

Not an API, this function is for internal use only.

7.34.3.81 pal_lld_writegroup

#define pal_lld_writegroup(
    port,
    mask,
    offset,
    bits )

Value:

do {   \   (void)port;   \   (void)mask;   \   (void)offset;   \   (void)bits;   \   while (false)

Writes a group of bits.

Note

The PAL Driver provides a default software implementation of this functionality. Implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>port</td>
<td>port identifier</td>
</tr>
<tr>
<td>in</td>
<td>mask</td>
<td>group mask</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>group bit offset within the port</td>
</tr>
<tr>
<td>in</td>
<td>bits</td>
<td>bits to be written. Values exceeding the group width are masked.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.34.3.82  pal_lld_setgroupmode

#define pal_lld_setgroupmode(
    port,
    mask,
    offset,
    mode ) _pal_lld_setgroupmode(port, mask << offset, mode)

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

| in  | port   | port identifier |
| in  | mask   | group mask      |
| in  | offset | group bit offset within the port |
| in  | mode   | group mode      |

Function Class:

Not an API, this function is for internal use only.

7.34.3.83  pal_lld_readpad

#define pal_lld_readpad(
    port,
    pad ) PAL_LOW

Reads a logical state from an I/O pad.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

| in  | port   | port identifier |
| in  | pad    | pad number within the port |
Returns

The logical state.

Return values

<table>
<thead>
<tr>
<th>PAL_LOW</th>
<th>low logical state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL_HIGH</td>
<td>high logical state.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.34.3.84 pal_lld_writepad

```c
#define pal_lld_writepad(
    port,
    pad,
    bit )
```

Value:

```c
do {
    (void)port;
    (void)pad;
    (void)bit;
} while (false)
```

Writes a logical state on an output pad.

Note

This function is not meant to be invoked directly by the application code. The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>bit</td>
<td>logical value, the value must be PAL_LOW or PAL_HIGH</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.34.3.85  pal_lld_setpad

#define pal_lld_setpad(  
  port,  
  pad )

Value:
  do {  
    (void)port;  
    (void)pad;  
  } while (false)

Sets a pad logical state to PAL_HIGH.

Note
  The PAL Driver provides a default software implementation of this functionality, implement this function if can  
  optimize it by using special hardware functionalities or special coding.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:
  Not an API, this function is for internal use only.

7.34.3.86  pal_lld_clearpad

#define pal_lld_clearpad(  
  port,  
  pad )

Value:
  do {  
    (void)port;  
    (void)pad;  
  } while (false)

Clears a pad logical state to PAL_LOW.

Note
  The PAL Driver provides a default software implementation of this functionality, implement this function if can  
  optimize it by using special hardware functionalities or special coding.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>
7.34 PAL Driver

Function Class:
Not an API, this function is for internal use only.

7.34.3.87  pal_lld_togglepad

#define pal_lld_togglepad(  
    port,  
    pad  
)

Value:
  do {  
    (void)port;  
    (void)pad;  
  } while (false)

Toggles a pad logical state.

Note
The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.34.3.88  pal_lld_setpadmode

#define pal_lld_setpadmode(  
    port,  
    pad,  
    mode  
)

Value:
  do {  
    (void)port;  
    (void)pad;  
    (void)mode;  
  } while (false)

Pad mode setup.
This function programs a pad with the specified mode.

Note
The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.
Programming an unknown or unsupported mode is silently ignored.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>pad mode</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.34.3.89  pal_lld_get_pad_event

#define pal_lld_get_pad_event(
    port,
    pad ) &_pal_events[0]; (void)(port); (void)pad

Returns a PAL event structure associated to a pad.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.34.3.90  pal_lld_get_line_event

#define pal_lld_get_line_event(
    line ) &_pal_events[0]; (void)line

Returns a PAL event structure associated to a line.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
</table>
Function Class:
Not an API, this function is for internal use only.

## 7.34.4 Typedef Documentation

### 7.34.4.1 palcallback_t

typedef void(* palcallback_t) (void *arg)

Type of a PAL event callback.

### 7.34.4.2 ioportmask_t

typedef uint32_t ioportmask_t

Digital I/O port sized unsigned type.

### 7.34.4.3 iomode_t

typedef uint32_t iomode_t

Digital I/O modes.

### 7.34.4.4 ioline_t

typedef uint32_t ioline_t

Type of an I/O line.

### 7.34.4.5 ioportid_t

typedef uint32_t ioportid_t

Port Identifier.

This type can be a scalar or some kind of pointer, do not make any assumption about it, use the provided macros when populating variables of this type.
7.34.4.6 iopadid_t

typedef uint32_t iopadid_t

Type of an pad identifier.

7.34.5 Function Documentation

7.34.5.1 palReadBus()

ioportmask_t palReadBus (  
    const IOBus * bus )

Read from an I/O bus.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons, you may need to enclose port I/O operations between osalSysLock () and osalSysUnlock ()

The function internally uses the palReadGroup () macro. The use of this function is preferred when you value code size, readability and error checking over speed.

The function can be called from any context.

Parameters

| in | bus | the I/O bus, pointer to a IOBus structure |

Returns

The bus logical states.

Function Class:

Special function, this function has special requirements see the notes.

7.34.5.2 palWriteBus()

void palWriteBus (  
    const IOBus * bus,  
    ioportmask_t bits )

Write to an I/O bus.
Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bus</th>
<th>the I/O bus, pointer to a IOBus structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bits</td>
<td>the bits to be written on the I/O bus. Values exceeding the bus width are masked so most significant bits are lost.</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

7.34.5.3 palSetBusMode()

void palSetBusMode ( const IOBus * bus, iomode_t mode )

Programs a bus with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bus</th>
<th>the I/O bus, pointer to a IOBus structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>the mode</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.
7.34.5.4  palSetPadCallbackI()

```c
void palSetPadCallbackI ( 
    ioportid_t port, 
    iopadid_t pad, 
    palcallback_t cb, 
    void * arg )
```

Associates a callback to a port/pad.

**Parameters**

- **in** `port` port identifier
- **in** `pad` pad number within the port
- **in** `cb` event callback function
- **in** `arg` callback argument

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.34.5.5  palSetLineCallbackI()

```c
void palSetLineCallbackI ( 
    ioline_t line, 
    palcallback_t cb, 
    void * arg )
```

Associates a callback to a line.

**Parameters**

- **in** `line` line identifier
- **in** `cb` event callback function
- **in** `arg` callback argument

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.34.5.6  palWaitPadTimeoutS()

```c
msg_t palWaitPadTimeoutS ( 
    ioportid_t port,
```
Waits for an edge on the specified port/pad.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation state.

Return values

| MSG_OK | if an edge has been detected. |
| MSG_TIMEOUT | if a timeout occurred before an edge could be detected. |
| MSG_RESET | if the event has been disabled while the thread was waiting for an edge. |

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITY</code> no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation state.

Return values

| MSG_OK | if an edge has been detected. |
| MSG_TIMEOUT | if a timeout occurred before an edge could be detected. |
| MSG_RESET | if the event has been disabled while the thread was waiting for an edge. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
osalSysLock
palWaitPadTimeout
osalSysUnlock
palWaitPadTimeoutS
osalThreadEnqueueTimeoutS
```

7.34.5.8 `palWaitLineTimeoutS()`

```c
msg_t palWaitLineTimeoutS ( 
    ioline_t line,
    sysinterval_t timeout )
```

Waits for an edge on the specified line.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>operation timeout</td>
</tr>
</tbody>
</table>

Returns

The operation state.

Return values

| MSG_OK          | if an edge has been detected. |
| MSG_TIMEOUT     | if a timeout occurred before an edge could be detected. |
| MSG_RESET       | if the event has been disabled while the thread was waiting for an edge. |

Function Class:

This is an **S-Class API**, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.34.5.9 **palWaitLineTimeout()**

```c
msg_t palWaitLineTimeout (  
  ioline_t line,  
  sysinterval_t timeout )  
```

Waits for an edge on the specified line.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>operation timeout</td>
</tr>
</tbody>
</table>

Returns

The operation state.
Return values

| MSG_OK       | if an edge has been detected. |
| MSG_TIMEOUT  | if a timeout occurred before an edge could be detected. |
| MSG_RESET    | if the event has been disabled while the thread was waiting for an edge. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.34.5.10  _pal_lld_init()

```c
void _pal_lld_init ( void )
```

STM32 I/O ports configuration.

Function Class:

Not an API, this function is for internal use only.

7.34.5.11  _pal_lld_setgroupmode()

```c
void _pal_lld_setgroupmode ( ioportid_t port, 
                          ioportmask_t mask, 
                          iomode_t mode )
```

Pads mode setup.

This function programs a pads group belonging to the same port with the specified mode.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>port</td>
<td>the port identifier</td>
</tr>
<tr>
<td>in</td>
<td>mask</td>
<td>the group mask</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>the mode</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.35 Peripheral Interfaces

HAL Abstract Peripheral Interfaces.

7.35.1 Detailed Description

HAL Abstract Peripheral Interfaces.

Modules

- Generic Flash Interface

  HAL Generic Flash Driver Interface.
7.36 PWM Driver

Generic PWM Driver.

7.36.1 Detailed Description

Generic PWM Driver.
This module implements a generic PWM (Pulse Width Modulation) driver.

Precondition

In order to use the PWM driver the HAL_USE_PWM option must be enabled in halconf.h.

7.36.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

7.36.3 PWM Operations.

This driver abstracts a generic PWM timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. An optional callback can be generated when this happens.
- An array of PWM_CHANNELS PWM channels, each channel has an output, a comparator and is able to invoke an optional callback when a comparator match with the main counter happens.

A PWM channel output can be in two different states:

- **IDLE**, when the channel is disabled or after a match occurred.
- **ACTIVE**, when the channel is enabled and a match didn’t occur yet in the current PWM cycle.

Note that the two states can be associated to both logical zero or one in the PWMChannelConfig structure.
Macros

- `#define PWM_CHANNELS 4`
  Number of PWM channels per PWM driver.
- `#define pwm_lld_change_period(pwmp, period)`
  Changes the period the PWM peripheral.

PWM output mode macros

- `#define PWM_OUTPUT_MASK 0x0FU`
  Standard output modes mask.
- `#define PWM_OUTPUT_DISABLED 0x00U`
  Output not driven, callback only.
- `#define PWM_OUTPUT_ACTIVE_HIGH 0x01U`
  Positive PWM logic, active is logic level one.
- `#define PWM_OUTPUT_ACTIVE_LOW 0x02U`
  Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- `#define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)`
  Converts from fraction to pulse width.
- `#define PWM_DEGREES_TO_WIDTH(pwmp, degrees) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)`
  Converts from degrees to pulse width.
- `#define PWM_PERCENTAGE_TO_WIDTH(pwmp, percentage) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)`
  Converts from percentage to pulse width.

Macro Functions

- `#define pwmChangePeriodI(pwmp, value)`
  Changes the period the PWM peripheral.
- `#define pwmEnableChannelI(pwmp, channel, width)`
  Enables a PWM channel.
- `#define pwmDisableChannelI(pwmp, channel)`
  Disables a PWM channel.
- `#define pwmIsChannelEnabledI(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)`
  Returns a PWM channel status.
- `#define pwmEnablePeriodicNotificationI(pwmp) pwm_lld_enable_periodic_notification(pwmp)`
  Enables the periodic activation edge notification.
- `#define pwmDisablePeriodicNotificationI(pwmp) pwm_lld_disable_periodic_notification(pwmp)`
  Disables the periodic activation edge notification.
- `#define pwmEnableChannelNotificationI(pwmp, channel) pwm_lld_enable_channel_notification(pwmp, channel)`
  Enables a channel de-activation edge notification.
- `#define pwmDisableChannelNotificationI(pwmp, channel) pwm_lld_disable_channel_notification(pwmp, channel)`
  Disables a channel de-activation edge notification.
PLATFORM configuration options

- #define PLATFORM_PWM_USE_PWM1 FALSE
  PWMD1 driver enable switch.

Typedefs

- typedef struct PWMDriver PWMDriver
  Type of a structure representing a PWM driver.
- typedef void(∗ pwmcallback_t) (PWMDriver ∗pwmp)
  Type of a PWM notification callback.
- typedef uint32_t pwmmode_t
  Type of a PWM mode.
- typedef uint8_t pwmchannel_t
  Type of a PWM channel.
- typedef uint32_t pwmcnt_t
  Type of a PWM counter.
- typedef uint32_t pwmchnmsk_t
  Type of a channels mask.

Data Structures

- struct PWMChannelConfig
  Type of a PWM driver channel configuration structure.
- struct PWMConfig
  Type of a PWM driver configuration structure.
- struct PWMDriver
  Structure representing a PWM driver.

Functions

- void pwmInit (void)
  PWM Driver initialization.
- void pwmObjectInit (PWMDriver ∗pwmp)
  Initializes the standard part of a PWMDriver structure.
- msg_t pwmStart (PWMDriver ∗pwmp, const PWMConfig ∗config)
  Configures and activates the PWM peripheral.
- void pwmStop (PWMDriver ∗pwmp)
  Deactivates the PWM peripheral.
- void pwmChangePeriod (PWMDriver ∗pwmp, pwmcnt_t period)
  Changes the period the PWM peripheral.
- void pwmEnableChannel (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)
  Enables a PWM channel.
- void pwmDisableChannel (PWMDriver ∗pwmp, pwmchannel_t channel)
  Disables a PWM channel and its notification.
- void pwmEnablePeriodicNotification (PWMDriver ∗pwmp)
  Enables the periodic activation edge notification.
- void pwmDisablePeriodicNotification (PWMDriver ∗pwmp)
  Disables the periodic activation edge notification.
• void pwmEnableChannelNotification (PWMDriver *pwmp, pwmchannel_t channel)

  Enables a channel de-activation edge notification.

• void pwmDisableChannelNotification (PWMDriver *pwmp, pwmchannel_t channel)

  Disables a channel de-activation edge notification.

• void pwm_lld_init (void)

  Low level PWM driver initialization.

• void pwm_lld_start (PWMDriver *pwmp)

  Configures and activates the PWM peripheral.

• void pwm_lld_stop (PWMDriver *pwmp)

  Deactivates the PWM peripheral.

• void pwm_lld_enable_channel (PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)

  Enables a PWM channel.

• void pwm_lld_disable_channel (PWMDriver *pwmp, pwmchannel_t channel)

  Disables a PWM channel and its notification.

• void pwm_lld_enable_periodic_notification (PWMDriver *pwmp)

  Enables the periodic activation edge notification.

• void pwm_lld_disable_periodic_notification (PWMDriver *pwmp)

  Disables the periodic activation edge notification.

• void pwm_lld_enable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)

  Enables a channel de-activation edge notification.

• void pwm_lld_disable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)

  Disables a channel de-activation edge notification.

Enumerations

• enum pwmstate_t { PWM_UNINIT = 0, PWM_STOP = 1, PWM_READY = 2 }

  Driver state machine possible states.

Variables

• PWMDriver PWMD1

  PWMD1 driver identifier.

7.36.4  Macro Definition Documentation

7.36.4.1  PWM_OUTPUT_MASK

#define PWM_OUTPUT_MASK 0x0FU

Standard output modes mask.
7.36.2 **PWM_OUTPUT_DISABLED**

```c
#define PWM_OUTPUT_DISABLED 0x00U
```

Output not driven, callback only.

7.36.3 **PWM_OUTPUT_ACTIVE_HIGH**

```c
#define PWM_OUTPUT_ACTIVE_HIGH 0x01U
```

Positive PWM logic, active is logic level one.

7.36.4 **PWM_OUTPUT_ACTIVE_LOW**

```c
#define PWM_OUTPUT_ACTIVE_LOW 0x02U
```

Inverse PWM logic, active is logic level zero.

7.36.5 **PWM_FRACTION_TO_WIDTH**

```c
#define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)
```

Value:

```
((pwmcnt_t)((((pwmcnt_t)(pwmp)->period) * (pwmcnt_t)(numerator)) / (pwmcnt_t)(denominator)) / ((pwmcnt_t)(pwmp)->period))
```

Converts from fraction to pulse width.

**Note**

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see **PWM_COMPUTE_PSC**.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><strong>pwmp</strong></th>
<th>pointer to a <strong>PWM Driver</strong> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>denominator</strong></td>
<td>denominator of the fraction</td>
</tr>
<tr>
<td>in</td>
<td><strong>numerator</strong></td>
<td>numerator of the fraction</td>
</tr>
</tbody>
</table>
Returns
The pulse width to be passed to `pwmEnableChannel()`.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.4.6  PWM_DEGREES_TO_WIDTH

```c
#define PWM_DEGREES_TO_WIDTH( 
    pwmp, 
    degrees ) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)
```

Converts from degrees to pulse width.

Note
Be careful with rounding errors, this is integer math not magic. You can specify hundredths of degrees but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>pwmp</code></th>
<th>pointer to a <code>PWMDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>degrees</code></td>
<td>degrees as an integer between 0 and 36000</td>
</tr>
</tbody>
</table>

Returns
The pulse width to be passed to `pwmEnableChannel()`.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.4.7  PWM_PERCENTAGE_TO_WIDTH

```c
#define PWM_PERCENTAGE_TO_WIDTH( 
    pwmp, 
    percentage ) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)
```

Converts from percentage to pulse width.

Note
Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`. 
7.36 PWM Driver

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>pwmp</code></th>
<th>pointer to a <code>PWMDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>percentage</code></td>
<td>percentage as an integer between 0 and 10000</td>
</tr>
</tbody>
</table>

Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.4.8 `pwmChangePeriodI`

```c
#define pwmChangePeriodI(
    pwmp,
    value
)
```

Value:

```c
{  
    (pwmp)->period = (value);
    pwm_lld_change_period(pwmp, value);
}
```

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart()`.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The PWM unit period is changed to the new value.

Note

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>pwmp</code></th>
<th>pointer to a <code>PWMDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>value</code></td>
<td>new cycle time in ticks</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.4.9  pwmEnableChannelI

#define pwmEnableChannelI(
pwmp,
    channel,
    width )

Value:

do {
    (pwmp)->enabled |= ((pwmchnmsk_t)1U « (pwmchnmsk_t)(channel));
    pwm_lld_enable_channel(pwmp, channel, width);
} while (false)

Enables a PWM channel.

Precondition

The PWM unit must have been activated using pwmStart().

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pwmp</th>
<th>pointer to a PWMDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>channel</td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
<tr>
<td>in</td>
<td>width</td>
<td>PWM pulse width as clock pulses number</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.4.10  pwmDisableChannelI

#define pwmDisableChannelI(
pwmp,
    channel )

ChibiOS/HAL
Value:

```c
    do {
        (pwmp)->enabled &= ~((pwmchnmsk_t)1U « (pwmchnmsk_t)(channel));
        pwm_lld_disable_channel(pwmp, channel);
    } while (false)
```

Disables a PWM channel.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pwmp</td>
<td>pointer to a PWMDriver object</td>
</tr>
<tr>
<td>in channel</td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.4.11 `pwmIsChannelEnabledI`

```c
#define pwmIsChannelEnabledI( pwmp, channel ) (((pwmp)->enabled & ((pwmchnmsk_t)1U « (pwmchnmsk_t)(channel))) != 0U)
```

Returns a PWM channel status.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pwmp</td>
<td>pointer to a PWMDriver object</td>
</tr>
<tr>
<td>in channel</td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

`7.36.4.12 pwmEnablePeriodicNotificationI`

```c
#define pwmEnablePeriodicNotificationI(pwmp) pwm_lld_enable_periodic_notification(pwmp)
```

Enables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

| in  | pwmp | pointer to a PWMDriver object |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

`7.36.4.13 pwmDisablePeriodicNotificationI`

```c
#define pwmDisablePeriodicNotificationI(pwmp) pwm_lld_disable_periodic_notification(pwmp)
```

Disables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already disabled then the call has no effect.
7.36 PWM Driver

Parameters

| in | pwmp | pointer to a PWMDriver object |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.4.14 pwmEnableChannelNotificationI

#define pwmEnableChannelNotificationI(
    pwmp,
    channel
) pwm_lld_enable_channel_notification(pwmp, channel)

Enables a channel de-activation edge notification.

Precondition

The PWM unit must have been activated using pwmStart().

The channel must have been activated using pwmEnableChannel().

Note

If the notification is already enabled then the call has no effect.

Parameters

| in | pwmp | pointer to a PWMDriver object |
| in | channel | PWM channel identifier (0...channels-1) |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.4.15 pwmDisableChannelNotificationI

#define pwmDisableChannelNotificationI(
    pwmp,
    channel
) pwm_lld_disable_channel_notification(pwmp, channel)

Disables a channel de-activation edge notification.
Precondition

The PWM unit must have been activated using \texttt{pwmStart()}. The channel must have been activated using \texttt{pwmEnableChannel()}.

Note

If the notification is already disabled then the call has no effect.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\texttt{pwmp}</td>
<td>pointer to a \texttt{PWMDriver} object</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{channel}</td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>

Function Class:

This is an \texttt{I-Class} API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.36.4.16 PWM_CHANNELS

\#define PWM_CHANNELS 4

Number of PWM channels per PWM driver.

### 7.36.4.17 PLATFORM_PWM_USE_PWM1

\#define PLATFORM_PWM_USE_PWM1 FALSE

PWMD1 driver enable switch.

If set to \texttt{TRUE} the support for PWM1 is included.

Note

The default is \texttt{FALSE}.
### 7.36.18 pwm_lld_change_period

```c
#define pwm_lld_change_period(
    pwmp,
    period
)
```

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart()`.

**Precondition**

The PWM unit must have been activated using `pwmStart()`.

**Postcondition**

The PWM unit period is changed to the new value.

**Note**

The function has effect at the next cycle start.

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwmp</code></td>
<td>pointer to a <code>PWMDriver</code> object</td>
</tr>
<tr>
<td><code>period</code></td>
<td>new cycle time in ticks</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

### 7.36.5 Typedef Documentation

#### 7.36.5.1 PWMDriver

```c
typedef struct PWMDriver PWMDriver
```

Type of a structure representing a PWM driver.

#### 7.36.5.2 pwmcallback_t

```c
typedef void(* pwmcallback_t)(PWMDriver *pwmp)
```

Type of a PWM notification callback.
Parameters

| in | pwxmp | pointer to a PWMDriver object |

7.36.5.3 pwmmode_t

typedef uint32_t pwmmode_t

Type of a PWM mode.

7.36.5.4 pwmchannel_t

typedef uint8_t pwmchannel_t

Type of a PWM channel.

7.36.5.5 pwmchnmsk_t

typedef uint32_t pwmchnmsk_t

Type of a channels mask.

7.36.5.6 pwmcnt_t

typedef uint32_t pwmcnt_t

Type of a PWM counter.

7.36.6 Enumeration Type Documentation

7.36.6.1 pwmstate_t

enum pwmstate_t

Driver state machine possible states.
7.36.7 Function Documentation

7.36.7.1 pwmInit()

```c
void pwmInit (
    void 
)
```

PWM Driver initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call Graph]

7.36.7.2 pwmObjectInit()

```c
void pwmObjectInit ( 
    PWMDriver * pwmp 
)
```

Initializes the standard part of a `PWMDriver` structure.
### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>pwmp</code></td>
<td>pointer to a <code>PWMDriver</code> object</td>
</tr>
</tbody>
</table>

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.36.7.3 `pwmStart()`

```c
msg_t pwmStart (PWMDriver * pwmp, const PWMConfig * config)
```

Configures and activates the PWM peripheral.

**Note**

Starting a driver that is already in the `PWM_READY` state disables all the active channels.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>pwmp</code></td>
<td>pointer to a <code>PWMDriver</code> object</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>config</code></td>
<td>pointer to a <code>PWMConfig</code> object</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.36 PWM Driver

7.36.7.4  pwmStop()

```c
void pwmStop (PWMDriver *pwmp)
```

Deactivates the PWM peripheral.

**Parameters**

| in | pwmp | pointer to a PWMDriver object |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph]

7.36.7.5  pwmChangePeriod()

```c
void pwmChangePeriod (PWMDriver *pwmp,
                      pwmcnt_t period)
```

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart()`.

**Precondition**

The PWM unit must have been activated using `pwmStart()`.

**Postcondition**

The PWM unit period is changed to the new value.

**Note**

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.
7.36.7.6  pwmEnableChannel()

void pwmEnableChannel (  
PWMDriver * pwmp,  
pwmchannel_t channel,  
pwmcnt_t width )

Enables a PWM channel.

Precondition

The PWM unit must have been activated using pwmStart().

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pwmp</th>
<th>pointer to a PWMDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>channel</td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
<tr>
<td>in</td>
<td>width</td>
<td>PWM pulse width as clock pulses number</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
pwmEnableChannel
  osalSysLock

osalSysUnlock
```

### 7.36.7.7 pwmDisableChannel()

```c
void pwmDisableChannel (  
    PWMDriver * pwmp,  
    pwmchannel_t channel )
```

Disables a PWM channel and its notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).
7.36.7.8 **pwmEnablePeriodicNotification()**

```c
void pwmEnablePeriodicNotification (  
    PWMDriver * pwmp  )
```

Enables the periodic activation edge notification.

**Precondition**

The PWM unit must have been activated using `pwmStart()`.

**Note**

If the notification is already enabled then the call has no effect.

**Parameters**

```markdown
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwmp</code></td>
<td>pointer to a PWMDriver object</td>
</tr>
</tbody>
</table>
```

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

7.36.7.9 \texttt{pwmDisablePeriodicNotification()}

\begin{verbatim}
    void pwmDisablePeriodicNotification (PWMDriver * pwmp )
\end{verbatim}

Disables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using \texttt{pwmStart()}.  

Note

If the notification is already disabled then the call has no effect.

Parameters

\begin{verbatim}
    in pwmp   pointer to a PWMDriver object
\end{verbatim}
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
pwmDisablePeriodicNotification
    osalSysLock
osalSysUnlock
```

### 7.36.7.10 pwmEnableChannelNotification()

```c
void pwmEnableChannelNotification ( 
    PWMDriver * pwmp, 
    pwmchannel_t channel )
```

Enables a channel de-activation edge notification.

**Precondition**
- The PWM unit must have been activated using `pwmStart()`.
- The channel must have been activated using `pwmEnableChannel()`.

**Note**
If the notification is already enabled then the call has no effect.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>pwmp</code></td>
<td>pointer to a <code>PWMDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>channel</code></td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
void pwmDisableChannelNotification (PWMDriver * pwmp, pwmchannel_t channel)
```

Disables a channel de-activation edge notification.

Precondition

- The PWM unit must have been activated using `pwmStart()`.
- The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pwmp</td>
<td>pointer to a PWMDriver object</td>
</tr>
<tr>
<td>in</td>
<td>channel</td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
pwmDisableChannelNotification

osalSysLock

osalSysUnlock
```

### 7.36.7.12 pwm_lld_init()

```c
void pwm_lld_init ( )
```

Low level PWM driver initialization.

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
pwm_lld_init

pwmObjectInit
```

### 7.36.7.13 pwm_lld_start()

```c
void pwm_lld_start ( PWMDriver * pwmp )
```

Configures and activates the PWM peripheral.

Note
Starting a driver that is already in the `PWM_READY` state disables all the active channels.
7.36 PWM Driver

Parameters

| in | pwmp | pointer to a PWMDriver object |

Function Class:

Not an API, this function is for internal use only.

7.36.7.14 pwm_lld_stop()

```c
void pwm_lld_stop ( 
    PWMDriver * pwmp )
```

Deactivates the PWM peripheral.

Parameters

| in | pwmp | pointer to a PWMDriver object |

Function Class:

Not an API, this function is for internal use only.

7.36.7.15 pwm_lld_enable_channel()

```c
void pwm_lld_enable_channel ( 
    PWMDriver * pwmp,
    pwmchannel_t channel,
    pwmcnt_t width )
```

Enables a PWM channel.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is active using the specified configuration.

Note

The function has effect at the next cycle start.
Channel notification is not enabled.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> <code>pwmp</code></td>
<td>pointer to a PWMDriver object</td>
</tr>
<tr>
<td><strong>in</strong> <code>channel</code></td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
<tr>
<td><strong>in</strong> <code>width</code></td>
<td>PWM pulse width as clock pulses number</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.36.7.16  pwm_lld_disable_channel()

```c
void pwm_lld_disable_channel ( 
    PWMDriver * pwmp, 
    pwmchannel_t channel 
)
```

Disables a PWM channel and its notification.

**Precondition**

The PWM unit must have been activated using `pwmStart()`.

**Postcondition**

The channel is disabled and its output line returned to the idle state.

**Note**

The function has effect at the next cycle start.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> <code>pwmp</code></td>
<td>pointer to a PWMDriver object</td>
</tr>
<tr>
<td><strong>in</strong> <code>channel</code></td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.36.7.17  pwm_lld_enable_periodic_notification()

```c
void pwm_lld_enable_periodic_notification ( 
    PWMDriver * pwmp 
)
```

Enables the periodic activation edge notification.
Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

| in  | `pwmp` | pointer to a `PWMDriver` object |

Function Class:

Not an API; this function is for internal use only.

7.36.7.18 `pwm_lld_disable_periodic_notification()`

```c
void pwm_lld_disable_periodic_notification ( 
    PWMDriver * pwmp )
```

Disables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

| in  | `pwmp` | pointer to a `PWMDriver` object |

Function Class:

Not an API; this function is for internal use only.

7.36.7.19 `pwm_lld_enable_channel_notification()`

```c
void pwm_lld_enable_channel_notification ( 
    PWMDriver * pwmp, 
    pwmchannel_t channel )
```

Enables a channel de-activation edge notification.
Precondition

The PWM unit must have been activated using `pwmStart()`. The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwmp</code></td>
<td>pointer to a <code>PWMDriver</code> object</td>
</tr>
<tr>
<td><code>channel</code></td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.36.7.20 `pwm_lld_disable_channel_notification()`

```c
void pwm_lld_disable_channel_notification ( 
    PWMDriver * pwmp, 
    pwmchannel_t channel )
```

Disables a channel de-activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`. The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwmp</code></td>
<td>pointer to a <code>PWMDriver</code> object</td>
</tr>
<tr>
<td><code>channel</code></td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.36.8 Variable Documentation
7.36.8.1 PWMD1

PWMDriver PWMD1

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.
7.37 I/O Bytes Queues

7.37.1 Detailed Description

Queues are mostly used in serial-like device drivers. Serial device drivers are usually designed to have a lower side (lower driver, it is usually an interrupt service routine) and an upper side (upper driver, accessed by the application threads).

There are several kind of queues:

- **Input queue**, unidirectional queue where the writer is the lower side and the reader is the upper side.
- **Output queue**, unidirectional queue where the writer is the upper side and the reader is the lower side.
- **Full duplex queue**, bidirectional queue. Full duplex queues are implemented by pairing an input queue and an output queue together.

Queue functions returned status value

- #define Q_OK MSG_OK
  
  Operation successful.

- #define Q_TIMEOUT MSG_TIMEOUT
  
  Timeout condition.

- #define Q_RESET MSG_RESET
  
  Queue has been reset.

- #define Q_EMPTY MSG_TIMEOUT
  
  Queue empty.

- #define Q_FULL MSG_TIMEOUT
  
  Queue full.

Macro Functions

- #define qSizeX(qp)
  
  Returns the queue's buffer size.

- #define qSpaceI(qp) ((qp) - q_counter)
  
  Queue space.

- #define qGetLink(qp) ((qp) - q_link)
  
  Returns the queue application-defined link.

- #define qSetLink(qp, lk) ((qp) - q_link = lk)
  
  Sets the queue application-defined link.

- #define iqGetFullI(iqp) qSpaceI(iqp)
  
  Returns the filled space into an input queue.

- #define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))
  
  Returns the empty space into an input queue.

- #define iqIsEmptyI(iqp) ((bool)(qSpaceI(iqp) == 0U))
  
  Evaluates to true if the specified input queue is empty.
7.37 I/O Bytes Queues

- `#define iqIsFull(iqp)`
  
  Evaluates to `true` if the specified input queue is full.

- `#define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)`
  
  Input queue read.

- `#define oqGetFull(oqp) (qSizeX(oqp) - qSpace(oqp))`
  
  Returns the filled space into an output queue.

- `#define oqGetEmpty(oqp) qSpace(oqp)`
  
  Returns the empty space into an output queue.

- `#define oqIsEmpty(oqp)`
  
  Evaluates to `true` if the specified output queue is empty.

- `#define oqIsFull(oqp) ((bool)(qSpace(oqp) == 0U))`
  
  Evaluates to `true` if the specified output queue is full.

- `#define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)`
  
  Output queue write.

**Typedefs**

- `typedef struct io_queue io_queue_t`
  
  Type of a generic I/O queue structure.

- `typedef void void_t (io_queue_t *qp)`
  
  Queue notification callback type.

- `typedef io_queue_t input_queue_t`
  
  Type of an input queue structure.

- `typedef io_queue_t output_queue_t`
  
  Type of an output queue structure.

**Data Structures**

- `struct io_queue`
  
  Generic I/O queue structure.

**Functions**

- `static size_t iq_read (input_queue_t *iqp, uint8_t *bp, size_t n)`
  
  Non-blocking input queue read.

- `static size_t oq_write (output_queue_t *oqp, const uint8_t *bp, size_t n)`
  
  Non-blocking output queue write.

- `void iqObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)`
  
  Initializes an input queue.

- `void iqResetI (input_queue_t *iqp)`
  
  Resets an input queue.

- `msg_t iqPutI (input_queue_t *iqp, uint8_t b)`
  
  Input queue write.

- `msg_t iqGetI (input_queue_t *iqp)`
  
  Input queue non-blocking read.

- `msg_t iqGetTimeout (input_queue_t *iqp, sysinterval_t timeout)`
  
  Input queue read with timeout.

- `size_t iqReadI (input_queue_t *iqp, uint8_t *bp, size_t n)`

ChibiOS/HAL
Input queue non-blocking read.

- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, sysinterval_t timeout)`
  
  Input queue read with timeout.

- `void oqObjectInit (output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)`
  
  Initializes an output queue.

- `void oqResetI (output_queue_t *oqp)`
  
  Resets an output queue.

- `msg_t oqPutI (output_queue_t *oqp, uint8_t b)`
  
  Output queue non-blocking write.

- `msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, sysinterval_t timeout)`
  
  Output queue write with timeout.

- `msg_t oqGetI (output_queue_t *oqp)`
  
  Output queue read.

- `size_t oqWriteI (output_queue_t *oqp, const uint8_t *bp, size_t n)`
  
  Output queue non-blocking write.

- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, sysinterval_t timeout)`
  
  Output queue write with timeout.

### 7.37.2 Macro Definition Documentation

#### 7.37.2.1 Q_OK

```c
#define Q_OK MSG_OK
```

Operation successful.

#### 7.37.2.2 Q_TIMEOUT

```c
#define Q_TIMEOUT MSG_TIMEOUT
```

Timeout condition.

#### 7.37.2.3 Q_RESET

```c
#define Q_RESET MSG_RESET
```

Queue has been reset.
7.37.2.4 QEMPTY

#define QEMPTY MSG_TIMEOUT

Queue empty.

7.37.2.5 QFULL

#define QFULL MSG_TIMEOUT

Queue full.

7.37.2.6 qSizeX

#define qSizeX(
    qp )

Value:
/*lint -save -e9033 [10.8] The cast is safe.*/
(size_t)((qp)->q_top - (qp)->q_buffer))
/*lint -restore*/

Returns the queue’s buffer size.

Parameters

| in | qp | pointer to a io_queue_t structure |

Returns

The buffer size.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.37.2.7 qSpaceI

#define qSpaceI(
    qp ) ((qp)->q_counter)

Queue space.

Returns the used space if used on an input queue or the empty space if used on an output queue.
Parameters

| in | qp | pointer to a io_queue_t structure |

Returns

The buffer space.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.37.2.8 qGetLink

```c
#define qGetLink( 
    qp ) {((qp)->q_link)
```

Returns the queue application-defined link.

Note

This function can be called in any context.

Parameters

| in | qp | pointer to a io_queue_t structure |

Returns

The application-defined link.

Function Class:

Special function, this function has special requirements see the notes.

### 7.37.2.9 qSetLink

```c
#define qSetLink( 
    qp, 
    lk ) {((qp)->q_link = lk)
```

Sets the queue application-defined link.

Note

This function can be called in any context.
### 7.37.2.10 iqGetFullI

```c
#define iqGetFullI(iqp) qSpaceI(iqp)
```

Returns the filled space into an input queue.

**Parameters**

| in | iq | pointer to an input_queue_t structure |

**Returns**

The number of full bytes in the queue.

**Return values**

| 0 | if the queue is empty. |

**Function Class:**

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.37.2.11 iqGetEmptyI

```c
#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))
```

Returns the empty space into an input queue.

**Parameters**

| in | iq | pointer to an input_queue_t structure |

**ChibiOS/HAL**
Returns

The number of empty bytes in the queue.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the queue is full.</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.37.2.12 iqIsEmptyI

#define iqIsEmptyI(
    iqp ) ((bool)(qSpaceI(iqp) == 0U))

Evaluates to \texttt{true} if the specified input queue is empty.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in iqp</td>
<td>pointer to an \texttt{input_queue_t} structure</td>
</tr>
</tbody>
</table>

Returns

The queue status.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the queue is not empty.</td>
</tr>
<tr>
<td>true</td>
<td>if the queue is empty.</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.37.2.13 iqIsFullI

#define iqIsFullI(
    iqp )

Value:

\[\{\texttt{bool}\}\{((iqp)->q_wrptr == (iqp)->q_rdptr) \&\& ((iqp)->q_counter != 0U)\}\]

Evaluates to \texttt{true} if the specified input queue is full.
Parameters

\[ \text{in } iqp \ \text{pointer to an input_queue_t structure} \]

Returns

The queue status.

Return values

- **false** if the queue is not full.
- **true** if the queue is full.

Function Class:

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.37.2.14 iqGet

```c
#define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)
```

Input queue read.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a byte arrives in the queue.

Parameters

\[ \text{in } iqp \ \text{pointer to an input_queue_t structure} \]

Returns

A byte value from the queue.

Return values

- **MSG_RESET** if the queue has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
### 7.37.2.15 oqGetFullI

```c
#define oqGetFullI(oqp) (qSizeX(oqp) - qSpaceI(oqp))
```

Returns the filled space into an output queue.

**Parameters**

- `in oqp` pointer to an `output_queue_t` structure

**Returns**

The number of full bytes in the queue.

**Return values**

- `0` if the queue is empty.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.37.2.16 oqGetEmptyI

```c
#define oqGetEmptyI(oqp) qSpaceI(oqp)
```

Returns the empty space into an output queue.

**Parameters**

- `in oqp` pointer to an `output_queue_t` structure

**Returns**

The number of empty bytes in the queue.

**Return values**

- `0` if the queue is full.
Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.37.2.17 oqIsEmptyI

```c
#define oqIsEmptyI(
   oqp ) ((bool)(((oqp)->q_wrptr == (oqp)->q_rdptr) && ((oqp)->q_counter != 0U))
```

**Value:**

Evaluates to **true** if the specified output queue is empty.

**Parameters**

| in | oqp | pointer to an output_queue_t structure |

**Returns**

The queue status.

**Return values**

| false | if the queue is not empty. |
| true | if the queue is empty. |

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.37.2.18 oqIsFullI

```c
#define oqIsFullI(
   oqp ) ((bool)(qSpaceI(oqp) == 0U))
```

**Evaluates to** **true** if the specified output queue is full.

**Parameters**

| in | oqp | pointer to an output_queue_t structure |

ChibiOS/HAL
Returns

The queue status.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if the queue is not full.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if the queue is full.</td>
</tr>
</tbody>
</table>

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.37.2.19 oqPut

```c
#define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)
```

**Output queue write.**

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>oqp</th>
<th>pointer to an output_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written in the queue</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the queue has been reset.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.37.3 Typedef Documentation
7.37 I/O Bytes Queues

7.37.1 io_queue_t

typedef struct io_queue io_queue_t

Type of a generic I/O queue structure.

7.37.2 qnotify_t

typedef void(* qnotify_t) (io_queue_t *qp)

Queue notification callback type.

Parameters

| in  | qp | the queue pointer |

7.37.3 input_queue_t

typedef io_queue_t input_queue_t

Type of an input queue structure.

This structure represents a generic asymmetrical input queue. Writing to the queue is non-blocking and can be performed from interrupt handlers or from within a kernel lock zone. Reading the queue can be a blocking operation and is supposed to be performed by a system thread.

7.37.4 output_queue_t

typedef io_queue_t output_queue_t

Type of an output queue structure.

This structure represents a generic asymmetrical output queue. Reading from the queue is non-blocking and can be performed from interrupt handlers or from within a kernel lock zone. Writing the queue can be a blocking operation and is supposed to be performed by a system thread.

7.37.4 Function Documentation

7.37.4.1 iq_read()

static size_t iq_read (  
    input_queue_t *iqp,  
    uint8_t *bp,  
    size_t n )  
    [static]

Non-blocking input queue read.

The function reads data from an input queue into a buffer. The operation completes when the specified amount of data has been transferred or when the input queue has been emptied.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>iqp</th>
<th>pointer to an input_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.

Function Class:
Not an API, this function is for internal use only.

7.37.4.2 oq_write()

static size_t oq_write {
    output_queue_t * oqp,
    const uint8_t * bp,
    size_t n ) [static]

Non-blocking output queue write.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or when the output queue has been filled.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>oqp</th>
<th>pointer to an output_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.

Function Class:
Not an API, this function is for internal use only.

7.37.4.3 iqObjectInit()

void iqObjectInit {
    input_queue_t * iqp,
Initializes an input queue.

A Semaphore is internally initialized and works as a counter of the bytes contained in the queue.

Note

The callback is invoked from within the S-Locked system state.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>iq</th>
<th>pointer to an input_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to a memory area allocated as queue buffer</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of the queue buffer</td>
</tr>
<tr>
<td>in</td>
<td>infy</td>
<td>pointer to a callback function that is invoked when data is read from the queue. The value can be NULL.</td>
</tr>
<tr>
<td>in</td>
<td>link</td>
<td>application defined pointer</td>
</tr>
</tbody>
</table>

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
| iqObjectInit | osalThreadQueueObjectInit |
```

7.37.4.4 iqResetI()

void iqResetI (
    input_queue_t * iqp)

Resets an input queue.

All the data in the input queue is erased and lost, any waiting thread is resumed with status MSG_RESET.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.
Parameters

| in  | iqp | pointer to an input_queue_t structure |

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

![Call Graph](chart)

### 7.37.4.5 iqPutI()

```c
msg_t iqPutI (input_queue_t *iqp, uint8_t b)
```

Input queue write.

A byte value is written into the low end of an input queue. The operation completes immediately.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>iqp</th>
<th>pointer to an input_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written in the queue</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the operation has been completed with success.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the queue is full.</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

![Call Graph]

7.37.4.6 iqGetI()

```c
msg_t iqGetI ( 
    input_queue_t * iqp )
```

Input queue non-blocking read.

This function reads a byte value from an input queue. The operation completes immediately.

Note

The callback is invoked after removing a character from the queue.

Parameters

| in  | iqp          | pointer to an input_queue_t structure |

Returns

A byte value from the queue.

Return values

| MSG_TIMEOUT | if the queue is empty. |
| MSG_RESET   | if the queue has been reset. |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
7.37.4.7 iqGetTimeout()

```c
msg_t iqGetTimeout ( 
    input_queue_t * iqp, 
    sysinterval_t timeout )
```

Input queue read with timeout.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a byte arrives in the queue or a timeout occurs.

**Note**

The callback is invoked after removing a character from the queue.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>iqp</th>
<th>pointer to an <code>input_queue_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

A byte value from the queue.

**Return values**

<table>
<thead>
<tr>
<th>MSG_TIMEOUT</th>
<th>if the specified time expired.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the queue has been reset.</td>
</tr>
</tbody>
</table>

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.37.4.8 iqReadI()

size_t iqReadI (  
    input_queue_t * iqp,  
    uint8_t * bp,  
    size_t n )

Input queue non-blocking read.

The function reads data from an input queue into a buffer. The operation completes immediately.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in iqp</td>
<td>pointer to an input_queue_t structure</td>
</tr>
<tr>
<td>out bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

![Call graph for iqReadI and iq_read]

7.37.4.9 iqReadTimeout()

size_t iqReadTimeout (  
    input_queue_t * iqp,  
    uint8_t * bp,  
    size_t n,  
    sysinterval_t timeout )

Input queue read with timeout.

The function reads data from an input queue into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

Note

The function is not atomic, if you need atomicity it is suggested to use a semaphore or a mutex for mutual exclusion.

The callback is invoked after removing each character from the queue.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>iap</th>
<th>pointer to an input_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

### Returns

The number of bytes effectively transferred.

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.37.4.10 oqObjectInit()

```c
void oqObjectInit ( 
    output_queue_t * oqp, 
    uint8_t * bp, 
    size_t size, 
    qnotify_t onfy, 
    void * link )
```

Initializes an output queue.

A Semaphore is internally initialized and works as a counter of the free bytes in the queue.

**Note**

The callback is invoked from within the S-Locked system state.
7.37 I/O Bytes Queues

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td><strong>oqp</strong></td>
<td>pointer to an output_queue_t structure</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>bp</strong></td>
<td>pointer to a memory area allocated as queue buffer</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>size</strong></td>
<td>size of the queue buffer</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>only</strong></td>
<td>pointer to a callback function that is invoked when data is written to the queue. The value can be NULL.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>link</strong></td>
<td>application defined pointer</td>
</tr>
</tbody>
</table>

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
  oqObjectInit  osalThreadQueueObjectInit
```

7.37.4.11 oqResetI()

```c
void oqResetI ( output_queue_t * oqp )
```

Resets an output queue.

All the data in the output queue is erased and lost, any waiting thread is resumed with status MSG_RESET.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

Parameters

|   | **oqp** | pointer to an output_queue_t structure |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
Here is the call graph for this function:

```
oqResetI → osalThreadDequeueAllI
```

### 7.37.4.12 oqPutI()

```c
msg_t oqPutI (output_queue_t *oqp, uint8_t b )
```

Output queue non-blocking write.

This function writes a byte value to an output queue. The operation completes immediately.

**Parameters**

| in | oqp | pointer to an output_queue_t structure |
| in | b   | the byte value to be written in the queue |

**Returns**

The operation status.

**Return values**

| MSG_OK | if the operation succeeded. |
| MSG_TIMEOUT | if the queue is full. |
| MSG_RESET | if the queue has been reset. |

**Function Class:**

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.37.4.13 oqPutTimeout()

```c
msg_t oqPutTimeout (output_queue_t *oqp, 
```

Output queue write with timeout.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue or a timeout occurs.

Note

The callback is invoked after putting the character into the queue.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>oqp</th>
<th>pointer to an output_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written in the queue</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the specified time expired.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the queue has been reset.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.37.4.14  oqGetI()

```c
msg_t oqGetI (output_queue_t *oqp)
```

Output queue read.

A byte value is read from the low end of an output queue. The operation completes immediately.

**Parameters**

| in  | oqp | pointer to an output_queue_t structure |

**Returns**

The byte value from the queue.

**Return values**

- **MSG_TIMEOUT** if the queue is empty.

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
  oqGetI ® osalThreadDequeueNextI
```

7.37.4.15  oqWriteI()

```c
size_t oqWriteI (output_queue_t *oqp, const uint8_t *bp, size_t n )
```

Output queue non-blocking write.

The function writes data from a buffer to an output queue. The operation completes immediately.
7.37 I/O Bytes Queues

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>oqp</th>
<th>pointer to an output_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
    oqWriteI oq_write
```

7.37.4.16  oqWriteTimeout()

```
size_t oqWriteTimeout (
    output_queue_t * oqp,
    const uint8_t * bp,
    size_t n,
    sysinterval_t timeout )
```

Output queue write with timeout.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

Note

The function is not atomic, if you need atomicity it is suggested to use a semaphore or a mutex for mutual exclusion.

The callback is invoked after putting each character into the queue.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in oqp</td>
<td>pointer to an output_queue_t structure</td>
</tr>
<tr>
<td>in bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
<tr>
<td>in timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.38 RTC Driver

Generic RTC Driver.

7.38.1 Detailed Description

Generic RTC Driver.

This module defines an abstract interface for a Real Time Clock Peripheral.

Precondition

In order to use the RTC driver the `HAL_USE_RTC` option must be enabled in `halconf.h`.

Macros

- `#define RTC_BASE_YEAR 1980U`
  
  Base year of the calendar.

- `#define _rtc_driver_methods_base_pers_storage_methods` BasePersistentStorage specific methods.

- `#define rtc_lld_driver_fields uint32_t dummy`
  Implementation-specific `RTC_DRIVER` fields.

Date/Time bit masks for FAT format

- `#define RTC_FAT_TIME_SECONDS_MASK 0x0000001FU`
- `#define RTC_FAT_TIME_MINUTES_MASK 0x000007E0U`
- `#define RTC_FAT_TIME_HOURS_MASK 0x0000F800U`
- `#define RTC_FAT_DATE_DAYS_MASK 0x001F0000U`
- `#define RTC_FAT_DATE_MONTHS_MASK 0x01E00000U`
- `#define RTC_FAT_DATE_YEARS_MASK 0xFE000000U`

Day of week encoding

- `#define RTC_DAY_CUTURDAY 0U`
- `#define RTC_DAY_MONDAY 1U`
- `#define RTC_DAY_TUESDAY 2U`
- `#define RTC_DAY_WEDNESDAY 3U`
- `#define RTC_DAY_THURSDAY 4U`
- `#define RTC_DAY_FRIDAY 5U`
- `#define RTC_DAY_SATURDAY 6U`
- `#define RTC_DAY_SUNDAY 7U`

Implementation capabilities

- `#define RTC_SUPPORTS_CALLBACKS TRUE`
  Callback support in the driver.

- `#define RTC_ALARMS 2`
  Number of alarms available.

- `#define RTC_HAS_STORAGE FALSE`
  Presence of a local persistent storage.
**PLATFORM configuration options**

- `#define PLATFORM_RTC_USE_RTC1 FALSE`
  
  RTCD1 driver enable switch.

**Typedefs**

- `typedef struct RTCDriver RTCDriver`
  
  Type of a structure representing an RTC driver.

- `typedef unsigned int rtcalarm_t`
  
  Type of an RTC alarm number.

- `typedef void(*rtccb_t)(RTCDriver *rtcp, rtcevent_t event)`
  
  Type of a generic RTC callback.

**Data Structures**

- `struct RTCDateTime`
  
  Type of a structure representing an RTC date/time stamp.

- `struct RTCDriverVMT`
  
  RTCDriver virtual methods table.

- `struct RTCDriver`
  
  Structure representing an RTC driver.

- `struct RTCAlarm`
  
  Type of a structure representing an RTC alarm time stamp.

**Functions**

- `void rtcInit (void)`
  
  RTC Driver initialization.

- `void rtcObjectInit (RTCDriver *rtcp)`
  
  Initializes a generic RTC driver object.

- `void rtcSetTime (RTCDriver *rtcp, const RTCDateTime *timespec)`
  
  Set current time.

- `void rtcGetTime (RTCDriver *rtcp, RTCDateTime *timespec)`
  
  Get current time.

- `void rtcSetAlarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)`
  
  Set alarm time.

- `void rtcGetAlarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)`
  
  Get current alarm.

- `void rtcSetCallback (RTCDriver *rtcp, rtccb_t callback)`
  
  Enables or disables RTC callbacks.

- `void rtcConvertDateTimeToStructTm (const RTCDateTime *timespec, struct tm *timp, uint32_t *tv_msec)`
  
  Convert RTCDateTime to broken-down time structure.

- `void rtcConvertStructTmToDateTime (const struct tm *timp, uint32_t tv_msec, RTCDateTime *timespec)`
  
  Convert broken-down time structure to RTCDateTime.

- `uint32_t rtcConvertDateTimeToFAT (const RTCDateTime *timespec)`
  
  Get current time in format suitable for usage in FAT file system.

- `void rtc_lld_init (void)`
  
  RTC driver identifier.
7.38 RTC Driver

- `void rtc_lld_set_time (RTCDriver *rtcp, const RTCDateTime *timespec)`
  
  Set current time.

- `void rtc_lld_get_time (RTCDriver *rtcp, RTCDateTime *timespec)`
  
  Get current time.

- `void rtc_lld_set_alarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)`
  
  Set alarm time.

- `void rtc_lld_get_alarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)`
  
  Get alarm time.

Enumerations

- `enum rtcevent_t`
  
  Type of an RTC event.

7.38.2 Macro Definition Documentation

7.38.2.1 RTC_BASE_YEAR

#define RTC_BASE_YEAR 1980U

Base year of the calendar.

7.38.2.2 _rtc_driver_methods

#define _rtc_driver_methods _base_pers_storage_methods

BasePersistentStorage specific methods.

7.38.2.3 RTC_SUPPORTS_CALLBACKS

#define RTC_SUPPORTS_CALLBACKS TRUE

Callback support int the driver.

7.38.2.4 RTC_ALARMS

#define RTC_ALARMS 2

Number of alarms available.
7.38.2.5 RTC_HAS_STORAGE

#define RTC_HAS_STORAGE FALSE

Presence of a local persistent storage.

7.38.2.6 PLATFORM_RTC_USE_RTC1

#define PLATFORM_RTC_USE_RTC1 FALSE

RTCD1 driver enable switch.
If set to TRUE the support for RTC1 is included.

Note
The default is FALSE.

7.38.2.7 rtc_lld_driver_fields

#define rtc_lld_driver_fields uint32_t dummy

Implementation-specific RTCDriver fields.

7.38.3 Typedef Documentation

7.38.3.1 RTCDriver

typedef struct RTCDriver RTCDriver

Type of a structure representing an RTC driver.

7.38.3.2 rtcalarm_t

typedef unsigned int rtcalarm_t

Type of an RTC alarm number.
7.38.3.3 rtcb_t

typedef void(* rtcb_t) (RTCDriver *rtcp, rtcevent_t event)
Type of a generic RTC callback.

7.38.4 Enumeration Type Documentation

7.38.4.1 rtcevent_t

enum rtcevent_t
Type of an RTC event.

7.38.5 Function Documentation

7.38.5.1 rtcInit()

void rtcInit ( 
    void 
)
RTC Driver initialization.

Note
This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.

Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
rctInit --> rtc_lld_init --> rtcObjectInit
```

7.38.5.2 rtcObjectInit()

void rtcObjectInit ( 
    RTCDriver * rtcp 
)
Initializes a generic RTC driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.
Parameters

| out  | rtcp | pointer to RTC driver structure |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

---

### 7.38.5.3 rtcSetTime()

```c
void rtcSetTime (  
    RTCDriver * rtcp,  
    const RTCDateTime * timespec )
```

Set current time.

**Note**

This function can be called from any context but limitations could be imposed by the low level implementation. It is guaranteed that the function can be called from thread context.

The function can be reentrant or not reentrant depending on the low level implementation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timespec</td>
<td>pointer to a RTCDateTime structure</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

[Call Graph Image]

ChibiOS/HAL
7.38.5.4 rtcGetTime()

```c
void rtcGetTime ( 
    RTCDriver * rtcp,
    RTCDateTime * timespec )
```

Get current time.

**Note**

This function can be called from any context but limitations could be imposed by the low level implementation. It is guaranteed that the function can be called from thread context.

The function can be reentrant or not reentrant depending on the low level implementation.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>timespec</td>
<td>pointer to a RTCDateTime structure</td>
</tr>
</tbody>
</table>

**Function Class:**

Special function, this function has special requirements see the notes.

**Here is the call graph for this function:**

```
rtcGetTime  ->  rtc_lld_get_time
```

7.38.5.5 rtcSetAlarm()

```c
void rtcSetAlarm ( 
    RTCDriver * rtcp,
    rtcalarm_t alarm,
    const RTCAlarm * alaramspec )
```

Set alarm time.

**Note**

This function can be called from any context but limitations could be imposed by the low level implementation. It is guaranteed that the function can be called from thread context.

The function can be reentrant or not reentrant depending on the low level implementation.
### rtcGetAlarm()

```c
void rtcGetAlarm(
    RTCDriver * rtcp,
    rtcalarm_t alarm,
    RTCAlarm * alarmspec
)
```

Get current alarm.

**Note**

If an alarm has not been set then the returned alarm specification is not meaningful.

This function can be called from any context but limitations could be imposed by the low level implementation. It is guaranteed that the function can be called from thread context.

The function can be reentrant or not reentrant depending on the low level implementation.
Here is the call graph for this function:

![Call Graph Diagram]

### 7.38.5.7 rtcSetCallback()

```c
void rtcSetCallback ( 
    RTCDriver ∗ rtcp, 
    rtccb_t callback )
```

Enables or disables RTC callbacks.

This function enables or disables the callback, use a NULL pointer in order to disable it.

**Note**

- This function can be called from any context but limitations could be imposed by the low level implementation.
- It is guaranteed that the function can be called from thread context.
- The function can be reentrant or not reentrant depending on the low level implementation.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>callback</td>
<td>callback function pointer or NULL</td>
</tr>
</tbody>
</table>

**Function Class:**

Special function, this function has special requirements see the notes.

### 7.38.5.8 rtcConvertDateTimeToStructTm()

```c
void rtcConvertDateTimeToStructTm ( 
    const RTCDateTime ∗ timespec, 
    struct tm ∗ timp, 
    uint32_t ∗ tv_msec )
```

Convert RTCDateTime to broken-down time structure.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>timespec</th>
<th>pointer to a RTCDateTime structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>timp</td>
<td>pointer to a broken-down time structure</td>
</tr>
<tr>
<td>out</td>
<td>tv_msec</td>
<td>pointer to milliseconds value or NULL</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.38.5.9 rtcConvertStructTmToDateTime()

```c
void rtcConvertStructTmToDateTime (
    const struct tm * timp,
    uint32_t tv_msec,
    RTCDateTime * timespec )
```

Convert broken-down time structure to RTCDateTime.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>timp</th>
<th>pointer to a broken-down time structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tv_msec</td>
<td>milliseconds value</td>
</tr>
<tr>
<td>out</td>
<td>timespec</td>
<td>pointer to a RTCDateTime structure</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.38.5.10 rtcConvertDateTimeToFAT()

```c
uint32_t rtcConvertDateTimeToFAT ( 
    const RTCDateTime * timespec )
```

Get current time in format suitable for usage in FAT file system.

Note
The information about day of week and DST is lost in DOS format, the second field loses its least significant bit.

Parameters

| out   | timespec | pointer to a RTCDateTime structure |
Returns

FAT date/time value.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.38.5.11 rtc_lld_init()

void rtc_lld_init ( void )

RTC driver identifier.
Enable access to registers.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
rtc_lld_init  rtcObjectInit
```

7.38.5.12 rtc_lld_set_time()

void rtc_lld_set_time ( RTCDriver * rtcp,
                       const RTCDateTime * timespec )

Set current time.

Note

Fractional part will be silently ignored. There is no possibility to set it on PLATFORM platform.
The function can be called from any context.
**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timespec</td>
<td>pointer to a RTCDateTime structure</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

### 7.38.5.13 rtc_lld_get_time()

```c
void rtc_lld_get_time ( 
    RTCDriver * rtcp,
    RTCDateTime * timespec )
```

Get current time.

**Note**

The function can be called from any context.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>timespec</td>
<td>pointer to a RTCDateTime structure</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

### 7.38.5.14 rtc_lld_set_alarm()

```c
void rtc_lld_set_alarm ( 
    RTCDriver * rtcp,
    rtcalarm_t alarm,
    const RTCAlarm * alarmspec )
```

Set alarm time.

**Note**

Default value after BKP domain reset for both comparators is 0.

Function does not performs any checks of alarm time validity.

The function can be called from any context.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>rtcp</th>
<th>pointer to RTC driver structure.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>alarm</td>
<td>alarm identifier. Can be 1 or 2.</td>
</tr>
<tr>
<td></td>
<td>alarmspec</td>
<td>pointer to a RTCAlarm structure.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.38.5.15 rtc_lld_get_alarm()

```c
void rtc_lld_get_alarm ( 
    RTCDriver * rtcp,
    rtcalarm_t alarm,
    RTCAlarm * alarmspec )
```

Get alarm time.

Note
The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>alarm</td>
<td>alarm identifier</td>
</tr>
<tr>
<td></td>
<td>alarmspec</td>
<td>pointer to a RTCAlarm structure</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.
7.39 SDC Driver

Generic SD Card Driver.

7.39.1 Detailed Description

Generic SD Card Driver.

This module implements a generic SDC (Secure Digital Card) driver.

Precondition

In order to use the SDC driver the \texttt{HAL\_USE\_SDC} option must be enabled in \texttt{halconf.h}.

7.39.2 Driver State Machine

This driver implements a state machine internally, see the Abstract I/O Block Device module documentation for details.

7.39.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Macros

- \texttt{\#define \_sdc\_driver\_methods \_mmcsd\_block\_device\_methods}
  
  \texttt{SDCDriver} specific methods.

SD card types

- \texttt{\#define SDC\_MODE\_CARDTYPE\_MASK 0xFU}
- \texttt{\#define SDC\_MODE\_CARDTYPE\_SDV11 0U}
- \texttt{\#define SDC\_MODE\_CARDTYPE\_SDV20 1U}
- \texttt{\#define SDC\_MODE\_CARDTYPE\_MMC 2U}
- \texttt{\#define SDC\_MODE\_HIGH\_CAPACITY 0x10U}

SDC bus error conditions

- \texttt{\#define SDC\_NO\_ERROR 0U}
- \texttt{\#define SDC\_CMD\_CRC\_ERROR 1U}
- \texttt{\#define SDC\_DATA\_CRC\_ERROR 2U}
- \texttt{\#define SDC\_DATA\_TIMEOUT 4U}
- \texttt{\#define SDC\_COMMAND\_TIMEOUT 8U}
- \texttt{\#define SDC\_TX\_UNDERRUN 16U}
- \texttt{\#define SDC\_RX\_OVERRUN 32U}
- \texttt{\#define SDC\_STARTBIT\_ERROR 64U}
- \texttt{\#define SDC\_OVERFLOW\_ERROR 128U}
- \texttt{\#define SDC\_UNHANDLED\_ERROR 0xFFFFFFFFU}
SDC configuration options

- `#define SDC_INIT_RETRY 100`
  Number of initialization attempts before rejecting the card.
- `#define SDC_MMC_SUPPORT FALSE`
  Include support for MMC cards.
- `#define SDC_NICE_WAITING TRUE`
  Delays insertions.
- `#define SDC_INIT_OCR_V20 0x50FF8000U`
  OCR initialization constant for V20 cards.
- `#define SDC_INIT_OCR 0x80100000U`
  OCR initialization constant for non-V20 cards.

Macro Functions

- `#define sdcIsCardInserted(sdcp) (sdc_lld_is_card_inserted(sdcp))`
  Returns the card insertion status.
- `#define sdcIsWriteProtected(sdcp) (sdc_lld_is_write_protected(sdcp))`
  Returns the write protect status.

PLATFORM configuration options

- `#define PLATFORM_SDC_USE_SDC1 FALSE`
  PWMD1 driver enable switch.

Typedefs

- `typedef uint32_t sdcmode_t`
  Type of card flags.
- `typedef uint32_t sdcflags_t`
  SDC Driver condition flags type.
- `typedef struct SDCDriver SDCDriver`
  Type of a structure representing an SDC driver.

Data Structures

- `struct SDCCConfig`
  Driver configuration structure.
- `struct SDCDriverVMT`
  SDCDriver virtual methods table.
- `struct SDCDriver`
  Structure representing an SDC driver.
Functions

- **static bool mode_detect (SDCDriver *sdcp)**
  Detects card mode.

- **static bool mmc_init (SDCDriver *sdcp)**
  Init procedure for MMC.

- **static bool sdc_init (SDCDriver *sdcp)**
  Init procedure for SDC.

- **static uint32_t mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)**
  Constructs CMD6 argument for MMC.

- **static uint32_t sdc_cmd6_construct (sd_switch_t mode, sd_switch_function_t function, uint32_t value)**
  Constructs CMD6 argument for SDC.

- **static uint16_t sdc_cmd6_extract_info (sd_switch_function_t function, const uint8_t *buf)**
  Extracts information from CMD6 answer.

- **static bool sdc_cmd6_check_status (sd_switch_function_t function, const uint8_t *buf)**
  Checks status after switching using CMD6.

- **static bool sdc_detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)**
  Reads supported bus clock and switch SDC to appropriate mode.

- **static bool mmc_detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)**
  Reads supported bus clock and switch MMC to appropriate mode.

- **static bool detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)**
  Reads supported bus clock and switch card to appropriate mode.

- **static bool sdc_set_bus_width (SDCDriver *sdcp)**
  Sets bus width for SDC.

- **static bool mmc_set_bus_width (SDCDriver *sdcp)**
  Sets bus width for MMC.

- **bool _sdc_wait_for_transfer_state (SDCDriver *sdcp)**
  Wait for the card to complete pending operations.

- **void sdcInit (void)**
  SDC Driver initialization.

- **void sdcObjectInit (SDCDriver *sdcp)**
  Initializes the standard part of a SDCDriver structure.

- **msg_t sdcStart (SDCDriver *sdcp, const SDCConfig *config)**
  Configures and activates the SDC peripheral.

- **void sdcStop (SDCDriver *sdcp)**
  Deactivates the SDC peripheral.

- **bool sdcConnect (SDCDriver *sdcp)**
  Performs the initialization procedure on the inserted card.

- **bool sdcDisconnect (SDCDriver *sdcp)**
  Brings the driver in a state safe for card removal.

- **bool sdcRead (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)**
  Reads one or more blocks.

- **bool sdcWrite (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)**
  Writes one or more blocks.

- **sdcflags_t sdcGetAndClearErrors (SDCDriver *sdcp)**
  Returns the errors mask associated to the previous operation.

- **bool sdcSync (SDCDriver *sdcp)**
  Waits for card idle condition.

- **bool sdcGetInfo (SDCDriver *sdcp, BlockDeviceInfo *bdip)**
  Returns the media info.

- **bool sdcErase (SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)**
Erases the supplied blocks.

• void sdc_lld_init (void)
  Low level SDC driver initialization.

• void sdc_lld_start (SDCDriver *sdcp)
  Configures and activates the SDC peripheral.

• void sdc_lld_stop (SDCDriver *sdcp)
  Deactivates the SDC peripheral.

• void sdc_lld_start_clk (SDCDriver *sdcp)
  Starts the SDIO clock and sets it to init mode (400kHz or less).

• void sdc_lld_set_data_clk (SDCDriver *sdcp, sdcbusclk_t clk)
  Sets the SDIO clock to data mode (25MHz or less).

• void sdc_lld_stop_clk (SDCDriver *sdcp)
  Stops the SDIO clock.

• void sdc_lld_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)
  Switches the bus to 4 bits mode.

• void sdc_lld_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)
  Sends an SDIO command with no response expected.

• bool sdc_lld_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)
  Sends an SDIO command with a short response expected.

• bool sdc_lld_send_cmd_short_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)
  Sends an SDIO command with a short response expected and CRC.

• bool sdc_lld_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)
  Sends an SDIO command with a long response expected and CRC.

• bool sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)
  Reads one or more blocks.

• bool sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)
  Writes one or more blocks.

• bool sdc_lld_sync (SDCDriver *sdcp)
  Waits for card idle condition.

Enumerations

• enum mmc_switch_t
  MMC switch mode.

• enum sd_switch_t
  SDC switch mode.

• enum sd_switch_function_t
  SDC switch function.

• enum sdcbusmode_t
  Type of SDIO bus mode.

• enum sdcbusclk_t
  Max supported clock.

Variables

• static const struct SDCDriverVMT sdc_vmt
  Virtual methods table.

• SDCDriver SDCD1
  SDCD1 driver identifier.
7.39.4  Macro Definition Documentation

7.39.4.1  SDC_INIT_RETRY

#define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.39.4.2  SDC_MMC_SUPPORT

#define SDC_MMC_SUPPORT FALSE

Include support for MMC cards.

Note

MMC support is not yet implemented so this option must be kept at FALSE.

7.39.4.3  SDC_NICE_WAITING

#define SDC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

7.39.4.4  SDC_INIT_OCR_V20

#define SDC_INIT_OCR_V20 0x50FF8000U

OCR initialization constant for V20 cards.
7.39.4.5 SDC_INIT_OCR

#define SDC_INIT_OCR 0x80100000U

OCR initialization constant for non-V20 cards.

7.39.4.6 sdcIsCardInserted

#define sdcIsCardInserted(sdp) (sdc_lld_is_card_inserted(sdp))

Returns the card insertion status.

Note

This macro wraps a low level function named `sdc_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.
Parameters

| in | sdc | pointer to the SDCDriver object |

Returns

The card state.

Return values

| false | card not inserted. |
| true  | card inserted. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.4.7 sdcIsWriteProtected

```
#define sdcIsWriteProtected(
    sdc
) (sdc_lld_is_write_protected(sdc))
```

Returns the write protect status.

Note

This macro wraps a low level function named sdc_lld_is_write_protected(), this function must be provided by the application because it is not part of the SDC driver.

Parameters

| in | sdc | pointer to the SDCDriver object |

Returns

The card state.

Return values

| false | not write protected. |
| true  | write protected. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.39.4.8 PLATFORM_SDC_USE_SDC1

#define PLATFORM_SDC_USE_SDC1 FALSE

PWMD1 driver enable switch.

If set to TRUE the support for PWM1 is included.

Note

The default is FALSE.

7.39.4.9 _sdc_driver_methods

#define _sdc_driver_methods _mmcsd_block_device_methods

SDCDriver specific methods.

7.39.5 Typedef Documentation

7.39.5.1 sdcmode_t

typedef uint32_t sdcmode_t

Type of card flags.

7.39.5.2 sdcflags_t

typedef uint32_t sdcflags_t

SDC Driver condition flags type.

7.39.5.3 SDCDriver

typedef struct SDCDriver SDCDriver

Type of a structure representing an SDC driver.
7.39.6 Enumeration Type Documentation

7.39.6.1 mmc_switch_t

enum mmc_switch_t
MMC switch mode.

7.39.6.2 sd_switch_t

enum sd_switch_t
SDC switch mode.

7.39.6.3 sd_switch_function_t

enum sd_switch_function_t
SDC switch function.

7.39.6.4 sdcbusmode_t

enum sdcbusmode_t
Type of SDIO bus mode.

7.39.6.5 sdcbusclk_t

enum sdcbusclk_t
Max supported clock.

7.39.7 Function Documentation

7.39.7.1 mode_detect()

static bool mode_detect ( 
    SDCDriver * sdcp ) [static]

Detects card mode.
Parameters

| in | sdp | pointer to the SDCDriver object |

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
mode_detect        sdc_lld_send_cmd_short_crc
```

### 7.39.2.2 mmc_init()

```c
static bool mmc_init (  
    SDCDriver * sdp) [static]
```

Init procedure for MMC.

Parameters

| in | sdp | pointer to the SDCDriver object |

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>
7.39.7.3  sdc_init()

static bool sdc_init ( 
    SDCDriver ∗ sdcp ) [static]

Init procedure for SDC.

Parameters

in  sdcp  pointer to the SDCDriver object

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.39.7.4  mmc_cmd6_construct()

static uint32_t mmc_cmd6_construct ( 
    mmc_switch_t access, 
    uint32_t idx, 
    uint32_t

ChibiOS/HAL
uchar_t value,
uchar_t cmd_set ) [static]

Constructs CMD6 argument for MMC.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>access</code></td>
<td>EXT_CSD access mode</td>
</tr>
<tr>
<td><code>idx</code></td>
<td>EXT_CSD byte number</td>
</tr>
<tr>
<td><code>value</code></td>
<td>value to be written in target field</td>
</tr>
<tr>
<td><code>cmd_set</code></td>
<td>switch current command set</td>
</tr>
</tbody>
</table>

### Returns

CMD6 argument.

### Function Class:

Not an API, this function is for internal use only.

---

#### 7.39.7.5 `sdc_cmd6_construct()`

```c
static uint32_t sdc_cmd6_construct (  
  sd_switch_t mode,
  sd_switch_function_t function,
  uint32_t value ) [static]
```

Constructs CMD6 argument for SDC.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mode</code></td>
<td>switch/test mode</td>
</tr>
<tr>
<td><code>function</code></td>
<td>function number to be switched</td>
</tr>
<tr>
<td><code>value</code></td>
<td>value to be written in target function</td>
</tr>
</tbody>
</table>

### Returns

CMD6 argument.

### Function Class:

Not an API, this function is for internal use only.

---

#### 7.39.7.6 `sdc_cmd6_extract_info()`

```c
static uint16_t sdc_cmd6_extract_info (  
  sd_switch_function_t function,
  const uint8_t * buf ) [static]
```

Extracts information from CMD6 answer.
Parameters

| in | function | function number to be switched |
| in | buf      | buffer with answer |

Returns

extracted answer.

Function Class:

Not an API, this function is for internal use only.

7.39.7.7  sdc_cmd6_check_status()

```c
static bool sdc_cmd6_check_status (  
    sd_switch_function_t function,  
    const uint8_t * buf ) [static]
```

Checks status after switching using CMD6.

Parameters

| in | function | function number to be switched |
| in | buf      | buffer with answer |

Returns

The operation status.

Return values

| HAL_SUCCESS | operation succeeded. |
| HAL_FAILED  | operation failed.    |

Function Class:

Not an API, this function is for internal use only.

7.39.7.8  sdc_detect_bus_clk()

```c
static bool sdc_detect_bus_clk (  
    SDCDriver * sdcp,  
    sdcbusclk_t * clk ) [static]
```

Reads supported bus clock and switch SDC to appropriate mode.

ChibiOS/HAL
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>clk</td>
<td>pointer to clock enum</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.39.7.9 mmc_detect_bus_clk()

static bool mmc_detect_bus_clk (  
    SDCDriver * sdc,  
    sdcbusclk_t * clk ) [static]

Reads supported bus clock and switch MMC to appropriate mode.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>clk</td>
<td>pointer to clock enum</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.39.7.10  **detect_bus_clk()**

```c
static bool detect_bus_clk (  
    SDCDriver * sdc,  
    sdcbusclk_t * clk ) [static]
```

Reads supported bus clock and switch card to appropriate mode.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in sdc</td>
<td>pointer to the SCDriver object</td>
</tr>
<tr>
<td>out clk</td>
<td>pointer to clock enum</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

**Function Class:**

Not an API, this function is for internal use only.

7.39.7.11  **sdc_set_bus_width()**

```c
static bool sdc_set_bus_width (  
    SDCDriver * sdc ) [static]
```

Sets bus width for SDC.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in sdc</td>
<td>pointer to the SCDriver object</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>
7.39.7.12 mmc_set_bus_width()

static bool mmc_set_bus_width ( SDCDriver * sdcp ) [static]

Sets bus width for MMC.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdcp</td>
<td>pointer to the SDCDriver object</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

- **HAL_SUCCESS**: operation succeeded.
- **HAL_FAILED**: operation failed.

Here is the call graph for this function:

![Call graph for mmc_set_bus_width](image)

7.39.7.13 _sdc_wait_for_transfer_state()

bool _sdc_wait_for_transfer_state ( SDCDriver * sdcp )

Wait for the card to complete pending operations.
Parameters

| in | sdcp | pointer to the SDCDriver object |

Returns

The operation status.

Return values

| HAL_SUCCESS | operation succeeded. |
| HAL_FAILED  | operation failed. |

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
_sdc_wait_for_transfer
  _state
    sdc_lld_send_cmd_short_crc
```

### 7.39.7.14 sdcInit()

```c
void sdcInit (  
    void  )
```

SDC Driver initialization.

**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
    sdcInit → sdc_lld_init → sdcObjectInit
```

### 7.39.7.15 sdcObjectInit()

```c
void sdcObjectInit ( SDCDriver * sdcp )
```

Initializes the standard part of a SDCDriver structure.

**Parameters**

- `sdcp` pointer to the SDCDriver object

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.39.7.16 sdcStart()

```c
msg_t sdcStart ( SDCDriver * sdcp,
                  const SDCConfig * config )
```

Configures and activates the SDC peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sdcp</code></td>
<td>pointer to the SDCDriver object</td>
</tr>
<tr>
<td><code>config</code></td>
<td>pointer to the SDCConfig object, can be NULL if the driver supports a default configuration or requires no configuration</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
sdcStart
   osalSysLock
   sdc_lld_start
```

### 7.39.7.17 sdcStop()

```c
void sdcStop ( SDCDriver * sdcp )
```

Deactivates the SDC peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
</table>

ChibiOS_HAL
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
sdcStop
    \arrow{osalSysLock}
    \arrow{osalSysUnlock}
    \arrow{sdc_lld_stop}
```

### 7.39.7.18  sdcConnect()

```c
bool sdcConnect (
    SDCDriver * sdcp )
```

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the BLK_READY state where it is possible to perform read and write operations.

**Parameters**

- **in sdcp** pointer to the SDCDriver object

**Returns**

- The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HAL_SUCCESS</strong></td>
<td>operation succeeded.</td>
</tr>
<tr>
<td><strong>HAL_FAILED</strong></td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.39.7.19  sdcDisconnect()

bool sdcDisconnect ( 
    SDCDriver * sdp )

Brings the driver in a state safe for card removal.

Parameters

  in sdp  pointer to the SDCDriver object

Returns

  The operation status.

Return values

<table>
<thead>
<tr>
<th>Operation</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>succeeded</td>
<td>HAL_SUCCESS</td>
</tr>
<tr>
<td>failed</td>
<td>HAL_FAILED</td>
</tr>
</tbody>
</table>

Function Class:

  Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.39.7.20  sdcRead()

bool sdcRead ( 
    SDCDriver * sdp, 
    uint32_t startblk, 
    uint8_t * buf, 
    uint32_t n )

Reads one or more blocks.
Precondition

The driver must be in the BLK_READY state after a successful sdcConnect() invocation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to read</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the read buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to read</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.7.21 sdcWrite()

bool sdcWrite (  
    SDCDriver * sdcp,  
    uint32_t startblk,  
    const uint8_t * buf,  
    uint32_t n )

Writes one or more blocks.

Precondition

The driver must be in the BLK_READY state after a successful sdcConnect() invocation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to write</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the write buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to write</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.7.22 sdcGetAndClearErrors()

sdcflags_t sdcGetAndClearErrors (
    SDCDriver * sdcp
)

Returns the errors mask associated to the previous operation.

Parameters

| in | sdcp | pointer to the SDCDriver object |

Returns

The errors mask.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
| sdcGetAndClearErrors  | osalSysLock |
```

7.39.7.23 sdcSync()

bool sdcSync (
    SDCDriver * sdcp
)

Waits for card idle condition.
Parameters

| in     | sdcp | pointer to the SDCDriver object |

Returns

The operation status.

Return values

| HAL_SUCCESS | the operation succeeded. |
| HAL_FAILED  | the operation failed.   |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
sdcSync → sdc_lld_sync
```

### 7.39.7.24 sdcGetInfo()

```c
bool sdcGetInfo ( 
    SDCDriver * sdcp,
    BlockDeviceInfo * bdip )
```

Returns the media info.

Parameters

| in     | sdcp | pointer to the SDCDriver object |
| out    | bdip | pointer to a BlockDeviceInfo structure |

Returns

The operation status.
Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{HAL_SUCCESS}</td>
<td>the operation succeeded.</td>
</tr>
<tr>
<td>\textit{HAL_FAILED}</td>
<td>the operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.7.25 \texttt{sdcErase()}

\begin{verbatim}
bool sdcErase (  
    SDCDriver * sdcp,  
    uint32_t startblk,  
    uint32_t endblk )
\end{verbatim}

Erases the supplied blocks.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{in}</td>
<td>\texttt{sdcp}</td>
</tr>
<tr>
<td>\textbf{in}</td>
<td>\texttt{startblk}</td>
</tr>
<tr>
<td>\textbf{in}</td>
<td>\texttt{endblk}</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{HAL_SUCCESS}</td>
<td>the operation succeeded.</td>
</tr>
<tr>
<td>\textit{HAL_FAILED}</td>
<td>the operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.7.26 \texttt{sdc\_lld\_init()}

\begin{verbatim}
void sdc\_lld\_init (  
    void )
\end{verbatim}

Low level SDC driver initialization.
Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
sdc_lld_init sdcObjectInit
```

### 7.39.7.27 sdc_lld_start()

```c
void sdc_lld_start (  
    SDCDriver * sdcp );
```

Configures and activates the SDC peripheral.

**Parameters**

| in | sdcp | pointer to the SDCDriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.39.7.28 sdc_lld_stop()

```c
void sdc_lld_stop (  
    SDCDriver * sdcp );
```

Deactivates the SDC peripheral.

**Parameters**

| in | sdcp | pointer to the SDCDriver object |

Function Class:

Not an API, this function is for internal use only.
7.39.7.29  sdc_lld_start_clk()

```c
void sdc_lld_start_clk ( SDCDriver * sdcp )
```

Starts the SDIO clock and sets it to init mode (400kHz or less).

**Parameters**

| in   | sdcp | pointer to the SDCDriver object |

**Function Class:**

Not an API, this function is for internal use only.

7.39.7.30  sdc_lld_set_data_clk()

```c
void sdc_lld_set_data_clk ( SDCDriver * sdcp,
                           sdcbusclk_t clk )
```

Sets the SDIO clock to data mode (25MHz or less).

**Parameters**

| in   | sdcp | pointer to the SDCDriver object |
| in   | clk  | the clock mode |

**Function Class:**

Not an API, this function is for internal use only.

7.39.7.31  sdc_lld_stop_clk()

```c
void sdc_lld_stop_clk ( SDCDriver * sdcp )
```

Stops the SDIO clock.

**Parameters**

| in   | sdcp | pointer to the SDCDriver object |
Function Class:
   Not an API, this function is for internal use only.

7.39.7.32  sdc_lld_set_bus_mode()

    void sdc_lld_set_bus_mode (  
        SDCDriver ∗ sdp,  
        sdcbusmode_t mode )

Switches the bus to 4 bits mode.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>bus mode</td>
</tr>
</tbody>
</table>

Function Class:
   Not an API, this function is for internal use only.

7.39.7.33  sdc_lld_send_cmd_none()

    void sdc_lld_send_cmd_none (  
        SDCDriver ∗ sdp,  
        uint8_t cmd,  
        uint32_t arg )

Sends an SDIO command with no response expected.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>card command</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>command argument</td>
</tr>
</tbody>
</table>

Function Class:
   Not an API, this function is for internal use only.

7.39.7.34  sdc_lld_send_cmd_short()

    bool sdc_lld_send_cmd_short (  
        SDCDriver ∗ sdp,  
        uint8_t cmd,  
        uint32_t arg )
uint8_t cmd,
uint32_t arg,
uint32_t *resp)

Sends an SDIO command with a short response expected.

Note
The CRC is not verified.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcpl pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd card command</td>
</tr>
<tr>
<td>in</td>
<td>arg command argument</td>
</tr>
<tr>
<td>out</td>
<td>resp pointer to the response buffer (one word)</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.39.7.35 sdc_lld_send_cmd_short_crc()

bool sdc_lld_send_cmd_short_crc(
    SDCDriver *sdcpl,
    uint8_t cmd,
    uint32_t arg,
    uint32_t *resp)

Sends an SDIO command with a short response expected and CRC.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcpl pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd card command</td>
</tr>
<tr>
<td>in</td>
<td>arg command argument</td>
</tr>
<tr>
<td>out</td>
<td>resp pointer to the response buffer (one word)</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.39.7.36 sdc_lld_send_cmd_long_crc()

bool sdc_lld_send_cmd_long_crc (SDCDriver * sdcp, uint8_t cmd, uint32_t arg, uint32_t * resp )

Sends an SDIO command with a long response expected and CRC.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdcp</td>
<td>pointer to the SDCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmd</td>
<td>card command</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>command argument</td>
</tr>
<tr>
<td>out</td>
<td>resp</td>
<td>pointer to the response buffer (four words)</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.
7.39 SDC Driver

7.39.7.37 sdc_lld_read()

```c
bool sdc_lld_read ( 
    SDCDriver * sdcp,
    uint32_t startblk,
    uint8_t * buf,
    uint32_t n )
```

Reads one or more blocks.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdcp</td>
<td>pointer to the SDCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to read</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the read buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to read</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **HAL_SUCCESS**: operation succeeded.
- **HAL_FAILED**: operation failed.

**Function Class:**

Not an API, this function is for internal use only.

7.39.7.38 sdc_lld_write()

```c
bool sdc_lld_write ( 
    SDCDriver * sdcp,
    uint32_t startblk,
    const uint8_t * buf,
    uint32_t n )
```

Writes one or more blocks.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdcp</td>
<td>pointer to the SDCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to write</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the write buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to write</td>
</tr>
</tbody>
</table>
Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>operation status</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

### 7.39.7.39 sdc_lld_sync()

```c
bool sdc_lld_sync (SDCDriver * sdcp)
```

Waits for card idle condition.

Parameters
- **in** `sdcp` pointer to the `SDCDriver` object

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>operation status</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>the operation succeeded</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>the operation failed</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.39.8 Variable Documentation

#### 7.39.8.1 sdc_vmt

```c
const struct SDCDriverVMT sdc_vmt [static]
```
Initial value:

= {
  (size_t)0,
  (bool (*)(void *))sdc_lld_is_card_inserted,
  (bool (*)(void *))sdc_lld_is_write_protected,
  (bool (*)(void *))sdcConnect,
  (bool (*)(void *))sdcDisconnect,
  (bool (*)(void *, uint32_t, uint8_t *, uint32_t))sdcRead,
  (bool (*)(void *, uint32_t, const uint8_t *, uint32_t))sdcWrite,
  (bool (*)(void *))sdcSync,
  (bool (*)(void *, BlockDeviceInfo */) sdcGetInfo
}

Virtual methods table.

7.39.8.2 SDCD1

SDCDriver SDCD1

SDCD1 driver identifier.
7.40 Serial Driver

Generic Serial Driver.

7.40.1 Detailed Description

Generic Serial Driver.

This module implements a generic full duplex serial driver. The driver implements a `SerialDriver` interface and uses I/O Queues for communication between the upper and the lower driver. Event flags are used to notify the application about incoming data, outgoing data and other I/O events. The module also contains functions that make the implementation of the interrupt service routines much easier.

Precondition

In order to use the SERIAL driver the `HAL_USE_SERIAL` option must be enabled in `halconf.h`.

7.40.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

![State Machine Diagram]

Macros

- `#define _serial_driver_methods _base_asynchronous_channel_methods`
  SerialDriver specific methods.
- `#define _serial_driver_data`
  SerialDriver specific data.
Serial status flags (legacy)

- #define SD_PARITY_ERROR CHN_PARITY_ERROR
- #define SD_FRAMING_ERROR CHN_FRAMING_ERROR
- #define SD_OVERRUN_ERROR CHN_OVERRUN_ERROR
- #define SD_NOISE_ERROR CHN_NOISE_ERROR
- #define SD_BREAK_DETECTED CHN_BREAK_DETECTED
- #define SD_QUEUE_FULL_ERROR CHN_BUFFER_FULL_ERROR

Serial configuration options

- #define SERIAL_DEFAULT_BITRATE 38400
  Default bit rate.
- #define SERIAL_BUFFERS_SIZE 16
  Serial buffers size.

Macro Functions

- #define sdPutI(sdp, b) oqPutI(&(sdp)->oqueue, b)
  Direct write to a SerialDriver.
- #define sdPut(sdp, b) oqPut(&(sdp)->oqueue, b)
  Direct write to a SerialDriver.
- #define sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->oqueue, b, t)
  Direct write to a SerialDriver with timeout specification.
- #define sdGetI(sdp) iqGetI(&(sdp)->iqueue)
  Direct read from a SerialDriver.
- #define sdGet(sdp) iqGet(&(sdp)->iqueue)
  Direct read from a SerialDriver.
- #define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->iqueue, t)
  Direct read from a SerialDriver with timeout specification.
- #define sdWriteI(sdp, b, n) oqWriteI(&(sdp)->oqueue, b, n)
  Direct non-blocking write to a SerialDriver.
- #define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)
  Direct blocking write to a SerialDriver.
- #define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->oqueue, b, n, t)
  Direct blocking write to a SerialDriver with timeout specification.
- #define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)
  Direct non-blocking write to a SerialDriver.
- #define sdReadI(sdp, b, n) iqReadI(&(sdp)->iqueue, b, n)
  Direct non-blocking read from a SerialDriver.
- #define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n)
  Direct non-blocking read from a SerialDriver.
- #define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->iqueue, b, n, t)
  Direct blocking read from a SerialDriver.
- #define sdAsynchronousRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)
  Direct non-blocking read from a SerialDriver.

PLATFORM configuration options

- #define PLATFORM_SERIAL_USE_USART1 FALSE
  USART1 driver enable switch.
Typedefs

- typedef struct hal_serial_driver SerialDriver
  Structure representing a serial driver.
- typedef struct hal_serial_config SerialConfig
  PLATFORM Serial Driver configuration structure.

Data Structures

- struct SerialDriverVMT
  SerialDriver virtual methods table.
- struct hal_serial_driver
  Full duplex serial driver class.
- struct hal_serial_config
  PLATFORM Serial Driver configuration structure.

Functions

- void sdInit (void)
  Serial Driver initialization.
- void sdObjectInit (SerialDriver *sdp, qnotify_t inotify, qnotify_t onotify)
  Initializes a generic serial driver object.
- msg_t sdStart (SerialDriver *sdp, const SerialConfig *config)
  Configures and starts the driver.
- void sdStop (SerialDriver *sdp)
  Stops the driver.
- void sdIncomingDataI (SerialDriver *sdp, uint8_t b)
  Handles incoming data.
- msg_t sdRequestDataI (SerialDriver *sdp)
  Handles outgoing data.
- bool sdPutWouldBlock (SerialDriver *sdp)
  Direct output check on a SerialDriver.
- bool sdGetWouldBlock (SerialDriver *sdp)
  Direct input check on a SerialDriver.
- msg_t sdControl (SerialDriver *sdp, unsigned int operation, void *arg)
  Control operation on a serial port.
- void sd_lld_init (void)
  Low level serial driver initialization.
- void sd_lld_start (SerialDriver *sdp, const SerialConfig *config)
  Low level serial driver configuration and (re)start.
- void sd_lld_stop (SerialDriver *sdp)
  Low level serial driver stop.

Enumerations

- enum sdstate_t { SD_UNINIT = 0, SD_STOP = 1, SD_READY = 2 }
  Driver state machine possible states.
Variables

- SerialDriver SD1
  USART1 serial driver identifier.
- static const SerialConfig default_config
  Driver default configuration.

7.40.3 Macro Definition Documentation

7.40.3.1 SERIAL_DEFAULT_BITRATE

#define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.40.3.2 SERIAL_BUFFERS_SIZE

#define SERIAL_BUFFERS_SIZE 16

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

Note

The default is 16 bytes for both the transmission and receive buffers.
This is a global setting and it can be overridden by low level driver specific settings.

7.40.3.3 _serial_driver_methods

#define _serial_driver_methods _base_asynchronous_channel_methods

SerialDriver specific methods.
7.40.3.4 sdPutI

#define sdPutI(
    sdp,
    b ) oqPutI(&(sdp)->oqueue, b)

Direct write to a SerialDriver.

Note
This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

Function Class:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.40.3.5 sdPut

#define sdPut(
    sdp,
    b ) oqPut(&(sdp)->oqueue, b)

Direct write to a SerialDriver.

Note
This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.40.3.6 sdPutTimeout

#define sdPutTimeout(
    sdp,
    b,
    t ) oqPutTimeout(&(sdp)->oqueue, b, t)

Direct write to a SerialDriver with timeout specification.

Note
This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.40 Serial Driver

7.40.3.7  sdGetI

#define sdGetI( 
    sdp ) iqGetI(&(sdp)->iqueue)

Direct read from a SerialDriver.

Note
This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

Function Class:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.40.3.8  sdGet

#define sdGet( 
    sdp ) iqGet(&(sdp)->iqueue)

Direct read from a SerialDriver.

Note
This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.40.3.9  sdGetTimeout

#define sdGetTimeout( 
    sdp, 
    t ) iqGetTimeout(&(sdp)->iqueue, t)

Direct read from a SerialDriver with timeout specification.

Note
This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.40.3.10  sdWriteI

#define sdWriteI(
  sdp,
  b,
  n)
  oqWriteI(&(sdp)->queue, b, n)

Direct non-blocking write to a SerialDriver.

Note
This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write from different channels implementations.

Function Class:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.40.3.11  sdWrite

#define sdWrite(
  sdp,
  b,
  n)
  oqWriteTimeout(&(sdp)->queue, b, n, TIME_INFINITE)

Direct blocking write to a SerialDriver.

Note
This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write from different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.40.3.12  sdWriteTimeout

#define sdWriteTimeout(
  sdp,
  b,
  n,
  t)
  oqWriteTimeout(&(sdp)->queue, b, n, t)

Direct blocking write to a SerialDriver with timeout specification.

Note
This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.40.3.13 sdAsynchronousWrite

```c
#define sdAsynchronousWrite(
    sdp,
    b,
    n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)
```

Direct non-blocking write to a SerialDriver.

**Note**

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.40.3.14 sdReadI

```c
#define sdReadI(
    sdp,
    b,
    n) iqReadI(&(sdp)->iqueue, b, n)
```

Direct non-blocking read from a SerialDriver.

**Note**

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

**Function Class:**

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.40.3.15 sdRead

```c
#define sdRead(
    sdp,
    b,
    n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)
```

Direct blocking read from a SerialDriver.

**Note**

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.40.3.16 sdReadTimeout

#define sdReadTimeout(
    sdp,
    b,
    n,
    t ) iqReadTimeout(&(sdp)->queue, b, n, t)

Direct blocking read from a SerialDriver with timeout specification.

Note
This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.40.3.17 sdAsynchronousRead

#define sdAsynchronousRead(
    sdp,
    b,
    n ) iqReadTimeout(&(sdp)->queue, b, n, TIME_IMMEDIATE)

Direct non-blocking read from a SerialDriver.

Note
This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.40.3.18 PLATFORM_SERIAL_USE_USART1

#define PLATFORM_SERIAL_USE_USART1 FALSE

USART1 driver enable switch.
If set to TRUE the support for USART1 is included.

Note
The default is FALSE.
7.40 Serial Driver

7.40.3.19 _serial_driver_data

#define _serial_driver_data

Value:

_value_base_asynchronous_channel_data

/* Driver state. */
volatile sdstate_t state;

/* Input queue. */
input_queue_t iqueue;

/* Output queue. */
output_queue_t oqueue;

/* Input circular buffer. */
uint8_t ib[SERIAL_BUFFERS_SIZE];

/* Output circular buffer. */
uint8_t ob[SERIAL_BUFFERS_SIZE];

/* End of the mandatory fields. */

SerialDriver specific data.

7.40.4 Typedef Documentation

7.40.4.1 SerialDriver

typedef struct hal_serial_driver SerialDriver

Structure representing a serial driver.

7.40.4.2 SerialConfig

typedef struct hal_serial_config SerialConfig

PLATFORM Serial Driver configuration structure.

An instance of this structure must be passed to sdStart() in order to configure and start a serial driver operations.

Note

This structure content is architecture dependent, each driver implementation defines its own version and the custom static initializers.

7.40.5 Enumeration Type Documentation

7.40.5.1 sdstate_t

enum sdstate_t

Driver state machine possible states.
7.40.6 Function Documentation

7.40.6.1 sdInit()

void sdInit (  
    void )

Serial Driver initialization.

Note

This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

\[ \text{sdInit} \rightarrow \text{sd_lld_init} \rightarrow \text{sdObjectInit} \]

7.40.6.2 sdObjectInit()

void sdObjectInit (  
    SerialDriver * sdp,  
    qnotify_t inotify,  
    qnotify_t onotify )

Initializes a generic serial driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.
7.40 Serial Driver

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>sdp</th>
<th>pointer to a SerialDriver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>notify</td>
<td>pointer to a callback function that is invoked when some data is read from the Queue. The value can be NULL.</td>
</tr>
<tr>
<td>in</td>
<td>onotify</td>
<td>pointer to a callback function that is invoked when some data is written in the Queue. The value can be NULL.</td>
</tr>
</tbody>
</table>

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.40.6.3 sdStart()

\[
\text{msg_t \ sdStart (}
\begin{align*}
  \text{SerialDriver * sdp,} \\
  \text{const SerialConfig * config}
\end{align*}
\]

Configures and starts the driver.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to a SerialDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>the architecture-dependent serial driver configuration. If this parameter is set to NULL then a default configuration is used.</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
### 7.40.6.4 sdStop()

```c
void sdStop (SerialDriver * sdp )
```

Stops the driver.

Any thread waiting on the driver’s queues will be awakened with the message `MSG_RESET`.

**Parameters**

- **in sdp** pointer to a `SerialDriver` object

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.40.6.5 sdIncomingDataI()

```c
void sdIncomingDataI (SerialDriver * sdp, uint8_t b )
```

Handles incoming data.

This function must be called from the input interrupt service routine in order to enqueue incoming data and generate the related events.
Note

The incoming data event is only generated when the input queue becomes non-empty.
In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to a SerialDriver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte to be written in the driver's Input Queue</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
sdIncomingDataI
  iqPut
  osalThreadDequeueNextI
```

7.40.6.6 sdRequestDataI()

`msg_t sdRequestDataI`

`SerialDriver * sdp`)

Handles outgoing data.

Must be called from the output interrupt service routine in order to get the next byte to be transmitted.

Note

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

Parameters

| in   | sdp | pointer to a SerialDriver structure |
Returns

The byte value read from the driver's output queue.

Return values

- **MSG_TIMEOUT** if the queue is empty (the lower driver usually disables the interrupt source when this happens).

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

7.40.6.7  **sdPutWouldBlock()**

```c
bool sdPutWouldBlock ( SerialDriver * sdp )
```

Direct output check on a **SerialDriver**.

Note

This function bypasses the indirect access to the channel and checks directly the output queue. This is faster but cannot be used to check different channels implementations.

Parameters

- **in**  **sdp**  pointer to a **SerialDriver** structure

Returns

The queue status.

Return values

- **false** if the next write operation would not block.
- **true** if the next write operation would block.
7.40 Serial Driver

**Deprecated**

**Function Class:**
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
sdPutWouldBlock
    osalSysLock
osalSysUnlock
```

### 7.40.6.8 sdGetWouldBlock()

```c
bool sdGetWouldBlock (
    SerialDriver * sdp
)
```

**Direct input check on a SerialDriver.**

**Note**
This function bypasses the indirect access to the channel and checks directly the input queue. This is faster but cannot be used to check different channels implementations.

**Parameters**

| in  | sdp | pointer to a SerialDriver structure |

**Returns**

The queue status.

**Return values**

| false | if the next write operation would not block. |
| true  | if the next write operation would block.     |

**Deprecated**
7.40.6.9  sdControl()

msg_t sdControl (  
    SerialDriver * sdp,  
    unsigned int operation,  
    void * arg )

Control operation on a serial port.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdp</td>
</tr>
<tr>
<td>in</td>
<td>operation</td>
</tr>
<tr>
<td>in,out</td>
<td>arg</td>
</tr>
</tbody>
</table>

Returns

The control operation status.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>in case of success.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>in case of operation timeout.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>in case of operation reset.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.40.6.10  sd_lld_init()

```c
void sd_lld_init (  
    void  )
```

Low level serial driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
sd_lld_init  sdObjectInit
```

7.40.6.11  sd_lld_start()

```c
void sd_lld_start (  
    SerialDriver * sdp,  
    const SerialConfig * config )
```

Low level serial driver configuration and (re)start.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to a SerialDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>the architecture-dependent serial driver configuration. If this parameter is set to NULL then a default configuration is used.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.40.6.12  sd_lld_stop()

```c
void sd_lld_stop (  
    SerialDriver * sdp  )
```

Low level serial driver stop.

De-initializes the USART, stops the associated clock, resets the interrupt vector.
Parameters

| in  | sdp | pointer to a SerialDriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.40.7 Variable Documentation

#### 7.40.7.1 SD1

SerialDriver SD1

USART1 serial driver identifier.

#### 7.40.7.2 default_config

const SerialConfig default_config [static]

Initial value:
```
= { 38400 }  
```

Driver default configuration.
7.41 Serial over USB Driver

Serial over USB Driver.

7.41.1 Detailed Description

Serial over USB Driver.

This module implements an USB Communication Device Class (CDC) as a normal serial communication port accessible from the device application.

Precondition

In order to use the USB over Serial driver the HAL_USE_SERIAL_USB option must be enabled in halconf.h.

7.41.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

![Driver State Machine Diagram]

Macros

- #define _serial_usb_driver_data
  
  SerialDriver specific data.

- #define _serial_usb_driver_methods _base_asynchronous_channel_methods
  
  SerialUSBDriver specific methods.
SERIAL_USB configuration options

- `#define SERIAL_USB_BUFFERS_SIZE 256`
  Serial over USB buffers size.
- `#define SERIAL_USB_BUFFERS_NUMBER 2`
  Serial over USB number of buffers.

Typedefs

- `typedef struct SerialUSBDriver SerialUSBDriver`
  Structure representing a serial over USB driver.

Data Structures

- `struct SerialUSBConfig`
  Serial over USB Driver configuration structure.
- `struct SerialUSBDriverVMT`
  SerialDriver virtual methods table.
- `struct SerialUSBDriver`
  Full duplex serial driver class.

Functions

- `static void ibnotify (io_buffers_queue_t *bqp)`
  Notification of empty buffer released into the input buffers queue.
- `static void obnotify (ioBuffersQueue_t *bqp)`
  Notification of filled buffer inserted into the output buffers queue.
- `void sduInit (void)`
  Serial Driver initialization.
- `void sduObjectInit (SerialUSBDriver *sdup)`
  Initializes a generic full duplex driver object.
- `msg_t sduStart (SerialUSBDriver *sdup, const SerialUSBConfig *config)`
  Configures and starts the driver.
- `void sduStop (SerialUSBDriver *sdup)`
  Stops the driver.
- `void sduSuspendHook1 (SerialUSBDriver *sdup)`
  USB device suspend handler.
- `void sduWakeupHook1 (SerialUSBDriver *sdup)`
  USB device wakeup handler.
- `void sduConfigureHook1 (SerialUSBDriver *sdup)`
  USB device configured handler.
- `bool sduRequestsHook (USBDriver *usbp)`
  Default requests hook.
- `void sduSOFHook1 (SerialUSBDriver *sdup)`
  SOF handler.
- `void sduDataTransmitted (USBDriver *usbp, usbep_t ep)`
  Default data transmitted callback.
- `void sduDataReceived (USBDriver *usbp, usbep_t ep)`
  Default data received callback.
- `void sduInterruptTransmitted (USBDriver *usbp, usbep_t ep)`
  Default data received callback.
- `msg_t sduControl (USBDriver *usbp, unsigned int operation, void *arg)`
  Control operation on a serial USB port.
Enumerations

- enum sdustate_t { SDU_UNINIT = 0, SDU_STOP = 1, SDU_READY = 2 }

Driver state machine possible states.

7.41.3 Macro Definition Documentation

7.41.3.1 SERIAL_USB_BUFFERS_SIZE

#define SERIAL_USB_BUFFERS_SIZE 256

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

Note

The default is 256 bytes for both the transmission and receive buffers.

7.41.3.2 SERIAL_USB_BUFFERS_NUMBER

#define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.41.3.3 _serial_usb_driver_data

#define _serial_usb_driver_data

Value:

_base_asynchronous_channel_data

/* Driver state.*/

sdustate_t state;

/* Input buffers queue.*/

input_buffers_queue_t ibqueue;

/* Output queue.*/

output_buffers_queue_t obqueue;

/* Input buffer.*/

uint8_t ib[BQ_BUFFER_SIZE(SERIAL_USB_BUFFERS_NUMBER, SERIAL_USB_BUFFERS_SIZE)];

/* Output buffer.*/

uint8_t ob[BQ_BUFFER_SIZE(SERIAL_USB_BUFFERS_NUMBER, SERIAL_USB_BUFFERS_SIZE)];

/* End of the mandatory fields.*/

/* Current configuration data.*/

const SerialUSBConfig *config;

SerialDriver specific data.
7.41.3.4 _serial_usb_driver_methods

#define _serial_usb_driver_methods _base_asynchronous_channel_methods

SerialUSB Driver specific methods.

7.41.4 Typedef Documentation

7.41.4.1 SerialUSB Driver

typedef struct SerialUSB Driver SerialUSB Driver

Structure representing a serial over USB driver.

7.41.5 Enumeration Type Documentation

7.41.5.1 sdustate_t

enum sdustate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDU_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SDU_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>SDU_READY</td>
<td>Ready.</td>
</tr>
</tbody>
</table>

7.41.6 Function Documentation

7.41.6.1 ibnotify()

static void ibnotify (  
    io_buffers_queue_t * bqp ) [static]

Notification of empty buffer released into the input buffers queue.
### 7.41.6.2 obnotify()

```c
static void obnotify (io_buffers_queue_t *bqp) [static]
```

Notification of filled buffer inserted into the output buffers queue.

**Parameters**

- `in bqp` the buffers queue pointer.

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.41.6.3 sduInit()

```c
void sduInit (void)
```

Serial Driver initialization.

**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

**Function Class:**

- **Initializer**, this function just initializes an object and can be invoked before the kernel is initialized.
### 7.41.6.4 sduObjectInit()

```c
void sduObjectInit (SerialUSBDriver * sdup)
```

Initializes a generic full duplex driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

**Parameters**

| Out | sdup | Pointer to a `SerialUSBDriver` structure |

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.41.6.5 sduStart()

```c
msg_t sduStart (SerialUSBDriver * sdup,
                const SerialUSBConfig * config)
```

Configures and starts the driver.

**Parameters**

| In  | sdup | Pointer to a `SerialUSBDriver` object |
| In  | config | The serial over USB driver configuration |

**Returns**

The operation status.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
ChibiOS/HAL
```
7.41.6.6 sduStop()

void sduStop {
    SerialUSBDriver * sdup
}

Stops the driver.

Any thread waiting on the driver’s queues will be awakened with the message MSG_RESET.

Parameters

| in  | sdup  | pointer to a SerialUSBDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.41.6.7 sduSuspendHookI()

void sduSuspendHookI ( 
    SerialUSBDriver * sdup
)

USB device suspend handler.

Generates a CHN_DISCONNECT event and puts queues in non-blocking mode, this way the application cannot get stuck in the middle of an I/O operations.

Note

If this function is not called from an ISR then an explicit call to osalOsRescheduleS() in necessary afterward.
7.41.6.8 sduWakeupHookI()

```c
void sduWakeupHookI ( 
    SerialUSBDriver * sdup )
```

USB device wakeup handler.

Generates a CHN_CONNECT event and resumes normal queues operations.

**Note**

If this function is not called from an ISR then an explicit call to `osalOsRescheduleS()` is necessary afterward.

**Parameters**

| in  | sdup | pointer to a SerialUSBDriver object |

**Function Class:**

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.41.6.9 sduConfigureHookI()

```c
void sduConfigureHookI ( 
    SerialUSBDriver * sdup )
```

USB device configured handler.

**Parameters**

| in  | sdup | pointer to a SerialUSBDriver object |

**Function Class:**

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

![Call Graph](image)

### 7.41.6.10  sduRequestsHook()

```c
bool sduRequestsHook ( 
    USBDriver * usbp )
```

Default requests hook.

Applications wanting to use the Serial over USB driver can use this function as requests hook in the USB configuration. The following requests are emulated:

- CDC_GET_LINE_CODING.
- CDC_SET_LINE_CODING.
- CDC_SET_CONTROL_LINE_STATE.

**Parameters**

- `usbp` pointer to the `USBDriver` object

**Returns**

- The hook status.

**Return values**

<table>
<thead>
<tr>
<th>true</th>
<th>Message handled internally.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>Message not handled.</td>
</tr>
</tbody>
</table>
7.41.6.11   sduSOFHookI()

void sduSOFHookI (  
    SerialUSBDriver * sdup  )

SOF handler.

The SOF interrupt is used for automatic flushing of incomplete buffers pending in the output queue.

Parameters

|   in | sdup | pointer to a SerialUSBDriver object |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
    obqGetFullBufferI
    sduSOFHookI
    obqTryFlushI
    usbStartTransmitI
    usb_lld_start_in
```

7.41.6.12   sduDataTransmitted()

void sduDataTransmitted (  
    USBDriver * usbp,  
    usbep_t ep  )

Default data transmitted callback.

The application must use this function as callback for the IN data endpoint.

Parameters

|   in  | usbp | pointer to the USBDriver object |
|      | ep   | IN endpoint number              |
7.41 Serial over USB Driver

Here is the call graph for this function:

7.41.6.13 sduDataReceived()

```c
void sduDataReceived (
    USBDriver * usbp,
    usbep_t ep )
```

Default data received callback.

The application must use this function as callback for the OUT data endpoint.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
<tr>
<td>in</td>
<td>ep</td>
<td>OUT endpoint number</td>
</tr>
</tbody>
</table>

Here is the call graph for this function:
7.41.6.14  sduInterruptTransmitted()

void sduInterruptTransmitted (  
    USBDriver ∗ usbp,  
    usbep_t ep )

Default data received callback.

The application must use this function as callback for the IN interrupt endpoint.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

7.41.6.15  sduControl()

msg_t sduControl (  
    USBDriver ∗ usbp,  
    unsigned int operation,  
    void ∗ arg )

Control operation on a serial USB port.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to a USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>operation</td>
<td>control operation code</td>
</tr>
<tr>
<td>in,out</td>
<td>arg</td>
<td>operation argument</td>
</tr>
</tbody>
</table>

Returns

The control operation status.

Return values

- **MSG_OK** in case of success.
- **MSG_TIMEOUT** in case of operation timeout.
- **MSG_RESET** in case of operation reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
### 7.42 SIO Driver

Generic SIO Driver.

#### 7.42.1 Detailed Description

Generic SIO Driver.

This driver abstracts a generic serial communication channel, usually an UART, this driver is similar to Serial and UART drivers but follows a different concept:

- Very close to HW.
- No buffering done in SW, the driver relies on the peripheral internal FIFO, if any.
- Asynchronous, the API is always non blocking.
- Callbacks capable, operations completion and other events are notified using callbacks.
- Very short code paths, especially in ISRs.

**Precondition**

In order to use the SIO driver the `HAL_USE_SIO` option must be enabled in `halconf.h`.

#### 7.42.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).
Macros

- `#define _sio_driver_methods _base_channel_methods`
  
  SIO Driver specific methods.

- `#define sioSetCallbackX(siop, f) (siop)-&gt;cb = (f)`
  
  Associates a callback to the SIO instance.

- `#define siosRXEmptyX(siop) sio_lld_is_rx_empty(siop)`
  
  Determines the state of the RX FIFO.

- `#define siosRXIdleX(siop) sio_lld_is_rx_idle(siop)`
  
  Determines the activity state of the receiver.

- `#define sioHasRXErrorsX(siop) sio_lld_has_rx_errors(siop)`
  
  Determines if RX has pending errors to be read and cleared.

- `#define siosTXFullX(siop) sio_lld_is_tx_full(siop)`
  
  Determines the state of the TX FIFO.

- `#define siosTXOngoingX(siop) sio_lld_is_tx_ongoing(siop)`
  
  Determines the transmission state.

- `#define sioWriteEnableFlagsX(siop, mask)`
  
  Writes the enabled events mask.

- `#define sioSetEnableFlagsX(siop, mask)`
  
  Sets flags into the enabled events flags mask.

- `#define sioClearEnableFlagsX(siop, mask)`
  
  Clears flags from the enabled events flags mask.

- `#define sioGetEnableFlagsX(siop) (siop)-&gt;enabled`
  
  Return the enabled condition flags mask.

- `#define sioGetAndClearErrorsX(siop) sio_lld_get_and_clear_errors(siop)`
  
  Get and clears SIO error event flags.

- `#define sioGetAndClearEventsX(siop) sio_lld_get_and_clear_events(siop)`
  
  Get and clears SIO event flags.

- `#define sioGetEventsX(siop) sio_lld_get_events(siop)`
  
  Returns the pending SIO event flags.

- `#define sioGetX(siop) sio_lld_get(siop)`
  
  Returns one frame from the RX FIFO.

- `#define sioPutX(siop, data) sio_lld_put(siop, data)`
  
  Pushes one frame into the TX FIFO.

- `#define sioAsyncReadX(siop, size, buffer) sio_lld_read(siop, size, buffer)`
  
  Reads data from the RX FIFO.

- `#define sioAsyncWriteX(siop, size, buffer) sio_lld_write(siop, size, buffer)`
  
  Writes data into the TX FIFO.

- `#define sioControlX(siop, operation, arg) sio_lld_control(siop, operation, arg)`
  
  Control operation on a serial port.

- `#define ___sio_reloc_field(v, m, s, d) (((v) &amp; m) &gt;&gt; (s)) &lt;&lt; (d))`
  
  Relocates a bit field.

- `#define sio_lld_driver_fields uint32_t dummy`
  
  Low level fields of the SIO driver structure.

- `#define sio_lld_config_fields uint32_t dummy`
  
  Low level fields of the SIO configuration structure.

- `#define sio_lld_is_rx_empty(siop) false`
  
  Determines the state of the RX FIFO.

- `#define sio_lld_is_rx_idle(siop) false`
  
  Determines the activity state of the receiver.
7.42 SIO Driver

Determines if RX has pending error events to be read and cleared.

• \#define sio_lld_is_tx_full(siop) false

Determines the state of the TX FIFO.

• \#define sio_lld_is_tx_ongoing(siop) false

Determines the transmission state.

SIO masks offsets

• \#define SIO_EV_RXNOTEMPTY_POS 2 /* CHN_INPUT_AVAILABLE */
• \#define SIO_EV_TXNOTFULL_POS 3 /* CHN_OUTPUT_EMPTY */
• \#define SIO_EV_TXDONE_POS 4 /* CHN_TRANSMISSION_END */
• \#define SIO_EV_ALL_ERRORS_POS SIO_EV_PARITY_ERR_POS
• \#define SIO_EV_PARITY_ERR_POS 5 /* CHN_PARITY_ERROR */
• \#define SIO_EV_FRAMING_ERR_POS 6 /* CHN_FRAMING_ERROR */
• \#define SIO_EV_NOISE_ERR_POS 7 /* CHN_NOISE_ERROR */
• \#define SIO_EV_OVERRUN_ERR_POS 8 /* CHN_OVERRUN_ERROR */
• \#define SIO_EV_RXIDLE_POS 9 /* CHN_IDLE_DETECTED */
• \#define SIO_EV_RXBREAK_POS 10 /* CHN_BREAK_DETECTED */

Event flags (compatible with channel and serial events)

• \#define SIO_EV_NONE 0U
• \#define SIO_EV_RXNOTEMPTY (1U << SIO_EV_RXNOTEMPTY_POS)
• \#define SIO_EV_TXNOTFULL (1U << SIO_EV_TXNOTFULL_POS)
• \#define SIO_EV_ALL_DATA (SIO_EV_RXNOTEMPTY | SIO_EV_TXNOTFULL)
• \#define SIO_EV_TXDONE (1U << SIO_EV_TXDONE_POS)
• \#define SIO_EV_PARITY_ERR (1U << SIO_EV_PARITY_ERR_POS)
• \#define SIO_EV_FRAMING_ERR (1U << SIO_EV_FRAMING_ERR_POS)
• \#define SIO_EV_NOISE_ERR (1U << SIO_EV_NOISE_ERR_POS)
• \#define SIO_EV_OVERRUN_ERR (1U << SIO_EV_OVERRUN_ERR_POS)
• \#define SIO_EV_RXIDLE (1U << SIO_EV_RXIDLE_POS)
• \#define SIO_EV_RXBREAK (1U << SIO_EV_RXBREAK_POS)
• \#define SIO_EV_ALL_ERRORS
• \#define SIO_EV_ALL_EVENTS

Additional messages

• \#define SIO_MSG_ERRORS 1

SIO configuration options

• \#define SIO_DEFAULT_BITRATE 38400
  Default bit rate.
• \#define SIO_USE_SYNCHRONIZATION TRUE
  Support for thread synchronization API.
• \#define SIO_USE_STREAMS_INTERFACE SIO_USE_SYNCHRONIZATION
  Support for streams interface.
Low level driver helper macros

- `#define __sio_callback(siop)`
  SIO callback.
- `#define __sio_wakeup_errors(siop)`
  Wakes up because RX errors.
- `#define __sio_wakeup_rx(siop)`
  Wakes up the RX-waiting thread.
- `#define __sio_wakeup_rxidle(siop)`
  Wakes up the RX-idle-waiting thread.
- `#define __sio_wakeup_tx(siop)`
  Wakes up the TX-waiting thread.
- `#define __sio_wakeup_txend(siop)`
  Wakes up the TXend-waiting thread.

PLATFORM configuration options

- `#define PLATFORM_SIO_USE_SIO1 FALSE`
  SIO driver enable switch.

Typedefs

- `typedef eventflags_t sioevents_t`  
  Type of event flags.
- `typedef struct hal_sio_driver SIODriver`  
  Type of structure representing a SIO driver.
- `typedef struct hal_sio_config SIOConfig`  
  Type of structure representing a SIO configuration.
- `typedef void(∗siocb_t)(SIODriver ∗siop)`  
  Generic SIO notification callback type.

Data Structures

- `struct hal_sio_config`  
  Driver configuration structure.
- `struct sio_driver_vmt`  
  SIO driver virtual methods table.
- `struct hal_sio_driver`  
  Structure representing a SIO driver.
Functions

- **void sioInit (void)**
  
  SIO Driver initialization.

- **void sioObjectInit (SIODriver *siop)**
  
  Initializes the standard part of a SIO Driver structure.

- **msg_t sioStart (SIODriver *siop, const SIOConfig *config)**
  
  Configures and activates the SIO peripheral.

- **void sioStop (SIODriver *siop)**
  
  Deactivates the SIO peripheral.

- **void sioWriteEnableFlags (SIODriver *siop, sioevents_t mask)**
  
  Writes the enabled events flags mask.

- **void sioSetEnableFlags (SIODriver *siop, sioevents_t mask)**
  
  Sets flags into the enabled events flags mask.

- **void sioClearEnableFlags (SIODriver *siop, sioevents_t mask)**
  
  Clears flags from the enabled events flags mask.

- **sioevents_t sioGetAndClearErrors (SIODriver *siop)**
  
  Get and clears SIO error event flags.

- **sioevents_t sioGetAndClearEvents (SIODriver *siop)**
  
  Get and clears SIO event flags.

- **sioevents_t sioGetEvents (SIODriver *siop)**
  
  Returns the pending SIO event flags.

- **size_t sioAsyncRead (SIODriver *siop, uint8_t *buffer, size_t n)**
  
  Reads data from the RX FIFO.

- **size_t sioAsyncWrite (SIODriver *siop, const uint8_t *buffer, size_t n)**
  
  Writes data into the TX FIFO.

- **msg_t sioSynchronizeRX (SIODriver *siop, sysinterval_t timeout)**
  
  Synchronizes with RX FIFO data availability.

- **msg_t sioSynchronizeRXIdle (SIODriver *siop, sysinterval_t timeout)**
  
  Synchronizes with RX going idle.

- **msg_t sioSynchronizeTX (SIODriver *siop, sysinterval_t timeout)**
  
  Synchronizes with TX FIFO space availability.

- **msg_t sioSynchronizeTXEnd (SIODriver *siop, sysinterval_t timeout)**
  
  Synchronizes with TX completion.

- **void sio_lld_init (void)**
  
  Low level SIO driver initialization.

- **msg_t sio_lld_start (SIODriver *siop)**
  
  Configures and activates the SIO peripheral.

- **void sio_lld_stop (SIODriver *siop)**
  
  Deactivates the SIO peripheral.

- **void sio_lld_update_enable_flags (SIODriver *siop)**
  
  Enable flags change notification.

- **sioevents_t sio_lld_get_and_clear_errors (SIODriver *siop)**
  
  Get and clears SIO error event flags.

- **sioevents_t sio_lld_get_and_clear_events (SIODriver *siop)**
  
  Get and clears SIO event flags.

- **sioevents_t sio_lld_get_events (SIODriver *siop)**
  
  Returns the pending SIO event flags.

- **size_t sio_lld_read (SIODriver *siop, uint8_t *buffer, size_t n)**
  
  Reads data from the RX FIFO.

- **size_t sio_lld_write (SIODriver *siop, const uint8_t *buffer, size_t n)**
  
  Writes data into the TX FIFO.
Module Documentation

Writes data into the TX FIFO.

- `msg_t sio_lld_get (SIODriver *siop)`
  Returns one frame from the RX FIFO.

- `void sio_lld_put (SIODriver *siop, uint_fast16_t data)`
  Pushes one frame into the TX FIFO.

- `msg_t sio_lld_control (SIODriver *siop, unsigned int operation, void *arg)`
  Control operation on a serial port.

- `void sio_lld_serve_interrupt (SIODriver *siop)`
  Serves an UART interrupt.

Enumerations

- `enum siostate_t { SIO_UNINIT = 0, SIO_STOP = 1, SIO_READY = 2 }`
  Driver state machine possible states.

Variables

- `SIODriver SIO1`
  SIO1 driver identifier.

7.42.3 Macro Definition Documentation

7.42.3.1 SIO_DEFAULT_BITRATE

#define SIO_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.42.3.2 SIO_USE_SYNCHRONIZATION

#define SIO_USE_SYNCHRONIZATION TRUE

Support for thread synchronization API.

7.42.3.3 SIO_USE_STREAMS_INTERFACE

#define SIO_USE_STREAMS_INTERFACE SIO_USE_SYNCHRONIZATION

Support for streams interface.
7.42 SIO Driver

7.42.3.4 _sio_driver_methods

#define _sio_driver_methods _base_channel_methods

SIODriver specific methods.

7.42.3.5 sioSetCallbackX

#define sioSetCallbackX(
    siop,
    f ) (siop)->cb = (f)

Associates a callback to the SIO instance.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>f</td>
<td>callback to be associated</td>
</tr>
</tbody>
</table>

7.42.3.6 sioIsRXEmptyX

#define sioIsRXEmptyX(
    siop ) sio_lld_is_rx_empty(siop)

Determines the state of the RX FIFO.

Parameters

| in  | siop | pointer to the SIODriver object |

Returns

The RX FIFO state.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if RX FIFO is not empty.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if RX FIFO is empty.</td>
</tr>
</tbody>
</table>

Function Class:

This is an X-Class API, this function can be invoked from any context.
7.42.3.7 sioIsRXIdleX

#define sioIsRXIdleX(
    siop ) sio_lld_is_rx_idle(siop)

Determines the activity state of the receiver.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>siop</td>
<td>pointer to the SIODriver object</td>
</tr>
</tbody>
</table>

Returns

The RX activity state.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if RX is in active state.</td>
</tr>
<tr>
<td>true</td>
<td>if RX is in idle state.</td>
</tr>
</tbody>
</table>

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.8 sioHasRXErrorsX

#define sioHasRXErrorsX(
    siop ) sio_lld_has_rx_errors(siop)

Determines if RX has pending errors to be read and cleared.

Note

Only errors are handled, data and idle events are not considered.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>siop</td>
<td>pointer to the SIODriver object</td>
</tr>
</tbody>
</table>

Returns

The RX error events.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if RX has no pending events.</td>
</tr>
<tr>
<td>true</td>
<td>if RX has pending events.</td>
</tr>
</tbody>
</table>
Function Class:
This is an **X-Class API**, this function can be invoked from any context.

7.42.3.9 sioIsTXFullX

```c
#define sioIsTXFullX(
    
    siop ) sio_lld_is_tx_full(siop)
```

Determines the state of the TX FIFO.

**Parameters**

```
  in  siop  pointer to the SIODriver object
```

**Returns**

The TX FIFO state.

**Return values**

- `false` if TX FIFO is not full.
- `true` if TX FIFO is full.

Function Class:
This is an **X-Class API**, this function can be invoked from any context.

7.42.3.10 sioIsTXOngoingX

```c
#define sioIsTXOngoingX(
    
    siop ) sio_lld_is_tx_ongoing(siop)
```

Determines the transmission state.

**Parameters**

```
  in  siop  pointer to the SIODriver object
```

**Returns**

The TX FIFO state.
Return values

| false | if transmission is over. |
| true  | if transmission is ongoing. |

Function Class:

This is an X-Class API, this function can be invoked from any context.

### 7.42.3.11 sioWriteEnableFlagsX

```c
#define sioWriteEnableFlagsX(
    siop,
    mask )
```

**Value:**

```c
    do {
        \( (siop)->enabled = (mask); \)
        sio_lld_update_enable_flags(siop);
    } while (false)
```

Writes the enabled events mask.

**Parameters**

| in  | siop | pointer to the SIODriver object |
| in  | mask | enabled events mask to be written |

Function Class:

This is an X-Class API, this function can be invoked from any context.

### 7.42.3.12 sioSetEnableFlagsX

```c
#define sioSetEnableFlagsX(
    siop,
    mask )
```

**Value:**

```c
    do {
        \( (siop)->enabled |= (mask); \)
        sio_lld_update_enable_flags(siop);
    } while (false)
```

Sets flags into the enabled events flags mask.

**Parameters**

| in  | siop | pointer to the SIODriver object |
| in  | mask | enabled events mask to be set |
Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.13  sioClearEnableFlagsX

#define sioClearEnableFlagsX(
    siop,
    mask )

Value:

    do {
        do {
            (siop)->enabled &= ~(mask);
        } while (false)
        sio_lld_update_enable_flags(siop);
    } while (false)

Clears flags from the enabled events flags mask.

Parameters

| in  | siop | pointer to the SIODriver object |
| in  | mask | enabled events mask to be cleared |

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.14  sioGetEnableFlagsX

#define sioGetEnableFlagsX(
    siop ) (siop)->enabled

Return the enabled condition flags mask.

Parameters

| in  | siop | pointer to the SIODriver object |

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.15  sioGetAndClearErrorsX

#define sioGetAndClearErrorsX(
    siop ) sio_lld_get_and_clear_errors(siop)

ChibiOS/HAL
Get and clears SIO error event flags.

Parameters

| in  | siop | pointer to the SIODriver object |

Returns

The pending error event flags.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.16 sioGetAndClearEventsX

#define sioGetAndClearEventsX(
    siop ) sio_lld_get_and_clear_events(siop)

Get and clears SIO event flags.

Parameters

| in  | siop | pointer to the SIODriver object |

Returns

The pending event flags.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.17 sioGetEventsX

#define sioGetEventsX(
    siop ) sio_lld_get_events(siop)

Returns the pending SIO event flags.

Parameters

| in  | siop | pointer to the SIODriver object |
7.42 SIO Driver

Returns

The pending event flags.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.18 sioGetX

#define sioGetX(
    siop ) sio_lld_get(siop)

Returns one frame from the RX FIFO.

Note

If the FIFO is empty then the returned value is unpredictable.

Parameters

| in  | siop  | pointer to the SIODriver object |

Returns

The frame from RX FIFO.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.19 sioPutX

#define sioPutX(
    siop,
    data ) sio_lld_put(siop, data)

Pushes one frame into the TX FIFO.

Note

If the FIFO is full then the behavior is unpredictable.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>siop</td>
<td>pointer to the SIO Driver object</td>
</tr>
<tr>
<td>data</td>
<td>frame to be written</td>
</tr>
</tbody>
</table>

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.20  sioAsyncReadX

```c
#define sioAsyncReadX(
    siop,
    size,
    buffer ) sio_lld_read(siop, size, buffer)
```

Reads data from the RX FIFO.

This function is non-blocking, data is read if present and the effective amount is returned.

Note

This function can be called from any context but it is meant to be called from the rxne_cb callback handler.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>siop</td>
<td>pointer to the SIO Driver object</td>
</tr>
<tr>
<td>size</td>
<td>maximum number of frames to read</td>
</tr>
<tr>
<td>buffer</td>
<td>buffer for the received data</td>
</tr>
</tbody>
</table>

Returns

The number of received frames.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.42.3.21  sioAsyncWriteX

```c
#define sioAsyncWriteX(
    siop,
    size,
    buffer ) sio_lld_write(siop, size, buffer)
```

Writes data into the TX FIFO.

This function is non-blocking, data is written if there is space in the FIFO and the effective amount is returned.
Note

This function can be called from any context but it is meant to be called from the `txnf_cb` callback handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>maximum number of frames to read</td>
</tr>
<tr>
<td>out</td>
<td>buffer</td>
<td>buffer containing the data to be transmitted</td>
</tr>
</tbody>
</table>

Returns

The number of transmitted frames.

Function Class:

This is an **X-Class API**, this function can be invoked from any context.

7.42.3.22  sioControlX

```c
#define sioControlX(
    siop,
    operation,
    arg ) sio_lld_control(siop, operation, arg)
```

Control operation on a serial port.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>operation</td>
<td>control operation code</td>
</tr>
<tr>
<td>in, out</td>
<td>arg</td>
<td>operation argument</td>
</tr>
</tbody>
</table>

Returns

The control operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>in case of success.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>in case of operation timeout.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>in case of operation reset.</td>
</tr>
</tbody>
</table>

Function Class:

This is an **X-Class API**, this function can be invoked from any context.
7.42.3.23 __sio_callback

#define __sio_callback(
    siop )

Value:
    do {
        \    
            if ((siop)->cb != NULL) {   \    
                (siop)->cb(siop);    \    
            }                           \    
        | while (false)                 \    
    } while (false)

SIO callback.

Parameters

  in  siop  pointer to the SIODriver object

Function Class:
    Not an API, this function is for internal use only.

7.42.3.24 __sio_wakeup_errors

#define __sio_wakeup_errors(
    siop )

Value:
    do {
        \    
            osalSysLockFromISR(); \    
            osalThreadResumeI(&(siop)->sync_rx, SIO_MSG_ERRORS); \    
            osalThreadResumeI(&(siop)->sync_rxidle, SIO_MSG_ERRORS); \    
            osalSysUnlockFromISR(); \    
        | while (false) \    
    } while (false)

Wakes up because RX errors.

Parameters

  in  siop  pointer to the SIODriver object

Function Class:
    Not an API, this function is for internal use only.

7.42.3.25 __sio_wakeup_rx

#define __sio_wakeup_rx(
    siop )
Value:
\[
\text{do} \{ \\
    \text{osalSysLockFromISR();} \\
    \text{osalThreadResume1(&(siop)->sync_rx, MSG_OK);} \\
    \text{osalSysUnlockFromISR();} \\
    \text{while (false)}
\}
\]
Wakes up the RX-waiting thread.

Parameters

\[
\text{in } \text{siop} \text{ pointer to the SIODriver object}
\]

Function Class:
Not an API, this function is for internal use only.

7.42.3.26 __sio_wakeup_rxidle

\[
\text{#define __sio_wakeup_rxidle(} \\
\text{siop)}
\]

Value:
\[
\text{do} \{ \\
    \text{osalSysLockFromISR();} \\
    \text{osalThreadResume1(&(siop)->sync_rxidle, MSG_OK);} \\
    \text{osalSysUnlockFromISR();} \\
    \text{while (false)}
\}
\]
Wakes up the RX-idle-waiting thread.

Parameters

\[
\text{in } \text{siop} \text{ pointer to the SIODriver object}
\]

Function Class:
Not an API, this function is for internal use only.

7.42.3.27 __sio_wakeup_tx

\[
\text{#define __sio_wakeup_tx(} \\
\text{siop)}
\]

Value:
\[
\text{do} \{ \\
    \text{osalSysLockFromISR();} \\
    \text{osalThreadResume1(&(siop)->sync_tx, MSG_OK);} \\
    \text{osalSysUnlockFromISR();} \\
    \text{while (false)}
\}
\]
Wakes up the TX-waiting thread.
Parameters

| in  | siop | pointer to the SIODriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.42.3.28 __sio_wakeup_txend

```c
#define __sio_wakeup_txend(
    siop )
```

Value:

```c
void {
    osalSysLockFromISR();
    osalThreadResumeI4((siop)->sync_txend, MSG_OK);
    osalSysUnlockFromISR();
} while (false)
```

Wakes up the TXend-waiting thread.

Parameters

| in  | siop | pointer to the SIODriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.42.3.29 __sio_reloc_field

```c
#define __sio_reloc_field(
    v,  
    m,  
    s,  
    d ) (((v) & m) >> (s)) << (d)
```

Relocates a bit field.

Parameters

| in  | v  | value |
| in  | m  | mask of the bit field |
| in  | s  | source bit offset |
| in  | d  | destination bit offset |
### 7.42.3.30 PLATFORM_SIO_USE_SIO1

```c
#define PLATFORM_SIO_USE_SIO1 FALSE
```

SIO driver enable switch.

If set to **TRUE** the support for SIO1 is included.

**Note**

The default is **FALSE**.

### 7.42.3.31 sio_lld_driver_fields

```c
#define sio_lld_driver_fields uint32_t dummy
```

Low level fields of the SIO driver structure.

### 7.42.3.32 sio_lld_config_fields

```c
#define sio_lld_config_fields uint32_t dummy
```

Low level fields of the SIO configuration structure.

### 7.42.3.33 sio_lld_is_rx_empty

```c
#define sio_lld_is_rx_empty(
    siop ) false
```

Determines the state of the RX FIFO.

**Parameters**

```
    in   siop   pointer to the SIODriver object
```

**Returns**

The RX FIFO state.
Return values

| false | if RX FIFO is not empty |
| true  | if RX FIFO is empty    |

Function Class:

Not an API, this function is for internal use only.

7.42.3.34  sio_lld_is_rx_idle

#define sio_lld_is_rx_idle(  
siop ) false

Determines the activity state of the receiver.

Parameters

| in  siop   | pointer to the SIODriver object |

Returns

The RX activity state.

Return values

| false   | if RX is in active state. |
| true    | if RX is in idle state.   |

Function Class:

Not an API, this function is for internal use only.

7.42.3.35  sio_lld_has_rx_errors

#define sio_lld_has_rx_errors(  
siop ) false

Determines if RX has pending error events to be read and cleared.

Note

Only error and protocol errors are handled, data events are not considered.
Parameters

in  siop  pointer to the SIODriver object

Returns

The RX error events.

Return values

false  if RX has no pending events
true   if RX has pending events

Function Class:

Not an API, this function is for internal use only.

7.42.3.36  sio_lld_is_tx_full

#define sio_lld_is_tx_full(
    siop ) false

Determines the state of the TX FIFO.

Parameters

in  siop  pointer to the SIODriver object

Returns

The TX FIFO state.

Return values

false  if TX FIFO is not full
true   if TX FIFO is full

Function Class:

Not an API, this function is for internal use only.

7.42.3.37  sio_lld_is_tx_ongoing

#define sio_lld_is_tx_ongoing(
    siop ) false
Determines the transmission state.

Parameters

| in | siop | pointer to the SIODriver object |

Returns

The TX FIFO state.

Return values

- false if transmission is idle
- true if transmission is ongoing

Function Class:

Not an API, this function is for internal use only.

7.42.4 Typedef Documentation

7.42.4.1 sioevents_t

typedef eventflags_t sioevents_t

Type of event flags.

7.42.4.2 SIODriver

typedef struct hal_sio_driver SIODriver

Type of structure representing a SIO driver.

7.42.4.3 SIOConfig

typedef struct hal_sio_config SIOConfig

Type of structure representing a SIO configuration.

7.42.4.4 siocb_t

typedef void(* siocb_t) (SIODriver *siop)

Generic SIO notification callback type.
Parameters

* in  `siop`  pointer to the SIODriver object

### 7.42.5 Enumeration Type Documentation

#### 7.42.5.1 siostate_t

```c
enum siostate_t
```

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SIO_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>SIO_READY</td>
<td>Ready.</td>
</tr>
</tbody>
</table>

### 7.42.6 Function Documentation

#### 7.42.6.1 sioInit()

```c
void sioInit (  
    void )
```

SIO Driver initialization.

**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
sioInit  sio_lld_init  sioObjectInit
```

### 7.42.6.2 sioObjectInit()

```c
void sioObjectInit ( 
    SIODriver * siop )
```

Initializes the standard part of a SIODriver structure.

Parameters

| out | siop | pointer to the SIODriver object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.42.6.3 sioStart()

```c
msg_t sioStart ( 
    SIODriver * siop, 
    const SIOConfig * config )
```

Configures and activates the SIO peripheral.

Parameters

| in  | siop | pointer to the SIODriver object |
| in  | config | pointer to the SIOConfig object, can be NULL if the default configuration is desired |
Returns

The operation status.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
sioStart
osalSysLock
sio_lld_start
```

### 7.42.6.4 sioStop()

```c
void sioStop ( SIODriver * siop )
```

Deactivates the SIO peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>siop</td>
<td>pointer to the SIODriver object</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
+------------------------+
| osalSysLock            |
+------------------------+
    |                     |
    v                     |
+------------------------+
| sioStop                |
+------------------------+
    |                     |
    v                     |
+------------------------+
| osalThreadResumeI      |
+------------------------+
    |                     |
    v                     |
+------------------------+
| sio_lld_stop           |
```

### 7.42.6.5 sioWriteEnableFlags()

```c
void sioWriteEnableFlags ( 
    SIODriver * siop, 
    sioevents_t mask )
```

Writes the enabled events flags mask.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mask</td>
<td>enabled events mask to be written</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
+------------------------+
| osalSysLock            |
+------------------------+
    |                     |
    v                     |
+------------------------+
| sioWriteEnableFlags    |
    |                     |
    v                     |
+------------------------+
| osalSysUnlock          |
```

7.42.6.6 sioSetEnableFlags()

void sioSetEnableFlags ( 
    SIODriver ∗ siop,
    sioevents_t mask )

Sets flags into the enabled events flags mask.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mask</td>
<td>enabled events mask to be set</td>
</tr>
</tbody>
</table>

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.42.6.7 sioClearEnableFlags()

void sioClearEnableFlags ( 
    SIODriver ∗ siop,
    sioevents_t mask )

Clears flags from the enabled events flags mask.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mask</td>
<td>enabled events mask to be cleared</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.42.6.8  sioGetAndClearErrors()

```c
sioevents_t sioGetAndClearErrors (  
    SIODriver * siop  )
```

Get and clears SIO error event flags.

Parameters

| in | siop | pointer to the SIODriver object |

Returns

The pending error event flags.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
7.42 SIO Driver

7.42.6.9  sioGetAndClearEvents()

sioevents_t  sioGetAndClearEvents (  
   SIODriver  ∗  siop  )

Get and clears SIO event flags.

Parameters

|     |   siop   | pointer to the SIODriver object |

Returns

The pending event flags.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
sioGetAndClearEvents
   ↘
   ↙
osalSysLock
salSysUnlock
   ↘
   ↙
sioGetAndClearEvents
```

7.42.6.10  sioGetEvents()

sioevents_t  sioGetEvents (  
   SIODriver  ∗  siop  )

Returns the pending SIO event flags.

Parameters

|     |   siop   | pointer to the SIODriver object |

ChibiOS/HAL
Returns
The pending event flags.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
sioGetEvents
       ↓
osalSysLock
```
```
sioGetEvents
       ↑
osalSysUnlock
```

7.42.6.11  sioAsyncRead()

```c
size_t sioAsyncRead (  
    SIODriver ∗ siop,  
    uint8_t ∗ buffer,  
    size_t n )
```

Reads data from the RX FIFO.

This function is non-blocking, data is read if present and the effective amount is returned.

Note
This function can be called from any context but it is meant to be called from the rxne_cb callback handler.

Parameters
<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buffer</td>
<td>buffer for the received data</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>maximum number of frames to read</td>
</tr>
</tbody>
</table>

Returns
The number of received frames.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.42.6.12 sioAsyncWrite()

```c
size_t sioAsyncWrite(   
    SIODriver * siop,   
    const uint8_t * buffer,   
    size_t n )
```

Writes data into the TX FIFO.

This function is non-blocking, data is written if there is space in the FIFO and the effective amount is returned.

**Note**

This function can be called from any context but it is meant to be called from the txnf_cb callback handler.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>buffer</td>
<td>buffer containing the data to be transmitted</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>maximum number of frames to read</td>
</tr>
</tbody>
</table>

**Returns**

The number of transmitted frames.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

```
sioAsyncWrite
  osalSysLock
  osalSysUnlock
```

7.42.6.13 **sioSynchronizeRX()**

```c
msg_t sioSynchronizeRX (  
    SIODriver * siop,  
    sysinterval_t timeout )
```

Synchronizes with RX FIFO data availability.

**Note**

The exact behavior depends on low level FIFO settings such as thresholds, etc.

This function can only be called by a single thread at time.

**Parameters**

- **in** `siop` pointer to an `SIODriver` structure
- **in** `timeout` synchronization timeout

**Returns**

The synchronization result.

**Return values**

| MSG_OK          | if there is data in the RX FIFO. |
| MSG_TIMEOUT     | if synchronization timed out.    |
| MSG_RESET       | if the operation has been stopped while waiting. |
| SIO_MSG_ERRORS  | if RX errors occurred before or during wait. |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

```plaintext
sioSynchronizeRX
  ↓ osalSysLock
osalSysLock
  ↓
osalSysUnlock
  ↓
sioSynchronizeRX
```

### 7.42.6.14 sioSynchronizeRXIdle()

```c
msg_t sioSynchronizeRXIdle ( 
    SIODriver * siop, 
    sysinterval_t timeout )
```

Synchronizes with RX going idle.

**Note**

This function can only be called by a single thread at time.

**Parameters**

| in | siop | pointer to an SIODriver structure |
| in | timeout | synchronization timeout |

**Returns**

The synchronization result.

**Return values**

| MSG_OK | if RW went idle. |
| MSG_TIMEOUT | if synchronization timed out. |
| MSG_RESET | if the operation has been stopped while waiting. |
| SIO_MSG_ERRORS | if RX errors occurred before or during wait. |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph Image]

7.42.6.15 sioSynchronizeTX()

```c
msg_t sioSynchronizeTX (  
    SIODriver * siop,  
    sysinterval_t timeout )
```

Synchronizes with TX FIFO space availability.

**Note**

The exact behavior depends on low level FIFO settings such as thresholds, etc.
This function can only be called by a single thread at time.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>siop</code> pointer to an SIODriver structure</td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code> synchronization timeout</td>
</tr>
</tbody>
</table>

**Returns**

The synchronization result.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if there is space in the TX FIFO.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if synchronization timed out.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>operation has been stopped while waiting.</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
sioSynchronizeTX
    osalSysLock
```

### 7.42.6.16 `sioSynchronizeTXEnd()`

```c
msg_t sioSynchronizeTXEnd (  
    SIODriver * siop,  
    sysinterval_t timeout )
```

Synchronizes with TX completion.

**Note**

This function can only be called by a single thread at time.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>siop</code></th>
<th>pointer to an <code>SIODriver</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>timeout</code></td>
<td>synchronization timeout</td>
</tr>
</tbody>
</table>

**Returns**

The synchronization result.

**Return values**

<table>
<thead>
<tr>
<th><code>MSG_OK</code></th>
<th>if TX operation finished.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_TIMEOUT</code></td>
<td>if synchronization timed out.</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

7.42.6.17  sio_lld_init()

```c
void sio_lld_init ( 
    void )
```

Low level SIO driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

![Call Graph](image)

7.42.6.18  sio_lld_start()

```c
msg_t sio_lld_start ( 
    SIODriver * siop )
```

Configures and activates the SIO peripheral.
Parameters

in  siop  pointer to the SIODriver object

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.

7.42.6.19  sio_lld_stop()

void sio_lld_stop (  
    SIODriver * siop )

Deactivates the SIO peripheral.

Parameters

in  siop  pointer to the SIODriver object

Function Class:

Not an API, this function is for internal use only.

7.42.6.20  sio_lld_update_enable_flags()

void sio_lld_update_enable_flags (  
    SIODriver * siop )

Enable flags change notification.

Parameters

in  siop  pointer to the SIODriver object

7.42.6.21  sio_lld_get_and_clear_errors()

sioevents_t sio_lld_get_and_clear_errors (  
    SIODriver * siop )

ChibiOS/HAL
Get and clears SIO error event flags.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>siop</td>
</tr>
</tbody>
</table>

Returns

The pending event flags.

Function Class:

Not an API, this function is for internal use only.

7.42.6.22  sio_lld_get_and_clear_events()

sioevents_t sio_lld_get_and_clear_events (SIODriver * siop)

Get and clears SIO event flags.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>siop</td>
</tr>
</tbody>
</table>

Returns

The pending event flags.

Function Class:

Not an API, this function is for internal use only.

7.42.6.23  sio_lld_get_events()

sioevents_t sio_lld_get_events (SIODriver * siop)

Returns the pending SIO event flags.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>siop</td>
</tr>
</tbody>
</table>
Returns
The pending event flags.

Function Class:
Not an API, this function is for internal use only.

7.42.6.24  sio_lld_read()

size_t sio_lld_read ( 
    SIODriver * siop, 
    uint8_t * buffer, 
    size_t n )

Reads data from the RX FIFO.
The function is not blocking, it writes frames until there is space available without waiting.

Parameters

| in  | siop  | pointer to an SIODriver structure |
| in  | buffer | pointer to the buffer for read frames |
| in  | n      | maximum number of frames to be read |

Returns
The number of frames copied from the buffer.

Return values

| 0  | if the TX FIFO is full |

7.42.6.25  sio_lld_write()

size_t sio_lld_write ( 
    SIODriver * siop, 
    const uint8_t * buffer, 
    size_t n )

Writes data into the TX FIFO.
The function is not blocking, it writes frames until there is space available without waiting.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to an SIODriver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buffer</td>
<td>pointer to the buffer for read frames</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>maximum number of frames to be written</td>
</tr>
</tbody>
</table>

Returns

The number of frames copied from the buffer.

Return values

| 0 | if the TX FIFO is full |

7.42.6.26 sio_lld_get()

```c
msg_t sio_lld_get ( 
    SIODriver * siop )
```

Returns one frame from the RX FIFO.

Note

If the FIFO is empty then the returned value is unpredictable.

Parameters

| in | siop | pointer to the SIODriver object |

Returns

The frame from RX FIFO.

Function Class:

Not an API, this function is for internal use only.

7.42.6.27 sio_lld_put()

```c
void sio_lld_put ( 
    SIODriver * siop, 
    uint_fast16_t data )
```

Pushes one frame into the TX FIFO.

Note

If the FIFO is full then the behavior is unpredictable.
7.42 SIO Driver

Parameters

| in | siop | pointer to the SIODriver object |
| in | data | frame to be written |

Function Class:

Not an API, this function is for internal use only.

7.42.6.28 sio_lld_control()

```c
msg_t sio_lld_control ( 
    SIODriver * siop, 
    unsigned int operation, 
    void * arg )
```

Control operation on a serial port.

Parameters

| in | siop | pointer to the SIODriver object |
| in | operation | control operation code |
| in,out | arg | operation argument |

Returns

The control operation status.

Return values

| MSG_OK   | in case of success. |
| MSG_TIMEOUT | in case of operation timeout. |
| MSG_RESET   | in case of operation reset. |

Function Class:

Not an API, this function is for internal use only.

7.42.6.29 sio_lld_serve_interrupt()

```c
void sio_lld_serve_interrupt ( 
    SIODriver * siop )
```

Serves an UART interrupt.
Parameters

| in  | siop | pointer to the SIODriver object |

Function Class:

Not an API, this function is for internal use only.

7.42.7 Variable Documentation

7.42.7.1 SIOD1

SIODriver SIOD1

SIO1 driver identifier.
7.43 SPI Driver Selector

SPI Driver selector.

7.43.1 Detailed Description

SPI Driver selector.

This module is a selector between SPI drivers version 1 and version 2.
7.44 SPIv1 Driver

Generic SPI Driver (legacy).

7.44.1 Detailed Description

Generic SPI Driver (legacy).

This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

In order to use the SPI driver the HAL_USE_SPI option must be enabled in halconf.h.

7.44.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

The driver is not thread safe for performance reasons, if you need to access the SPI bus from multiple threads then use the spiAcquireBus() and spiReleaseBus() APIs in order to gain exclusive access.

Macros

- #define SPI_SUPPORTS_CIRCULAR TRUE
  Circular mode support flag.
- #define spi_lld_driver_fields
  Low level fields of the SPI driver structure.
- #define spi_lld_config_fields
  Low level fields of the SPI configuration structure.
Chip Select modes

- \#define SPI_SELECT_MODE_NONE
  
- \#define SPI_SELECT_MODE_PAD 1 \*/@brief Legacy mode. */
- \#define SPI_SELECT_MODE_PORT 2 \*/@brief Fastest mode. */
- \#define SPI_SELECT_MODE_LINE 3 \*/@brief Packed mode. */
- \#define SPI_SELECT_MODE_LLD 4 \*/@brief LLD-defined mode. */

SPI configuration options

- \#define SPI_USE_WAIT TRUE
  
  Enables synchronous APIs.
- \#define SPI_USE_MUTUAL_EXCLUSION TRUE
  
  Enables the spiAcquireBus() and spiReleaseBus() APIs.
- \#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD
  
  Handling method for SPI CS line.

Macro Functions

- \#define spiIsBufferComplete(spip) ((bool)((spip)->state == SPI_COMPLETE))
  
  Buffer state.
- \#define spiSelect(spip)
  
  Asserts the slave select signal and prepares for transfers.
- \#define spiUnselect(spip)
  
  Deasserts the slave select signal.
- \#define spiStartIgnore(spip, n)
  
  Ignores data on the SPI bus.
- \#define spiStartExchange(spip, n, txbuf, rxbuf)
  
  Exchanges data on the SPI bus.
- \#define spiStartSend(spip, n, txbuf)
  
  Sends data over the SPI bus.
- \#define spiStartReceive(spip, n, rxbuf)
  
  Receives data from the SPI bus.
- \#define spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)
  
  Exchanges one frame using a polled wait.

Low level driver helper macros

- \#define _spi_wakeup_isr(spip)
  
  Wakes up the waiting thread.
- \#define _spi_isr_code(spip)
  
  Common ISR code when circular mode is not supported.
- \#define _spi_isr_half_code(spip)
  
  Half buffer filled ISR code in circular mode.
- \#define _spi_isr_full_code(spip)
  
  Full buffer filled ISR code in circular mode.
PLATFORM configuration options

- `#define PLATFORM_SPI_USE_SPI1 FALSE`
  SPI1 driver enable switch.

Typedefs

- `typedef struct hal_spi_driver SPIDriver`
  Type of a structure representing an SPI driver.
- `typedef struct hal_spi_config SPIConfig`
  Type of a SPI driver configuration structure.
- `typedef void(* spicallback_t) (SPIDriver *spip)`
  SPI notification callback type.

Data Structures

- `struct hal_spi_config`
  Driver configuration structure.
- `struct hal_spi_driver`
  Structure representing an SPI driver.

Functions

- `void spi_lld_init (void)`
  Low level SPI driver initialization.
- `void spi_lld_start (SPIDriver *spip)`
  Configures and activates the SPI peripheral.
- `void spi_lld_stop (SPIDriver *spip)`
  Deactivates the SPI peripheral.
- `void spi_lld_select (SPIDriver *spip)`
  Asserts the slave select signal and prepares for transfers.
- `void spi_lld_unselect (SPIDriver *spip)`
  Deasserts the slave select signal.
- `void spi_lld_ignore (SPIDriver *spip, size_t n)`
  Ignores data on the SPI bus.
- `void spi_lld_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
  Exchanges data on the SPI bus.
- `void spi_lld_send (SPIDriver *spip, size_t n, const void *txbuf)`
  Sends data over the SPI bus.
- `void spi_lld_receive (SPIDriver *spip, size_t n, void *rxbuf)`
  Receives data from the SPI bus.
- `void spi_lld_abort (SPIDriver *spip)`
  Aborts the ongoing SPI operation, if any.
- `uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)`
  Exchanges one frame using a polled wait.
Enums

```c
enum spistate_t {
    SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,
    SPI_COMPLETE = 4, SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2,
    SPI_ACTIVE = 3, SPI_COMPLETE = 4 }
```

Driver state machine possible states.

Variables

- SPIDriver SPID1

  SPI1 driver identifier.

7.44.3 Macro Definition Documentation

7.44.3.1 SPI_USE_WAIT

```c
#define SPI_USE_WAIT TRUE
```

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.44.3.2 SPI_USE_MUTUAL_EXCLUSION

```c
#define SPI_USE_MUTUAL_EXCLUSION TRUE
```

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.44.3.3 SPI_SELECT_MODE

```c
#define SPI_SELECT_MODE_SPI_SELECT_MODE_PAD
```

Handling method for SPI CS line.

Note

Disabling this option saves both code and data space.
7.44.3.4 spiIsBufferComplete

#define spiIsBufferComplete(spip) ((bool)((spip)->state == SPI_COMPLETE))

Buffer state.

Note
This function is meant to be called from the SPI callback only.

Parameters

| in | spip | pointer to the SPIDriver object |

Returns
The buffer state.

Return values

| false | if the driver filled/sent the first half of the buffer. |
| true  | if the driver filled/sent the second half of the buffer. |

Function Class:
Special function, this function has special requirements see the notes.

7.44.3.5 spiSelectI

#define spiSelectI(spip)

Value:
do {
    spi_lld_select(spip);
} while (false)

Asserts the slave select signal and prepares for transfers.

Parameters

| in | spip | pointer to the SPIDriver object |

Function Class:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
### 7.44.3.6 spiUnselectI

#define spiUnselectI(
    spip
)

**Value:**

```c
    do {
        spi_lld_unselect(spip);
    } while (false)
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

**Parameters**

- **in spip** pointer to the SPIDriver object

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.44.3.7 spiStartIgnoreI

#define spiStartIgnoreI(
    spip,
    n
)

**Value:**

```c
    (spip)->state = SPI_ACTIVE;
    spi_lld_ignore(spip, n);
```

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

**Precondition**

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

**Postcondition**

At the end of the operation the configured callback is invoked.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in spip</td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td>in n</td>
<td>number of words to be ignored</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 7.44.3.8 spiStartExchange_l

```c
#define spiStartExchange_l(
    spip,
    n,
    txbuf,
    rxbuf )
```

**Value:**

```
{   (spip)->state = SPI_ACTIVE;
   spi_lld_exchange(spip, n, txbuf, rxbuf);
}
```

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

**Precondition**

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in spip</td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td>in n</td>
<td>number of words to be exchanged</td>
</tr>
<tr>
<td>in txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
<tr>
<td>out rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.44.3.9 spiStartSendI

#define spiStartSendI(
    spip, n, txbuf
)

Value:

{ (spip)->state = SPI_ACTIVE; 
spi_lld_send(spip, n, txbuf); }

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

Precondition

A slave must have been selected using spiSelect() or spiSelectI().

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>spip</th>
<th>pointer to the SPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of words to send</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
7.44.3.10  spiStartReceiveI

```c
#define spiStartReceiveI(spip, n, rxbuf)
```

**Value:**

```c
{ (spip)->state = SPI_ACTIVE;
  spi_lld_receive(spip, n, rxbuf);
}
```

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

**Precondition**

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>spip</code>   pointer to the <code>SPIDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code>      number of words to receive</td>
</tr>
<tr>
<td>out</td>
<td><code>rxbuf</code>  the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.44.3.11  spiPolledExchange

```c
#define spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)
```

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.
Note

This API is implemented as a macro in order to minimize latency.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>spip</th>
<th>pointer to the SPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>frame</td>
<td>the data frame to send over the SPI bus</td>
</tr>
</tbody>
</table>

Returns

The received data frame from the SPI bus.

7.44.3.12 _spi_wakeup_isr

#define _spi_wakeup_isr (spip)

Value:

{
    osalSysLockFromISR();
    osalThreadResume(&(spip)->thread, MSG_OK);
    osalSysUnlockFromISR();
}

Wakes up the waiting thread.

Parameters

| in  | spip | pointer to the SPIDriver object |

Function Class:

Not an API, this function is for internal use only.

7.44.3.13 _spi_isr_code

#define _spi_isr_code (spip)

Value:

if ((spip)->config->end_cb) {
    (spip)->state = SPI_COMPLETE;
    (spip)->config->end_cb(spip);
    if ((spip)->state == SPI_COMPLETE)
        (spip)->state = SPI_READY;
    else
        (spip)->state = SPI_READY;
    _spi_wakeup_isr (spip);
    }

ChibiOS/HAL
Common ISR code when circular mode is not supported.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

\textbf{in} \quad \textbf{spip} \quad \textbf{pointer to the SPIDriver object}

Function Class:

Not an API, this function is for internal use only.

7.44.3.14 \_spi_isr_half_code

\textbf{#define} \_spi_isr_half_code ( \\
\quad \textbf{spip} )

\textbf{Value:}

\{ \\
\quad \textbf{if} \ ((\textbf{spip})->\textbf{config}->\textbf{end}\_\textbf{cb}) \ { \\
\quad\quad \textbf{(spip)}->\textbf{config}->\textbf{end}\_\textbf{cb}(\textbf{spip}); \} \\
\}

Half buffer filled ISR code in circular mode.

This code handles the portable part of the ISR code:

- Callback invocation.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

\textbf{in} \quad \textbf{spip} \quad \textbf{pointer to the SPIDriver object}
Function Class:
Not an API, this function is for internal use only.

7.44.3.15 _spi_isr_full_code

#define _spi_isr_full_code (spip)

Value:

{    if ((spip)->config->end_cb) {        (spip)->state = SPI_COMPLETE;        (spip)->config->end_cb(spip);        if ((spip)->state == SPI_COMPLETE)        (spip)->state = SPI_ACTIVE;    }
}

Full buffer filled ISR code in circular mode.

This code handles the portable part of the ISR code:

• Callback invocation.
• Driver state transitions.

Note
This macro is meant to be used in the low level drivers implementation only.

Parameters
in spip pointer to the SPIDriver object

Function Class:
Not an API, this function is for internal use only.

7.44.3.16 SPI_SUPPORTS_CIRCULAR

#define SPI_SUPPORTS_CIRCULAR TRUE

Circular mode support flag.
7.44.3.17 PLATFORM_SPI_USE_SPI1

#define PLATFORM_SPI_USE_SPI1 FALSE
SPI1 driver enable switch.
If set to TRUE the support for SPI1 is included.

Note
The default is FALSE.

7.44.3.18 spi_lld_driver_fields

#define spi_lld_driver_fields
Value:
/* Dummy field, it is not needed.*/
uint32_t dummy
Low level fields of the SPI driver structure.

7.44.3.19 spi_lld_config_fields

#define spi_lld_config_fields
Value:
/* Dummy configuration, it is not needed.*/
uint32_t dummy
Low level fields of the SPI configuration structure.

7.44.4 Typedef Documentation

7.44.4.1 SPIDriver

typedef struct hal_spi_driver SPIDriver
Type of a structure representing an SPI driver.

7.44.4.2 SPIConfig

typedef struct hal_spi_config SPIConfig
Type of a SPI driver configuration structure.

7.44.4.3 spicallback_t

typedef void(* spicallback_t) (SPIDriver *spip)
SPI notification callback type.
Parameters

in | *spip* | pointer to the SPI driver object triggering the callback

7.44.5 Enumeration Type Documentation

7.44.5.1 `spistate_t`

```c
enum spistate_t
```

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SPI_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>SPI_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>SPI_ACTIVE</td>
<td>Exchanging data.</td>
</tr>
<tr>
<td>SPI_COMPLETE</td>
<td>Asynchronous operation complete.</td>
</tr>
<tr>
<td>SPI_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SPI_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>SPI_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>SPI_ACTIVE</td>
<td>Exchanging data.</td>
</tr>
<tr>
<td>SPI_COMPLETE</td>
<td>Asynchronous operation complete.</td>
</tr>
</tbody>
</table>

7.44.6 Function Documentation

7.44.6.1 `spi_lld_init()`

```c
void spi_lld_init (  
  void )
```

Low level SPI driver initialization.
Function Class:
Not an API, this function is for internal use only.

### 7.44.6.2 spi_lld_start()

```c
msg_t spi_lld_start (SPIDriver * spip)
```

Configures and activates the SPI peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>spip</td>
<td>pointer to the SPIDriver object</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

### 7.44.6.3 spi_lld_stop()

```c
void spi_lld_stop (SPIDriver * spip)
```

Deactivates the SPI peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>spip</td>
<td>pointer to the SPIDriver object</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

### 7.44.6.4 spi_lld_select()

```c
void spi_lld_select (SPIDriver * spip)
```

Asserts the slave select signal and prepares for transfers.
7.44 SPIv1 Driver

Parameters

| in  | spip | pointer to the SPIDriver object |

Function Class:
Not an API, this function is for internal use only.

7.44.6.5 spi_lld_unselect()

void spi_lld_unselect (  
    SPIDriver * spip )

Deasserts the slave select signal.
The previously selected peripheral is unselected.

Parameters

| in  | spip | pointer to the SPIDriver object |

Function Class:
Not an API, this function is for internal use only.

7.44.6.6 spi_lld_ignore()

msg_t spi_lld_ignore (  
    SPIDriver * spip,  
    size_t n )

Ignores data on the SPI bus.
This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

Postcondition
At the end of the operation the configured callback is invoked.

Parameters

| in  | spip | pointer to the SPIDriver object |
| in  | n    | number of words to be ignored   |
Function Class:
Not an API, this function is for internal use only.

### 7.44.6.7 spi_lld_exchange()  

```c
msg_t spi_lld_exchange (  
    SPIDriver * spip,  
    size_t n,  
    const void * txbuf,  
    void * rxbuf )
```

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

**Postcondition**
At the end of the operation the configured callback is invoked.

**Note**
The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>spip</td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of words to be exchanged</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

### 7.44.6.8 spi_lld_send()  

```c
msg_t spi_lld_send (  
    SPIDriver * spip,  
    size_t n,  
    const void * txbuf )
```

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.
Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>spip</th>
<th>pointer to the SPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of words to send</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.44.6.9 spi_lld_receive()

msg_t spi_lld_receive ( 
    SPIDriver * spip,  
    size_t n,  
    void * rxbuf )

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>spip</th>
<th>pointer to the SPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of words to receive</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

### 7.44.6.10 spi_lld_abort()

```c
void spi_lld_abort (SPIDriver *spip)
```

Aborts the ongoing SPI operation, if any.

**Parameters**
- `in spip` pointer to the SPIDriver object

Function Class:
Not an API, this function is for internal use only.

### 7.44.6.11 spi_lld_polled_exchange()

```c
uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)
```

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

**Parameters**
- `in spip` pointer to the SPIDriver object
- `in frame` the data frame to send over the SPI bus

**Returns**
- The received data frame from the SPI bus.

Function Class:
Not an API, this function is for internal use only.

### 7.44.7 Variable Documentation
7.44 SPIv1 Driver

7.44.7.1 SPID1

SPI Driver SPID1

SPI1 driver identifier.
7.45 SPIv2 Driver

Generic SPI Driver.

7.45.1 Detailed Description

Generic SPI Driver.

This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

In order to use the SPI driver the `HAL_USE_SPI` option must be enabled in `halconf.h`.

7.45.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

The driver is not thread safe for performance reasons, if you need to access the SPI bus from multiple threads then use the `spiAcquireBus()` and `spiReleaseBus()` APIs in order to gain exclusive access.

Macros

- `#define SPI_SUPPORTS_CIRCULAR TRUE`
  
  Circular mode support flag.

- `#define SPI_SUPPORTS_SLAVE_MODE TRUE`
  
  Slave mode support flag.

- `#define spi_lld_driver_fields`
  
  Low level fields of the SPI driver structure.

- `#define spi_lld_config_fields`
  
  Low level fields of the SPI configuration structure.
Chip Select modes

- \#define SPI_SELECT_MODE_NONE
- \#define SPI_SELECT_MODE_PAD 1 /** @brief Legacy mode. */
- \#define SPI_SELECT_MODE_PORT 2 /** @brief Fastest mode. */
- \#define SPI_SELECT_MODE_LINE 3 /** @brief Packed mode. */
- \#define SPI_SELECT_MODE_LLD 4 /** @brief LLD-defined mode. */

SPI configuration options

- \#define SPI_USE_SYNCHRONIZATION FALSE
  Support for thread synchronization API.
- \#define SPI_USE_ASSERT_ON_ERROR TRUE
  Inserts an assertion on function errors before returning.
- \#define SPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the \texttt{spiAcquireBus()} and \texttt{spiReleaseBus()} APIs.
- \#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD
  Handling method for SPI CS line.

Macro Functions

- \#define spiIsBufferComplete(spip) ((bool)((spip)->state == SPI_COMPLETE))
  Buffer state.
- \#define spiSelectI(spip)
  Asserts the slave select signal and prepares for transfers.
- \#define spiUnselectI(spip)
  Deasserts the slave select signal.
- \#define spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)
  Exchanges one frame using a polled wait.
- \#define spiAbortI(spip) spiStopTransferI(spip, NULL)
  Compatibility API with SPI driver v1.
- \#define spiAbort(spip) spiStopTransfer(spip, NULL)
  Compatibility API with SPI driver v1.

Low level driver helper macros

- \#define __spi_wakeup_isr(spip, msg)
  Wakes up the waiting thread.
- \#define __spi_isr_complete_code(spip)
  Common ISR code in linear mode.
- \#define __spi_isr_half_code(spip)
  Half buffer filled ISR code in circular mode.
- \#define __spi_isr_full_code(spip)
  Full buffer filled ISR code in circular mode.
- \#define __spi_isr_error_code(spip, msg)
  ISR error reporting code.
PLATFORM configuration options

- 
  
  #define PLATFORM_SPI_USE_SPI1 FALSE
  
  SPI1 driver enable switch.

Typedefs

- 
  
  typedef struct hal_spi_driver SPIDriver
  
  Type of a structure representing an SPI driver.

- 
  
  typedef struct hal_spi_config SPIConfig
  
  Type of a SPI driver configuration structure.

- 
  
  typedef void(spicb_t)(SPIDriver *spip)
  
  SPI notification callback type.

Data Structures

- 
  
  struct hal_spi_config
  
  Driver configuration structure.

- 
  
  struct hal_spi_driver
  
  Structure representing an SPI driver.

Functions

- 
  
  void spi_lld_init (void)
  
  Low level SPI driver initialization.

- 
  
  msg_t spi_lld_start (SPIDriver *spip)
  
  Configures and activates the SPI peripheral.

- 
  
  void spi_lld_stop (SPIDriver *spip)
  
  Deactivates the SPI peripheral.

- 
  
  void spi_lld_select (SPIDriver *spip)
  
  Asserts the slave select signal and prepares for transfers.

- 
  
  void spi_lld_unselect (SPIDriver *spip)
  
  Deasserts the slave select signal.

- 
  
  msg_t spi_lld_ignore (SPIDriver *spip, size_t n)
  
  Ignores data on the SPI bus.

- 
  
  msg_t spi_lld_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  
  Exchanges data on the SPI bus.

- 
  
  msg_t spi_lld_send (SPIDriver *spip, size_t n, const void *txbuf)
  
  Sends data over the SPI bus.

- 
  
  msg_t spi_lld_receive (SPIDriver *spip, size_t n, void *rxbuf)
  
  Receives data from the SPI bus.

- 
  
  msg_t spi_lld_stop_transfer (SPIDriver *spip, size_t *sizep)
  
  Aborts the ongoing SPI operation, if any.

- 
  
  uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)
  
  Exchanges one frame using a polled wait.
Enumerations

- enum spistate_t {
  SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,
  SPI_COMPLETE = 4, SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2,
  SPI_ACTIVE = 3, SPI_COMPLETE = 4 } 

  Driver state machine possible states.

Variables

- SPIDriver SPID1
  
  SPI1 driver identifier.

7.45.3 Macro Definition Documentation

7.45.3.1 SPI_USE_SYNCHRONIZATION

#define SPI_USE_SYNCHRONIZATION FALSE

Support for thread synchronization API.

7.45.3.2 SPI_USE_ASSERT_ON_ERROR

#define SPI_USE_ASSERT_ON_ERROR TRUE

Inserts an assertion on function errors before returning.

7.45.3.3 SPI_USE_MUTUAL_EXCLUSION

#define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the spiAcquireBus() and spiReleaseBus() APIs.

Note

Disabling this option saves both code and data space.
7.45.3.4 SPI_SELECT_MODE

#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD

Handling method for SPI CS line.

Note
Disabling this option saves both code and data space.

7.45.3.5 spiIsBufferComplete

#define spiIsBufferComplete(
    spip
) ((bool)((spip)->state == SPI_COMPLETE))

Buffer state.

Note
This function is meant to be called from the SPI callback only.

Parameters

| in | spip | pointer to the SPI Driver object |

Returns
The buffer state.

Return values

| false | if the driver filled/sent the first half of the buffer. |
| true  | if the driver filled/sent the second half of the buffer. |

Function Class:
Special function, this function has special requirements see the notes.

7.45.3.6 spiSelectI

#define spiSelectI(
    spip
)

Value:

do {
    spi_lld_select(spip);
} while (false)

Asserts the slave select signal and prepares for transfers.
Parameters

in \textit{spip} pointer to the SPIDriver object

Function Class:

This is an \texttt{I-Class} API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.45.3.7 \texttt{spiUnselectI}

#define \texttt{spiUnselectI}(  
spip)

Value:

\begin{verbatim}
do {  
  spi_lld_unselect(spip);  
} while (false)
\end{verbatim}

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in \textit{spip} pointer to the SPIDriver object

Function Class:

This is an \texttt{I-Class} API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.45.3.8 \texttt{spiPolledExchange}

#define \texttt{spiPolledExchange}(  
  spip,  
  frame) spi_lld_polled_exchange(spip, frame)

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

Note

This API is implemented as a macro in order to minimize latency.
Module Documentation

Parameters

| in | spip | pointer to the SPIDriver object |
| in | frame | the data frame to send over the SPI bus |

Returns

The received data frame from the SPI bus.

7.45.3.9 spiAbortI

#define spiAbortI(
    spip ) spiStopTransferI(spip, NULL)

Compatibility API with SPI driver v1.

Parameters

| in | spip | pointer to the SPIDriver object |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.45.3.10 spiAbort

#define spiAbort(
    spip ) spiStopTransfer(spip, NULL)

Compatibility API with SPI driver v1.

Parameters

| in | spip | pointer to the SPIDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.45.3.11 __spi_wakeup_isr

```c
#define __spi_wakeup_isr(
    spip,
    msg )

Value:

\{ osalSysLockFromISR();
\}
osalThreadResumeI(&(spip)->sync_transfer, MSG_OK);
osalSysUnlockFromISR();
\}

Wakes up the waiting thread.

Parameters

| in  | spip | pointer to the SPI_Driver object |
| in  | msg  | wakeup message                  |

Function Class:

Not an API, this function is for internal use only.

7.45.3.12 __spi_isr_complete_code

```c
#define __spi_isr_complete_code(
    spip )

Value:

\{ if ((spip)->config->data_cb) { (spip)->state = SPI_COMPLETE; (spip)->config->data_cb(spip); if ((spip)->state == SPI_COMPLETE) (spip)->state = SPI_READY; } else { (spip)->state = SPI_READY; } __spi_wakeup_isr(spip, MSG_OK); \}

Common ISR code in linear mode.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.
Parameters

in spip pointer to the SPIDriver object

Function Class:
Not an API, this function is for internal use only.

7.45.3.13 __spi_isr_half_code

#define __spi_isr_half_code(
    spip )

Value:

if ((spip)->config->data_cb) {
    (spip)->config->data_cb(spip);
}

Half buffer filled ISR code in circular mode.

This code handles the portable part of the ISR code:

- Callback invocation.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in spip pointer to the SPIDriver object

Function Class:
Not an API, this function is for internal use only.

7.45.3.14 __spi_isr_full_code

#define __spi_isr_full_code(
    spip )

Value:

if ((spip)->config->data_cb) {

ChibiOS/HAL
Full buffer filled ISR code in circular mode.

This code handles the portable part of the ISR code:

- Callback invocation.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

### Parameters

| in | spip | pointer to the SPI Driver object |

Function Class:

Not an API, this function is for internal use only.

### 7.45.3.15 __spi_isr_error_code

```c
#define __spi_isr_error_code(spip, msg)  
   if ((spip)->config->error_cb) {  
      (spip)->config->error_cb(spip);  
   }  
   __spi_wakeup_isr(spip, msg); 
```

ISR error reporting code.

Note

This macro is meant to be used in the low level drivers implementation only.

### Parameters

| in | spip | pointer to the SPI Driver object |
| in | msg | error code |
Function Class:
Not an API, this function is for internal use only.

### 7.45.3.16 SPI_SUPPORTS_CIRCULAR

```c
#define SPI_SUPPORTS_CIRCULAR TRUE
```
Circular mode support flag.

### 7.45.3.17 SPI_SUPPORTS_SLAVE_MODE

```c
#define SPI_SUPPORTS_SLAVE_MODE TRUE
```
Slave mode support flag.

### 7.45.3.18 PLATFORM_SPI_USE_SPI1

```c
#define PLATFORM_SPI_USE_SPI1 FALSE
```
SPI1 driver enable switch.

If set to `TRUE` the support for SPI1 is included.

**Note**

The default is `FALSE`.

### 7.45.3.19 spi_lld_driver_fields

```c
#define spi_lld_driver_fields
```

**Value:**

```c
/* Dummy field, it is not needed.*/
uint32_t dummy
```

Low level fields of the SPI driver structure.
7.45 SPIv2 Driver

7.45.3.20 spi_lld_config_fields

#define spi_lld_config_fields

Value:
/* Dummy configuration, it is not needed. */
uint32_t dummy

Low level fields of the SPI configuration structure.

7.45.4 Typedef Documentation

7.45.4.1 SPIDriver

typedef struct hal_spi_driver SPIDriver

Type of a structure representing an SPI driver.

7.45.4.2 SPIConfig

typedef struct hal_spi_config SPIConfig

Type of a SPI driver configuration structure.

7.45.4.3 spicb_t

typedef void(* spicb_t) (SPIDriver *spip)

SPI notification callback type.

Parameters

| in  | spip | pointer to the SPIDriver object triggering the callback |

7.45.5 Enumeration Type Documentation

7.45.5.1 spistate_t

enum spistate_t
Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SPI_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>SPI_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>SPI_ACTIVE</td>
<td>Exchanging data.</td>
</tr>
<tr>
<td>SPI_COMPLETE</td>
<td>Asynchronous operation complete.</td>
</tr>
<tr>
<td>SPI_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SPI_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>SPI_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>SPI_ACTIVE</td>
<td>Exchanging data.</td>
</tr>
<tr>
<td>SPI_COMPLETE</td>
<td>Asynchronous operation complete.</td>
</tr>
</tbody>
</table>

7.45.6 Function Documentation

7.45.6.1 spi_lld_init()

```c
void spi_lld_init (    
    void )
```

Low level SPI driver initialization.

Function Class:

Not an API, this function is for internal use only.

7.45.6.2 spi_lld_start()

```c
msg_t spi_lld_start (    
    SPIDriver * spip )
```

Configures and activates the SPI peripheral.
Parameters

| in  | spip | pointer to the SPIDriver object |

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.

### 7.45.6.3 spi_lld_stop()

```c
void spi_lld_stop ( 
    SPIDriver * spip )
```

Deactivates the SPI peripheral.

Parameters

| in  | spip | pointer to the SPIDriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.45.6.4 spi_lld_select()

```c
void spi_lld_select ( 
    SPIDriver * spip )
```

Asserts the slave select signal and prepares for transfers.

Parameters

| in  | spip | pointer to the SPIDriver object |

Function Class:

Not an API, this function is for internal use only.
7.45.6.5 spi_lld_unselect()

```c
void spi_lld_unselect ( 
    SPIDriver * spip )
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

**Parameters**

| in   | spip | pointer to the SPIDriver object |

**Function Class:**

Not an API, this function is for internal use only.

7.45.6.6 spi_lld_ignore()

```c
msg_t spi_lld_ignore ( 
    SPIDriver * spip, 
    size_t n )
```

Ignores data on the SPI bus.

This synchronous function performs the transmission of a series of idle words on the SPI bus and ignores the received data.

**Precondition**

- In order to use this function the option `SPI_USE_SYNCHRONIZATION` must be enabled.

**Parameters**

| in   | spip | pointer to the SPIDriver object |
| in   | n    | number of words to be ignored |

**Returns**

The operation status.

**Function Class:**

Not an API, this function is for internal use only.
7.45 SPIv2 Driver

7.45.6.7 spi_lld_exchange()

```c
msg_t spi_lld_exchange ( 
    SPIDriver * spip, 
    size_t n, 
    const void * txbuf, 
    void * rxbuf )
```

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Note**

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>spip</th>
<th>pointer to the SPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of words to be exchanged</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Function Class:**

Not an API, this function is for internal use only.

7.45.6.8 spi_lld_send()

```c
msg_t spi_lld_send ( 
    SPIDriver * spip, 
    size_t n, 
    const void * txbuf )
```

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.
Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.
7.45 SPIv2 Driver

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>spip</th>
<th>pointer to the SPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of words to send</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.

7.45.6.9 spi_lld_receive()

msg_t spi_lld_receive ( 
    SPIDriver * spip, 
    size_t n, 
    void * rxbuf )

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>spip</th>
<th>pointer to the SPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of words to receive</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.
7.45.6.10  spi_lld_stop_transfer()

msg_t spi_lld_stop_transfer ( 
   SPIDriver * spip,
   size_t * sizep 
)

Aborts the ongoing SPI operation, if any.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>spip</td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td>out</td>
<td>sizep</td>
<td>pointer to the counter of frames not yet transferred or NULL</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.

7.45.6.11  spi_lld_polled_exchange()

uint16_t spi_lld_polled_exchange ( 
   SPIDriver * spip,
   uint16_t frame 
)

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>spip</td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td>frame</td>
<td>the data frame to send over the SPI bus</td>
</tr>
</tbody>
</table>

Returns

The received data frame from the SPI bus.

7.45.7  Variable Documentation
7.45.7.1 SPI1

SPI1 driver identifier.
7.46 ST Driver

Generic System Tick Driver.

7.46.1 Detailed Description

Generic System Tick Driver.

This module implements a system tick timer in order to support the underlying operating system.

Functions

- void stInit (void)
  
  ST Driver initialization.

- void stBind (void)
  
  Enables an alarm interrupt on the invoking core.

- systime_t stGetCounter (void)
  
  Returns the time counter value.

- void stStartAlarm (systime_t abstime)
  
  Starts the alarm zero.

- void stStopAlarm (void)
  
  Stops the alarm zero interrupt.

- void stSetAlarm (systime_t abstime)
  
  Sets the alarm zero time.

- systime_t stGetAlarm (void)
  
  Returns the alarm zero current time.

- bool stIsAlarmActive (void)
  
  Determines if the alarm zero is active.

- void stSetCallback (unsigned alarm, st_callback_t cb)
  
  Associates a callback to an alarm.

- void stBindAlarmN (unsigned alarm)
  
  Enables an alarm interrupt on the invoking core.

- void stStartAlarmN (unsigned alarm, systime_t abstime)
  
  Starts an additional alarm.

- void stStopAlarmN (unsigned alarm)
  
  Stops an additional alarm.

- void stSetAlarmN (unsigned alarm, systime_t abstime)
  
  Sets an additional alarm time.

- systime_t stGetAlarmN (unsigned alarm)
  
  Returns an additional alarm current time.

- bool stIsAlarmActiveN (unsigned alarm)
  
  Determines if the specified alarm is active.

- void st_lld_init (void)
  
  Low level ST driver initialization.

- static systime_t st_lld_get_counter (void)
  
  Returns the time counter value.

- static void st_lld_start_alarm (systime_t abstime)
  
  Starts the alarm.

- static void st_lld_stop_alarm (void)
7.46 ST Driver

Stops the alarm interrupt.

• static void st_lld_set_alarm (systime_t abstime)
  Sets the alarm time.
• static systime_t st_lld_get_alarm (void)
  Returns the current alarm time.
• static bool st_lld_is_alarm_active (void)
  Determines if the alarm is active.

Variables

• st_callback_t st_callbacks [ST_LLD_NUM_ALARMS]
  Callback pointers for each alarm.

7.46.2 Function Documentation

7.46.2.1 stInit()

void stInit {
  void
}

ST Driver initialization.

Note

This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.46.2.2 stBind()

void stBind {
  void
}

Enables an alarm interrupt on the invoking core.

Note

Must be called before any other alarm-related function.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.46.2.3 stGetCounter()

```c
systime_t stGetCounter ( void )
```

Returns the time counter value.

**Note**

This functionality is only available in free running mode, the behaviour in periodic mode is undefined.

**Returns**

The counter value.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
stGetCounter -> st_lld_get_counter
```

7.46.2.4 stStartAlarm()

```c
void stStartAlarm ( systime_t abstime )
```

Starts the alarm zero.

**Note**

Makes sure that no spurious alarms are triggered after this call.

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

**Parameters**

- `abstime` the time to be set for the first alarm
7.46 ST Driver

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
图
```

7.46.2.5 stStopAlarm()

```c
void stStopAlarm (  
    void  )
```

Stops the alarm zero interrupt.

Note
This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
图
```

7.46.2.6 stSetAlarm()

```c
void stSetAlarm (  
    systime_t abstime  )
```

Sets the alarm zero time.

Note
This functionality is only available in free running mode, the behavior in periodic mode is undefined.
Parameters

| in | abstime | the time to be set for the next alarm |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
7.46.2.7 stGetAlarm()

systime_t stGetAlarm ( void )

Returns the alarm zero current time.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Returns

The currently set alarm time.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:
```
7.46.2.8 stIsAlarmActive()

```c
bool stIsAlarmActive ( 
    void 
)
```

Determined if the alarm zero is active.

Returns

The alarm status.

Return values

| false  | if the alarm is not active. |
| true   | is the alarm is active     |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
stIsAlarmActive  st_lld_is_alarm_active
```

7.46.2.9 stSetCallback()

```c
void stSetCallback ( 
    unsigned alarm, 
    st_callback_t cb )
```

Associates a callback to an alarm.

Note

Makes sure that no spurious alarms are triggered after this call.

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>alarm</th>
<th>alarm channel number (0..ST_LLD_NUM_ALARMS-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
<td>alarm callback or NULL</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.46.2.10  stBindAlarmN()

void stBindAlarmN (  
    unsigned alarm  )

Enables an alarm interrupt on the invoking core.

Note

Must be called before any other alarm-related function.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>alarm</td>
<td>alarm channel number (0..ST_LLD_NUM_ALARMS-1)</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.46.2.11  stStartAlarmN()

void stStartAlarmN (  
    unsigned alarm,  
    systime_t abstime  )

Starts an additional alarm.

Note

Makes sure that no spurious alarms are triggered after this call.

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>alarm</td>
<td>alarm channel number (0..ST_LLD_NUM_ALARMS-1)</td>
</tr>
<tr>
<td>in</td>
<td>abstime</td>
<td>the time to be set for the first alarm</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.46.2.12  stStopAlarmN()

void stStopAlarmN (  
    unsigned alarm  )

Stops an additional alarm.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Parameters

| in  | alarm | alarm channel number (0..ST_LLD_NUM_ALARMS-1) |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.46.2.13  stSetAlarmN()

void stSetAlarmN (  
    unsigned alarm,  
    systime_t abstime  )

Sets an additional alarm time.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Parameters

| in  | alarm | alarm channel number (0..ST_LLD_NUM_ALARMS-1) |
| in  | abstime | the time to be set for the next alarm |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.46.2.14  stGetAlarmN()

```c
systime_t stGetAlarmN (unsigned alarm)
```

Returns an additional alarm current time.

**Note**

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

**Parameters**

- **in**: `alarm` — alarm channel number (0..ST_LLD_NUM_ALARMS-1)

**Returns**

The currently set alarm time.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.46.2.15  stIsAlarmActiveN()

```c
bool stIsAlarmActiveN (unsigned alarm)
```

Determines if the specified alarm is active.

**Parameters**

- **in**: `alarm` — alarm channel number (0..ST_LLD_NUM_ALARMS-1)

**Returns**

The alarm status.

**Return values**

- `false` if the alarm is not active.
- `true` if the alarm is active

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.46.2.16  st_lld_init()

```c
void st_lld_init (  
   void  )
```

Low level ST driver initialization.

Function Class:
Not an API, this function is for internal use only.

7.46.2.17  st_lld_get_counter()

```c
static systime_t st_lld_get_counter (  
   void  ) [inline], [static]
```

Returns the time counter value.

Returns
The counter value.

Function Class:
Not an API, this function is for internal use only.

7.46.2.18  st_lld_start_alarm()

```c
static void st_lld_start_alarm (  
   systime_t abstime  ) [inline], [static]
```

Starts the alarm.

Note
Makes sure that no spurious alarms are triggered after this call.

Parameters

| in | abstime | the time to be set for the first alarm |
Function Class:
    Not an API, this function is for internal use only.

7.46.2.19  st_lld_stop_alarm()

static void st_lld_stop_alarm (  
    void ) [inline], [static]

Stops the alarm interrupt.

Function Class:
    Not an API, this function is for internal use only.

7.46.2.20  st_lld_set_alarm()

static void st_lld_set_alarm (  
    systime_t abstime ) [inline], [static]

Sets the alarm time.

Parameters

|   in   | abstime | the time to be set for the next alarm |

Function Class:
    Not an API, this function is for internal use only.

7.46.2.21  st_lld_get_alarm()

static systime_t st_lld_get_alarm (  
    void ) [inline], [static]

Returns the current alarm time.

Returns

    The currently set alarm time.

Function Class:
    Not an API, this function is for internal use only.
7.46.2.22  st_lld_is_alarm_active()

static bool st_lld_is_alarm_active (  
    void ) [inline], [static]

Determines if the alarm is active.

Returns
The alarm status.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if the alarm is not active.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>is the alarm is active</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.46.3  Variable Documentation

7.46.3.1  st_callbacks

st_callback_t st_callbacks[ST_LLD_NUM_ALARMS]

Callback pointers for each alarm.

Note
If some alarms have static callbacks defined in the LLD then some of the pointers might be unused (never called through).
7.47 TRNG Driver

Generic True Random Numbers Generator Driver.

7.47.1 Detailed Description

Generic True Random Numbers Generator Driver.

This module implements a generic TRNG driver.

Precondition

In order to use the TRNG driver the `HAL_USE_TRNG` option must be enabled in `halconf.h`.

Macros

- `#define trng_lld_driver_fields`
  
  `Low level fields of the TRNG driver structure.`

- `#define trng_lld_config_fields`
  
  `Low level fields of the TRNG configuration structure`.

PLATFORM configuration options

- `#define PLATFORM_TRNG_USE_TRNG1 FALSE`
  
  `TRNGD1 driver enable switch.`

Typedefs

- `typedef struct hal_trng_driver TRNGDriver`
  
  `Type of a structure representing a TRNG driver.`

- `typedef struct hal_trng_config TRNGConfig`
  
  `Driver configuration structure.`

Data Structures

- `struct hal_trng_config`
  
  `Driver configuration structure`.

- `struct hal_trng_driver`
  
  `Structure representing a TRNG driver.`
Functions

- **void trngInit (void)**
  TRNG Driver initialization.
- **void trngObjectInit (TRNGDriver *trngp)**
  Initializes the standard part of a TRNGDriver structure.
- **msg_t trngStart (TRNGDriver *trngp, const TRNGConfig *config)**
  Configures and activates the TRNG peripheral.
- **void trngStop (TRNGDriver *trngp)**
  Deactivates the TRNG peripheral.
- **bool trngGenerate (TRNGDriver *trngp, size_t size, uint8_t *out)**
  True random numbers generator.
- **void trng_lld_init (void)**
  Low level TRNG driver initialization.
- **void trng_lld_start (TRNGDriver *trngp)**
  Configures and activates the TRNG peripheral.
- **void trng_lld_stop (TRNGDriver *trngp)**
  Deactivates the TRNG peripheral.
- **bool trng_lld_generate (TRNGDriver *trngp, size_t size, uint8_t *out)**
  True random numbers generator.

Enumerations

- **enum trngstate_t { TRNG_UNINIT = 0, TRNG_STOP = 1, TRNG_READY = 2, TRNG_RUNNING = 3 }**
  Driver state machine possible states.

Variables

- **TRNGDriver TRNGD1**
  TRNGD1 driver identifier.

7.47.2 Macro Definition Documentation

7.47.2.1 PLATFORM_TRNG_USE_TRNG1

#define PLATFORM_TRNG_USE_TRNG1 FALSE

TRNGD1 driver enable switch.

If set to TRUE the support for TRNGD1 is included.

Note

The default is FALSE.
7.47.2.2 trng_lld_driver_fields

#define trng_lld_driver_fields

Value:
/* Dummy field, it is not needed. */
uint32_t dummy

Low level fields of the TRNG driver structure.

7.47.2.3 trng_lld_config_fields

#define trng_lld_config_fields

Value:
/* Dummy configuration, it is not needed. */
uint32_t dummy

Low level fields of the TRNG configuration structure.

7.47.3 Typedef Documentation

7.47.3.1 TRNGDriver

typedef struct hal_trng_driver TRNGDriver

Type of a structure representing a TRNG driver.

7.47.3.2 TRNGConfig

typedef struct hal_trng_config TRNGConfig

Driver configuration structure.

Note
It could be empty on some architectures.

7.47.4 Enumeration Type Documentation

7.47.4.1 trngstate_t

enum trngstate_t

Driver state machine possible states.
7.47 TRNG Driver

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRNG_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>TRNG_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>TRNG_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>TRNG_RUNNING</td>
<td>Generating random number.</td>
</tr>
</tbody>
</table>

7.47.5 Function Documentation

7.47.5.1 trngInit()

```c
void trngInit (  
    void  )
```

TRNG Driver initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
trngInit -> trng_lld_init -> trngoObjectInit
```

7.47.5.2 trngObjectInit()

```c
void trngObjectInit (  
    TRNGDriver * trngp  )
```

Initializes the standard part of a TRNGDriver structure.
Parameters

| out | trngp | pointer to the TRNGDriver object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.47.5.3 trngStart()

```c
msg_t trngStart (  
    TRNGDriver * trnp,  
    const TRNGConfig * config )
```

Configures and activates the TRNG peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>trngp</th>
<th>pointer to the TRNGDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the TRNGConfig object or NULL for default configuration</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    osalSysLock
      
      trngStart
        
      trng_lld_start
```

ChibiOS/HAL
7.47.5.4 trngStop()

```c
void trngStop (  
    TRNGDriver * trnp )
```

Deactivates the TRNG peripheral.
7.47.5.5 trngGenerate()

bool trngGenerate (\
    TRNGDriver * trngp, \
    size_t size, \
    uint8_t * out )

True random numbers generator.

Note
The function is blocking and likely performs polled waiting inside the low level implementation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>trngp</th>
<th>pointer to the TRNGDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of output buffer</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>output buffer</td>
</tr>
</tbody>
</table>

Returns
The operation status.
Return values

| false | if a random number has been generated. |
| true  | if an HW error occurred. |

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
trngGenerate → trng_lld_generate
```

### 7.47.5.6 trng_lld_init()

```c
void trng_lld_init ( void )
```

Low level TRNG driver initialization.

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
trng_lld_init → trngObjectInit
```

### 7.47.5.7 trng_lld_start()

```c
void trng_lld_start ( TRNGDriver * trngp )
```

Configures and activates the TRNG peripheral.
Module Documentation

7.47.5.8 trng_lld_stop()

```c
void trng_lld_stop (
    TRNGDriver * trngp )
```

Deactivates the TRNG peripheral.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>trngp</td>
<td>pointer to the TRNGDriver object</td>
</tr>
</tbody>
</table>

Function Class:
Not an API, this function is for internal use only.

7.47.5.9 trng_lld_generate()

```c
bool trng_lld_generate (  
    TRNGDriver * trngp,  
    size_t size,  
    uint8_t * out )
```

True random numbers generator.

Note
The function is blocking and likely performs polled waiting inside the low level implementation.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>trngp</td>
<td>pointer to the TRNGDriver object</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of output buffer</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>output buffer</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if a random number has been generated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if an HW error occurred.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.47.6 Variable Documentation

7.47.6.1 TRNGD1

TRNGDriver TRNGD1

TRNGD1 driver identifier.
7.48 UART Driver

Generic UART Driver.

7.48.1 Detailed Description

Generic UART Driver.

This driver abstracts a generic UART (Universal Asynchronous Receiver Transmitter) peripheral, the API is designed to be:

- Unbuffered and copy-less, transfers are always directly performed from/to the application-level buffers without extra copy operations.
- Asynchronous, the API is always non blocking.
- Callbacks capable, operations completion and other events are notified using callbacks.

Special hardware features like deep hardware buffers, DMA transfers are hidden to the user but fully supportable by the low level implementations.

This driver model is best used where communication events are meant to drive an higher level state machine, as example:

- RS485 drivers.
- Multipoint network drivers.
- Serial protocol decoders.

If your application requires a synchronous buffered driver then the Serial Driver should be used instead.

Precondition

In order to use the UART driver the HAL_USE_UART option must be enabled in halconf.h.

7.48.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).
7.48.2.1 Transmitter sub State Machine

The follow diagram describes the transmitter state machine, this diagram is valid while the driver is in the UART ← T_READY state. This state machine is automatically reset to the TX_IDLE state each time the driver enters the UART_READY state.

![Transmitter State Machine Diagram]

7.48.2.2 Receiver sub State Machine

The follow diagram describes the receiver state machine, this diagram is valid while the driver is in the UART ←_READY state. This state machine is automatically reset to the RX_IDLE state each time the driver enters the UARTREADY state.

![Receiver State Machine Diagram]

UART status flags

- #define UART_NO_ERROR 0
  No pending conditions.
• \#define UART_PARITY_ERROR 4
  Parity error happened.

• \#define UART_FRAMING_ERROR 8
  Framing error happened.

• \#define UART_OVERRUN_ERROR 16
  Overflow happened.

• \#define UART_NOISE_ERROR 32
  Noise on the line.

• \#define UART_BREAK_DETECTED 64
  Break detected.

UART error conditions
  • \#define UART_ERR_NOT_ACTIVE (size_t)-1

UART configuration options
  • \#define UART_USE_WAIT FALSE
    Enables synchronous APIs.
  • \#define UART_USE_MUTUAL_EXCLUSION FALSE
    Enables the uartAcquireBus() and uartReleaseBus() APIs.

Low level driver helper macros
  • \#define _uart_wakeup_tx1_isr(uartp)
    Wakes up the waiting thread in case of early TX complete.
  • \#define _uart_wakeup_tx2_isr(uartp)
    Wakes up the waiting thread in case of late TX complete.
  • \#define _uart_wakeup_rx_complete_isr(uartp)
    Wakes up the waiting thread in case of RX complete.
  • \#define _uart_wakeup_rx_error_isr(uartp)
    Wakes up the waiting thread in case of RX error.
  • \#define _uart_wakeup_rx_cm_isr(uartp)
    Wakes up the waiting thread in case of RX character match.
  • \#define _uart_wakeup_rx_timeout_isr(uartp)
    Wakes up the waiting thread in case of RX timeout.
  • \#define _uart_tx1_isr_code(uartp)
    Common ISR code for early TX.
  • \#define _uart_tx2_isr_code(uartp)
    Common ISR code for late TX.
  • \#define _uart_rx_complete_isr_code(uartp)
    Common ISR code for RX complete.
  • \#define _uart_rx_error_isr_code(uartp, errors)
    Common ISR code for RX error.
  • \#define _uart_rx_idle_code(uartp)
    Common ISR code for RX on idle.
  • \#define _uart_timeout_isr_code(uartp)
    Timeout ISR code for receiver.
  • \#define _uart_rx_char_match_isr_code(uartp)
    Character match ISR code for receiver.
PLATFORM configuration options

• #define PLATFORM_UART_USE_UART1 FALSE
  UART driver enable switch.

Typedefs

• typedef uint32_t uartflags_t
  UART driver condition flags type.
• typedef struct hal_uart_driver UARTDriver
  Type of structure representing an UART driver.
• typedef void(*uartcb_t)(UARTDriver *uartp)
  Generic UART notification callback type.
• typedef void(*uartccb_t)(UARTDriver *uartp, uint16_t c)
  Character received UART notification callback type.
• typedef void(*uartecb_t)(UARTDriver *uartp, uartflags_t e)
  Receive error UART notification callback type.
• typedef struct hal_uart_config UARTConfig
  Driver configuration structure.

Data Structures

• struct hal_uart_config
  Driver configuration structure.
• struct hal_uart_driver
  Structure representing an UART driver.

Functions

• void uartInit (void)
  UART Driver initialization.
• void uartObjectInit (UARTDriver *uartp)
  Initializes the standard part of a UARTDriver structure.
• msg_t uartStart (UARTDriver *uartp, const UARTConfig *config)
  Configures and activates the UART peripheral.
• void uartStop (UARTDriver *uartp)
  Deactivates the UART peripheral.
• void uartStartSend (UARTDriver *uartp, size_t n, const void *txbuf)
  Starts a transmission on the UART peripheral.
• void uartStartSend (UARTDriver *uartp, size_t n, const void *txbuf)
  Starts a transmission on the UART peripheral.
• size_t uartStopSend (UARTDriver *uartp)
  Stops any ongoing transmission.
• size_t uartStopSend (UARTDriver *uartp)
  Stops any ongoing transmission.
• void uartStartReceive (UARTDriver *uartp, size_t n, void *rbuf)
  Starts a receive operation on the UART peripheral.
• void uartStartReceive (UARTDriver *uartp, size_t n, void *rbuf)
  Starts a receive operation on the UART peripheral.
• size_t uartStopReceive (UARTDriver *uartp)
  Stops any ongoing receive operation.
• size_t uartStopReceiveI (UARTDriver *uartp)
  Stops any ongoing receive operation.
• msg_t uartSendTimeout (UARTDriver +uartp, size_t *np, const void *txbuf, sysinterval_t timeout)
  Performs a transmission on the UART peripheral.
• msg_t uartSendFullTimeout (UARTDriver +uartp, size_t *np, const void *txbuf, sysinterval_t timeout)
  Performs a transmission on the UART peripheral.
• msg_t uartReceiveTimeout (UARTDriver +uartp, size_t *np, void *rxbuf, sysinterval_t timeout)
  Performs a receive operation on the UART peripheral.
• void uartAcquireBus (UARTDriver *uartp)
  Gains exclusive access to the UART bus.
• void uartReleaseBus (UARTDriver *uartp)
  Releases exclusive access to the UART bus.
• void uart_lld_init (void)
  Low level UART driver initialization.
• void uart_lld_start (UARTDriver *uartp)
  Configures and activates the UART peripheral.
• void uart_lld_stop (UARTDriver *uartp)
  Deactivates the UART peripheral.
• void uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)
  Starts a transmission on the UART peripheral.
• size_t uart_lld_stop_send (UARTDriver *uartp)
  Stops any ongoing transmission.
• void uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rxbuf)
  Starts a receive operation on the UART peripheral.
• size_t uart_lld_stop_receive (UARTDriver *uartp)
  Stops any ongoing receive operation.

Enumerations

• enum uartstate_t { UART_UNINIT = 0, UART_STOP = 1, UART_READY = 2 }
  Driver state machine possible states.
• enum uartxstate_t { UART_TX_IDLE = 0, UART_TX_ACTIVE = 1, UART_TX_COMPLETE = 2 }
  Transmitter state machine states.
• enum uartrxstate_t { UART_RX_IDLE = 0, UART_RX_ACTIVE = 1, UART_RX_COMPLETE = 2 }
  Receiver state machine states.

Variables

• UARTDriver UARTD1
  UART1 driver identifier.

7.48.3  Macro Definition Documentation
7.48 UART Driver

7.48.3.1 UART_NO_ERROR

#define UART_NO_ERROR 0

No pending conditions.

7.48.3.2 UART_PARITY_ERROR

#define UART_PARITY_ERROR 4

Parity error happened.

7.48.3.3 UART_FRAMING_ERROR

#define UART_FRAMING_ERROR 8

Framing error happened.

7.48.3.4 UART_OVERRUN_ERROR

#define UART_OVERRUN_ERROR 16

Overflow happened.

7.48.3.5 UART_NOISE_ERROR

#define UART_NOISE_ERROR 32

Noise on the line.

7.48.3.6 UART_BREAK_DETECTED

#define UART_BREAK_DETECTED 64

Break detected.
### 7.48.3.7 UART_USE_WAIT

```c
#define UART_USE_WAIT FALSE
```

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.

### 7.48.3.8 UART_USE_MUTUAL_EXCLUSION

```c
#define UART_USE_MUTUAL_EXCLUSION FALSE
```

Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

**Note**

Disabling this option saves both code and data space.

### 7.48.3.9 _uart_wakeup_tx1_isr

```c
#define _uart_wakeup_tx1_isr(uartp)
```

**Value:**

```c
if ((uartp)->early == true) {
    osalSysLockFromISR();
    osalThreadResumeI(&(uartp)->threadtx, MSG_OK);
    osalSysUnlockFromISR();
}
```

Wakes up the waiting thread in case of early TX complete.

**Parameters**

- **in** `uartp` pointer to the UARTDriver object

**Function Class:**

Not an API, this function is for internal use only.
7.48 UART Driver

7.48.3.10 _uart_wakeup_tx2_isr

#define _uart_wakeup_tx2_isr(
   uartp )

Value:
{
   if ((uartp)->early == false) {
      osalSysLockFromISR();
      osalThreadResumeI(&(uartp)->threadtx, MSG_OK);
      osalSysUnlockFromISR();
   }
}

Wakes up the waiting thread in case of late TX complete.

Parameters

   in  uartp  pointer to the UARTDriver object

Function Class:

   Not an API, this function is for internal use only.

7.48.3.11 _uart_wakeup_rx_complete_isr

#define _uart_wakeup_rx_complete_isr(
   uartp )

Value:
{
   osalSysLockFromISR();
   osalThreadResumeI(&(uartp)->threadrx, MSG_OK);
   osalSysUnlockFromISR();
}

Wakes up the waiting thread in case of RX complete.

Parameters

   in  uartp  pointer to the UARTDriver object

Function Class:

   Not an API, this function is for internal use only.

7.48.3.12 _uart_wakeup_rx_error_isr

#define _uart_wakeup_rx_error_isr(
   uartp )

ChibiOS/HAL
Value:
{
  osalSysLockFromISR();
  osalThreadResumeI(&(uartp)->threadrx, MSG_RESET);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread in case of RX error.

Parameters

| in | uartp | pointer to the UARTDriver object |

Function Class:
Not an API, this function is for internal use only.

7.48.3.13 _uart_wakeup_rx_cm_isr

#define _uart_wakeup_rx_cm_isr(
    uartp
)

Value:
{
  osalSysLockFromISR();
  osalThreadResumeI(&(uartp)->threadrx, MSG_TIMEOUT);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread in case of RX character match.

Parameters

| in | uartp | pointer to the UARTDriver object |

Function Class:
Not an API, this function is for internal use only.

7.48.3.14 _uart_wakeup_rx_timeout_isr

#define _uart_wakeup_rx_timeout_isr(
    uartp
)

Value:
{
  osalSysLockFromISR();
  osalThreadResumeI(&(uartp)->threadrx, MSG_TIMEOUT);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread in case of RX timeout.
Parameters

```
in  uartp  pointer to the UARTDriver object
```

Function Class:

Not an API, this function is for internal use only.

### 7.48.3.15 _uart_tx1_isr_code

```c
#define _uart_tx1_isr_code(uartp)  
\{
    (uartp)->txstate = UART_TX_COMPLETE;  
    if ((uartp)->config->txend1_cb != NULL) {  
        (uartp)->config->txend1_cb(uartp);  
    }  
    if ((uartp)->txstate == UART_TX_COMPLETE) {  
        (uartp)->txstate = UART_TX_IDLE;  
    }  
    _uart_wakeup_tx1_isr(uartp);  
}
```

Common ISR code for early TX.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

Parameters

```
in  uartp  pointer to the UARTDriver object
```

Function Class:

Not an API, this function is for internal use only.

### 7.48.3.16 _uart_tx2_isr_code

```c
#define _uart_tx2_isr_code(uartp)  
```

ChibiOS/HAL
Common ISR code for late TX.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

| in  | uartp | pointer to the UARTDriver object |

Function Class:

Not an API, this function is for internal use only.

7.48.3.17 _uart_rx_complete_isr_code

#define _uart_rx_complete_isr_code{
    uartp
}

Common ISR code for RX complete.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.
Parameters

| in | uartp | pointer to the UARTDriver object |

Function Class:

Not an API, this function is for internal use only.

### 7.48.3.18 _uart_rx_error_isr_code

```c
#define _uart_rx_error_isr_code(uartp, errors)
if ((uartp)->config->rxerr_cb != NULL) {
    (uartp)->config->rxerr_cb(uartp, errors);
}
_uart_wakeup_rx_error_isr(uartp);
```

Common ISR code for RX error.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

Parameters

| in | uartp | pointer to the UARTDriver object |
| in | errors | mask of errors to be reported |

Function Class:

Not an API, this function is for internal use only.

### 7.48.3.19 _uart_rx_idle_code

```c
#define _uart_rx_idle_code(uartp)
```

ChibiOS/HAL
Value:
{
  if ((uartp)->config->rxchar_cb != NULL) {
    (uartp)->config->rxchar_cb(uartp, (uartp)->rxbuf);
  }
}

Common ISR code for RX on idle.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note
This macro is meant to be used in the low level drivers implementation only.

Parameters
in uartp pointer to the UARTDriver object

Function Class:
Not an API, this function is for internal use only.

7.48.3.20 _uart_timeout_isr_code

#define _uart_timeout_isr_code(
uartp )

Value:
{
  if ((uartp)->config->timeout_cb != NULL) {
    (uartp)->config->timeout_cb(uartp);
  } _uart_wakeup_rx_timeout_isr(uartp);
}

Timeout ISR code for receiver.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note
This macro is meant to be used in the low level drivers implementation only.
Parameters

| in | uartp | pointer to the UARTDriver object |

Function Class:

Not an API, this function is for internal use only.

7.48.3.21 _uart_rx_char_match_isr_code

#define _uart_rx_char_match_isr_code(uartp)

Value:

```c
if ((uartp)->config->rx_cm_cb != NULL) {
  (uartp)->config->rx_cm_cb(uartp);
}
_uart_wakeup_rx_cm_isr(uartp);
```

Character match ISR code for receiver.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

| in | uartp | pointer to the UARTDriver object |

Function Class:

Not an API, this function is for internal use only.

7.48.3.22 PLATFORM_UART_USE_UART1

#define PLATFORM_UART_USE_UART1 FALSE

UART driver enable switch.

If set to TRUE the support for UART1 is included.
7.48.4 Typedef Documentation

7.48.4.1 uartflags_t

typedef uint32_t uartflags_t

UART driver condition flags type.

7.48.4.2 UARTDriver

typedef struct hal_uart_driver UARTDriver

Type of structure representing an UART driver.

7.48.4.3 uartcb_t

typedef void(* uartcb_t) (UARTDriver *uartp)

Generic UART notification callback type.

Parameters

| in | uartp | pointer to the UARTDriver object |

7.48.4.4 uartccb_t

typedef void(* uartccb_t) (UARTDriver *uartp, uint16_t c)

Character received UART notification callback type.

Parameters

| in | uartp | pointer to the UARTDriver object triggering the callback |
| in | c     | received character |
7.48.4.5 uartcb_t

typedef void(* uartcb_t) (UARTDriver *uartp, uartflags_t e)

Receive error UART notification callback type.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>uartp</th>
<th>pointer to the UARTDriver object triggering the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>e</td>
<td>receive error mask</td>
</tr>
</tbody>
</table>

7.48.4.6 UARTConfig

typedef struct hal_uart_config UARTConfig

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

7.48.5 Enumeration Type Documentation

7.48.5.1 uartstate_t

enum uartstate_t

Driver state machine possible states.

Enumerator

<table>
<thead>
<tr>
<th>UART_UNINIT</th>
<th>Not initialized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>UART_READY</td>
<td>Ready.</td>
</tr>
</tbody>
</table>
7.48.5.2 uarttxstate_t

enum uarttxstate_t

Transmitter state machine states.

Enumerator

<table>
<thead>
<tr>
<th>UART_TX_IDLE</th>
<th>Not transmitting.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_TX_ACTIVE</td>
<td>Transmitting.</td>
</tr>
<tr>
<td>UART_TX_COMPLETE</td>
<td>Buffer complete.</td>
</tr>
</tbody>
</table>

7.48.5.3 uartrxstate_t

enum uartrxstate_t

Receiver state machine states.

Enumerator

<table>
<thead>
<tr>
<th>UART_RX_IDLE</th>
<th>Not receiving.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_RX_ACTIVE</td>
<td>Receiving.</td>
</tr>
<tr>
<td>UART_RX_COMPLETE</td>
<td>Buffer complete.</td>
</tr>
</tbody>
</table>

7.48.6 Function Documentation

7.48.6.1 uartInit()

void uartInit (  
    void )

UART Driver initialization.

Note

This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.
Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call Graph](image)

### 7.48.6.2 uartObjectInit()

```c
void uartObjectInit ( UARTDriver * uartp )
```

Initializes the standard part of a UARTDriver structure.

**Parameters**

| out | uartp | pointer to the UARTDriver object |

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call Graph](image)

### 7.48.6.3 uartStart()

```c
msg_t uartStart ( UARTDriver * uartp ,
                 const UARTConfig * config )
```

Configures and activates the UART peripheral.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>uartp</th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the UARTConfig object</td>
</tr>
</tbody>
</table>

### Returns

The operation status.

### Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](call_graph.png)

#### 7.48.6.4 uartStop()

```c
void uartStop (UARTDriver * uartp )
```

Deactivates the UART peripheral.

### Parameters

| in  | uartp   | pointer to the UARTDriver object |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
uartStop
  osalSysLock
osalSysUnlock
uart_lld_stop
```

### 7.48.6.5 uartStartSend()

```c
void uartStartSend (UARTDriver *uartp,
                   size_t n,
                   const void *txbuf )
```

Starts a transmission on the UART peripheral.

**Note**

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>uartp</code> pointer to the UARTDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code> number of data frames to send</td>
</tr>
<tr>
<td>in</td>
<td><code>txbuf</code> the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
uartStartSend
  osalSysLock
osalSysUnlock
uart_lld_start_send
```

### 7.48.6.6 uartStartSendI()

```c
void uartStartSendI (
    UARTDriver * uartp,
    size_t n,
    const void * txbuf )
```

Starts a transmission on the UART peripheral.

**Note**

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

This function has to be invoked from a lock zone.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>uartp pointer to the UARTDriver object</td>
</tr>
<tr>
<td>in</td>
<td>n number of data frames to send</td>
</tr>
<tr>
<td>in</td>
<td>txbuf the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

7.48.6.7 uartStopSend()

size_t uartStopSend (UARTDriver * uartp )

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.

Parameters

| in | uartp | pointer to the UARTDriver object |

Returns

The number of data frames not transmitted by the stopped transmit operation.

Return values

UART_ERR_NOT_ACTIVE if there was no transmit operation in progress.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
uartStopSend
osalSysLock
uart_lld_stop_send
```

### 7.48.6.8 uartStopSendI()

```c
size_t uartStopSendI ( 
    UARTDriver * uartp )
```

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.
This function has to be invoked from a lock zone.

Parameters

- **in uartp** pointer to the UARTDriver object

Returns

The number of data frames not transmitted by the stopped transmit operation.

Return values

- `UART_ERR_NOT_ACTIVE` if there was no transmit operation in progress.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
Here is the call graph for this function:

```c
void uartStartReceive (UARTDriver * uartp, size_t n, void * rxbuf )
```

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>uarp</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
<tr>
<td>in</td>
<td>rxbuf</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

```
osalSysLock
uartStartReceive
osalSysUnlock
uart_lld_start_receive
```

---

### uartStartReceive()

```c
void uartStartReceiveI (UARTDriver * uartp, size_t n, void * rxbuf )
```

Starts a receive operation on the UART peripheral.

**Note**

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

This function has to be invoked from a lock zone.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>uartp</strong> pointer to the UARTDriver object</td>
</tr>
<tr>
<td>in</td>
<td><strong>n</strong> number of data frames to receive</td>
</tr>
<tr>
<td>out</td>
<td><strong>rxbuf</strong> the pointer to the receive buffer</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
uartStartReceiveI <- uart_lld_start_receive
```

7.48.6.11  uartStopReceive()

```c
size_t uartStopReceive (UARTDriver * uartp)
```

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.

Parameters

- **uartp**: pointer to the UARTDriver object

Returns

The number of data frames not received by the stopped receive operation.

Return values

- UART_ERR_NOT_ACTIVE if there was no receive operation in progress.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

7.48.6.12 uartStopReceive()

```c
size_t uartStopReceiveI (UARTDriver * uartp)
```

Stops any ongoing receive operation.

**Note**

Stopping a receive operation also suppresses the receive callbacks.
This function has to be invoked from a lock zone.

**Parameters**

| in | uartp | pointer to the UARTDriver object |

**Returns**

The number of data frames not received by the stopped receive operation.

**Return values**

- **UART_ERR_NOT_ACTIVE** if there was no receive operation in progress.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
Here is the call graph for this function:

```
uartStopReceiveI -> uart_lld_stop_receive
```

### 7.48.6.13 `uartSendTimeout()`

```c
msg_t uartSendTimeout (UARTDriver * uartp,
                       size_t * np,
                       const void * txbuf,
                       sysinterval_t timeout )
```

**Performs a transmission on the UART peripheral.**

**Note**

The function returns when the specified number of frames have been sent to the UART or on timeout.

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

This function implements a software timeout, it does not use any underlying HW timeout mechanism.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>uartp</code></td>
</tr>
<tr>
<td>in, out</td>
<td><code>np</code></td>
</tr>
<tr>
<td>in</td>
<td><code>txbuf</code></td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code></td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

| MSG_OK          | if the operation completed successfully. |
| MSG_TIMEOUT     | if the operation timed out. |
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.48.6.14  uartSendFullTimeout()

```c
msg_t uartSendFullTimeout ( UARTDriver ∗ uartp, size_t ∗ np, const void ∗ txbuf, sysinterval_t timeout )
```

Performs a transmission on the UART peripheral.

**Note**
The function returns when the specified number of frames have been physically transmitted or on timeout. The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

This function implements a software timeout, it does not use any underlying HW timeout mechanism.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><code>uartp</code></td>
</tr>
<tr>
<td><strong>in,out</strong></td>
<td><code>np</code></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>txbuf</code></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>timeout</code></td>
</tr>
</tbody>
</table>

**Returns**

The operation status.
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation timed out.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
osalSysLock
uartSendFullTimeout
osalThreadSuspendTimeoutS
uart_lld_start_send
```

7.48.6.15 `uartReceiveTimeout()`

```c
msg_t uartReceiveTimeout (UARTDriver * uartp, size_t * np, void * rxbuf, sysinterval_t timeout )
```

Performs a receive operation on the UART peripheral.

**Note**

The function returns when the specified number of frames have been received or on error/timeout.

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

This function implements a software timeout, it does not use any underlying HW timeout mechanism.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>uartp</code></th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td><code>np</code></td>
<td>number of data frames to receive, on exit the number of frames actually received</td>
</tr>
<tr>
<td>in</td>
<td><code>rxbuf</code></td>
<td>the pointer to the receive buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code></td>
<td>operation timeout</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation timed out.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>in case of a receive error.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
uartReceiveTimeout
  ↘      osalSysLock
  ↗  osalThreadSuspendTimeoutS
uart_lld_start_receive
```

### 7.48.6.16 uartAcquireBus()

```c
void uartAcquireBus (UARTDriver * uartp )
```

Gains exclusive access to the UART bus.

This function tries to gain ownership to the UART bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option UART_USE_MUTUAL_EXCLUSION must be enabled.

Parameters

| in   | uartp | pointer to the UARTDriver object |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
uartAcquireBus -> osalMutexLock
```

### 7.48.6.17 `uartReleaseBus()`

```c
void uartReleaseBus ( UARTDriver * uartp )
```

Releases exclusive access to the UART bus.

**Precondition**

In order to use this function the option `UART_USE_MUTUAL_EXCLUSION` must be enabled.

**Parameters**

| in    | `uartp` | pointer to the UARTDriver object |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
uartReleaseBus -> osalMutexUnlock
```
7.48.6.18  uart_lld_init()

```c
void uart_lld_init (  
    void  )
```

Low level UART driver initialization.

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
uart_lld_init  uartObjectInit  osalMutexObjectInit
```

7.48.6.19  uart_lld_start()

```c
void uart_lld_start (  
    UARTDriver * uartp  )
```

Configures and activates the UART peripheral.

Parameters
- `in uarp` pointer to the UARTDriver object

Function Class:
Not an API, this function is for internal use only.

7.48.6.20  uart_lld_stop()

```c
void uart_lld_stop (  
    UARTDriver * uarp  )
```

Deactivates the UART peripheral.
7.48 UART Driver

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uartp</code></td>
<td>pointer to the UARTDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.48.6.21 `uart_lld_start_send()`

```c
void uart_lld_start_send (UARTDriver * uartp, size_t n, const void * txbuf)
```

Starts a transmission on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uartp</code></td>
<td>pointer to the UARTDriver object</td>
</tr>
<tr>
<td><code>n</code></td>
<td>number of data frames to send</td>
</tr>
<tr>
<td><code>txbuf</code></td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.48.6.22 `uart_lld_stop_send()`

```c
size_t uart_lld_stop_send (UARTDriver * uartp)
```

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.
Parameters

| in  | uartp | pointer to the UARTDriver object |

Returns

The number of data frames not transmitted by the stopped transmit operation.

Function Class:

Not an API, this function is for internal use only.

7.48.6.23  uart_lld_start_receive()

void uart_lld_start_receive (UARTDriver * uartp,
size_t n,
void * rxbuf )

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

| in  | uartp | pointer to the UARTDriver object |
| in  | n     | number of data frames to send    |
| out | rxbuf | the pointer to the receive buffer |

Function Class:

Not an API, this function is for internal use only.

7.48.6.24  uart_lld_stop_receive()

size_t uart_lld_stop_receive (UARTDriver * uartp )

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.
Parameters

```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uartp</code></td>
<td>pointer to the UARTDriver object</td>
</tr>
</tbody>
</table>
```

Returns

The number of data frames not received by the stopped receive operation.

Function Class:

Not an API, this function is for internal use only.

### 7.48.7 Variable Documentation

#### 7.48.7.1 UARTD1

```
UARTDriver UARTD1
```

UART1 driver identifier.
7.49 USB Driver

Generic USB Driver.

7.49.1 Detailed Description

Generic USB Driver.

This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

Precondition

In order to use the USB driver the HAL_USE_USB option must be enabled in halconf.h.

7.49.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).

7.49.3 USB Operations

The USB driver is quite complex and USB is complex in itself, it is recommended to study the USB specification before trying to use the driver.
7.49.3.1 USB Implementation

The USB driver abstracts the inner details of the underlying USB hardware. The driver works asynchronously and communicates with the application using callbacks. The application is responsible of the descriptors and strings required by the USB device class to be implemented and of the handling of the specific messages sent over the endpoint zero. Standard messages are handled internally to the driver. The application can use hooks in order to handle custom messages or override the handling of the default handling of standard messages.

7.49.3.2 USB Endpoints

USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

- **IN endpoints** are used by the application to transmit data to the host.
- **OUT endpoints** are used by the application to receive data from the host.

The driver invokes a callback after finishing an IN or OUT transaction. States diagram for OUT endpoints in transaction mode:

```
States diagram for OUT endpoints in transaction mode:
```

```
EP_DISABLED
Disabled
EP_IDLE
Ready
usbInitEndpoint()
usbPrepareReceive()
usbStartReceiveI()
more packets
EP_BUSY
Receiving
USB RESET
usbDisableEndpointsI()
>out_cb<
USB RESET
usbDisableEndpointsI()
usbPrepareReceive()
usbStartReceiveI()
```

```
States diagram for IN endpoints in transaction mode:
```

```
EP_DISABLED
Disabled
EP_IDLE
Ready
usbInitEndpoint()
usbPrepareTransmit()
usbStartTransmitI()
more packets
EP_BUSY
Transmitting
USB RESET
usbDisableEndpointsI()
>in_cb<
USB RESET
usbDisableEndpointsI()
usbPrepareTransmit()
usbStartTransmitI()
```
7.49.3.3 USB Callbacks

The USB driver uses callbacks in order to interact with the application. There are several kinds of callbacks to be handled:

- Driver events callback. As example errors, suspend event, reset event etc.
- Messages Hook callback. This hook allows the application to implement handling of custom messages or to override the default handling of standard messages on endpoint zero.
- Descriptor Requested callback. When the driver endpoint zero handler receives a GET DESCRIPTOR message and needs to send a descriptor to the host it queries the application using this callback.
- Start of Frame callback. This callback is invoked each time a SOF packet is received.
- Endpoint callbacks. Each endpoint informs the application about I/O conditions using those callbacks.

Macros

- #define USB_USE_WAIT FALSE
  Enables synchronous APIs.
- #define USB_MAX_ENDPOINTS 4
  Maximum endpoint address.
- #define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW
  Status stage handling method.
- #define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS
  The address can be changed immediately upon packet reception.
- #define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW
  Method for set address acknowledge.
- #define usb_lld_get_frame_number(usbp) 0
  Returns the current frame number.
- #define usb_lld_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)
  Returns the exact size of a receive transaction.
- #define usb_lld_connect_bus(usbp)
  Connects the USB device.
- #define usb_lld_disconnect_bus(usbp)
  Disconnect the USB device.
- #define usb_lld_wakeup_host(usbp)
  Start of host wake-up procedure.

Helper macros for USB descriptors

- #define USB_DESC_INDEX(i) ((uint8_t)(i))
  Helper macro for index values into descriptor strings.
- #define USB_DESC_BYTE(b) ((uint8_t)(b))
  Helper macro for byte values into descriptor strings.
- #define USB_DESC_WORD(w)
  Helper macro for word values into descriptor strings.
- #define USB_DESC_BCD(bcd)
  Helper macro for BCD values into descriptor strings.
- #define USB_DESC_DEVICE_SIZE 18U
• #define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)
  Device Descriptor helper macro.
• #define USB_DESC_CONFIGURATION_SIZE 9U
  Configuration Descriptor size.
• #define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)
  Configuration Descriptor helper macro.
• #define USB_DESC_INTERFACE_SIZE 9U
  Interface Descriptor size.
• #define USB_DESC_INTERFACE(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)
  Interface Descriptor helper macro.
• #define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U
  Interface Association Descriptor size.
• #define USB_DESC_INTERFACE_ASSOCIATION(bFirstInterface, bInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, lInterface)
  Interface Association Descriptor helper macro.
• #define USB_DESC_ENDPOINT_SIZE 7U
  Endpoint Descriptor size.
• #define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)
  Endpoint Descriptor helper macro.

Endpoint types and settings

• #define USB_EP_MODE_TYPE 0x0003U
• #define USB_EP_MODE_TYPE_CTRL 0x0000U
• #define USB_EP_MODE_TYPE_ISOC 0x0001U
• #define USB_EP_MODE_TYPE_BULK 0x0002U
• #define USB_EP_MODE_TYPE_INTR 0x0003U

Macro Functions

• #define usbGetDriverStateI(usbp) ((usbp)->state)
  Returns the driver state.
• #define usbConnectBus(usbp) usb_lld_connect_bus(usbp)
  Connects the USB device.
• #define usbDisconnectBus(usbp) usb_lld_disconnect_bus(usbp)
  Disconnects the USB device.
• #define usbGetFrameNumberX(usbp) usb_lld_get_frame_number(usbp)
  Returns the current frame number.
• #define usbGetTransmitStatusI(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep)) != 0U)
  Returns the status of an IN endpoint.
• #define usbGetReceiveStatusI(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep)) != 0U)
  Returns the status of an OUT endpoint.
• #define usbGetReceiveTransactionSizeX(usbp, ep) usb_lld_get_transaction_size(usbp, ep)
  Returns the exact size of a receive transaction.
• #define usbSetupTransfer(usbp, buf, n, endcb)
  Request transfer setup.
• #define usbReadSetup(usbp, ep, buf) usb_lld_read_setup(usbp, ep, buf)
  Reads a setup packet from the dedicated packet buffer.
Low level driver helper macros

- `#define _usb_isr_invoke_event_cb(usbp, evt)`
  Common ISR code, usb event callback.
- `#define _usb_isr_invoke_sof_cb(usbp)`
  Common ISR code, SOF callback.
- `#define _usb_isr_invoke_setup_cb(usbp, ep)`
  Common ISR code, setup packet callback.
- `#define _usb_isr_invoke_in_cb(usbp, ep)`
  Common ISR code, IN endpoint callback.
- `#define _usb_isr_invoke_out_cb(usbp, ep)`
  Common ISR code, OUT endpoint event.

PLATFORM configuration options

- `#define PLATFORM_USB_USE_USB1 FALSE`
  USB driver enable switch.

Typedefs

- `typedef struct USBDriver USBDriver`  
  Type of a structure representing an USB driver.
- `typedef uint8_t usbep_t`  
  Type of an endpoint identifier.
- `typedef void(∗ usbcallback_t) (USBDriver ∗usbp)`  
  Type of an USB generic notification callback.
- `typedef void(∗ usbepcallback_t) (USBDriver ∗usbp, usbep_t ep)`  
  Type of an USB endpoint callback.
- `typedef void(∗ usbeventcb_t) (USBDriver ∗usbp, usbevent_t event)`  
  Type of an USB event notification callback.
- `typedef bool(∗ usbreqhandler_t) (USBDriver ∗usbp)`  
  Type of a requests handler callback.

Data Structures

- `struct USBDescriptor`  
  Type of an USB descriptor.
- `struct USBInEndpointState`  
  Type of an IN endpoint state structure.
- `struct USBOutEndpointState`  
  Type of an OUT endpoint state structure.
- `struct USBEndpointConfig`  
  Type of an USB endpoint configuration structure.
- `struct USBConfig`  
  Type of an USB driver configuration structure.
- `struct USBDriver`  
  Structure representing an USB driver.
Functions

• static void set_address (USBDriver *usbp)
  SET ADDRESS transaction callback.
• static bool default_handler (USBDriver *usbp)
  Standard requests handler.
• void usInit (void)
  USB Driver initialization.
• void usbObjectInit (USBDriver *usbp)
  Initializes the standard part of a USBDriver structure.
• msg_t usbStart (USBDriver *usbp, const USBConfig *config)
  Configures and activates the USB peripheral.
• void usbStop (USBDriver *usbp)
  Deactivates the USB peripheral.
• void usbInitEndpointI (USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)
  Enables an endpoint.
• void usbDisableEndpointsI (USBDriver *usbp)
  Disables all the active endpoints.
• void usbStartReceiveI (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)
  Starts a receive transaction on an OUT endpoint.
• void usbStartTransmitI (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)
  Starts a transmit transaction on an IN endpoint.
• msg_t usbReceive (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)
  Performs a receive transaction on an OUT endpoint.
• msg_t usbTransmit (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)
  Performs a transmit transaction on an IN endpoint.
• bool usbStallReceiveI (USBDriver *usbp, usbep_t ep)
  Stalls an OUT endpoint.
• bool usbStallTransmitI (USBDriver *usbp, usbep_t ep)
  Stalls an IN endpoint.
• void usbWakeupHost (USBDriver *usbp)
  Host wake-up procedure.
• void _usb_reset (USBDriver *usbp)
  USB reset routine.
• void _usb_suspend (USBDriver *usbp)
  USB suspend routine.
• void _usb_wakeup (USBDriver *usbp)
  USB wake-up routine.
• void _usb_ep0setup (USBDriver *usbp, usbep_t ep)
  Default EP0 SETUP callback.
• void _usb_ep0in (USBDriver *usbp, usbep_t ep)
  Default EP0 IN callback.
• void _usb_ep0out (USBDriver *usbp, usbep_t ep)
  Default EP0 OUT callback.
• void usb_lld_init (void)
  Low level USB driver initialization.
• void usb_lld_start (USBDriver *usbp)
  Configures and activates the USB peripheral.
• void usb_lld_stop (USBDriver *usbp)
  Deactivates the USB peripheral.
• void usb_lld_reset (USBDriver *usbp)

ChibiOS/HAL
USB low level reset routine.

- void usb_lld_set_address (USBDriver *usbp)
  Sets the USB address.

- void usb_lld_init_endpoint (USBDriver *usbp, usbep_t ep)
  Enables an endpoint.

- void usb_lld_disable_endpoints (USBDriver *usbp)
  Disables all the active endpoints except the endpoint zero.

- usbpstatus_t usb_lld_get_status_out (USBDriver *usbp, usbep_t ep)
  Returns the status of an OUT endpoint.

- usbpstatus_t usb_lld_get_status_in (USBDriver *usbp, usbep_t ep)
  Returns the status of an IN endpoint.

- void usb_lld_read_setup (USBDriver *usbp, usbep_t ep, uint8_t *buf)
  Reads a setup packet from the dedicated packet buffer.

- void usb_lld_prepare_receive (USBDriver *usbp, usbep_t ep)
  Prepares for a receive operation.

- void usb_lld_prepare_transmit (USBDriver *usbp, usbep_t ep)
  Prepares for a transmit operation.

- void usb_lld_start_out (USBDriver *usbp, usbep_t ep)
  Starts a receive operation on an OUT endpoint.

- void usb_lld_start_in (USBDriver *usbp, usbep_t ep)
  Starts a transmit operation on an IN endpoint.

- void usb_lld_stall_out (USBDriver *usbp, usbep_t ep)
  Brings an OUT endpoint in the stalled state.

- void usb_lld_stall_in (USBDriver *usbp, usbep_t ep)
  Brings an IN endpoint in the stalled state.

- void usb_lld_clear_out (USBDriver *usbp, usbep_t ep)
  Brings an OUT endpoint in the active state.

- void usb_lld_clear_in (USBDriver *usbp, usbep_t ep)
  Brings an IN endpoint in the active state.

Enumerations

- enum usbstate_t { USB_UNINIT = 0, USB_STOP = 1, USB_READY = 2, USB_SELECTED = 3, USB_ACTIVE = 4, USB_SUSPENDED = 5 }
  Type of a driver state machine possible states.

- enum usbpstatus_t { EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE = 2 }
  Type of an endpoint status.

- enum usbp0state_t { USB_EP0_STP_WAITING = 0U, USB_EP0_IN_TX = USB_IN_STATE | 1U, USB_EP0_IN_WAITING_TX0 = USB_IN_STATE | 2U, USB_EP0_IN_SENDING_STS = USB_IN_STATE | 3U, USB_EP0_OUT_WAITING_STS = USB_OUT_STATE | 4U, USB_EP0_OUT_RX = USB_OUT_STATE | 5U, USB_EP0_ERROR = 6U }
  Type of an endpoint zero state machine states.

- enum usbevent_t { USB_EVENT_RESET = 0, USB_EVENT_ADDRESS = 1, USB_EVENT_CONFIGURED = 2, USB_EVENT_UNCONFIGURED = 3, USB_EVENT_SUSPEND = 4, USB_EVENT_WAKEUP = 5, USB_EVENT_STALLED = 6 }
  Type of an enumeration of the possible USB events.
Variables

- const typedef USBDescriptor ∗(∗usbgetdescriptor_t)(USBDriver ∗usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)
  
  Type of an USB descriptor-retrieving callback.
- USBDriver USBD1
  
  USB1 driver identifier.
- union {
    USBInEndpointState in
    IN EP0 state.
    USBOutEndpointState out
    OUT EP0 state.
  } ep0_state
  
  EP0 state.
- static const USBEndpointConfig ep0config
  
  EP0 initialization structure.

7.49.4 Macro Definition Documentation

7.49.4.1 USB_DESC_INDEX

#define USB_DESC_INDEX(
    i
) ((uint8_t)(i))

Helper macro for index values into descriptor strings.

7.49.4.2 USB_DESC_BYTE

#define USB_DESC_BYTE(
    b
) {{uint8_t}{b}}

Helper macro for byte values into descriptor strings.

7.49.4.3 USB_DESC_WORD

#define USB_DESC_WORD(
    w
) 

Value:
{{uint8_t}{{w} & 255U},
 {{uint8_t}{{w} » 8} & 255U}

Helper macro for word values into descriptor strings.
7.49.4.4 USB_DESC_BCD

#define USB_DESC_BCD(
  bcd )

Value:
(uint8_t)((bcd) & 255U),
(uint8_t)((bcd) >> 8) & 255)

Helper macro for BCD values into descriptor strings.

7.49.4.5 USB_DESC_DEVICE

#define USB_DESC_DEVICE(
  bcdUSB, 
  bDeviceClass, 
  bDeviceSubClass, 
  bDeviceProtocol, 
  bMaxPacketSize, 
  idVendor, 
  idProduct, 
  bcdDevice, 
  iManufacturer, 
  iProduct, 
  iSerialNumber, 
  bNumConfigurations )

Value:
USB_DESC_BYTE(USB_DESC_DEVICE_SIZE),
USB_DESC_BYTE(USB_DESCRIPTOR_DEVICE),
USB_DESC_BCD(bcdUSB),
USB_DESC_BYTE(bDeviceClass),
USB_DESC_BYTE(bDeviceSubClass),
USB_DESC_BYTE(bDeviceProtocol),
USB_DESC_BYTE(bMaxPacketSize),
USB_DESC_WORD(idVendor),
USB_DESC_WORD(idProduct),
USB_DESC_BCD(bcdDevice),
USB_DESC_INDEX(iManufacturer),
USB_DESC_INDEX(iProduct),
USB_DESC_INDEX(iSerialNumber),
USB_DESC_BYTE(bNumConfigurations)

Device Descriptor helper macro.

7.49.4.6 USB_DESC_CONFIGURATION_SIZE

#define USB_DESC_CONFIGURATION_SIZE 9U

Configuration Descriptor size.
7.49 USB Driver

7.49.4.7 USB_DESC_CONFIGURATION

#define USB_DESC_CONFIGURATION(
    wTotalLength,
    bNumInterfaces,
    bConfigurationValue,
    iConfiguration,
    bmAttributes,
    bMaxPower )

Value:
    USB_DESC_BYTE(USB_DESC_CONFIGURATION_SIZE),
    USB_DESC_BYTE(USB_DESCRIPTOR_CONFIGURATION),
    USB_DESC_WORD(wTotalLength),
    USB_DESC_BYTE(bNumInterfaces),
    USB_DESC_BYTE(bConfigurationValue),
    USB_DESC_INDEX(iConfiguration),
    USB_DESC_BYTE(bmAttributes),
    USB_DESC_BYTE(bMaxPower)

Configuration Descriptor helper macro.

7.49.4.8 USB_DESC_INTERFACE_SIZE

#define USB_DESC_INTERFACE_SIZE 9U

Interface Descriptor size.

7.49.4.9 USB_DESC_INTERFACE

#define USB_DESC_INTERFACE(
    bInterfaceNumber,
    bAlternateSetting,
    bNumEndpoints,
    bInterfaceClass,
    bInterfaceSubClass,
    bInterfaceProtocol,
    iInterface )

Value:
    USB_DESC_BYTE(USB_DESC_INTERFACE_SIZE),
    USB_DESC_BYTE(USB_DESCRIPTOR_INTERFACE),
    USB_DESC_BYTE(bInterfaceNumber),
    USB_DESC_BYTE(bAlternateSetting),
    USB_DESC_BYTE(bNumEndpoints),
    USB_DESC_BYTE(bInterfaceClass),
    USB_DESC_BYTE(bInterfaceSubClass),
    USB_DESC_BYTE(bInterfaceProtocol),
    USB_DESC_INDEX(iInterface)

Interface Descriptor helper macro.
7.49.4.10 USB_DESC_INTERFACE_ASSOCIATION_SIZE

#define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U

Interface Association Descriptor size.

7.49.4.11 USB_DESC_INTERFACE_ASSOCIATION

#define USB_DESC_INTERFACE_ASSOCIATION(
    bFirstInterface,
    bInterfaceCount,
    bFunctionClass,
    bFunctionSubClass,
    bFunctionProtocol,
    iInterface
)

Value:
    USB_DESC_BYTE(USB_DESC_INTERFACE_ASSOCIATION_SIZE),
    USB_DESC_BYTE(USB_DESCRIPTOR_INTERFACE_ASSOCIATION),
    USB_DESC_BYTE(bFirstInterface),
    USB_DESC_BYTE(bInterfaceCount),
    USB_DESC_BYTE(bFunctionClass),
    USB_DESC_BYTE(bFunctionSubClass),
    USB_DESC_BYTE(bFunctionProtocol),
    USB_DESC_INDEX(iInterface)

Interface Association Descriptor helper macro.

7.49.4.12 USB_DESC_ENDPOINT_SIZE

#define USB_DESC_ENDPOINT_SIZE 7U

Endpoint Descriptor size.

7.49.4.13 USB_DESC_ENDPOINT

#define USB_DESC_ENDPOINT(
    bEndpointAddress,
    bmAttributes,
    wMaxPacketSize,
    bInterval
)

Value:
    USB_DESC_BYTE(USB_DESC_ENDPOINT_SIZE),
    USB_DESC_BYTE(USB_DESCRIPTOR_ENDPOINT),
    USB_DESC_BYTE(bEndpointAddress),
    USB_DESC_BYTE(bmAttributes),
    USB_DESC_WORD(wMaxPacketSize),
    USB_DESC_BYTE(bInterval)

Endpoint Descriptor helper macro.
7.49.4.14  USB_EP_MODE_TYPE

#define USB_EP_MODE_TYPE 0x0003U

Endpoint type mask.

7.49.4.15  USB_EP_MODE_TYPE_CTRL

#define USB_EP_MODE_TYPE_CTRL 0x0000U

Control endpoint.

7.49.4.16  USB_EP_MODE_TYPE_ISO

#define USB_EP_MODE_TYPE_ISO 0x0001U

Isochronous endpoint.

7.49.4.17  USB_EP_MODE_TYPE_BULK

#define USB_EP_MODE_TYPE_BULK 0x0002U

Bulk endpoint.

7.49.4.18  USB_EP_MODE_TYPE_INTR

#define USB_EP_MODE_TYPE_INTR 0x0003U

Interrupt endpoint.

7.49.4.19  USB_USE_WAIT

#define USB_USE_WAIT FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.49.4.20  usbGetDriverStateI

#define usbGetDriverStateI(
    usbp ) ((usbp)->state)

Returns the driver state.
Module Documentation

Parameters

| in  | usbp | pointer to the USBDriver object |

Returns

The driver state.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.49.4.21 usbConnectBus

#define usbConnectBus(
    usbp ) usb_lld_connect_bus(usbp)

Connects the USB device.

Parameters

| in  | usbp | pointer to the USBDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.49.4.22 usbDisconnectBus

#define usbDisconnectBus(
    usbp ) usb_lld_disconnect_bus(usbp)

Disconnect the USB device.

Parameters

| in  | usbp | pointer to the USBDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.49.4.23  **usbGetFrameNumberX**

```c
#define usbGetFrameNumberX( usbp ) usb_lld_get_frame_number(usbp)
```

Returns the current frame number.

**Parameters**

| in  | usbp | pointer to the USBDriver object |

**Returns**

The current frame number.

**Function Class:**

This is an **X-Class** API, this function can be invoked from any context.

7.49.4.24  **usbGetTransmitStatusI**

```c
#define usbGetTransmitStatusI( 
    usbp, 
    ep ) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) !=(0U))
```

Returns the status of an IN endpoint.

**Parameters**

| in  | usbp | pointer to the USBDriver object |
| in  | ep   | endpoint number |

**Returns**

The operation status.

**Return values**

| false | Endpoint ready. |
| true  | Endpoint transmitting. |

**Function Class:**

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
7.49.4.25 usbGetReceiveStatusI

#define usbGetReceiveStatusI(
    usbp,  
    ep ) (((usbp)->receiving & (uint16_t)(unsigned)1U << (unsigned)(ep))) != 0U

Returns the status of an OUT endpoint.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

| false | Endpoint ready.               |
| true  | Endpoint receiving.           |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.49.4.26 usbGetReceiveTransactionSizeX

#define usbGetReceiveTransactionSizeX(
    usbp,  
    ep ) usb_lld_get_transaction_size(usbp, ep)

Returns the exact size of a receive transaction.

The received size can be different from the size specified in usbStartReceiveI() because the last packet could have a size different from the expected one.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Returns

Received data size.
Function Class:

This is an X-Class API, this function can be invoked from any context.

7.49.4.27  usbSetupTransfer

#define usbSetupTransfer(
    usbp,
    buf,
    n,
    endcb)

Value:

\{(usbp)->ep0next = (buf);
    (usbp)->ep0n = (n);
    (usbp)->ep0endcb = (endcb);
\}

Request transfer setup.

This macro is used by the request handling callbacks in order to prepare a transaction over the endpoint zero.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>pointer to a buffer for the transaction data</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to be transferred</td>
</tr>
<tr>
<td>in</td>
<td>endcb</td>
<td>callback to be invoked after the transfer or NULL</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

7.49.4.28  usbReadSetup

#define usbReadSetup(
    usbp,
    ep,
    buf ) usb_lld_read_setup(usbp, ep, buf)

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the setup_cb callback in order to read the received setup packet.

Precondition

In order to use this function the endpoint must have been initialized as a control endpoint.

Note

This function can be invoked both in thread and IRQ context.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>buffer where to copy the packet data</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.

### 7.49.4.29 _usb_isr_invoke_event_cb

```c
#define _usb_isr_invoke_event_cb(
    usbp,
    evt )
```

**Value:**

```c
{
    if (((usbp)->config->event_cb) != NULL) {
        (usbp)->config->event_cb(usbp, evt);
    }
}
```

Common ISR code, usb event callback.

Function Class:

Not an API, this function is for internal use only.

### 7.49.4.30 _usb_isr_invoke_sof_cb

```c
#define _usb_isr_invoke_sof_cb(
    usbp )
```

**Value:**

```c
{
    if (((usbp)->config->sof_cb) != NULL) {
        (usbp)->config->sof_cb(usbp);
    }
}
```

Common ISR code, SOF callback.
Parameters

| in  | usbp | pointer to the USBDriver object |

Function Class:

Not an API, this function is for internal use only.

7.49.4.31 _usb_isr_invoke_setup_cb

```c
#define _usb_isr_invoke_setup_cb(
    usbp,
    ep )
```

Value:

```c
{ (usbp)->epc[ep]->setup_cb(usbp, ep); }
```

Common ISR code, setup packet callback.

Parameters

| in  | usbp | pointer to the USBDriver object |
| in  | ep   | endpoint number |

Function Class:

Not an API, this function is for internal use only.

7.49.4.32 _usb_isr_invoke_in_cb

```c
#define _usb_isr_invoke_in_cb(
    usbp,
    ep )
```

Value:

```c
{ (usbp)->transmitting &= ~(1 « (ep));
  if ((usbp)->epc[ep]->in_cb != NULL) {
    (usbp)->epc[ep]->in_cb(usbp, ep);
  }
  osalSysLockFromISR();
  osalThreadResumeI(&(usbp)->epc[ep]->in_state->thread, MSG_OK);
  osalSysUnlockFromISR();
}
```

Common ISR code, IN endpoint callback.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.49.4.33  _usb_isr_invoke_out_cb

#define _usb_isr_invoke_out_cb(
        usbp,  
        ep )

Value:

{ (usbp)->receiving &= ~(1 « (ep));  
  if ((usbp)->epc[ep]->out_cb != NULL) {  
    (usbp)->epc[ep]->out_cb(usbp, ep);  
  }  
osalSysLockFromISR();  
osalThreadResumeI(4 (usbp)->epc[ep]->out_state->thread,  
                           usbGetReceiveTransactionSizeX(usbp, ep));  
osalSysUnlockFromISR();  
}

Common ISR code, OUT endpoint event.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.49.4.34  USB_MAX_ENDPOINTS

#define USB_MAX_ENDPOINTS 4

Maximum endpoint address.
7.49.4.35  USB_EP0_STATUS_STAGE

#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW

Status stage handling method.

7.49.4.36  USB_SET_ADDRESS_MODE

#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS

The address can be changed immediately upon packet reception.

7.49.4.37  USB_SET_ADDRESS_ACK_HANDLING

#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW

Method for set address acknowledge.

7.49.4.38  PLATFORM_USB_USE_USB1

#define PLATFORM_USB_USE_USB1 FALSE

USB driver enable switch.

If set to TRUE the support for USB1 is included.

Note

The default is FALSE.

7.49.4.39  usb_lld_get_frame_number

#define usb_lld_get_frame_number(
    usbp ) 0

Returns the current frame number.

Parameters

| in  | usbp | pointer to the USBDriver object |

ChibiOS/HAL
Returns

The current frame number.

Function Class:

Not an API, this function is for internal use only.

7.49.4.40  usb_lld_get_transaction_size

#define usb_lld_get_transaction_size(
    usbp,
    ep ) ((usbp)->epc[ep]->out_state->rxcnt)

Returns the exact size of a receive transaction.

The received size can be different from the size specified in `usbStartReceiveI()` because the last packet could have a size different from the expected one.

Precondition

The OUT endpoint must have been configured in transaction mode in order to use this function.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Returns

Received data size.

Function Class:

Not an API, this function is for internal use only.

7.49.4.41  usb_lld_connect_bus

#define usb_lld_connect_bus(
    usbp )

Connects the USB device.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.49 USB Driver

7.49.42 usb_lld_disconnect_bus

#define usb_lld_disconnect_bus( 
    usbp )

Disconnect the USB device.

Function Class:
    Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.49.43 usb_lld_wakeup_host

#define usb_lld_wakeup_host( 
    usbp )

Start of host wake-up procedure.

Function Class:
    Not an API, this function is for internal use only.

7.49.5 Typedef Documentation

7.49.5.1 USBDriver

typedef struct USBDriver USBDriver

Type of a structure representing an USB driver.

7.49.5.2 usbep_t

typedef uint8_t usbep_t

Type of an endpoint identifier.

7.49.5.3 usbcallback_t

typedef void(* usbcallback_t) (USBDriver *usbp)

Type of an USB generic notification callback.
### Parameters

| in | usbp | pointer to the **USBDriver** object triggering the callback |

### 7.49.5.4 usbepcallback_t

typedef void(* usbepcallback_t) (USBDriver *usbp, usbep_t ep)

Type of an USB endpoint callback.

| in | usbp | pointer to the **USBDriver** object triggering the callback |
| in | ep   | endpoint number |

### 7.49.5.5 usbeventcb_t

typedef void(* usbeventcb) (USBDriver *usbp, usbevent_t event)

Type of an USB event notification callback.

| in | usbp | pointer to the **USBDriver** object triggering the callback |
| in | event | event type |

### 7.49.5.6 usbreqhandler_t

typedef bool(* usbreqhandler_t) (USBDriver *usbp)

Type of a requests handler callback.

The request is encoded in the **usb_setup** buffer.

| in | usbp | pointer to the **USBDriver** object triggering the callback |
Returns

The request handling exit code.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>Request not recognized by the handler.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Request handled.</td>
</tr>
</tbody>
</table>

7.49.6 Enumeration Type Documentation

7.49.6.1 usbstate_t

enum usbstate_t

Type of a driver state machine possible states.

Enumerator

<table>
<thead>
<tr>
<th>USB_UNINIT</th>
<th>Not initialized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>USB_READY</td>
<td>Ready, after bus reset.</td>
</tr>
<tr>
<td>USB_SELECTED</td>
<td>Address assigned.</td>
</tr>
<tr>
<td>USB_ACTIVE</td>
<td>Active, configuration selected.</td>
</tr>
<tr>
<td>USB_SUSPENDED</td>
<td>Suspended, low power mode.</td>
</tr>
</tbody>
</table>

7.49.6.2 usbepstatus_t

enum usbepstatus_t

Type of an endpoint status.

Enumerator

<table>
<thead>
<tr>
<th>EP_STATUS_DISABLED</th>
<th>Endpoint not active.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP_STATUS_STALLED</td>
<td>Endpoint opened but stalled.</td>
</tr>
<tr>
<td>EP_STATUS_ACTIVE</td>
<td>Active endpoint.</td>
</tr>
</tbody>
</table>
7.49.6.3 usbep0state_t

enum usbep0state_t

Type of an endpoint zero state machine states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_EP0_STP_WAITING</td>
<td>Waiting for SETUP data.</td>
</tr>
<tr>
<td>USB_EP0_IN_TX</td>
<td>Transmitting.</td>
</tr>
<tr>
<td>USB_EP0_IN_WAITING_TX0</td>
<td>Waiting transmit 0.</td>
</tr>
<tr>
<td>USB_EP0_IN_SENDING_STS</td>
<td>Sending status.</td>
</tr>
<tr>
<td>USB_EP0_OUT_WAITING_STS</td>
<td>Waiting status.</td>
</tr>
<tr>
<td>USB_EP0_OUT_RX</td>
<td>Receiving.</td>
</tr>
<tr>
<td>USB_EP0_ERROR</td>
<td>Error, EP0 stalled.</td>
</tr>
</tbody>
</table>

7.49.6.4 usbevent_t

enum usbevent_t

Type of an enumeration of the possible USB events.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_EVENT_RESET</td>
<td>Driver has been reset by host.</td>
</tr>
<tr>
<td>USB_EVENT_ADDRESS</td>
<td>Address assigned.</td>
</tr>
<tr>
<td>USB_EVENT_CONFIGURED</td>
<td>Configuration selected.</td>
</tr>
<tr>
<td>USB_EVENT_UNCONFIGURED</td>
<td>Configuration removed.</td>
</tr>
<tr>
<td>USB_EVENT_SUSPEND</td>
<td>Entering suspend mode.</td>
</tr>
<tr>
<td>USB_EVENT_WAKEUP</td>
<td>Leaving suspend mode.</td>
</tr>
<tr>
<td>USB_EVENT_STALLED</td>
<td>Endpoint 0 error, stalled.</td>
</tr>
</tbody>
</table>
7.49 USB Driver

7.49.7 Function Documentation

7.49.7.1 set_address()

static void set_address (USBDriver * usbp) [static]

SET ADDRESS transaction callback.

Parameters

| in  | usbp | pointer to the USBDriver object |

Here is the call graph for this function:

```
set_address       usb_lld_set_address
```

7.49.7.2 default_handler()

static bool default_handler (USBDriver * usbp) [static]

Standard requests handler.

This is the standard requests default handler, most standard requests are handled here, the user can override the standard handling using the requests_hook_cb hook in the USBConfig structure.

Parameters

| in  | usbp | pointer to the USBDriver object |

Returns

The request handling exit code.
Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td>Request not recognized by the handler or error.</td>
</tr>
<tr>
<td><code>true</code></td>
<td>Request handled.</td>
</tr>
</tbody>
</table>

7.49.7.3 usbInit()

```c
void usbInit (
    void )
```

USB Driver initialization.

**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

![Call Graph](image)

7.49.7.4 usbObjectInit()

```c
void usbObjectInit ( 
    USBDriver * usbp )
```

Initializes the standard part of a `USBDriver` structure.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>usbp</code></td>
</tr>
</tbody>
</table>
Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

### 7.49.7.5 usbStart()

```c
msg_t usbStart (    
    USBDriver * usbp,
    const USBConfig * config )
```

Configures and activates the USB peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the USBConfig object</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Function Class:**
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
usbStart
  ↓
osalSysLock
  ↓
usb_lld_start
```

### 7.49.7.6 usbStop()

```c
void usbStop (    
    USBDriver * usbp )
```

Deactivates the USB peripheral.
7.49.7.7 usbInitEndpointI()

void usbInitEndpointI (  
    USBDriver * usbp,  
    usbep_t ep,  
    const USBEndpointConfig * epcp )

Enables an endpoint.

This function enables an endpoint, both IN and/or OUT directions depending on the configuration structure.

Note

This function must be invoked in response of a SET_CONFIGURATION or SET_INTERFACE message.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
<tr>
<td>in</td>
<td>epcp</td>
<td>the endpoint configuration</td>
</tr>
</tbody>
</table>
Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
    usbInitEndpointI  ──> usb_init_endpoint
```

### 7.49.7.8 `usbDisableEndpointsI()`

```c
void usbDisableEndpointsI (  
    USBDriver * usbp )
```

Disables all the active endpoints.

This function disables all the active endpoints except the endpoint zero.

**Note**

This function must be invoked in response of a SET_CONFIGURAITON message with configuration number zero.

**Parameters**

- `in  usbp` pointer to the **USBDriver** object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
    usbDisableEndpointsI  ──> osalThreadResumeI
```
7.49.7.9 usbStartReceiveI()

```c
void usbStartReceiveI (  
    USBDriver * usbp,  
    usbep_t ep,  
    uint8_t * buf,  
    size_t n )
```

Starts a receive transaction on an OUT endpoint.

**Note**
This function is meant to be called from ISR context outside critical zones because there is a potentially slow operation inside.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>buffer where to copy the received data</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>transaction size. It is recommended a multiple of the packet size because the excess is discarded.</td>
</tr>
</tbody>
</table>

**Function Class:**
This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
usbStartReceiveI \(\rightarrow\) usb_lld_start_out
```

7.49.7.10 usbStartTransmitI()

```c
void usbStartTransmitI (  
    USBDriver * usbp,  
    usbep_t ep,  
    const uint8_t * buf,  
    size_t n )
```

Starts a transmit transaction on an IN endpoint.
Note

This function is meant to be called from ISR context outside critical zones because there is a potentially slow operation inside.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in usbp</code></td>
<td>pointer to the <code>USBDriver</code> object</td>
</tr>
<tr>
<td><code>in ep</code></td>
<td>endpoint number</td>
</tr>
<tr>
<td><code>in buf</code></td>
<td>buffer where to fetch the data to be transmitted</td>
</tr>
<tr>
<td><code>in n</code></td>
<td>transaction size</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
usbStartTransmitI(usb_lld_start_in)
```

### 7.49.7.11 `usbReceive()`

```c
msg_t usbReceive (
    USBDriver * usbp,
    usbep_t ep,
    uint8_t * buf,
    size_t n )
```

Performs a receive transaction on an OUT endpoint.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in usbp</code></td>
<td>pointer to the <code>USBDriver</code> object</td>
</tr>
<tr>
<td><code>in ep</code></td>
<td>endpoint number</td>
</tr>
<tr>
<td><code>out buf</code></td>
<td>buffer where to copy the received data</td>
</tr>
<tr>
<td><code>in n</code></td>
<td>transaction size. It is recommended a multiple of the packet size because the excess is discarded.</td>
</tr>
</tbody>
</table>
Returns

The received effective data size, it can be less than the amount specified.

Return values

| MSG_RESET | driver not in USB_ACTIVE state or the operation has been aborted by an USB reset or a transition to the USB_SUSPENDED state. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
usbReceive
  \- osalSysLock
    \- msg_t usbTransmit()
      \- \* usbp
        \- ep
          \- \* buf
            \- n
    \- osalSysUnlock
```

7.49.7.12 usbTransmit()

```
msg_t usbTransmit (  
    USBDriver * usbp,  
    usbep_t ep,  
    const uint8_t * buf,  
    size_t n )
```

Performs a transmit transaction on an IN endpoint.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
<tr>
<td>in</td>
<td>buf</td>
<td>buffer where to fetch the data to be transmitted</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>transaction size</td>
</tr>
</tbody>
</table>

Returns

The operation status.
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>operation performed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>driver not in USB_ACTIVE state or the operation has been aborted by an USB reset or a transition to the USB_SUSPENDED state.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
usbTransmit
  ↓
osalSysLock
  ↓
usbTransmit
  ↓
osalSysUnlock
```

### 7.49.7.13  usbStallReceiveI()

```c
bool usbStallReceiveI (  
  USBDriver * usbp, 
  usbep_t ep )
```

Stalls an OUT endpoint.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>false</th>
<th>Endpoint stalled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Endpoint busy, not stalled.</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
usbStallReceiveI  →  usb_lld_stall_out
```

### 7.49.7.14 usbStallTransmitI()

```c
bool usbStallTransmitI (  
    USBDriver ∗ usbp,  
    usbep_t ep )
```

Stalls an IN endpoint.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th><strong>usbp</strong></th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>ep</strong></td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>false</strong></td>
<td>Endpoint stalled.</td>
</tr>
<tr>
<td><strong>true</strong></td>
<td>Endpoint busy, not stalled.</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
usbStallTransmitI  usb_lld_stall_in
```

### 7.49.7.15 usbWakeupHost()

```c
void usbWakeupHost (  
  USBDriver * usbp )
```

Host wake-up procedure.

**Note**

It is silently ignored if the USB device is not in the USB_SUSPENDED state.

**Parameters**

- **in usbp** pointer to the USB Driver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.49.7.16 _usb_reset()

```c
void _usb_reset (  
  USBDriver * usbp )
```

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

7.49.7.17 _usb_suspend()

void _usb_suspend (  
    USBDriver * usbp  )

USB suspend routine.

This function must be invoked when an USB bus suspend condition is detected.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
</tbody>
</table>
7.49 USB Driver

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

![Call Graph Image]

7.49.7.18 _usb_wakeup()

```c
void _usb_wakeup ( 
    USBDriver * usbp )
```

USB wake-up routine.

This function must be invoked when an USB bus wake-up condition is detected.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
</table>

Function Class:
Not an API, this function is for internal use only.

7.49.7.19 _usb_ep0setup()

```c
void _usb_ep0setup ( 
    USBDriver * usbp, 
    usbep_t ep )
```

Default EP0 SETUP callback.

This function is used by the low level driver as default handler for EP0 SETUP events.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usbp</code></td>
<td>pointer to the <code>USBDriver</code> object</td>
</tr>
<tr>
<td><code>ep</code></td>
<td>endpoint number, always zero</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.49.7.20 _usb_ep0in()

```c
void _usb_ep0in ( 
    USBDriver * usbp, 
    usbep_t ep )
```

Default EP0 IN callback.

This function is used by the low level driver as default handler for EP0 IN events.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usbp</code></td>
<td>pointer to the <code>USBDriver</code> object</td>
</tr>
<tr>
<td><code>ep</code></td>
<td>endpoint number, always zero</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.49.7.21 _usb_ep0out()

```c
void _usb_ep0out ( 
    USBDriver * usbp, 
    usbep_t ep )
```

Default EP0 OUT callback.

This function is used by the low level driver as default handler for EP0 OUT events.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usbp</code></td>
<td>pointer to the <code>USBDriver</code> object</td>
</tr>
<tr>
<td><code>ep</code></td>
<td>endpoint number, always zero</td>
</tr>
</tbody>
</table>
Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
osalSysLockFromISR
osalSysUnlockFromISR
_usb_ep0out
usb_lld_stall_in
usb_lld_stall_out
usbStartTransmit!
```

### 7.49.7.22 usb_lld_init()

```c
void usb_lld_init ( void )
```

Low level USB driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

```
usb_lld_init
usbObjectInit
```

### 7.49.7.23 usb_lld_start()

```c
void usb_lld_start ( USBDriver * usbp )
```

Configures and activates the USB peripheral.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.49.7.24 usb_lld_stop()

```c
void usb_lld_stop ( 
    USBDriver * usbp )
```

Deactivates the USB peripheral.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.49.7.25 usb_lld_reset()

```c
void usb_lld_reset ( 
    USBDriver * usbp )
```

USB low level reset routine.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
    usb_lld_reset → usb_lld_init_endpoint
```

### 7.49.7.26 usb_lld_set_address()

```c
void usb_lld_set_address (  
    USBDriver * usbp  )
```

Sets the USB address.

**Parameters**
- **in** `usbp` pointer to the `USBDriver` object

Function Class:
Not an API, this function is for internal use only.

### 7.49.7.27 usb_lld_init_endpoint()

```c
void usb_lld_init_endpoint (  
    USBDriver * usbp,  
    usbep_t ep  )
```

Enables an endpoint.

**Parameters**
- **in** `usbp` pointer to the `USBDriver` object
- **in** `ep` endpoint number
Function Class:
Not an API, this function is for internal use only.

7.49.7.28   usb_lld_disable_endpoints()

void usb_lld_disable_endpoints (  
   USBDriver * usbp )

Disables all the active endpoints except the endpoint zero.

Parameters

|   in  |  usbp  | pointer to the USBDriver object |

Function Class:
Not an API, this function is for internal use only.

7.49.7.29   usb_lld_get_status_out()

usbepstatus_t usb_lld_get_status_out (  
   USBDriver * usbp,  
   usbep_t ep )

Returns the status of an OUT endpoint.

Parameters

|   in  |  usbp  | pointer to the USBDriver object |
|   in  |  ep    | endpoint number                |

Returns
The endpoint status.

Return values

| EP_STATUS_DISABLED | The endpoint is not active. |
| EP_STATUS_STALLED | The endpoint is stalled.     |
| EP_STATUS_ACTIVE  | The endpoint is active.      |
7.49 USB Driver

Function Class:
Not an API, this function is for internal use only.

7.49.7.30 usb_lld_get_status_in()

usbepstatus_t usb_lld_get_status_in (USBDriver * usbp, usbep_t ep);

Returns the status of an IN endpoint.

Parameters

| in | usbp | pointer to the USBDriver object |
| in | ep   | endpoint number                 |

Returns
The endpoint status.

Return values

| EP_STATUS_DISABLED | The endpoint is not active. |
| EP_STATUS_STALLED  | The endpoint is stalled.    |
| EP_STATUS_ACTIVE   | The endpoint is active.     |

Function Class:
Not an API, this function is for internal use only.

7.49.7.31 usb_lld_read_setup()

void usb_lld_read_setup (USBDriver * usbp, usbep_t ep, uint8_t * buf);

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the setup_cb callback in order to read the received setup packet.

Precondition
In order to use this function the endpoint must have been initialized as a control endpoint.

Postcondition
The endpoint is ready to accept another packet.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>buffer where to copy the packet data</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.49.7.32 usb_lld_prepare_receive()

```c
void usb_lld_prepare_receive (  
    USBDriver * usbp,  
    usbep_t ep )
```

Prepares for a receive operation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.49.7.33 usb_lld_prepare_transmit()

```c
void usb_lld_prepare_transmit (  
    USBDriver * usbp,  
    usbep_t ep )
```

Prepares for a transmit operation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>
7.49.7.34  usb_lld_start_out()

void usb_lld_start_out (  
    USBDriver * usbp,
    usbep_t ep )

Starts a receive operation on an OUT endpoint.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.49.7.35  usb_lld_start_in()

void usb_lld_start_in (  
    USBDriver * usbp,
    usbep_t ep )

Starts a transmit operation on an IN endpoint.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.49.7.36  usb_lld_stall_out()

void usb_lld_stall_out (  
    USBDriver * usbp,
    usbep_t ep )

Brings an OUT endpoint in the stalled state.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

7.49.7.37  usb_lld_stall_in()

```c
void usb_lld_stall_in ( 
    USBDriver * usbp,
    usbep_t ep )
```

Brings an IN endpoint in the stalled state.

Parameters:
- `usb`, pointer to the USBDriver object
- `ep`, endpoint number

Function Class:
Not an API, this function is for internal use only.

7.49.7.38  usb_lld_clear_out()

```c
void usb_lld_clear_out ( 
    USBDriver * usbp,
    usbep_t ep )
```

Brings an OUT endpoint in the active state.

Parameters:
- `usb`, pointer to the USBDriver object
- `ep`, endpoint number

Function Class:
Not an API, this function is for internal use only.

7.49.7.39  usb_lld_clear_in()

```c
void usb_lld_clear_in ( 
    USBDriver * usbp,
    usbep_t ep )
```

Brings an IN endpoint in the active state.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>usbp</strong></td>
<td>pointer to the <strong>USBDriver</strong> object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>ep</strong></td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

### 7.49.8 Variable Documentation

#### 7.49.8.1 usbgetdescriptor_t

```c
const typedef USBDescriptor *(*usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)
```

Type of an USB descriptor-retrieving callback.

#### 7.49.8.2 USBD1

**USBDriver USBD1**

USB1 driver identifier.

#### 7.49.8.3 ep0_state

```c
union { ... } ep0_state [static]
```

EP0 state.

**Note**

It is an union because IN and OUT endpoints are never used at the same time for EP0.

#### 7.49.8.4 in

```c
USBInEndpointState { ... } in
```

IN EP0 state.
7.49.8.5 out

USBOutEndpointState { ... } out

OUT EP0 state.

7.49.8.6 ep0config

const USBEndpointConfig ep0config [static]

Initial value:

= {
  USB_EP_MODE_TYPE_CTRL,
  _usb_ep0setup,
  _usb_ep0in,
  _usb_ep0out,
  0x40,
  0x40,
  &ep0_state.in,
  &ep0_state.out
  }

EP0 initialization structure.
7.50 **USB CDC Header**

USB CDC Support Header.

### 7.50.1 Detailed Description

USB CDC Support Header.

This header contains definitions and types related to USB CDC.

**CDC specific messages.**

- `#define CDC_SEND_ENCAPSULATED_COMMAND 0x00U`
- `#define CDC_GET_ENCAPSULATED_RESPONSE 0x01U`
- `#define CDC_SET_COMM_FEATURE 0x02U`
- `#define CDC_GET_COMM_FEATURE 0x03U`
- `#define CDC_CLEAR_COMM_FEATURE 0x04U`
- `#define CDC_SET_AUX_LINE_STATE 0x10U`
- `#define CDC_SET_HOOK_STATE 0x11U`
- `#define CDC_PULSE_SETUP 0x12U`
- `#define CDC_SEND_PULSE 0x13U`
- `#define CDC_SET_PULSE_TIME 0x14U`
- `#define CDC_RING_AUX_JACK 0x15U`
- `#define CDC_SET_LINE_CODING 0x20U`
- `#define CDC_GET_LINE_CODING 0x21U`
- `#define CDC_SET_CONTROL_LINE_STATE 0x22U`
- `#define CDC_SEND_BREAK 0x23U`
- `#define CDC_SET_RINGER_PARMS 0x30U`
- `#define CDC_GET_RINGER_PARMS 0x31U`
- `#define CDC_SET_OPERATION_PARMS 0x32U`
- `#define CDC_GET_OPERATION_PARMS 0x33U`

**CDC classes**

- `#define CDC_COMMUNICATION_INTERFACE_CLASS 0x02U`
- `#define CDC_DATA_INTERFACE_CLASS 0x0AU`

**CDC subclasses**

- `#define CDC_ABSTRACT_CONTROL_MODEL 0x02U`

**CDC descriptors**

- `#define CDC_CS_INTERFACE 0x24U`
CDC subdescriptors

- #define CDC_HEADER 0x00U
- #define CDC_CALL_MANAGEMENT 0x01U
- #define CDC_ABSTRACT_CONTROL_MANAGEMENT 0x02U
- #define CDC_UNION 0x06U

Line Control bit definitions.

- #define LC_STOP_1 0U
- #define LC_STOP_1P5 1U
- #define LC_STOP_2 2U
- #define LC_PARITY_NONE 0U
- #define LC_PARITY_ODD 1U
- #define LC_PARITY_EVEN 2U
- #define LC_PARITY_MARK 3U
- #define LC_PARITY_SPACE 4U

Data Structures

- struct cdc_linecoding_t
  Type of Line Coding structure.
7.51 WDG Driver

Generic WDG Driver.

7.51.1 Detailed Description

Generic WDG Driver.
This module defines an abstract interface for a watchdog timer.

Precondition

In order to use the WDG driver the HAL_USE_WDG option must be enabled in halconf.h.

Macros

• #define wdgReset(wdgp) wdg_lld_reset(wdgp)
  Resets WDG's counter.

Configuration options

• #define PLATFORM_WDG_USE_WDG1 FALSE
  WDG1 driver enable switch.

Typedefs

• typedef struct WDGDriver WDGDriver
  Type of a structure representing an WDG driver.

Data Structures

• struct WDGConfig
  Driver configuration structure.
• struct WDGDriver
  Structure representing an WDG driver.

Functions

• void wdgInit (void)
  WDG Driver initialization.
• msg_t wdgStart (WDGDriver *wdgp, const WDGConfig *config)
  Configures and activates the WDG peripheral.
• void wdgStop (WDGDriver *wdgp)
  Deactivates the WDG peripheral.
• void wdgReset (WDGDriver *wdgp)
  Resets WDG's counter.
• void wdg_lld_init (void)
  Low level WDG driver initialization.
• void wdg_lld_start (WDGDriver *wdgp)
  Configures and activates the WDG peripheral.
• void wdg_lld_stop (WDGDriver *wdgp)
  Deactivates the WDG peripheral.
• void wdg_lld_reset (WDGDriver *wdgp)
  Reloads WDG's counter.
Enumerations

- enum wdgstate_t { WDG_UNINIT = 0, WDG_STOP = 1, WDG_READY = 2 }

Driver state machine possible states.

7.51.2 Macro Definition Documentation

7.51.2.1 wdgResetI

#define wdgResetI(
    wdgp ) wdg_lld_reset(wdgp)

Resets WDG's counter.

Parameters

| in  | wdgp | pointer to the WDGDriver object |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.51.2.2 PLATFORM_WDG_USE_WDG1

#define PLATFORM_WDG_USE_WDG1 FALSE

WDG1 driver enable switch.

Note

The default is FALSE.

7.51.3 Typedef Documentation

7.51.3.1 WDGDriver

typedef struct WDGDriver WDGDriver

Type of a structure representing an WDG driver.
7.51.4 Enumeration Type Documentation

7.51.4.1 wdgstate_t

```c
enum wdgstate_t

Driver state machine possible states.

Enumerator

<table>
<thead>
<tr>
<th>WDG_UNINIT</th>
<th>Not initialized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDG_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>WDG_READY</td>
<td>Ready.</td>
</tr>
</tbody>
</table>
```

7.51.5 Function Documentation

7.51.5.1 wdgInit()

```c
void wdgInit ( void )
```

WDG Driver initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

- **Initializer**, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
wdgInit  wdg_lld_init
```

ChibiOS/HAL
7.51.5.2  wdgStart()

```c
msg_t wdgStart(
    WDGDriver * wdgp,
    const WDGConfig * config)
```

Configures and activates the WDG peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in wdgp</td>
<td>pointer to the WDGDriver object</td>
</tr>
<tr>
<td>in config</td>
<td>pointer to the WDGConfig object</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.51.5.3  wdgStop()

```c
void wdgStop(
    WDGDriver * wdgp)
```

Deactivates the WDG peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in wdgp</td>
<td>pointer to the WDGDriver object</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
wdgStop
osalSysLock
osalSysUnlock
wdg_lld_stop
```

### 7.51.5.4 wdgReset()

```c
void wdgReset ( 
    WDGDriver * wdgp )
```

Resets WDG’s counter.

**Parameters**

| in | wdgp | pointer to the WDGDriver object |

**Function Class:**
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
wdgReset
osalSysLock
osalSysUnlock
```
7.51.5.5  wdg_lld_init()

```c
void wdg_lld_init ( void )
```

Low level WDG driver initialization.

Function Class:
Not an API, this function is for internal use only.

7.51.5.6  wdg_lld_start()

```c
void wdg_lld_start ( WDGDriver ∗ wdgp )
```

Configures and activates the WDG peripheral.

Parameters
- `in wdgp` pointer to the `WDGDriver` object

Function Class:
Not an API, this function is for internal use only.

7.51.5.7  wdg_lld_stop()

```c
void wdg_lld_stop ( WDGDriver ∗ wdgp )
```

Deactivates the WDG peripheral.

Parameters
- `in wdgp` pointer to the `WDGDriver` object

Function Class:
Not an API, this function is for internal use only.
7.51.5.8  wdg_lld_reset()

```c
void wdg_lld_reset ( 
    WDGDriver * wdgp )
```

Reloads WDG's counter.

Parameters

| in  | wdgp | pointer to the WDGDriver object |

Function Class:

Not an API, this function is for internal use only.
7.52 WSPI Driver

Generic WSPI Driver.

7.52.1 Detailed Description

Generic WSPI Driver.

This module defines an abstract interface for a wide SPI communication interface (Quad SPI, Octal SPI and similar).

Precondition

In order to use the WSPI driver the HAL_USE_WSPI option must be enabled in halconf.h.

Macros

• #define wspi_lld_driver_fields
  Low level fields of the WSPI driver structure.
• #define wspi_lld_config_fields
  Low level fields of the WSPI configuration structure.

WSPI configuration options

• #define WSPI_USE_WAIT TRUE
  Enables synchronous APIs.
• #define WSPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the wspiAcquireBus() and wspiReleaseBus() APIs.

Transfer options

Note

The low level driver has the option to override the following definitions and use its own ones. In must take care to use the same name for the same function or compatibility is not ensured.

• #define WSPI_CFG_CMD_MODE_MASK (7LU << 0LU)
• #define WSPI_CFG_CMD_MODE_NONE (0LU << 0LU)
• #define WSPI_CFG_CMD_MODE_ONE_LINE (1LU << 0LU)
• #define WSPI_CFG_CMD_MODE_TWO_LINES (2LU << 0LU)
• #define WSPI_CFG_CMD_MODE_FOUR_LINES (3LU << 0LU)
• #define WSPI_CFG_CMD_MODE_EIGHT_LINES (4LU << 0LU)
• #define WSPI_CFG_CMD_DTR (1LU << 3LU)
• #define WSPI_CFG_CMD_SIZE_MASK (3LU << 4LU)
• #define WSPI_CFG_CMD_SIZE_8 (0LU << 4LU)
• #define WSPI_CFG_CMD_SIZE_16 (1LU << 4LU)
• #define WSPI_CFG_CMD_SIZE_24 (2LU << 4LU)
• #define WSPI_CFG_CMD_SIZE_32 (3LU << 4LU)
• #define WSPI_CFG_ADDR_MODE_MASK (7LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_NONE (0LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_ONE_LINE (1LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_TWO_LINES (2LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_FOUR_LINES (3LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_EIGHT_LINES (4LU << 8LU)
• #define WSPI_CFG_ADDR_DTR (1LU << 11LU)
• #define WSPI_CFG_ADDR_SIZE_MASK (3LU << 12LU)
• #define WSPI_CFG_ADDR_SIZE_8 (0LU << 12LU)
• #define WSPI_CFG_ADDR_SIZE_16 (1LU << 12LU)
• #define WSPI_CFG_ADDR_SIZE_24 (2LU << 12LU)
• #define WSPI_CFG_ADDR_SIZE_32 (3LU << 12LU)
• #define WSPI_CFG_ALT_MODE_MASK (7LU << 16LU)
• #define WSPI_CFG_ALT_MODE_NONE (0LU << 16LU)
• #define WSPI_CFG_ALT_MODE_ONE_LINE (1LU << 16LU)
• #define WSPI_CFG_ALT_MODE_TWO_LINES (2LU << 16LU)
• #define WSPI_CFG_ALT_MODE_FOUR_LINES (3LU << 16LU)
• #define WSPI_CFG_ALT_MODE_EIGHT_LINES (4LU << 16LU)
• #define WSPI_CFG_ALT_DTR (1LU << 19LU)
• #define WSPI_CFG_ALT_SIZE_MASK (3LU << 20LU)
• #define WSPI_CFG_ALT_SIZE_8 (0LU << 20LU)
• #define WSPI_CFG_ALT_SIZE_16 (1LU << 20LU)
• #define WSPI_CFG_ALT_SIZE_24 (2LU << 20LU)
• #define WSPI_CFG_ALT_SIZE_32 (3LU << 20LU)
• #define WSPI_CFG_DATA_MODE_MASK (7LU << 24LU)
• #define WSPI_CFG_DATA_MODE_NONE (0LU << 24LU)
• #define WSPI_CFG_DATA_MODE_ONE_LINE (1LU << 24LU)
• #define WSPI_CFG_DATA_MODE_TWO_LINES (2LU << 24LU)
• #define WSPI_CFG_DATA_MODE_FOUR_LINES (3LU << 24LU)
• #define WSPI_CFG_DATA_MODE_EIGHT_LINES (4LU << 24LU)
• #define WSPI_CFG_DATA_DTR (1LU << 27LU)
• #define WSPI_CFG_DQS_ENABLE (1LU << 29LU)
• #define WSPI_CFG_SIOO (1LU << 31LU)
• #define WSPI_CFG_ALL_DTR

Macro Functions

• #define wspiStartCommandI(wspip, cmdp)
  Sends a command without data phase.
• #define wspiStartSendI(wspip, cmdp, n, txbuf)
  Sends data over the WSPI bus.
• #define wspiStartReceiveI(wspip, cmdp, n, rxbuf)
  Receives data from the WSPI bus.
• #define wspiMapFlashI(wspip, cmdp, addrp) wspi_lld_map_flash(wspip, cmdp, addrp)
  Maps in memory space a WSPI flash device.
• #define wspiUnmapFlashI(wspip)
  Maps in memory space a WSPI flash device.

Low level driver helper macros

• #define _wspi_wakeup_isr(wspip, msg)
  Wakes up the waiting thread.
• #define _wspi_isr_code(wspip)
  Common ISR code.
• #define _wspi_error_code(wspip)
  Common error ISR code.
WSPI implementation capabilities

- #define WSPI_SUPPORTS_MEMMAP TRUE
- #define WSPI_DEFAULT_CFG_MASKS TRUE

Configuration options

- #define PLATFORM_WSPI_USE_WSPI1 FALSE
  WSPID1 driver enable switch.

Typedefs

- typedef struct hal_wspi_driver WSPIDriver
  Type of a structure representing an WSPI driver.
- typedef struct hal_wspi_config WSPIConfig
  Type of a structure representing an WSPI driver configuration.
- typedef void(∗wspicallback_t)(WSPIDriver ∗wspip)
  Type of a WSPI notification callback.

Data Structures

- struct wspi_command_t
  Type of a WSPI command descriptor.
- struct hal_wspi_config
  Driver configuration structure.
- struct hal_wspi_driver
  Structure representing an WSPI driver.

Functions

- void wspiInit (void)
  WSPI Driver initialization.
- void wspiObjectInit (WSPIDriver ∗wspip)
  Initializes the standard part of a WSPIDriver structure.
- msg_t wspiStart (WSPIDriver ∗wspip, const WSPIConfig ∗config)
  Configures and activates the WSPI peripheral.
- void wspiStop (WSPIDriver ∗wspip)
  Deactivates the WSPI peripheral.
- void wspiStartCommand (WSPIDriver ∗wspip, const wspi_command_t ∗cmdp)
  Sends a command without data phase.
- void wspiStartSend (WSPIDriver ∗wspip, const wspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)
  Sends a command with data over the WSPI bus.
- void wspiStartReceive (WSPIDriver ∗wspip, const wspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)
  Sends a command then receives data over the WSPI bus.
- bool wspiCommand (WSPIDriver ∗wspip, const wspi_command_t ∗cmdp)
  Sends a command without data phase.
- bool wspiSend (WSPIDriver ∗wspip, const wspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)
  Sends a command with data over the WSPI bus.
• bool wspiReceive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
  Sends a command then receives data over the WSPI bus.
• void wspiMapFlash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp)
  Maps in memory space a WSPI flash device.
• void wspiUnmapFlash (WSPIDriver *wspip)
  Unmaps from memory space a WSPI flash device.
• void wspiAcquireBus (WSPIDriver *wspip)
  Gains exclusive access to the WSPI bus.
• void wspiReleaseBus (WSPIDriver *wspip)
  Releases exclusive access to the WSPI bus.
• void wspi_lld_init (void)
  Low level WSPI driver initialization.
• void wspi_lld_start (WSPIDriver *wspip)
  Configures and activates the WSPI peripheral.
• void wspi_lld_stop (WSPIDriver *wspip)
  Deactivates the WSPI peripheral.
• void wspi_lld_command (WSPIDriver *wspip, const wspi_command_t *cmdp)
  Sends a command without data phase.
• void wspi_lld_send (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
  Sends a command with data over the WSPI bus.
• void wspi_lld_receive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
  Sends a command then receives data over the WSPI bus.
• void wspi_lld_map_flash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp)
  Maps in memory space a WSPI flash device.
• void wspi_lld_unmap_flash (WSPIDriver *wspip)
  Unmaps from memory space a WSPI flash device.

Enumerations

• enum wspistate_t {
  WSPI_UNINIT = 0, WSPI_STOP = 1, WSPI_READY = 2, WSPI_SEND = 3,
  WSPI_RECEIVE = 4, WSPI_COMPLETE = 5, WSPI_MEMMAP = 6 }
  Driver state machine possible states.

Variables

• WSPIDriver WSPID1
  WSPID1 driver identifier.

7.52.2 Macro Definition Documentation
7.52.2.1 WSPI_USE_WAIT

#define WSPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.52.2.2 WSPI_USE_MUTUAL_EXCLUSION

#define WSPI_USE_MUTUAL_EXCLUSION TRUE

Enables the wspiAcquireBus() and wspiReleaseBus() APIs.

Note

Disabling this option saves both code and data space.

7.52.2.3 wspiStartCommandI

#define wspiStartCommandI(
    wspip,
    cmdp)

Value:

\[
\begin{align*}
\text{osalDbgAssert}((\text{cmdp})->\text{cfg} & \& \text{WSPI_CFG_DATA_MODE_MASK}) == \\
\text{WSPI_CFG_DATA_MODE_NONE,} & \\
\text{"data mode specified"};
\end{align*}
\]

\[
\{\text{wspip}\}->\text{state} = \text{WSPI_SEND};
\]

\[
\text{wspi_lld_command(wspip, cmdp);}
\]

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>wspip</th>
<th>pointer to the WSPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
</tbody>
</table>
Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.52.2.4 wspiStartSendI

#define wspiStartSendI(
  wspip,
  cmdp,
  n,
  txbuf
)

Value:
{
  osalDbgAssert(((cmdp)->cfg & WSPI_CFG_DATA_MODE_MASK) != WSPI_CFG_DATA_MODE_NONE, "data mode required");
  (wspip)->state = WSPI_SEND;
  wspi_lld_send(wspip, cmdp, n, txbuf);
}

Sends data over the WSPI bus.

This asynchronous function starts a transmit operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

| in | wspip | pointer to the WSPIDriver object |
| in | cmdp  | pointer to the command descriptor |
| in | n     | number of bytes to send or zero if no data phase |
| in | txbuf | the pointer to the transmit buffer |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.52.2.5 wspiStartReceiveI

#define wspiStartReceiveI(
  wspip,
  cmdp,
  n,
  rxbuf
)
Receives data from the WSPI bus.

This asynchronous function starts a receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

| in   | wspip  | pointer to the WSPIDriver object |
| in   | cmdp   | pointer to the command descriptor |
| in   | n      | number of bytes to receive or zero if no data phase |
| out  | rxbuf  | the pointer to the receive buffer |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.52.2.6 wspiMapFlashI

#define wspiMapFlashI(
    wspip,
    cmdp,
    addrp ) wspi_lld_map_flash(wspip, cmdp, addrp)

Maps in memory space a WSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

| in   | wspip  | pointer to the WSPIDriver object |
| in   | cmdp   | pointer to the command descriptor |
| out  | addrp  | pointer to the memory start address of the mapped flash or NULL |
Function Class:
   This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.52.2.7  wspiUnmapFlashI

#define wspiUnmapFlashI(
    wspip ) wspi_lld_unmap_flash(wspip)

Maps in memory space a WSPI flash device.

Postcondition
   The memory flash device must be re-initialized for normal commands exchange.

Parameters
   in  wspip  pointer to the WSPIDriver object

Function Class:
   This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.52.2.8  _wspi_wakeup_isr

#define _wspi_wakeup_isr(
    wspip, msg )

Value:

   {  
      osalSysLockFromISR();  
      osalThreadResumeI(&(wspip)->thread, msg);  
      osalSysUnlockFromISR();  
   }

Wakes up the waiting thread.

Parameters
   in  wspip  pointer to the WSPIDriver object
   in  msg    the wakeup message
7.52.2.9 _wspi_isr_code

#define _wspi_isr_code(
  wspip
)

Value:

\{
  if ((wspip)->config->end_cb) {
    (wspip)->state = WSPI_COMPLETE;
    (wspip)->config->end_cb(wspip);
    if ((wspip)->state == WSPI_COMPLETE)
      (wspip)->state = WSPI_READY;
    else
      (wspip)->state = WSPI_READY;
    _wspi_wakeup_isr(wspip, MSG_OK);
  }
}

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

| in | wspip | pointer to the WSPIDriver object |

Function Class:

Not an API, this function is for internal use only.

7.52.2.10 _wspi_error_code

#define _wspi_error_code(
  wspip
)

Value:

\{
  \}
if ((wspip)->config->error_cb) {
    (wspip)->state = WSPI_COMPLETE;
    if ((wspip)->state == WSPI_COMPLETE)
        (wspip)->config->error_cb(wspip);
    else
        (wspip)->state = WSPI_READY;
    _wspi_wakeup_isr(wspip, MSG_RESET);
}

Common error ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

**Note**

This macro is meant to be used in the low level drivers implementation only.

**Parameters**

| in | wspip | pointer to the WSPIDriver object |

**Function Class:**

Not an API, this function is for internal use only.

### 7.52.2.11 PLATFORM_WSPI_USE_WSPI1

```c
#define PLATFORM_WSPI_USE_WSPI1 FALSE
```

WSPID1 driver enable switch.

If set to `TRUE` the support for WSPID1 is included.

**Note**

The default is `FALSE`.

### 7.52.2.12 wspi_lld_driver_fields

```c
#define wspi_lld_driver_fields
```

**Value:**

```c
/* Dummy field, it is not needed. */
uint32_t dummy
```

Low level fields of the WSPI driver structure.
7.52.2.13 wspi_lld_config_fields

#define wspi_lld_config_fields

Value:
/* Dummy configuration, it is not needed.*\
 uint32_t dummy

Low level fields of the WSPI configuration structure.

7.52.3 Typedef Documentation

7.52.3.1 WSPIDriver

typedef struct hal_wspi_driver WSPIDriver

Type of a structure representing an WSPI driver.

7.52.3.2 WSPIConfig

typedef struct hal_wspi_config WSPIConfig

Type of a structure representing an WSPI driver configuration.

7.52.3.3 wspicallback_t

typedef void(* wspicallback_t) (WSPIDriver *wspip)

Type of a WSPI notification callback.

Parameters

\begin{tabular}{ll}
\textbf{in} & \textbf{wspip} & pointer to the WSPIDriver object triggering the callback \\
\end{tabular}

7.52.4 Enumeration Type Documentation

7.52.4.1 wspistate_t

enum wspistate_t
Driver state machine possible states.

**Enumerator**

<table>
<thead>
<tr>
<th>WSPI_UNINIT</th>
<th>Not initialized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSPI_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>WSPI_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>WSPI_SEND</td>
<td>Sending data.</td>
</tr>
<tr>
<td>WSPI_RECEIVE</td>
<td>Receiving data.</td>
</tr>
<tr>
<td>WSPI_COMPLETE</td>
<td>Asynchronous operation complete.</td>
</tr>
<tr>
<td>WSPI_MEMMAP</td>
<td>In memory mapped mode.</td>
</tr>
</tbody>
</table>

### 7.52.5 Function Documentation

#### 7.52.5.1 wspiInit()

```c
void wspiInit (  
    void  )
```

WSPI Driver initialization.

**Note**

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

**Function Class:**

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:
7.52.5.2 wspiObjectInit()

```c
void wspiObjectInit (  
    WSPIDriver * wspip )
```

Initializes the standard part of a WSPIDriver structure.
Parameters

| out | wspip | pointer to the WSPIDriver object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
  wspiObjectInit  osalMutexObjectInit
```

7.52.5.3 wspiStart()

```c
msg_t wspiStart (  
    WSPIDriver * wspip,  
    const WSPIConfig * config )
```

Configures and activates the WSPI peripheral.

Parameters

| in | wspip | pointer to the WSPIDriver object |
| in | config | pointer to the WSPIConfig object |

Returns

The operation status.
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    osalSysLock

    wspiStart

    wspi_lld_start
```

### 7.52.5.4  `wspiStop()`

```c
void wspiStop (  
    WSPIDriver * wspip )
```

Deactivates the WSPI peripheral.

**Note**

Deactivating the peripheral also enforces a release of the slave select line.

**Parameters**

- `in` `wspip` pointer to the WSPIDriver object
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
    osalSysLock
    wspiStop
    osalSysUnlock
    wspi_lld_stop
```

7.52.5.5 wspiStartCommand()

```c
void wspiStartCommand ( 
    WSPIDriver * wspip, 
    const wspi_command_t * cmdp )
```

Sends a command without data phase.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wspip</code></td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td><code>cmdp</code></td>
<td>pointer to the command descriptor</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
wspiStartCommand
  ↓
osalSysLock
  ↓
osalSysUnlock
```

### 7.52.5.6 wspiStartSend()

```c
void wspiStartSend (
    WSPIDriver ∗ wspip,
    const wspi_command_t ∗ cmdp,
    size_t n,
    const uint8_t ∗ txbuf )
```

Sends a command with data over the WSPI bus.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to send</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

7.52.5.7 wspiStartReceive()

```c
void wspiStartReceive (  
    WSPIDriver * wspip,
    const wspi_command_t * cmdp,
    size_t n,
    uint8_t * rxbuf  )
```

Sends a command then receives data over the WSPI bus.

Postcondition
At the end of the operation the configured callback is invoked.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip  pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmdp   pointer to the command descriptor</td>
</tr>
<tr>
<td>in</td>
<td>n      number of bytes to send</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf  the pointer to the receive buffer</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>wspiStartReceive</th>
</tr>
</thead>
<tbody>
<tr>
<td>osalSysLock</td>
</tr>
<tr>
<td>osalSysUnlock</td>
</tr>
</tbody>
</table>
```

7.52.5.8  wspiCommand()

```c
bool wspiCommand (  
    WSPIDriver * wspip,  
    const wspi_command_t * cmdp )
```

Sends a command without data phase.

Precondition

In order to use this function the option WSPI_USE_WAIT must be enabled.

In order to use this function the driver must have been configured without callbacks (end_cb = NULL).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>wspip</th>
<th>pointer to the WSPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if the operation failed because HW issues.</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.52.5.9 wspiSend()

bool wspiSend (
    WSPIDriver * wspip,
    const wspi_command_t * cmdp,
    size_t n,
    const uint8_t * txbuf )

Sends a command with data over the WSPI bus.

Precondition
In order to use this function the option WSPI_USE_WAIT must be enabled.
In order to use this function the driver must have been configured without callbacks (end_cb = NULL).

Parameters

<table>
<thead>
<tr>
<th></th>
<th>wspip</th>
<th>pointer to the WSPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>number of bytes to send</td>
</tr>
<tr>
<td></td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>true</td>
<td>if the operation failed because HW issues.</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.52.5.10 wspiReceive()

bool wspiReceive ( 
    WSPIDriver * wspip, 
    const wspi_command_t * cmdp, 
    size_t n, 
    uint8_t * rxbuf )

Sends a command then receives data over the WSPI bus.
Precondition

In order to use this function the option WSPI_USE_WAIT must be enabled.

In order to use this function the driver must have been configured without callbacks \((end\_cb = \text{NULL})\).

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to send</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

- \text{false} if the operation succeeded.
- \text{true} if the operation failed because HW issues.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.52.5.11 \text{wspiMapFlash()}

```c
void wspiMapFlash ( 
    WSPIDriver * wspip, 
    const wspi_command_t * cmdp, 
    uint8_t ** addrp )
```

Maps in memory space a WSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>out</td>
<td>addrp</td>
<td>pointer to the memory start address of the mapped flash or NULL</td>
</tr>
</tbody>
</table>
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.52.5.12 wspiUnmapFlash()

```c
void wspiUnmapFlash (
    WSPIDriver * wspip )
```

Unmaps from memory space a WSPI flash device.

**Postcondition**

The memory flash device must be re-initialized for normal commands exchange.

**Parameters**

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
<td>pointer to the WSPIDriver object</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
wspiUnmapFlash
   °-
   osalSysLock
   °-
   osalSysUnlock
```

### 7.52.5.13 wspiAcquireBus()

```c
void wspiAcquireBus (  
    WSPIDriver * wspip )
```

Gains exclusive access to the WSPI bus.

This function tries to gain ownership to the WSPI bus, if the bus is already being used then the invoking thread is queued.

**Precondition**

In order to use this function the option `WSPI_USE_MUTUAL_EXCLUSION` must be enabled.
Parameters

- **in** `wspip` pointer to the `WSPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call graph](call_graph.png)

7.52.5.14  `wspiReleaseBus()`

```c
void wspiReleaseBus ( WSPIDriver *wspip )
```

Releases exclusive access to the WSPI bus.

Precondition

In order to use this function the option `WSPI_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

- **in** `wspip` pointer to the `WSPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call graph](call_graph.png)
7.52.5.15  wspi_lld_init()

void wspi_lld_init ( 
        void )

Low level WSPI driver initialization.

Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

7.52.5.16  wspi_lld_start()

void wspi_lld_start ( 
        WSPIDriver * wspip )

Configures and activates the WSPI peripheral.

Parameters
in  wspip  pointer to the WSPIDriver object

Function Class:
Not an API, this function is for internal use only.

7.52.5.17  wspi_lld_stop()

void wspi_lld_stop ( 
        WSPIDriver * wspip )

Deactivates the WSPI peripheral.
Parameters

| in  | wspip | pointer to the WSPIDriver object |

Function Class:
Not an API, this function is for internal use only.

7.52.5.18  wspi_lld_command()

```c
void wspi_lld_command (  
    WSPIDriver * wspip,  
    const wspi_command_t * cmdp )
```

Sends a command without data phase.

Postcondition
At the end of the operation the configured callback is invoked.

Parameters

| in  | wspip | pointer to the WSPIDriver object |
| in  | cmdp  | pointer to the command descriptor |

Function Class:
Not an API, this function is for internal use only.

7.52.5.19  wspi_lld_send()

```c
void wspi_lld_send (  
    WSPIDriver * wspip,  
    const wspi_command_t * cmdp,  
    size_t n,  
    const uint8_t * txbuf )
```

Sends a command with data over the WSPI bus.

Postcondition
At the end of the operation the configured callback is invoked.
7.52 WSPI Driver

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in wspip</td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>in n</td>
<td>number of bytes to send</td>
</tr>
<tr>
<td>in txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.52.5.20 wspi_lld_receive()

```c
void wspi_lld_receive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf )
```

Sends a command then receives data over the WSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in wspip</td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>in n</td>
<td>number of bytes to send</td>
</tr>
<tr>
<td>out rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.52.5.21 wspi_lld_map_flash()

```c
void wspi_lld_map_flash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp )
```

Maps in memory space a WSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>wspip</th>
<th>pointer to the WSPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>out</td>
<td>addrp</td>
<td>pointer to the memory start address of the mapped flash or NULL</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.

7.52.5.22  wspi_lld_unmap_flash()

```c
void wspi_lld_unmap_flash ( 
    WSPIDriver * wspip )
```

Unmaps from memory space a WSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

| in  | wspip | pointer to the WSPIDriver object |

Function Class:

Not an API, this function is for internal use only.

7.52.6  Variable Documentation

7.52.6.1  WSPID1

```c
WSPIDriver WSPID1
```

WSPID1 driver identifier.
7.53 HAL

Hardware Abstraction Layer.

7.53.1 Detailed Description

Hardware Abstraction Layer.

Under ChibiOS the set of the various device driver interfaces is called the HAL subsystem: Hardware Abstraction Layer. The HAL is the abstract interface between ChibiOS applications and hardware.

7.53.2 HAL Device Drivers Architecture

The HAL contains several kind of modules:

- Normal Device Drivers
- Complex Device Drivers
- Interfaces
- Inner Code

7.53.3 HAL Normal Device Drivers

Normal device are meant to interface the application to the underlying hardware through an high level API. Normal Device Drivers are split in two layers:

- High Level Device Driver (HLD). This layer contains the definitions of the driver's APIs and the platform independent part of the driver.
  An HLD is composed by two files:
  - hal_<driver>.c, the HLD implementation file. This file must be included in the Makefile in order to use the driver.
  - hal_<driver>.h, the HLD header file. This file is implicitly included by the HAL header file hal.h.

- Low Level Device Driver (LLD). This layer contains the platform dependent part of the driver.
  A LLD is composed by two files:
  - hal_<driver>_lld.c, the LLD implementation file. This file must be included in the Makefile in order to use the driver.
  - hal_<driver>_lld.h, the LLD header file. This file is implicitly included by the HLD header file.
7.53.4 HAL Complex Device Drivers

It is a class of device drivers that offer an high level API but do not use the hardware directly. Complex device drivers use other drivers for accessing the machine resources.

7.53.5 HAL Interfaces

An interface is a binary structure allowing the access to a service using virtual functions. This allows to create drivers that can be accessed using a common interface. The concept of interface is commonly found in object-oriented languages like Java or C++, their meaning in ChibiOS/HAL is exactly the same.

7.53.6 HAL Inner Code

Some modules are shared among multiple device drivers and are not necessarily meant to be used by the application layer.
Modules

- **Peripheral Interfaces**
  HAL Abstract Peripheral Interfaces.
- **Configuration**
  HAL Configuration.
- **Normal Drivers**
  HAL Normal Drivers.
- **Complex Drivers**
  HAL Complex Drivers.
- **Interfaces and Classes**
  HAL Interfaces and Classes.
- **Inner Code**
  HAL Inner Code.
- **Support Code**
  HAL Support Code.
- **OSAL**
  Operating System Abstraction Layer.
7.54 Configuration

HAL Configuration.

7.54.1 Detailed Description

HAL Configuration.

The file halconf.h contains the high level settings for all the drivers supported by the HAL. The low level, platform dependent, settings are contained in the mcuconf.h file instead and are describe in the various platforms reference manuals.

Macros

- `#define HAL_USE_PAL TRUE`  
  Enables the PAL subsystem.
- `#define HAL_USE_ADC TRUE`  
  Enables the ADC subsystem.
- `#define HAL_USE_CAN TRUE`  
  Enables the CAN subsystem.
- `#define HAL_USE_CRY TRUE`  
  Enables the cryptographic subsystem.
- `#define HAL_USE_DAC TRUE`  
  Enables the DAC subsystem.
- `#define HAL_USE_EFL TRUE`  
  Enables the EFlash subsystem.
- `#define HAL_USE_GPT TRUE`  
  Enables the GPT subsystem.
- `#define HAL_USE_I2C TRUE`  
  Enables the i2c subsystem.
- `#define HAL_USE_I2S TRUE`  
  Enables the I2S subsystem.
- `#define HAL_USE_ICU TRUE`  
  Enables the ICU subsystem.
- `#define HAL_USE_MAC TRUE`  
  Enables the MAC subsystem.
- `#define HAL_USE_MMC_SPI TRUE`  
  Enables the MMC_SPI subsystem.
- `#define HAL_USE_PWM TRUE`  
  Enables the PWM subsystem.
- `#define HAL_USE_RTC TRUE`  
  Enables the RTC subsystem.
- `#define HAL_USE_SDC TRUE`  
  Enables the SDC subsystem.
- `#define HAL_USE_SERIAL TRUE`  
  Enables the SERIAL subsystem.
- `#define HAL_USE_SERIAL_USB TRUE`  
  Enables the SERIAL over USB subsystem.
- `#define HAL_USE_SIO TRUE`
7.54 Configuration

Enables the SIO subsystem.

• #define HAL_USE_SPI TRUE
  Enables the SPI subsystem.

• #define HAL_USE_TRNG TRUE
  Enables the TRNG subsystem.

• #define HAL_USE_UART TRUE
  Enables the UART subsystem.

• #define HAL_USE_USB TRUE
  Enables the USB subsystem.

• #define HAL_USE_WDG TRUE
  Enables the WDG subsystem.

• #define HAL_USE_WSPI TRUE
  Enables the WSPI subsystem.

• #define PAL_USE_CALLBACKS FALSE
  Enables synchronous APIs.

• #define PAL_USE_WAIT FALSE
  Enables synchronous APIs.

• #define ADC_USE_WAIT TRUE
  Enables synchronous APIs.

• #define ADC_USE_MUTUAL_EXCLUSION TRUE
  Enables the adcAcquireBus() and adcReleaseBus() APIs.

• #define CAN_USE_SLEEP_MODE TRUE
  Sleep mode related APIs inclusion switch.

• #define CAN_ENFORCE_USE_CALLBACKS FALSE
  Enforces the driver to use direct callbacks rather than OSAL events.

• #define HAL_CRY_USE_FALLBACK FALSE
  Enables the SW fall-back of the cryptographic driver.

• #define HAL_CRY_ENFORCE_FALLBACK FALSE
  Makes the driver forcibly use the fall-back implementations.

• #define DAC_USE_WAIT TRUE
  Enables synchronous APIs.

• #define DAC_USE_MUTUAL_EXCLUSION TRUE
  Enables the dacAcquireBus() and dacReleaseBus() APIs.

• #define I2C_USE_MUTUAL_EXCLUSION TRUE
  Enables the mutual exclusion APIs on the I2C bus.

• #define MAC_USE_ZERO_COPY TRUE
  Enables the zero-copy API.

• #define MAC_USE_EVENTS TRUE
  Enables an event sources for incoming packets.

• #define MMC_IDLE_TIMEOUT_MS 1000
  Timeout before assuming a failure while waiting for card idle.

• #define MMC_USE_MUTUAL_EXCLUSION TRUE
  Mutual exclusion on the SPI bus.

• #define SDC_INIT_RETRY 100
  Number of initialization attempts before rejecting the card.

• #define SDC_MMC_SUPPORT TRUE
  Include support for MMC cards.

• #define SDC_NICE_WAITING TRUE
  Delays insertions.

• #define SDC_INIT_OCR_V20 0x50FF8000U
  OCR initialization constant for V20 cards.
• #define SDC_INIT_OCR 0x80100000U
  OCR initialization constant for non-V20 cards.
• #define SERIAL_DEFAULT_BITRATE 38400
  Default bit rate.
• #define SERIAL_BUFFERS_SIZE 16
  Serial buffers size.
• #define SIO_DEFAULT_BITRATE 38400
  Default bit rate.
• #define SIO_USE_SYNCHRONIZATION TRUE
  Support for thread synchronization API.
• #define SERIAL_USB_BUFFERS_SIZE 256
  Serial over USB buffers size.
• #define SERIAL_USB_BUFFERS_NUMBER 2
  Serial over USB number of buffers.
• #define SPI_USE_WAIT TRUE
  Enables synchronous APIs.
• #define SPI_USE_ASSERT_ON_ERROR TRUE
  Inserts an assertion on function errors before returning.
• #define SPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the spiAcquireBus() and spiReleaseBus() APIs.
• #define SPI_SELECT_MODE SPI_SELECT_MODE_PAD
  Handling method for SPI CS line.
• #define UART_USE_WAIT TRUE
  Enables synchronous APIs.
• #define UART_USE_MUTUAL_EXCLUSION TRUE
  Enables the uartAcquireBus() and uartReleaseBus() APIs.
• #define USB_USE_WAIT TRUE
  Enables synchronous APIs.
• #define WSPI_USE_WAIT TRUE
  Enables synchronous APIs.
• #define WSPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the wspiAcquireBus() and wspiReleaseBus() APIs.

7.54.2 Macro Definition Documentation

7.54.2.1 HAL_USE_PAL

#define HAL_USE_PAL TRUE

Enables the PAL subsystem.
7.54.2.2  HAL_USE_ADC

#define HAL_USE_ADC TRUE

Enables the ADC subsystem.

7.54.2.3  HAL_USE_CAN

#define HAL_USE_CAN TRUE

Enables the CAN subsystem.

7.54.2.4  HAL_USE_CRY

#define HAL_USE_CRY TRUE

Enables the cryptographic subsystem.

7.54.2.5  HAL_USE_DAC

#define HAL_USE_DAC TRUE

Enables the DAC subsystem.

7.54.2.6  HAL_USE_EFL

#define HAL_USE_EFL TRUE

Enables the EFlash subsystem.

7.54.2.7  HAL_USE_GPT

#define HAL_USE_GPT TRUE

Enables the GPT subsystem.
7.54.2.8 HAL_USE_I2C

#define HAL_USE_I2C TRUE

Enables the I2C subsystem.

7.54.2.9 HAL_USE_I2S

#define HAL_USE_I2S TRUE

Enables the I2S subsystem.

7.54.2.10 HAL_USE_ICU

#define HAL_USE_ICU TRUE

Enables the ICU subsystem.

7.54.2.11 HAL_USE_MAC

#define HAL_USE_MAC TRUE

Enables the MAC subsystem.

7.54.2.12 HAL_USE_MMC_SPI

#define HAL_USE_MMC_SPI TRUE

Enables the MMC_SPI subsystem.

7.54.2.13 HAL_USE_PWM

#define HAL_USE_PWM TRUE

Enables the PWM subsystem.
7.54.2.14 HAL_USE_RTC

#define HAL_USE_RTC TRUE

Enables the RTC subsystem.

7.54.2.15 HAL_USE_SDC

#define HAL_USE_SDC TRUE

Enables the SDC subsystem.

7.54.2.16 HAL_USE_SERIAL

#define HAL_USE_SERIAL TRUE

Enables the SERIAL subsystem.

7.54.2.17 HAL_USE_SERIAL_USB

#define HAL_USE_SERIAL_USB TRUE

Enables the SERIAL over USB subsystem.

7.54.2.18 HAL_USE_SIO

#define HAL_USE_SIO TRUE

Enables the SIO subsystem.

7.54.2.19 HAL_USE_SPI

#define HAL_USE_SPI TRUE

Enables the SPI subsystem.
7.54.2.20  HAL_USE_TRNG

#define HAL_USE_TRNG TRUE

Enables the TRNG subsystem.

7.54.2.21  HAL_USE_UART

#define HAL_USE_UART TRUE

Enables the UART subsystem.

7.54.2.22  HAL_USE_USB

#define HAL_USE_USB TRUE

Enables the USB subsystem.

7.54.2.23  HAL_USE_WDG

#define HAL_USE_WDG TRUE

Enables the WDG subsystem.

7.54.2.24  HAL_USE_WSPI

#define HAL_USE_WSPI TRUE

Enables the WSPI subsystem.

7.54.2.25  PAL_USE_CALLBACKS

#define PAL_USE_CALLBACKS FALSE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.
7.54.2.26  PAL_USE_WAIT

#define PAL_USE_WAIT FALSE
Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.54.2.27  ADC_USE_WAIT

#define ADC_USE_WAIT TRUE
Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.54.2.28  ADC_USE_MUTUAL_EXCLUSION

#define ADC_USE_MUTUAL_EXCLUSION TRUE
Enables the adcAcquireBus() and adcReleaseBus() APIs.

Note
Disabling this option saves both code and data space.

7.54.2.29  CAN_USE_SLEEP_MODE

#define CAN_USE_SLEEP_MODE TRUE
Sleep mode related APIs inclusion switch.

7.54.2.30  CAN_ENFORCE_USE_CALLBACKS

#define CAN_ENFORCE_USE_CALLBACKS FALSE
Enforces the driver to use direct callbacks rather than OSAL events.
7.54.2.31  HAL_CRY_USE_FALLBACK

#define HAL_CRY_USE_FALLBACK FALSE

Enables the SW fall-back of the cryptographic driver.
When enabled, this option, activates a fall-back software implementation for algorithms not supported by the under-
lying hardware.

Note
Fall-back implementations may not be present for all algorithms.

7.54.2.32  HAL_CRY_ENFORCE_FALLBACK

#define HAL_CRY_ENFORCE_FALLBACK FALSE

Makes the driver forcibly use the fall-back implementations.

7.54.2.33  DAC_USE_WAIT

#define DAC_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.54.2.34  DAC_USE_MUTUAL_EXCLUSION

#define DAC_USE_MUTUAL_EXCLUSION TRUE

Enables the dacAcquireBus() and dacReleaseBus() APIs.

Note
Disabling this option saves both code and data space.
7.54.2.35  **I2C_USE_MUTUAL_EXCLUSION**

```c
#define I2C_USE_MUTUAL_EXCLUSION TRUE
```

Enables the mutual exclusion APIs on the I2C bus.

7.54.2.36  **MAC_USE_ZERO_COPY**

```c
#define MAC_USE_ZERO_COPY TRUE
```

Enables the zero-copy API.

7.54.2.37  **MAC_USE_EVENTS**

```c
#define MAC_USE_EVENTS TRUE
```

Enables an event sources for incoming packets.

7.54.2.38  **MMC_IDLE_TIMEOUT_MS**

```c
#define MMC_IDLE_TIMEOUT_MS 1000
```

Timeout before assuming a failure while waiting for card idle.

**Note**

Time is in milliseconds.

7.54.2.39  **MMC_USE_MUTUAL_EXCLUSION**

```c
#define MMC_USE_MUTUAL_EXCLUSION TRUE
```

Mutual exclusion on the SPI bus.
7.54.2.40  SDC_INIT_RETRY

#define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.54.2.41  SDC_MMC_SUPPORT

#define SDC_MMC_SUPPORT TRUE

Include support for MMC cards.

Note

MMC support is not yet implemented so this option must be kept at FALSE.

7.54.2.42  SDC_NICE_WAITING

#define SDC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

7.54.2.43  SDC_INIT_OCR_V20

#define SDC_INIT_OCR_V20 0x50FF8000U

OCR initialization constant for V20 cards.

7.54.2.44  SDC_INIT_OCR

#define SDC_INIT_OCR 0x80100000U

OCR initialization constant for non-V20 cards.
7.54 Configuration

7.54.2.45 SERIAL_DEFAULT_BITRATE

#define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.54.2.46 SERIAL_BUFFERS_SIZE

#define SERIAL_BUFFERS_SIZE 16

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

Note

The default is 16 bytes for both the transmission and receive buffers.

7.54.2.47 SIO_DEFAULT_BITRATE

#define SIO_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.54.2.48 SIO_USE_SYNCHRONIZATION

#define SIO_USE_SYNCHRONIZATION TRUE

Support for thread synchronization API.

7.54.2.49 SERIAL_USB_BUFFERS_SIZE

#define SERIAL_USB_BUFFERS_SIZE 256

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

Note

The default is 256 bytes for both the transmission and receive buffers.
7.54.2.50  SERIAL_USB_BUFFERS_NUMBER

#define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note
   The default is 2 buffers.

7.54.2.51  SPI_USE_WAIT

#define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note
   Disabling this option saves both code and data space.

7.54.2.52  SPI_USE_ASSERT_ON_ERROR

#define SPI_USE_ASSERT_ON_ERROR TRUE

Inserts an assertion on function errors before returning.

7.54.2.53  SPI_USE_MUTUAL_EXCLUSION

#define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the spiAcquireBus() and spiReleaseBus() APIs.

Note
   Disabling this option saves both code and data space.
7.54.2.54  SPI_SELECT_MODE

#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD

Handling method for SPI CS line.

Note
Disabling this option saves both code and data space.

7.54.2.55  UART_USE_WAIT

#define UART_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.54.2.56  UART_USE_MUTUAL_EXCLUSION

#define UART_USE_MUTUAL_EXCLUSION TRUE

Enables the uartAcquireBus() and uartReleaseBus() APIs.

Note
Disabling this option saves both code and data space.

7.54.2.57  USB_USE_WAIT

#define USB_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.
7.54.2.58  WSPI_USE_WAIT

#define WSPI_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.54.2.59  WSPI_USE_MUTUAL_EXCLUSION

#define WSPI_USE_MUTUAL_EXCLUSION TRUE

Enables the \texttt{wspiAcquireBus()} and \texttt{wspiReleaseBus()} APIs.

Note
Disabling this option saves both code and data space.
7.55  Normal Drivers

HAL Normal Drivers.

7.55.1  Detailed Description

HAL Normal Drivers.

Modules

- HAL Driver
  Hardware Abstraction Layer.
- ADC Driver
  Generic ADC Driver.
- CAN Driver
  Generic CAN Driver.
- Crypto Driver
  Generic Crypto Driver.
- DAC Driver
  Generic DAC Driver.
- EFL Driver
  Generic Embedded Flash Driver.
- GPT Driver
  Generic GPT Driver.
- I2C Driver
  Generic I2C Driver.
- I2S Driver
  Generic I2S Driver.
- ICU Driver
  Generic ICU Driver.
- MAC Driver
  Generic MAC Driver.
- PAL Driver
  I/O Ports Abstraction Layer.
- PWM Driver
  Generic PWM Driver.
- RTC Driver
  Generic RTC Driver.
- SDC Driver
  Generic SD Card Driver.
- Serial Driver
  Generic Serial Driver.
- SIO Driver
  Generic SIO Driver.
- SPI Driver Selector
  SPI Driver selector.
- SPIv1 Driver
  Generic SPI Driver (legacy).
• SPIv2 Driver
  Generic SPI Driver.
• ST Driver
  Generic System Tick Driver.
• TRNG Driver
  Generic True Random Numbers Generator Driver.
• UART Driver
  Generic UART Driver.
• USB Driver
  Generic USB Driver.
• WDG Driver
  Generic WDG Driver.
• WSPI Driver
  Generic WSPI Driver.
7.56 Complex Drivers

HAL Complex Drivers.

7.56.1 Detailed Description

HAL Complex Drivers.

Modules

• **Buffered SIO Driver**
  
  Buffered SIO Driver.

• **Managed Flash Storage Driver**
  
  Managed Flash Storage Driver.

• **MMC over SPI Driver**
  
  Generic MMC driver.

• **Serial NOR Flash Driver**
  
  Serial NOR Flash driver.

• **Serial over USB Driver**
  
  Serial over USB Driver.
7.57 Interfaces and Classes

HAL Interfaces and Classes.

7.57.1 Detailed Description

HAL Interfaces and Classes.

Modules

- Blocks
- Streams
- Base Object
7.58 Inner Code

HAL Inner Code.

7.58.1 Detailed Description

HAL Inner Code.

Modules

- I/O Buffers Queues
- MMC/SD Block Device
- I/O Bytes Queues
7.59  Support Code

HAL Support Code.

7.59.1  Detailed Description

HAL Support Code.

Modules

- MII/RMII Header
  MII/RMII Support Header.
- USB CDC Header
  USB CDC Support Header.
7.60 OSAL

Operating System Abstraction Layer.

7.60.1 Detailed Description

Operating System Abstraction Layer.

The OSAL

The OSAL is the link between ChibiOS/HAL and services provided by operating systems like:

- Critical Zones handling.
- Interrupts handling.
- Runtime Errors management.
- Inter-task synchronization.
- Task-ISR synchronization.
- Time management.
- Events.

ChibiOS/HAL is designed to tightly integrate with the underlying RTOS in order to provide the best experience to developers and minimize integration issues. This section describes the API that OSALs are expected to expose to the HAL.

RTOS Requirements

The OSAL API closely resembles the ChibiOS/RT API, for obvious reasons, however an OSAL module can be implemented for any reasonably complete RTOS or even a RTOS-less bare metal machine, if required. In order to be able to support an HAL an RTOS should support the following minimal set of features:

- Task-level critical zones API.
- ISR-level critical zones API, only required on those CPU architectures supporting preemptable ISRs like Cortex-Mx cores.
- Ability to invoke API functions from inside a task critical zone. Functions that are required to support this feature are marked with an "I" or "S" letter at the end of the name.
- Ability to invoke API functions from inside an ISR critical zone. Functions that are required to support this feature are marked with an "I" letter at the end of the name.
- Tasks Queues or Counting Semaphores with Timeout capability.
- Ability to suspend a task and wakeup it from ISR with Timeout capability.
- Event flags, the mechanism can be simulated using callbacks in case the RTOS does not support it.
- Mutual Exclusion mechanism like Semaphores or Mutexes.

All the above requirements can be satisfied even on naked HW with a very think SW layer. In case that the HAL is required to work without an RTOS.
Supported RTOSes

The RTOSes supported out of the box are:

- ChibiOS/RT
- ChibiOS/NIL

Implementations have also been successfully created on RTOSes not belonging to the ChibiOS products family but are not supported as a core feature of ChibiOS/HAL.

Macros

- `#define OSAL_DBG_ENABLE_ASSERTS FALSE`
  Enables OSAL assertions.
- `#define OSAL_DBG_ENABLE_CHECKS FALSE`
  Enables OSAL functions parameters checks.

Common constants

- `#define FALSE 0`
- `#define TRUE 1`
- `#define OSAL_SUCCESS false`
- `#define OSAL_FAILED true`

Messages

- `#define MSG_OK (msg_t)0`
- `#define MSG_TIMEOUT (msg_t)-1`
- `#define MSG_RESET (msg_t)-2`

Special time constants

- `#define TIME_IMMEDIATE ((sysinterval_t)0)`
- `#define TIME_INFINITE ((sysinterval_t)-1)`

Systick modes.

- `#define OSAL_ST_MODE_NONE 0`
- `#define OSAL_ST_MODE_PERIODIC 1`
- `#define OSAL_ST_MODE_FREERUNNING 2`

Systick parameters.

- `#define OSAL_ST_RESOLUTION 32`
  Size in bits of the `systick_t` type.
- `#define OSAL_ST_FREQUENCY 1000`
  Required systick frequency or resolution.
- `#define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC`
  Systick mode required by the underlying OS.
### IRQ-related constants

- `#define OSAL_IRQ_PRIORITY_LEVELS 16U`
  Total priority levels.
- `#define OSAL_IRQ_MAXIMUM_PRIORITY 0U`
  Highest IRQ priority for HAL drivers.

### Debug related macros

- `#define osalDbgAssert(c, remark)`
  Condition assertion.
- `#define osalDbgCheck(c)`
  Function parameters check.
- `#define osalDbgCheckClassI()`
  I-Class state check.
- `#define osalDbgCheckClassS()`
  S-Class state check.

### IRQ service routines wrappers

- `#define OSAL_IRQ_IS_VALID_PRIORITY(n) (((n) >= OSAL_IRQ_MAXIMUM_PRIORITY) && ((n) < OSAL_IRQ_PRIORITY_LEVELS))`
  Priority level verification macro.
- `#define OSAL_IRQ_PROLOGUE()`
  IRQ prologue code.
- `#define OSAL_IRQ_EPILOGUE()`
  IRQ epilogue code.
- `#define OSAL_IRQ_HANDLER(id) void id(void)`
  IRQ handler function declaration.

### Time conversion utilities

- `#define OSAL_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)OSAL_ST_FREQUENCY))`
  Seconds to time interval.
- `#define OSAL_MS2I(msecs)`
  Milliseconds to time interval.
- `#define OSAL_US2I(usecs)`
  Microseconds to time interval.
- `#define OSAL_I2S(interval)`
  Time interval to seconds.
- `#define OSAL_I2MS(interval)`
  Time interval to milliseconds.
- `#define OSAL_I2US(interval)`
  Time interval to microseconds.
Time conversion utilities for the realtime counter

- #define OSAL_S2RTC(freq, sec) ((freq) \times (sec))
  Seconds to realtime counter.
- #define OSAL_MS2RTC(freq, msec) ((rtcnt_t)(((freq) + 999UL) / 1000UL) \times (msec))
  Milliseconds to realtime counter.
- #define OSAL_US2RTC(freq, usec) ((rtcnt_t)(((freq) + 999999UL) / 1000000UL) \times (usec))
  Microseconds to realtime counter.

Sleep macros using absolute time

- #define osalThreadSleepSeconds(secs) osalThreadSleep(OSAL_S2I(secs))
  Delays the invoking thread for the specified number of seconds.
- #define osalThreadSleepMilliseconds(msecs) osalThreadSleep(OSAL_MS2I(msecs))
  Delays the invoking thread for the specified number of milliseconds.
- #define osalThreadSleepMicroseconds(usecs) osalThreadSleep(OSAL_US2I(usecs))
  Delays the invoking thread for the specified number of microseconds.

Typedefs

- typedef uint32_t syssts_t
  Type of a system status word.
- typedef int32_t msg_t
  Type of a message.
- typedef uint32_t systime_t
  Type of system time counter.
- typedef uint32_t sysinterval_t
  Type of system time interval.
- typedef uint64_t time_conv_t
  Type of time conversion variable.
- typedef uint32_t rtcnt_t
  Type of realtime counter.
- typedef void * thread_reference_t
  Type of a thread reference.
- typedef uint32_t eventflags_t
  Type of an event flags mask.
- typedef struct event_source event_source_t
  Type of an event flags object.
- typedef void(* eventcallback_t) (event_source_t * esp)
  Type of an event source callback.
- typedef uint32_t mutex_t
  Type of a mutex.

Data Structures

- struct event_source
  Events source object.
- struct threads_queue_t
  Type of a thread queue.
Functions

- **void osallInit (void)**
  
  OSAL module initialization.

- **void osaSysHalt (const char *reason)**
  
  System halt with error message.

- **void osaSysPolledDelayX (rtcnt_t cycles)**
  
  Polled delay.

- **void osaOsTimerHandlerI (void)**
  
  System timer handler.

- **void osaOsRescheduleS (void)**
  
  Checks if a reschedule is required and performs it.

- **systime_t osaOsGetSystemTimeX (void)**
  
  Current system time.

- **void osaThreadSleepS (sysinterval_t time)**
  
  Suspends the invoking thread for the specified time.

- **void osaThreadSleep (sysinterval_t time)**
  
  Suspends the invoking thread for the specified time.

- **msg_t osaThreadSuspendS (thread_reference_t *trp)**
  
  Sends the current thread sleeping and sets a reference variable.

- **msg_t osaThreadSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)**
  
  Sends the current thread sleeping and sets a reference variable.

- **void osaThreadResumeI (thread_reference_t *trp, msg_t msg)**
  
  Wakes up a thread waiting on a thread reference object.

- **void osaThreadResumeS (thread_reference_t *trp, msg_t msg)**
  
  Wakes up a thread waiting on a thread reference object.

- **msg_t osaThreadEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)**
  
  Enqueues the caller thread.

- **void osaThreadDequeueNextI (threads_queue_t *tqp, msg_t msg)**
  
  Dequeues and wakes up one thread from the queue, if any.

- **void osaThreadDequeueAllI (threads_queue_t *tqp, msg_t msg)**
  
  Dequeues and wakes up all threads from the queue.

- **void osaEventBroadcastFlagsI (event_source_t *esp, eventflags_t flags)**
  
  Add flags to an event source object.

- **void osaEventBroadcastFlags (event_source_t *esp, eventflags_t flags)**
  
  Add flags to an event source object.

- **void osaEventSetCallback (event_source_t *esp, eventcallback_t cb, void *param)**
  
  Event callback setup.

- **void osaMutextLock (mutex_t *mp)**
  
  Locks the specified mutex.

- **void osaMutextUnlock (mutex_t *mp)**
  
  Unlocks the specified mutex.

- **static void osaSysDisable (void)**
  
  Disables interrupts globally.

- **static void osaSysEnable (void)**
  
  Enables interrupts globally.

- **static void osaSysLock (void)**
  
  Enters a critical zone from thread context.

- **static void osaSysUnlock (void)**
  
  Leaves a critical zone from thread context.

- **static void osaSysLockFromISR (void)**
  

En ters a critical zone from ISR context.

- static void osalSysUnlockFromISR (void)
  Leaves a critical zone from ISR context.
- static syssts_t osalSysGetStatusAndLockX (void)
  Returns the execution status and enters a critical zone.
- static void osalSysRestoreStatusX (syssts_t sts)
  Restores the specified execution status and leaves a critical zone.
- static systime_t osalTimeAddX (systime_t systime, sysinterval_t interval)
  Adds an interval to a system time returning a system time.
- static sysinterval_t osalTimeDiffX (systime_t start, systime_t end)
  Subtracts two system times returning an interval.
- static bool osalTimeIsInRangeX (systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time window.
- static void osalThreadQueueObjectInit (threads_queue_t *tqp)
  Initializes a threads queue object.
- static void osalEventObjectInit (event_source_t *esp)
  Initializes an event source object.
- static void osalMutexObjectInit (mutex_t *mp)
  Initializes a mutex object.

Variables

- const char *osal_hal t_msg
  Pointer to a halt error message.
- const char *osal_halt_msg
  Pointer to a halt error message.

7.60.2 Macro Definition Documentation

7.60.2.1 OSAL ST_RESOLUTION

#define OSAL_ST_RESOLUTION 32

Size in bits of the systick_t type.

7.60.2.2 OSAL ST_FREQUENCY

#define OSAL_ST_FREQUENCY 1000

Required systick frequency or resolution.
7.60 OSAL

7.60.2.3 OSAL_ST_MODE

#define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC

Systick mode required by the underlying OS.

7.60.2.4 OSAL_IRQ_PRIORITY_LEVELS

#define OSAL_IRQ_PRIORITY_LEVELS 16U

Total priority levels.
Implementation not mandatory.

7.60.2.5 OSAL_IRQ_MAXIMUM_PRIORITY

#define OSAL_IRQ_MAXIMUM_PRIORITY 0U

Highest IRQ priority for HAL drivers.
Implementation not mandatory.

7.60.2.6 OSAL_DBG_ENABLE_ASSERTS

#define OSAL_DBG_ENABLE_ASSERTS FALSE

Enables OSAL assertions.

7.60.2.7 OSAL_DBG_ENABLE_CHECKS

#define OSAL_DBG_ENABLE_CHECKS FALSE

Enables OSAL functions parameters checks.

7.60.2.8 osalDbgAssert

#define osalDbgAssert (c, remark)

Value:

do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (OSAL_DBG_ENABLE_ASSERTS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            osalSysHalt(__func__);  
        }  
    }  
}  
while (false)

Condition assertion.
If the condition check fails then the OSAL panics with a message and halts.

Note

The condition is tested only if the OSAL_ENABLE_ASSERTIONS switch is enabled.
The remark string is not currently used except for putting a comment in the code about the assertion.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>c</th>
<th>the condition to be verified to be true</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>remark</td>
<td>a remark string</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.9 osalDbgCheck

```c
#define osalDbgCheck(
    c )
```

Value:

```c
/*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
if (OSAL_DBG_ENABLE_CHECKS != FALSE) {
    if (!c) {
        /*lint -restore*/
        osalSysHalt(__func__);
    }
} while (false)
```

Function parameters check.

If the condition check fails then the OSAL panics and halts.

Note
The condition is tested only if the OSAL_ENABLE_CHECKS switch is enabled.

Parameters

| in  | c       | the condition to be verified to be true |

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.10 osalDbgCheckClassI

```c
#define osalDbgCheckClassI( )
```

I-Class state check.

Note
Implementation is optional.
7.60.2.11 osalDbgCheckClassS

#define osalDbgCheckClassS()

S-Class state check.

Note
   Implementation is optional.

7.60.2.12 OSAL_IRQ_IS_VALID_PRIORITY

#define OSAL_IRQ_IS_VALID_PRIORITY(n) (((n) >= OSAL_IRQ_MAXIMUM_PRIORITY) && ((n) < OSAL_IRQ_PRIORITY_LEVELS))

Priority level verification macro.

7.60.2.13 OSAL_IRQ_PROLOGUE

#define OSAL_IRQ_PROLOGUE()

IRQ prologue code.
This macro must be inserted at the start of all IRQ handlers.

7.60.2.14 OSAL_IRQ_EPILOGUE

#define OSAL_IRQ_EPILOGUE()

IRQ epilogue code.
This macro must be inserted at the end of all IRQ handlers.

7.60.2.15 OSAL_IRQ_HANDLER

#define OSAL_IRQ_HANDLER(id) void id(void)

IRQ handler function declaration.
This macro hides the details of an ISR function declaration.

Parameters
   in id a vector name as defined in vectors.s
7.60.2.16 OSAL_S2I

#define OSAL_S2I(
    secs
) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)OSAL_ST_FREQUENCY))

Seconds to time interval.
Converts from seconds to system ticks number.

Note
The result is rounded upward to the next tick boundary.
Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

| in  | secs  | number of seconds |

Returns
The number of ticks.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.17 OSAL_MS2I

#define OSAL_MS2I(
    msecs
) Value:
    ((sysinterval_t)(((time_conv_t)(msecs) * (time_conv_t)OSAL_ST_FREQUENCY) + (time_conv_t)999) / (time_conv_t)1000))

Milliseconds to time interval.
Converts from milliseconds to system ticks number.

Note
The result is rounded upward to the next tick boundary.
Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.
Parameters

| in  | msecs | number of milliseconds |

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.18 OSAL_US2I

#define OSAL_US2I(
    usecs )

Value:

(sysinterval_t)(((time_conv_t)(usecs) * (time_conv_t)OSAL_S7_FREQUENCY) + 999999) / (time_conv_t)1000000

Microseconds to time interval.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

| in  | usecs | number of microseconds |

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.60.2.19 OSAL_I2S

#define OSAL_I2S(
    interval)

Value:
(time_secs_t)((time_conv_t)(interval) +
    (time_conv_t)OSAL_ST_FREQUENCY -
    (time_conv_t)1) / (time_conv_t)OSAL_ST_FREQUENCY)

Time interval to seconds.
Converts from system ticks number to seconds.

Note
The result is rounded up to the next second boundary.
Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

| in  | interval | interval in ticks |

Returns
The number of seconds.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.20 OSAL_I2MS

#define OSAL_I2MS(
    interval)

Value:
(time_msecs_t)(((time_conv_t)(interval) * (time_conv_t)1000) +
    (time_conv_t)OSAL_ST_FREQUENCY - (time_conv_t)1) / (time_conv_t)OSAL_ST_FREQUENCY)

Time interval to milliseconds.
Converts from system ticks number to milliseconds.

Note
The result is rounded up to the next millisecond boundary.
Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.
Parameters

in interval interval in ticks

Returns

The number of milliseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.21 OSAL_I2US

#define OSAL_I2US(
    interval)

Value:

(time_msecs_t)(((time_conv_t)(interval) * (time_conv_t)1000000) +
    (time_conv_t)OSAL_ST_FREQUENCY - (time_conv_t)1) /
    (time_conv_t)OSAL_ST_FREQUENCY)

Time interval to microseconds.

Converts from system ticks number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in interval interval in ticks

Returns

The number of microseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.60.2.22 OSAL_S2RTC

#define OSAL_S2RTC(
    freq,
    sec ) ((freq) * (sec))

Seconds to realtime counter.

Converts from seconds to realtime counter cycles.

Note

The macro assumes that freq >= 1.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>freq</td>
<td>clock frequency, in Hz, of the realtime counter</td>
</tr>
<tr>
<td>in</td>
<td>sec</td>
<td>number of seconds</td>
</tr>
</tbody>
</table>

Returns

The number of cycles.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.23 OSAL_MS2RTC

#define OSAL_MS2RTC(
    freq,
    msec ) (((freq) + 999UL) / 1000UL) * (msec)

Milliseconds to realtime counter.

Converts from milliseconds to realtime counter cycles.

Note

The result is rounded upward to the next millisecond boundary.

The macro assumes that freq >= 1000.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>freq</td>
<td>clock frequency, in Hz, of the realtime counter</td>
</tr>
<tr>
<td>in</td>
<td>msec</td>
<td>number of milliseconds</td>
</tr>
</tbody>
</table>
Returns

The number of cycles.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.24 OSAL_US2RTC

#define OSAL_US2RTC(
    freq,
    usec ) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) * (usec))

Microseconds to realtime counter.

Converts from microseconds to realtime counter cycles.

Note

The result is rounded upward to the next microsecond boundary.
The macro assumes that freq >= 1000000.

Parameters

| in freq | clock frequency, in Hz, of the realtime counter |
| in usec | number of microseconds |

Returns

The number of cycles.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.25 osalThreadSleepSeconds

#define osalThreadSleepSeconds(
    secs ) osalThreadSleep(OSAL_S2I(secs))

Delays the invoking thread for the specified number of seconds.

Note

The specified time is rounded up to a value allowed by the real system tick clock.
The maximum specifiable value is implementation dependent.
Parameters

| in  | secs | time in seconds, must be different from zero |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.26  osalThreadSleepMilliseconds

#define osalThreadSleepMilliseconds(
msecs ) osalThreadSleep(OSAL_MS2I(msecs))

Delays the invoking thread for the specified number of milliseconds.

Note

The specified time is rounded up to a value allowed by the real system tick clock.
The maximum specifiable value is implementation dependent.

Parameters

| in  | msecs | time in milliseconds, must be different from zero |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.2.27  osalThreadSleepMicroseconds

#define osalThreadSleepMicroseconds(
usecs ) osalThreadSleep(OSAL_US2I(usecs))

Delays the invoking thread for the specified number of microseconds.

Note

The specified time is rounded up to a value allowed by the real system tick clock.
The maximum specifiable value is implementation dependent.

Parameters

| in  | usecs | time in microseconds, must be different from zero |
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.3 Typedef Documentation

7.60.3.1 syssts_t

typedef uint32_t syssts_t

Type of a system status word.

7.60.3.2 msg_t

typedef int32_t msg_t

Type of a message.

7.60.3.3 systime_t

typedef uint32_t systime_t

Type of system time counter.

7.60.3.4 sysinterval_t

typedef uint32_t sysinterval_t

Type of system time interval.

7.60.3.5 time_conv_t

typedef uint64_t time_conv_t

Type of time conversion variable.

Note

This type must have double width than other time types, it is only used internally for conversions.
7.60.3.6 rtcnt_t

typedef uint32_t rtcnt_t

Type of realtime counter.

7.60.3.7 thread_reference_t

typedef void* thread_reference_t

Type of a thread reference.

7.60.3.8 eventflags_t

typedef uint32_t eventflags_t

Type of an event flags mask.

7.60.3.9 event_source_t

typedef struct event_source event_source_t

Type of an event flags object.

Note
The content of this structure is not part of the API and should not be relied upon. Implementers may define
this structure in an entirely different way.
Retrieval and clearing of the flags are not defined in this API and are implementation-dependent.

7.60.3.10 eventcallback_t

typedef void(* eventcallback_t) (event_source_t *esp)

Type of an event source callback.

Note
This type is not part of the OSAL API and is provided exclusively as an example and for convenience.
7.60.3.11 mutex_t

typedef uint32_t mutex_t

Type of a mutex.

Note
If the OS does not support mutexes or there is no OS then then mechanism can be simulated.

7.60.4 Function Documentation

7.60.4.1 osalInit()

void osalInit ( void )

OSAL module initialization.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.4.2 osalSysHalt()

void osalSysHalt ( const char * reason )

System halt with error message.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>reason</th>
<th>the halt message pointer</th>
</tr>
</thead>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
osalSysHalt osalSysDisable
```

7.60.4.3 osalSysPolledDelayX()

```c
void osalSysPolledDelayX ( 
    rtcnt_t cycles )
```

Polled delay.

Note

The real delay is always few cycles in excess of the specified value.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cycles</th>
<th>number of cycles</th>
</tr>
</thead>
</table>

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.60.4.4 osalOsTimerHandlerl()

```c
void osalOsTimerHandlerl ( 
    void )
```

System timer handler.

The handler is used for scheduling and Virtual Timers management.

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
7.60.4.5  osalOsRescheduleS()

void osalOsRescheduleS ( void )

Checks if a reschedule is required and performs it.

Note

I-Class functions invoked from thread context must not reschedule by themselves, an explicit reschedule using this function is required in this scenario.

Not implemented in this simplified OSAL.

Function Class:

This is an S-Class API, this function can be invoked from within a system lock zone by threads only.

7.60.4.6  osalOsGetSystemTimeX()

systime_t osalOsGetSystemTimeX ( void )

Current system time.

Returns the number of system ticks since the osalInit() invocation.

Note

The counter can reach its maximum and then restart from zero.

This function can be called from any context but its atomicity is not guaranteed on architectures whose word size is less than systime_t size.

Returns

The system time in ticks.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.60.4.7  osalThreadSleepS()

void osalThreadSleepS ( sysinterval_t time )

Suspends the invoking thread for the specified time.
### osalThreadSleep()

```c
void osalThreadSleep ( 
    sysinterval_t time 
)
```

Suspends the invoking thread for the specified time.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the delay in system ticks, the special values are handled as follow:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</td>
</tr>
</tbody>
</table>

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### osalThreadSuspendS()

```c
msg_t osalThreadSuspendS ( 
    thread_reference_t * trp 
)
```

Sends the current thread sleeping and sets a reference variable.

**Note**

This function must reschedule, it can only be called from thread context.

**Parameters**

| in  | trp | a pointer to a thread reference object |
Returns
The wake up message.

Function Class:
This is an S-Class API, this function can be invoked from within a system lock zone by threads only.

7.60.4.10 osalThreadSuspendTimeoutS()

```c
msg_t osalThreadSuspendTimeoutS ( 
    thread_reference_t * trp, 
    sysinterval_t timeout )
```

Sends the current thread sleeping and sets a reference variable.

Note
This function must reschedule, it can only be called from thread context.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>trp</code></td>
<td>a pointer to a thread reference object</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>the timeout in system ticks, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> the thread is not enqueued and the function returns MSG_TIMEOUT as if a timeout occurred.</td>
</tr>
</tbody>
</table>

Returns
The wake up message.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_TIMEOUT</code></td>
<td>if the operation timed out.</td>
</tr>
</tbody>
</table>

Function Class:
This is an S-Class API, this function can be invoked from within a system lock zone by threads only.

7.60.4.11 osalThreadResumeI()

```c
void osalThreadResumeI ( 
    thread_reference_t * trp, 
    msg_t msg )
```
Wakes up a thread waiting on a thread reference object.

Note
This function must not reschedule because it can be called from ISR context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>trp</th>
<th>a pointer to a thread reference object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message code</td>
</tr>
</tbody>
</table>

Function Class:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.60.4.12 osalThreadResumeS()

void osalThreadResumeS ( 
    thread_reference_t * trp, 
    msg_t msg )

Wakes up a thread waiting on a thread reference object.

Note
This function must reschedule, it can only be called from thread context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>trp</th>
<th>a pointer to a thread reference object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message code</td>
</tr>
</tbody>
</table>

Function Class:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.60.4.13 osalThreadEnqueueTimeoutS()

msg_t osalThreadEnqueueTimeoutS ( 
    threads_queue_t * tqp, 
    sysinterval_t timeout )

Enqueues the caller thread.
The caller thread is enqueued and put to sleep until it is dequeued or the specified timeouts expires.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tqp</th>
<th>pointer to the threads queue object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the timeout in system ticks, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE the thread is not enqueued and the function returns MSG_TIMEOUT as if a timeout occurred.</td>
</tr>
</tbody>
</table>

Returns

The message from osalQueueWakeupOneI() or osalQueueWakeupAllI() functions.

Return values

| MSG_TIMEOUT | if the thread has not been dequeued within the specified timeout or if the function has been invoked with TIME_IMMEDIATE as timeout specification. |

Function Class:

This is an S-Class API, this function can be invoked from within a system lock zone by threads only.

7.60.4.14 osalThreadDequeueNextI()

void osalThreadDequeueNextI ( 
    threads_queue_t * tqp,
    msg_t msg )

Dequeues and wakes up one thread from the queue, if any.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tqp</th>
<th>pointer to the threads queue object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message code</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.60.4.15 osalThreadDequeueAllI()

void osalThreadDequeueAllI ( 
    threads_queue_t * tqp,
    msg_t msg )
Dequeues and wakes up all threads from the queue.
Parameters

| in  | esp  | pointer to the event flags object |
| in  | flags | flags to be ORed to the flags mask |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.60.4.16 osalEventBroadcastFlagsI()

```c
void osalEventBroadcastFlagsI (  
   event_source_t * esp,  
   eventflags_t flags )
```

Add flags to an event source object.

Parameters

| in  | esp  | pointer to the event flags object |
| in  | flags | flags to be ORed to the flags mask |

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.60.4.17 osalEventBroadcastFlags()

```c
void osalEventBroadcastFlags (  
   event_source_t * esp,  
   eventflags_t flags )
```

Add flags to an event source object.

Parameters

| in  | esp  | pointer to the event flags object |
| in  | flags | flags to be ORed to the flags mask |
Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

```
osalEventBroadcastFlagsI
osalEventBroadcastFlags
osalSysLock
osalSysUnlock
```

7.60.4.18  osalEventSetCallback()

```c
void osalEventSetCallback (  
    event_source_t * esp,  
    eventcallback_t cb,  
    void * param  
)
```

Event callback setup.

Note

The callback is invoked from ISR context and can only invoke I-Class functions. The callback is meant to wakeup the task that will handle the event by calling `osalEventGetAndClearFlagsI()`.

This function is not part of the OSAL API and is provided exclusively as an example and for convenience.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>esp</th>
<th>pointer to the event flags object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
<td>pointer to the callback function</td>
</tr>
<tr>
<td>in</td>
<td>param</td>
<td>parameter to be passed to the callback function</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.60.4.19 osalMutexLock()

```c
void osalMutexLock ( mutex_t * mp )
```

Locks the specified mutex.

**Postcondition**

The mutex is locked and inserted in the per-thread stack of owned mutexes.

**Parameters**

| in, out | mp | pointer to the mutex_t object |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.4.20 osalMutexUnlock()

```c
void osalMutexUnlock ( mutex_t * mp )
```

Unlocks the specified mutex.

**Note**

The HAL guarantees to release mutex in reverse lock order. The mutex being unlocked is guaranteed to be the last locked mutex by the invoking thread. The implementation can rely on this behavior and eventually ignore the mp parameter which is supplied in order to support those OSes not supporting a stack of the owned mutexes.

**Parameters**

| in, out | mp | pointer to the mutex_t object |

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.60.4.21 osalSysDisable()

```c
static void osalSysDisable ( void ) [inline], [static]
```

Disables interrupts globally.
Function Class:
Special function, this function has special requirements see the notes.

7.60.4.22 osalSysEnable()

static void osalSysEnable (  
void ) [inline], [static]

Enables interrupts globally.

Function Class:
Special function, this function has special requirements see the notes.

7.60.4.23 osalSysLock()

static void osalSysLock (  
void ) [inline], [static]

Enters a critical zone from thread context.

Note
This function cannot be used for reentrant critical zones.

Function Class:
Special function, this function has special requirements see the notes.

7.60.4.24 osalSysUnlock()

static void osalSysUnlock (  
void ) [inline], [static]

Leaves a critical zone from thread context.

Note
This function cannot be used for reentrant critical zones.

Function Class:
Special function, this function has special requirements see the notes.
7.60.4.25  osalSysLockFromISR()

static void osalSysLockFromISR (  
    void ) [inline], [static]

Enters a critical zone from ISR context.

Note
This function cannot be used for reentrant critical zones.

Function Class:
Special function, this function has special requirements see the notes.

7.60.4.26  osalSysUnlockFromISR()

static void osalSysUnlockFromISR (  
    void ) [inline], [static]

Leaves a critical zone from ISR context.

Note
This function cannot be used for reentrant critical zones.

Function Class:
Special function, this function has special requirements see the notes.

7.60.4.27  osalSysGetStatusAndLockX()

static syssts_t osalSysGetStatusAndLockX (  
    void ) [inline], [static]

Returns the execution status and enters a critical zone.

This functions enters into a critical zone and can be called from any context. Because its flexibility it is less efficient than chSysLock() which is preferable when the calling context is known.

Postcondition
The system is in a critical zone.

Returns
The previous system status, the encoding of this status word is architecture-dependent and opaque.

Function Class:
This is an X-Class API, this function can be invoked from any context.
7.60.4.28 osalSysRestoreStatusX()

```c
static void osalSysRestoreStatusX ( syssts_t sts ) [inline], [static]
```

Restores the specified execution status and leaves a critical zone.

**Note**

A call to `chSchRescheduleS()` is automatically performed if exiting the critical zone and if not in ISR context.

**Parameters**

| in | sts | the system status to be restored. |

**Function Class:**

This is an **X-Class API**, this function can be invoked from any context.

7.60.4.29 osalTimeAddX()

```c
static systime_t osalTimeAddX ( systime_t systime,
                                sysinterval_t interval ) [inline], [static]
```

Adds an interval to a system time returning a system time.

**Parameters**

| in | systime | base system time |
| in | interval | interval to be added |

**Returns**

The new system time.

**Function Class:**

This is an **X-Class API**, this function can be invoked from any context.

7.60.4.30 osalTimeDiffX()

```c
static sysinterval_t osalTimeDiffX ( systime_t start,
                                     systime_t end ) [inline], [static]
```

Subtracts two system times returning an interval.
Parameters

| in | start | first system time |
| in | end   | second system time |

Returns

The interval representing the time difference.

Function Class:

This is an X-Class API, this function can be invoked from any context.

7.60.4.31 osalTimeIsInRangeX()

static bool osalTimeIsInRangeX (systime_t time, systime_t start, systime_t end) [inline], [static]

Checks if the specified time is within the specified time window.

Note

When start==end then the function returns always false because the time window has zero size.

This function can be called from any context.

Parameters

| in | time | the time to be verified |
| in | start | the start of the time window (inclusive) |
| in | end   | the end of the time window (non inclusive) |

Return values

| true | current time within the specified time window. |
| false| current time not within the specified time window. |

Function Class:

This is an X-Class API, this function can be invoked from any context.
7.60.4.32  osalThreadQueueObjectInit()

static void osalThreadQueueObjectInit (  
    threads_queue_t * tqp ) [inline], [static]

Initializes a threads queue object.

Parameters

| out | tqp  | pointer to the threads queue object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.60.4.33  osalEventObjectInit()

static void osalEventObjectInit (  
    event_source_t * esp ) [inline], [static]

Initializes an event source object.

Parameters

| out | esp  | pointer to the event source object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.60.4.34  osalMutexObjectInit()

static void osalMutexObjectInit (  
    mutex_t * mp ) [inline], [static]

Initializes a mutex_t object.

Parameters

| out | mp   | pointer to the mutex_t object |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.
7.60.5 Variable Documentation

7.60.5.1 osal_halt_msg [1/2]

const char* osal_halt_msg

Pointer to a halt error message.

Note

The message is meant to be retrieved by the debugger after the system halt caused by an unexpected error.

7.60.5.2 osal_halt_msg [2/2]

const char* osal_halt_msg

Pointer to a halt error message.

Note

The message is meant to be retrieved by the debugger after the system halt caused by an unexpected error.
8.1 BaseAsynchronousChannel Struct Reference

Base asynchronous channel class.

#include <hal_channels.h>

Inheritance diagram for BaseAsynchronousChannel:
Data Fields

- const struct BaseAsynchronousChannelVMT * vmt
  Virtual Methods Table.

8.1.1 Detailed Description

Base asynchronous channel class.

This class extends BaseChannel by adding event sources fields for asynchronous I/O for use in an event-driven environment.

8.1.2 Field Documentation

8.1.2.1 vmt

const struct BaseAsynchronousChannelVMT * BaseAsynchronousChannel::vmt

Virtual Methods Table.
8.2 BaseAsynchronousChannelVMT Struct Reference

BaseAsynchronousChannel virtual methods table.

#include <hal_channels.h>

Inheritance diagram for BaseAsynchronousChannelVMT:
Collaboration diagram for BaseAsynchronousChannelVMT:

8.2.1 Detailed Description

BaseAsynchronousChannel virtual methods table.

8.3 BaseBlockDevice Struct Reference

Base block device class.

#include <hal_ioblock.h>
Inheritance diagram for BaseBlockDevice:
Collaboration diagram for BaseBlockDevice:

Data Fields

- const struct BaseBlockDeviceVMT * vmt
  Virtual Methods Table.

8.3.1 Detailed Description

Base block device class.

This class represents a generic, block-accessible, device.

8.3.2 Field Documentation

8.3.2.1 vmt

const struct BaseBlockDeviceVMT* BaseBlockDevice::vmt

Virtual Methods Table.
8.4 BaseBlockDeviceVMT Struct Reference

BaseBlockDevice virtual methods table.

#include <hal_ioblock.h>

Inheritance diagram for BaseBlockDeviceVMT:

Collaboration diagram for BaseBlockDeviceVMT:

8.4.1 Detailed Description

BaseBlockDevice virtual methods table.
8.5 BaseBufferedStream Struct Reference

Buffered stream class.

#include <hal_streams.h>

Inheritance diagram for BaseBufferedStream:
Data Fields

- const struct BaseBufferedStreamVMT * vmt
  
  Virtual Methods Table.

8.5.1 Detailed Description

Buffered stream class.

This class extends BaseSequentialStream to represent a generic blocking buffered sequential data stream.

8.5.2 Field Documentation

8.5.2.1 vmt

const struct BaseBufferedStreamVMT * BaseBufferedStream::vmt

Virtual Methods Table.
8.6 BaseBufferedStreamVMT Struct Reference

BaseBufferedStream virtual methods table.

#include <hal_streams.h>

Inheritance diagram for BaseBufferedStreamVMT:

Collaboration diagram for BaseBufferedStreamVMT:

8.6.1 Detailed Description

BaseBufferedStream virtual methods table.
8.7 BaseChannel Struct Reference

Base channel class.

#include <hal_channels.h>

Inheritance diagram for BaseChannel:

BaseObject
   + vmt

BaseSequentialStream
   + vmt

BaseChannel
   + vmt

BaseAsynchronousChannel
   + vmt

hal_buffered_serial
   + vmt

hal_serial_driver
   + vmt

SerialUSBDriver
   + vmt
Collaboration diagram for BaseChannel:

```
BaseObject

BaseSequentialStream

BaseObject

BaseSequentialStreamVMT

+ vmt

BaseChannelVMT

+ vmt

BaseChannel
```

Data Fields

- `const struct BaseChannelVMT * vmt`

  Virtual Methods Table.

8.7.1 Detailed Description

Base channel class.

This class represents a generic, byte-wide, I/O channel. This class introduces generic I/O primitives with timeout specification.

8.7.2 Field Documentation

8.7.2.1 vmt

`const struct BaseChannelVMT* BaseChannel::vmt`

Virtual Methods Table.
8.8 BaseChannelVMT Struct Reference

`BaseChannel` virtual methods table.

```c
#include <hal_channels.h>
```

Inheritance diagram for BaseChannelVMT:
Collaboration diagram for BaseChannelVMT:

8.8.1 Detailed Description

BaseChannel virtual methods table.

8.9 BaseFlash Struct Reference

Base flash class.

#include <hal_flash.h>
Inheritance diagram for BaseFlash:
Collaboration diagram for BaseFlash:

![Collaboration Diagram](image)

**Data Fields**

- const struct BaseFlashVMT * vmt
  
  Virtual Methods Table.

### 8.9.1 Detailed Description

Base flash class.

### 8.9.2 Field Documentation

#### 8.9.2.1 vmt

const struct BaseFlashVMT * BaseFlash::vmt

Virtual Methods Table.
8.10 BaseFlashVMT Struct Reference

BaseFlash virtual methods table.

#include <hal_flash.h>

Inheritance diagram for BaseFlashVMT:

Collaboration diagram for BaseFlashVMT:

8.10.1 Detailed Description

BaseFlash virtual methods table.

8.11 BaseObject Struct Reference

Base object class.

#include <hal_objects.h>
Inheritance diagram for BaseObject:

![Inheritance Diagram](image)

Collaboration diagram for BaseObject:

![Collaboration Diagram](image)

**Data Fields**

- const struct BaseObjectVMT * vmt

  *Virtual Methods Table.*

**8.11.1 Detailed Description**

Base object class.

This class represents a generic object including a virtual methods table (VMT).
8.11.2 Field Documentation

8.11.2.1 vmt

`const struct BaseObjectVMT* BaseObject::vmt`

Virtual Methods Table.

8.12 BaseObjectVMT Struct Reference

BaseObject virtual methods table.

#include <hal_objects.h>

Collaboration diagram for BaseObjectVMT:

```
BaseObjectVMT
```

8.12.1 Detailed Description

BaseObject virtual methods table.

8.13 BasePersistentStorage Struct Reference

Base persistent storage class.

#include <hal_persistent.h>
Inheritance diagram for BasePersistentStorage:

![Inheritance Diagram]

Collaboration diagram for BasePersistentStorage:

![Collaboration Diagram]

Data Fields

- const struct BasePersistentStorageVMT * vmt

*Virtual Methods Table.*
8.13.1 Detailed Description

Base persistent storage class.

8.13.2 Field Documentation

8.13.2.1 vmt

const struct BasePersistentStorageVMT* BasePersistentStorage::vmt

Virtual Methods Table.

8.14 BasePersistentStorageVMT Struct Reference

BasePersistentStorage virtual methods table.

#include <hal_persistent.h>

Collaboration diagram for BasePersistentStorageVMT:

![Collaboration Diagram]

8.14.1 Detailed Description

BasePersistentStorage virtual methods table.
8.15 BaseSequentialStream Struct Reference

Base stream class.

#include <hal_streams.h>

Inheritance diagram for BaseSequentialStream:
Collaboration diagram for BaseSequentialStream:

Data Fields

- const struct BaseSequentialStreamVMT * vmt
  
  Virtual Methods Table.

8.15.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

8.15.2 Field Documentation

8.15.2.1 vmt

const struct BaseSequentialStreamVMT* BaseSequentialStream::vmt

Virtual Methods Table.
8.16 BaseSequentialStreamVMT Struct Reference

BaseSequentialStream virtual methods table.

#include <hal_streams.h>

Inheritance diagram for BaseSequentialStreamVMT:

![](image)

Collaboration diagram for BaseSequentialStreamVMT:

![](image)

8.16.1 Detailed Description

BaseSequentialStream virtual methods table.
8.17 BlockDeviceInfo Struct Reference

Block device info.

```
#include <hal_ioblock.h>
```

Collaboration diagram for BlockDeviceInfo:

```
BlockDeviceInfo
+ blk_size
+ blk_num
```

Data Fields

- `uint32_t blk_size`
  
  Block size in bytes.

- `uint32_t blk_num`
  
  Total number of blocks.

8.17.1 Detailed Description

Block device info.

8.17.2 Field Documentation

8.17.2.1 blk_size

```
uint32_t BlockDeviceInfo::blk_size
```

Block size in bytes.
8.17.2.2 blk_num

uint32_t BlockDeviceInfo::blk_num

Total number of blocks.

8.18 BufferedSerialVMT Struct Reference

BufferedSerial virtual methods table.

#include <hal_buffered_serial.h>

Inheritance diagram for BufferedSerialVMT:

[Diagram of inheritance hierarchy]
Collaboration diagram for BufferedSerialVMT:

```
#include <bufstreams.h>
```

8.18.1 Detailed Description

BufferedSerial virtual methods table.

8.19 BufferedStreamAdapter Struct Reference

Buffered stream adapter object.

#include <bufstreams.h>

Inherits BufferedStream.

ChibiOS/HAL
Collaboration diagram for BufferedStreamAdapter:

Data Fields

- const struct BufferedStreamAdapterVMT * vmt
  Virtual Methods Table.

8.19.1 Detailed Description

Buffered stream adapter object.

8.19.2 Field Documentation

8.19.2.1 vmt

const struct BufferedStreamAdapterVMT * BufferedStreamAdapter::vmt

Virtual Methods Table.
8.20 BufferedStreamAdapterVMT Struct Reference

BufferedStreamAdapter virtual methods table.

#include <bufstreams.h>

Collaboration diagram for BufferedStreamAdapterVMT:

![Collaboration diagram for BufferedStreamAdapterVMT](image)

8.20.1 Detailed Description

BufferedStreamAdapter virtual methods table.

8.21 CANRxFrame Struct Reference

CAN received frame.

#include <hal_can_lld.h>

Collaboration diagram for CANRxFrame:

![Collaboration diagram for CANRxFrame](image)
Data Fields

- `uint8_t FMI`
  
  Filter id.

- `uint16_t TIME`
  
  Time stamp.

- `uint8_t DLC:4`
  
  Data length.

- `uint8_t RTR:1`
  
  Frame type.

- `uint8_t IDE:1`
  
  Identifier type.

- `uint32_t SID:11`
  
  Standard identifier.

- `uint32_t EID:29`
  
  Extended identifier.

- `uint8_t data8 [8]`
  
  Frame data.

- `uint16_t data16 [4]`
  
  Frame data.

- `uint32_t data32 [2]`
  
  Frame data.

8.21.1 Detailed Description

CAN received frame.

Note

Accessing the frame data as word16 or word32 is not portable because machine data endianness, it can be still useful for a quick filling.

8.21.2 Field Documentation

8.21.2.1 FMI

`uint8_t CANRxFrame::FMI`

Filter id.
8.21.2.2 TIME

uint16_t CANRxFrame::TIME

Time stamp.

8.21.2.3 DLC

uint8_t CANRxFrame::DLC

Data length.

8.21.2.4 RTR

uint8_t CANRxFrame::RTR

Frame type.

8.21.2.5 IDE

uint8_t CANRxFrame::IDE

Identifier type.

8.21.2.6 SID

uint32_t CANRxFrame::SID

Standard identifier.

8.21.2.7 EID

uint32_t CANRxFrame::EID

Extended identifier.
8.21.2.8 data8

uint8_t CANRxFrame::data8[8]

Frame data.

8.21.2.9 data16

uint16_t CANRxFrame::data16[4]

Frame data.

8.21.2.10 data32

uint32_t CANRxFrame::data32[2]

Frame data.

8.22 CANTxFrame Struct Reference

CAN transmission frame.

#include <hal_can_lld.h>

Collaboration diagram for CANTxFrame:
8.22 CAN Tx Frame Struct Reference

Data Fields

- `uint8_t DLC:4`
  Data length.

- `uint8_t RTR:1`
  Frame type.

- `uint8_t IDE:1`
  Identifier type.

- `uint32_t SID:11`
  Standard identifier.

- `uint32_t EID:29`
  Extended identifier.

- `uint8_t data8 [8]`
  Frame data.

- `uint16_t data16 [4]`
  Frame data.

- `uint32_t data32 [2]`
  Frame data.

8.22.1 Detailed Description

CAN transmission frame.

Note

Accessing the frame data as word16 or word32 is not portable because machine data endianness, it can be still useful for a quick filling.

8.22.2 Field Documentation

8.22.2.1 DLC

`uint8_t CANTxFrame::DLC`

Data length.
8.22.2.2  RTR

uint8_t CANTxFrame::RTR
Frame type.

8.22.2.3  IDE

uint8_t CANTxFrame::IDE
Identifier type.

8.22.2.4  SID

uint32_t CANTxFrame::SID
Standard identifier.

8.22.2.5  EID

uint32_t CANTxFrame::EID
Extended identifier.

8.22.2.6  data8

uint8_t CANTxFrame::data8[8]
Frame data.

8.22.2.7  data16

uint16_t CANTxFrame::data16[4]
Frame data.
8.23 cdc_linecoding_t Struct Reference

Type of Line Coding structure.

#include <hal_usb_cdc.h>

Collaboration diagram for cdc_linecoding_t:

```
cdc_linecoding_t
+ dwDTERate
+ bCharFormat
+ bParityType
+ bDataBits
```

8.23.1 Detailed Description

Type of Line Coding structure.

8.24 CRYConfig Struct Reference

Driver configuration structure.

#include <hal_crypto_lld.h>

Collaboration diagram for CRYConfig:

```
CRYConfig
+ dummy
```
8.24.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.25 CRYDriver Struct Reference

Structure representing an CRY driver.

#include <hal_crypto_lld.h>

Collaboration diagram for CRYDriver:

Data Fields

- crystate_t state
  
  Driver state.

- const CRYConfig * config
  
  Current configuration data.

8.25.1 Detailed Description

Structure representing an CRY driver.

8.25.2 Field Documentation
8.25.2.1 state

crystate_t CRYDriver::state

Driver state.

8.25.2.2 config

const CRYConfig* CRYDriver::config

Current configuration data.

8.26 EFlashDriverVMT Struct Reference

EFlash virtual methods table.

#include <hal_efl.h>

Inheritance diagram for EFlashDriverVMT:
Collaboration diagram for EFlashDriverVMT:

```
+-------------------+
| BaseFlashVMT      |
|                   |
+-------------------+
|                   |
| EFlashDriverVMT   |
```

8.26.1 Detailed Description

EFlash virtual methods table.

8.27 event_source Struct Reference

Events source object.

```c
#include <osal.h>
```

Collaboration diagram for event_source:
Data Fields

- volatile eventflags_t flags
  Stored event flags.
- eventcallback_t cb
  Event source callback.
- void * param
  User defined field.

8.27.1 Detailed Description

Events source object.

Note

The content of this structure is not part of the API and should not be relied upon. Implementers may define this structure in an entirely different way.

Retrieval and clearing of the flags are not defined in this API and are implementation-dependent.

8.27.2 Field Documentation

8.27.2.1 flags

volatile eventflags_t event_source::flags

Stored event flags.

8.27.2.2 cb

eventcallback_t event_source::cb

Event source callback.

8.27.2.3 param

void* event_source::param

User defined field.
8.28 FileStream Struct Reference

Base file stream class.

#include <hal_files.h>

Inheritance diagram for FileStream:
Collaboration diagram for FileStream:

Data Fields

- const struct FileStreamVMT * vmt
  
  Virtual Methods Table.

8.28.1 Detailed Description

Base file stream class.

This class represents a generic file data stream.

8.28.2 Field Documentation

8.28.2.1 vmt

const struct FileStreamVMT* FileStream::vmt

Virtual Methods Table.
8.29 FileStreamVMT Struct Reference

FileStream virtual methods table.

#include <hal_files.h>

Inheritance diagram for FileStreamVMT:

Collaboration diagram for FileStreamVMT:

8.29.1 Detailed Description

FileStream virtual methods table.
8.30 flash_descriptor_t Struct Reference

Type of a flash device descriptor.

#include <hal_flash.h>

Collaboration diagram for flash_descriptor_t:

---

Data Fields

- uint32_t attributes
  
  Device attributes.

- uint32_t page_size
  
  Size of write page.

- flash_sector_t sectors_count
  
  Number of sectors in the device.

- const flash_sector_descriptor_t *sectors
  
  List of sectors for devices with non-uniform sector sizes.

- uint32_t sectors_size
  
  Size of sectors for devices with uniform sector size.

- uint8_t * address
  
  Flash address if memory mapped or zero.

- uint32_t size
  
  Flash size.

---

8.30.1 Detailed Description

Type of a flash device descriptor.
8.30.2 Field Documentation

8.30.2.1 attributes

uint32_t flash_descriptor_t::attributes

Device_attributes.

8.30.2.2 page_size

uint32_t flash_descriptor_t::page_size

Size of write page.

8.30.2.3 sectors_count

flash_sector_t flash_descriptor_t::sectors_count

Number of sectors in the device.

8.30.2.4 sectors

const flash_sector_descriptor_t* flash_descriptor_t::sectors

List of sectors for devices with non-uniform sector sizes.

Note

If NULL then the device has uniform sectors size equal to sector_size.

8.30.2.5 sectors_size

uint32_t flash_descriptor_t::sectors_size

Size of sectors for devices with uniform sector size.

Note

If zero then the device has non uniform sectors described by the sectors array.
8.30.2.6 address

`uint8_t* flash_descriptor_t::address`

Flash address if memory mapped or zero.

Note

Conventionally, non memory mapped devices have address NULL.

8.30.2.7 size

`uint32_t flash_descriptor_t::size`

Flash size.

8.31 flash_sector_descriptor_t Struct Reference

Flash sector descriptor.

#include <hal_flash.h>

Collaboration diagram for flash_sector_descriptor_t:

```
flash_sector_descriptor_t
+ offset
+ size
```

Data Fields

- `flash_offset_t offset`
  Sector offset.
- `uint32_t size`
  Sector size.

8.31.1 Detailed Description

Flash sector descriptor.
8.31.2 Field Documentation

8.31.2.1 offset

flash_offset_t flash_sector_descriptor_t::offset
Sector offset.

8.31.2.2 size

uint32_t flash_sector_descriptor_t::size
Sector size.

8.32 GPTConfig Struct Reference

Driver configuration structure.
#include <hal_gpt_lld.h>
Collaboration diagram for GPTConfig:

Data Fields

- gptfreq_t frequency
  Timer clock in Hz.
- gptcallback_t callback
  Timer callback pointer.
8.32 GPTConfig Struct Reference

8.32.1 Detailed Description

Driver configuration structure.

Note
It could be empty on some architectures.

8.32.2 Field Documentation

8.32.2.1 frequency

`gptfreq_t GPTConfig::frequency`

Timer clock in Hz.

Note
The low level can use assertions in order to catch invalid frequency specifications.

8.32.2.2 callback

`gptcallback_t GPTConfig::callback`

Timer callback pointer.

Note
This callback is invoked on GPT counter events.
8.33 GPTDriver Struct Reference

Structure representing a GPT driver.

```c
#include <hal_gpt_lld.h>
```

Collaboration diagram for GPTDriver:

![Collaboration Diagram](image)

### Data Fields

- `gptstate_t state`
  
  Driver state.

- `const GPTConfig * config`
  
  Current configuration data.

### Detailed Description

Structure representing a GPT driver.

### Field Documentation

#### 8.33.2.1 state

```c

const GPTConfig *config
```

Driver state.

- [ChibiOS/HAL](https://www.chibios-rt.com)
8.34.1 Detailed Description

Driver configuration structure.

8.35 hal_adc_configuration_group Struct Reference

Conversion group configuration structure.

#include <hal_adc.h>
Collaboration diagram for hal_adc_configuration_group:

Data Fields

- bool circular
  Enables the circular buffer mode for the group.
- adc_channels_num_t num_channels
  Number of the analog channels belonging to the conversion group.
- adccallback_t end_cb
  Callback function associated to the group or NULL.
- adcerrrorcallback_t error_cb
  Error callback or NULL.

8.35.1 Detailed Description

Conversion group configuration structure.

This implementation-dependent structure describes a conversion operation.

Note

The use of this configuration structure requires knowledge of STM32 ADC cell registers interface, please refer to the STM32 reference manual for details.
8.35.2 Field Documentation

8.35.2.1 circular

bool hal_adc_configuration_group::circular

Enables the circular buffer mode for the group.

8.35.2.2 num_channels

adc_channels_num_t hal_adc_configuration_group::num_channels

Number of the analog channels belonging to the conversion group.

8.35.2.3 end_cb

adccallback_t hal_adc_configuration_group::end_cb

Callback function associated to the group or NULL.

8.35.2.4 error_cb

adcerrorcallback_t hal_adc_configuration_group::error_cb

Error callback or NULL.
8.36 hal_adc_driver Struct Reference

Structure representing an ADC driver.

#include <hal_adc.h>

Collaboration diagram for hal_adc_driver:

Data Fields

- **adcstate_t state**
  
  Driver state.
- **const ADCConfig * config**
  
  Current configuration data.
- **adcsample_t * samples**
  
  Current samples buffer pointer or NULL.
- **size_t depth**
  
  Current samples buffer depth or 0.
• const ADCConversionGroup * grpp
  Current conversion group pointer or NULL.
• thread_reference_t thread
  Waiting thread.
• mutex_t mutex
  Mutex protecting the peripheral.

8.36.1 Detailed Description

Structure representing an ADC driver.

8.36.2 Field Documentation

8.36.2.1 state

adcstate_t hal_adc_driver::state

Driver state.

8.36.2.2 config

const ADCConfig* hal_adc_driver::config

Current configuration data.

8.36.2.3 samples

adcsample_t* hal_adc_driver::samples

Current samples buffer pointer or NULL.

8.36.2.4 depth

size_t hal_adc_driver::depth

Current samples buffer depth or 0.
8.36.2.5 grpp

const ADCConversionGroup* hal_adc_driver::grpp

Current conversion group pointer or NULL.

8.36.2.6 thread

thread_reference_t hal_adc_driver::thread

Waiting thread.

8.36.2.7 mutex

mutex_t hal_adc_driver::mutex

Mutex protecting the peripheral.

8.37 hal_buffered_serial Struct Reference

Buffered serial channel class.

#include <hal_buffered_serial.h>

Inheritance diagram for hal_buffered_serial:
8.37.1 Detailed Description

Buffered serial channel class.

This class extends `BaseAsynchronousChannel` by adding physical I/O queues.

8.37.2 Field Documentation

8.37.2.1 vmt

```c
const struct BufferedSerialVMT* hal_buffered_serial::vmt
```

Virtual Methods Table.
8.38 hal_can_config Struct Reference

Type of a CAN configuration structure.

#include <hal_can_lld.h>

Collaboration diagram for hal_can_config:

8.38.1 Detailed Description

Type of a CAN configuration structure.

8.39 hal_can_driver Struct Reference

Structure representing an CAN driver.

#include <hal_can_lld.h>

Collaboration diagram for hal_can_driver:
Data Fields

- **canstate_t state**
  Driver state.
- **const CANConfig * config**
  Current configuration data.
- **threads_queue_t txqueue**
  Transmission threads queue.
- **threads_queue_t rxqueue**
  Receive threads queue.
- **event_source_t rxtfull_event**
  One or more frames become available.
- **event_source_t txempty_event**
  One or more transmission mailbox become available.
- **event_source_t error_event**
  A CAN bus error happened.
- **event_source_t sleep_event**
  Entering sleep state event.
- **event_source_t wakeup_event**
  Exiting sleep state event.

### 8.39.1 Detailed Description

Structure representing a CAN driver.

### 8.39.2 Field Documentation

#### 8.39.2.1 state

canstate_t hal_can_driver::state

Driver state.

#### 8.39.2.2 config

const CANConfig* hal_can_driver::config

Current configuration data.
8.39.2.3 txqueue

threads_queue_t hal_can_driver::txqueue

Transmission threads queue.

8.39.2.4 rxqueue

threads_queue_t hal_can_driver::rxqueue

Receive threads queue.

8.39.2.5 rxfull_event

event_source_t hal_can_driver::rxfull_event

One or more frames become available.

Note
After broadcasting this event it will not be broadcasted again until the received frames queue has been completely emptied. It is **not** broadcasted for each received frame. It is responsibility of the application to empty the queue by repeatedly invoking `chReceive()` when listening to this event. This behavior minimizes the interrupt served by the system because CAN traffic.

The flags associated to the listeners will indicate which receive mailboxes become non-empty.

8.39.2.6 txempty_event

event_source_t hal_can_driver::txempty_event

One or more transmission mailbox become available.

Note
The flags associated to the listeners will indicate which transmit mailboxes become empty.
8.39.2.7 error_event

```
event_source_t hal_can_driver::error_event
```

A CAN bus error happened.

Note

The flags associated to the listeners will indicate the error(s) that have occurred.

8.39.2.8 sleep_event

```
event_source_t hal_can_driver::sleep_event
```

Entering sleep state event.

8.39.2.9 wakeup_event

```
event_source_t hal_can_driver::wakeup_event
```

Exiting sleep state event.

8.40 hal_dac_config Struct Reference

Driver configuration structure.

```
#include <hal_dac.h>
```

Collaboration diagram for hal_dac_config:

```
hal_dac_config
+ dac_lld_config_fields
```

8.40.1 Detailed Description

Driver configuration structure.
8.41 hal_dac_conversion_group Struct Reference

DAC Conversion group structure.

```c
#include <hal_dac.h>
```

Collaboration diagram for hal_dac_conversion_group:

![Collaboration diagram for hal_dac_conversion_group](image)

### Data Fields

- `uint32_t num_channels`
  
  *Number of DAC channels.*

- `daccallback_t end_cb`
  
  *Operation complete callback or NULL.*

- `dacerrorcallback_t error_cb`
  
  *Error handling callback or NULL.*

### 8.41.1 Detailed Description

DAC Conversion group structure.

### 8.41.2 Field Documentation
8.41.2.1 num_channels

```c
uint32_t hal_dac_conversion_group::num_channels
```

Number of DAC channels.

8.41.2.2 end_cb

```c
daccallback_t hal_dac_conversion_group::end_cb
```

Operation complete callback or NULL.

8.41.2.3 error_cb

```c
dacerrorcallback_t hal_dac_conversion_group::error_cb
```

Error handling callback or NULL.

8.42 hal_dac_driver Struct Reference

Structure representing a DAC driver.

```c
#include <hal_dac.h>
```

ChibiOS/HAL
Collaboration diagram for hal_dac_driver:

Data Fields

- **dacstate_t state**
  
  Driver state.

- **const DACConversionGroup * grpp**
  
  Conversion group.

- **dacsample_t * samples**
  
  Samples buffer pointer.

- **size_t depth**
  
  Samples buffer size.

- **const DACConfig * config**
  
  Current configuration data.

- **thread_reference_t thread**
  
  Waiting thread.

- **mutex_t mutex**
  
  Mutex protecting the bus.
8.42.1 Detailed Description

Structure representing a DAC driver.

8.42.2 Field Documentation

8.42.2.1 state

dacstate_t hal_dac_driver::state

Driver state.

8.42.2.2 grpp

const DACConversionGroup* hal_dac_driver::grpp

Conversion group.

8.42.2.3 samples

dacsample_t* hal_dac_driver::samples

Samples buffer pointer.

8.42.2.4 depth

size_t hal_dac_driver::depth

Samples buffer size.

8.42.2.5 config

const DACConfig* hal_dac_driver::config

Current configuration data.
8.42.6 thread

\texttt{thread\_reference\_t \ hal\_dac\_driver::thread}

Waiting thread.

8.42.7 mutex

\texttt{mutex\_t \ hal\_dac\_driver::mutex}

Mutex protecting the bus.

8.43 \texttt{hal\_efl\_config} Struct Reference

Type of a structure representing a flash driver configuration.

#include \texttt{<hal\_efl\_h>}

Collaboration diagram for \texttt{hal\_efl\_config}:

![Collaboration diagram]

8.43.1 Detailed Description

Type of a structure representing a flash driver configuration.
Structure representing an embedded flash driver.

#include <hal_efl.h>

Inheritance diagram for hal_efl_driver:
Collaboration diagram for hal_efl_driver:

Data Fields

- const struct EFlashDriverVMT *vmt
  EFlashDriver Virtual Methods Table.
- _efl_driver_data mutex_t mutex
  Mutex protecting EFL.

8.44.1 Detailed Description

Structure representing an embedded flash driver.

8.44.2 Field Documentation

8.44.2.1 vmt

    const struct EFlashDriverVMT* hal_efl_driver::vmt

EFlashDriver Virtual Methods Table.
8.45 hal_i2c_config Struct Reference

I2C driver configuration structure.

#include <hal_i2c_lld.h>

Collaboration diagram for hal_i2c_config:

```
hal_i2c_config
    + dummy
```

8.45.1 Detailed Description

I2C driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.46 hal_i2c_driver Struct Reference

Structure representing an I2C driver.

#include <hal_i2c_lld.h>
Collaboration diagram for hal_i2c_driver:

```
hal_i2c_config
+ dummy

+config

hal_i2c_driver
+ state
+ errors
+ mutex

hal_i2c_config
+ dummy
+ config
```

Data Fields

- `i2cstate_t state`
  
  Driver state.

- `const I2CConfig * config`
  
  Current configuration data.

- `i2cflags_t errors`
  
  Error flags.

8.46.1 Detailed Description

Structure representing an I2C driver.

8.46.2 Field Documentation

8.46.2.1 state

```c
i2cstate_t hal_i2c_driver::state
```

Driver state.
8.46.2.2 config

const I2CConfig * hal_i2c_driver::config

Current configuration data.

8.46.2.3 errors

i2cflags_t hal_i2c_driver::errors

Error flags.

8.47 hal_i2s_config Struct Reference

Driver configuration structure.

#include <hal_i2s.h>

Collaboration diagram for hal_i2s_config:

Data Fields

- const void * tx_buffer
  Transmission buffer pointer.
- void * rx_buffer
  Receive buffer pointer.
- size_t size
  TX and RX buffers size as number of samples.
- i2scallback_t end_cb
  Callback function called during streaming.
8.47.1 Detailed Description

Driver configuration structure.

8.47.2 Field Documentation

8.47.2.1 tx_buffer

const void* hal_i2s_config::tx_buffer

Transmission buffer pointer.

Note

Can be NULL if TX is not required.

8.47.2.2 rx_buffer

void* hal_i2s_config::rx_buffer

Receive buffer pointer.

Note

Can be NULL if RX is not required.

8.47.2.3 size

size_t hal_i2s_config::size

TX and RX buffers size as number of samples.

8.47.2.4 end_cb

i2scallback_t hal_i2s_config::end_cb

Callback function called during streaming.
8.48 hal_i2s_driver Struct Reference

Structure representing an I2S driver.

#include <hal_i2s.h>

Collaboration diagram for hal_i2s_driver:

Data Fields

- **i2sstate_t state**
  Driver state.
- **const I2SConfig * config**
  Current configuration data.

8.48.1 Detailed Description

Structure representing an I2S driver.

8.48.2 Field Documentation

8.48.2.1 state

i2sstate_t hal_i2s_driver::state

Driver state.
8.48.2.2  config

```c
const I2SConfig* hal_i2s_driver::config
```

Current configuration data.

## 8.49 hal_mac_config Struct Reference

Driver configuration structure.

```c
#include <hal_mac.h>
```

Collaboration diagram for hal_mac_config:

![Collaboration diagram](image)

### 8.49.1 Detailed Description

Driver configuration structure.

**Note**

Implementations may extend this structure to contain more, architecture dependent, fields.

## 8.50 hal_mac_driver Struct Reference

Structure representing a MAC driver.

```c
#include <hal_mac.h>
```
Collaboration diagram for hal_mac_driver:

Data Fields

- **macstate_t state**
  
  Driver state.
- **const MACConfig ∗ config**
  
  Current configuration data.
- **threads_queue_t tdqueue**
  
  Transmit threads queue.
- **threads_queue_t rdqueue**
  
  Receive threads queue.
- **event_source_t es**
  
  MAC events source.
- **eventflags_t flags**
  
  Locally held event flags for callback use.
- **maccb_t cb**
  
  Events callback.
- **void ∗ arg**
  
  User argument.

8.50.1 Detailed Description

Structure representing a MAC driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.
8.50.2 Field Documentation

8.50.2.1 state

`macstate_t hal_mac_driver::state`

Driver state.

8.50.2.2 config

`const MACConfig* hal_mac_driver::config`

Current configuration data.

8.50.2.3 tdqueue

`threads_queue_t hal_mac_driver::tdqueue`

Transmit threads queue.

8.50.2.4 rdqueue

`threads_queue_t hal_mac_driver::rdqueue`

Receive threads queue.

8.50.2.5 es

`event_source_t hal_mac_driver::es`

MAC events source.
8.50.2.6 flags

`eventflags_t hal_mac_driver::flags`

Locally held event flags for callback use.

8.50.2.7 cb

`maccb_t hal_mac_driver::cb`

Events callback.

Note

Can be NULL.

8.50.2.8 arg

`void* hal_mac_driver::arg`

User argument.

Note

Can be retrieved through the `ethp` argument of the callback.

8.51 hal_mac_receive_descriptor Struct Reference

Structure representing a MAC receive descriptor.

```
#include <hal_mac.h>
```

Collaboration diagram for hal_mac_receive_descriptor:
Data Fields

- `size_t offset`
  Current read offset.
- `size_t size`
  Available data size.

8.51.1 Detailed Description

Structure representing a MAC receive descriptor.

8.51.2 Field Documentation

8.51.2.1 offset

```c
size_t hal_mac_receive_descriptor::offset
```

Current read offset.

8.51.2.2 size

```c
size_t hal_mac_receive_descriptor::size
```

Available data size.

8.52 hal_mac_transmit_descriptor Struct Reference

Structure representing a MAC transmit descriptor.

```c
#include <hal_mac.h>
```

Collaboration diagram for hal_mac_transmit_descriptor:
Data Fields

- `size_t offset`
  
  Current write offset.

- `size_t size`
  
  Available space size.

### 8.52.1 Detailed Description

Structure representing a MAC transmit descriptor.

### 8.52.2 Field Documentation

#### 8.52.2.1 offset

size_t hal_mac_transmit_descriptor::offset

Current write offset.

#### 8.52.2.2 size

size_t hal_mac_transmit_descriptor::size

Available space size.

### 8.53 hal_serial_config Struct Reference

PLATFORM Serial Driver configuration structure.

#include <hal_serial_lld.h>

Collaboration diagram for hal_serial_config:
Data Fields

- `uint32_t speed`
  
  Bit rate.

8.53.1 Detailed Description

PLATFORM Serial Driver configuration structure.

An instance of this structure must be passed to `sdStart()` in order to configure and start a serial driver operations.

Note

This structure content is architecture dependent, each driver implementation defines its own version and the custom static initializers.

8.53.2 Field Documentation

8.53.2.1 speed

`uint32_t hal_serial_config::speed`

Bit rate.

8.54 hal_serial_driver Struct Reference

Full duplex serial driver class.

#include <hal_serial.h>
Inheritance diagram for hal_serial_driver:
Collaboration diagram for hal_serial_driver:

Data Fields

- `const struct SerialDriverVMT* vmt`
  
  Virtual Methods Table.

8.54.1 Detailed Description

Full duplex serial driver class.

This class extends `BaseAsynchronousChannel` by adding physical I/O queues.

8.54.2 Field Documentation

8.54.2.1 vmt

`const struct SerialDriverVMT* hal_serial_driver::vmt`

Virtual Methods Table.
8.55 hal_sio_config Struct Reference

Driver configuration structure.

#include <hal_sio.h>

Collaboration diagram for hal_sio_config:

Note
Implementations may extend this structure to contain more, architecture dependent, fields.

8.56 hal_sio_driver Struct Reference

Structure representing a SIO driver.

#include <hal_sio.h>
Data Fields

- const struct sio_driver_vmt *vmt
  Virtual Methods Table.
- siostate_t state
  Driver state.
- const SIOConfig *config
  Current configuration data.
- sioevents_t enabled
  Enabled event flags.
- siocb_t cb
  Events callback.
- void *arg
  User argument.
- thread_reference_t sync_rx
  Synchronization point for RX.
- thread_reference_t sync_rxidle
  Synchronization point for RX idle.
- thread_reference_t sync_tx
  Synchronization point for TX.
8.56.1 Detailed Description

Structure representing a SIO driver.

Note
Implementations may extend this structure to contain more, architecture dependent, fields.

8.56.2 Field Documentation

8.56.2.1 vmt

const struct sio_driver_vmt* hal_sio_driver::vmt

Virtual Methods Table.

8.56.2.2 state

siostate_t hal_sio_driver::state

Driver state.

8.56.2.3 config

const SIOConfig* hal_sio_driver::config

Current configuration data.

8.56.2.4 enabled

sioevents_t hal_sio_driver::enabled

Enabled event flags.
8.56.2.5  cb

siocb_t hal_sio_driver::cb

Events callback.

Note

Can be NULL.

8.56.2.6  arg

void* hal_sio_driver::arg

User argument.

Note

Can be retrieved through the siop argument of the callback.

8.56.2.7  sync_rx

thread_reference_t hal_sio_driver::sync_rx

Synchronization point for RX.

8.56.2.8  sync_rxidle

thread_reference_t hal_sio_driver::sync_rxidle

Synchronization point for RX idle.

8.56.2.9  sync_tx

thread_reference_t hal_sio_driver::sync_tx

Synchronization point for TX.
### 8.56.2.10 sync_txend

```c
thread_reference_t hal_sio_driver::sync_txend
```
Synchronization point for TX-end.

### 8.57 hal_spi_config Struct Reference

Driver configuration structure.

```c
#include <hal_spi_v2.h>
```

Collaboration diagram for hal_spi_config:

![Collaboration Diagram](image)

#### Data Fields

- **bool circular**
  
  *Enables the circular buffer mode.*
- **spicallback_t end_cb**
  
  *Operation complete callback or NULL.*
- **ioline_t ssline**
  
  *The chip select line.*
- **ioprotid_t ssport**
The chip select port.

- `ioprtmask_t ssmask`
  The chip select port mask.

- `uint_fast8_t sspad`
  The chip select pad number.

- `bool slave`
  Enables the slave mode.

- `spicb_t data_cb`
  Operation data callback or `NULL`.

- `spicb_t error_cb`
  Operation error callback or `NULL`.

8.57.1 Detailed Description

Driver configuration structure.

8.57.2 Field Documentation

8.57.2.1 circular

```cpp
bool hal_spi_config::circular
```

Enables the circular buffer mode.

8.57.2.2 end_cb

```cpp
spicallback_t hal_spi_config::end_cb
```

Operation complete callback or `NULL`.

8.57.2.3 ssline

```cpp
ioline_t hal_spi_config::ssline
```

The chip select line.

Note

Only used in master mode.
8.57.2.4 ssport

ioportid_t hal_spi_config::ssport

The chip select port.

8.57.2.5 ssmask

ioportmask_t hal_spi_config::ssmask

The chip select port mask.

8.57.2.6 sspad

uint_fast8_t hal_spi_config::sspad

The chip select pad number.

8.57.2.7 slave

bool hal_spi_config::slave

Enables the slave mode.

8.57.2.8 data_cb

spicb_t hal_spi_config::data_cb

Operation data callback or NULL.

8.57.2.9 error_cb

spicb_t hal_spi_config::error_cb

Operation error callback or NULL.
8.58  hal_spi_driver Struct Reference

Structure representing an SPI driver.

#include <hal_spi_v2.h>

Collaboration diagram for hal_spi_driver:

---

**Data Fields**

- **spistate_t state**
  
  Driver state.

- **const SPIConfig * config**
  
  Current configuration data.

- **thread_reference_t thread**
  
  Waiting thread.

- **mutex_t mutex**
  
  Mutex protecting the peripheral.

- **thread_reference_t sync_transfer**
  
  Synchronization point for transfer.

---

8.58.1  Detailed Description

Structure representing an SPI driver.
8.58.2 Field Documentation

8.58.2.1 state

spistate_t hal_spi_driver::state

Driver state.

8.58.2.2 config

const SPIConfig * hal_spi_driver::config

Current configuration data.

8.58.2.3 thread

thread_reference_t hal_spi_driver::thread

Waiting thread.

8.58.2.4 mutex

mutex_t hal_spi_driver::mutex

Mutex protecting the peripheral.

8.58.2.5 sync_transfer

thread_reference_t hal_spi_driver::sync_transfer

Synchronization point for transfer.
8.59  hal_trng_config Struct Reference

Driver configuration structure.

```c
#include <hal_trng.h>
```

Collaboration diagram for hal_trng_config:

```
hal_trng_config
+ trng_lld_config_fields
```

8.59.1  Detailed Description

Driver configuration structure.

8.60  hal_trng_driver Struct Reference

Structure representing a TRNG driver.

```c
#include <hal_trng.h>
```

Collaboration diagram for hal_trng_driver:

```
hal_trng_driver
+ state
+ trng_lld_driver_fields
```

hal_trng_config
+ trng_lld_config_fields

hal_trng_driver
+ config

hal_trng_config
+ trng_lld_config_fields
Data Fields

- `trngstate_t state`
  
  Driver state.

- `const TRNGConfig* config`

  Current configuration data.

### 8.60.1 Detailed Description

Structure representing a TRNG driver.

### 8.60.2 Field Documentation

#### 8.60.2.1 state

```c
trngstate_t hal_trng_driver::state
```

Driver state.

#### 8.60.2.2 config

```c
const TRNGConfig* hal_trng_driver::config
```

Current configuration data.

### 8.61 hal_uart_config Struct Reference

Driver configuration structure.

```c
#include <hal_uart_lld.h>
```
Collaboration diagram for hal_uart_config:

Data Fields

- uartcb_t txend1_cb
  
  End of transmission buffer callback.

- uartcb_t txend2_cb
  
  Physical end of transmission callback.

- uartcb_t rxend_cb
  
  Receive buffer filled callback.

- uartcb_t rxchar_cb
  
  Character received while out of the UART_RECEIVE state.

- uartcb_t rxerr_cb
  
  Receive error callback.

8.61.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.
8.61.2 Field Documentation

8.61.2.1 txend1_cb

```cpp
uartcb_t hal_uart_config::txend1_cb
```

End of transmission buffer callback.

8.61.2.2 txend2_cb

```cpp
uartcb_t hal_uart_config::txend2_cb
```

Physical end of transmission callback.

8.61.2.3 rxend_cb

```cpp
uartcb_t hal_uart_config::rxend_cb
```

Receive buffer filled callback.

8.61.2.4 rxchar_cb

```cpp
uartccb_t hal_uart_config::rxchar_cb
```

Character received while out if the UART_RECEIVE state.

8.61.2.5 rxerr_cb

```cpp
uartecb_t hal_uart_config::rxerr_cb
```

Receive error callback.
8.62  hal_uart_driver Struct Reference

Structure representing an UART driver.

```
#include <hal_uart_lld.h>
```

Collaboration diagram for hal_uart_driver:

```
hal_uart_driver
+ state
 + txstate
 + rxstate
 + early
 + threadrx
 + threadtx
 + mutex

hal_uart_config
+ rxend_cb
+ rxchar_cb
+ rxerr_cb
+ txend2_cb
+ txend1_cb
+ config
```

Data Fields

- `uartstate_t state`
  Driver state.
- `uarttxstate_t txstate`
  Transmitter state.
- `uartrxstate_t rxstate`
  Receiver state.
- `const UARTConfig * config`
  Current configuration data.
- `bool early`
  Synchronization flag for transmit operations.
- `thread_reference_t threadrx`
  Waiting thread on RX.
- `thread_reference_t threadtx`
  Waiting thread on TX.
- `mutex_t mutex`
  Mutex protecting the peripheral.
8.62.1 Detailed Description

Structure representing an UART driver.

Note
Implementations may extend this structure to contain more, architecture dependent, fields.

8.62.2 Field Documentation

8.62.2.1 state

`uartstate_t hal_uart_driver::state`

Driver state.

8.62.2.2 txstate

`uarttxstate_t hal_uart_driver::txstate`

Transmitter state.

8.62.2.3 rxstate

`uartrxstate_t hal_uart_driver::rxstate`

Receiver state.

8.62.2.4 config

`const UARTConfig* hal_uart_driver::config`

Current configuration data.
### 8.62.2.5 early

```cpp
bool hal_uart_driver::early
```
Synchronization flag for transmit operations.

### 8.62.2.6 threadrx

```cpp
thread_reference_t hal uart_driver::threadrx
```
Waiting thread on RX.

### 8.62.2.7 threadtx

```cpp
thread_reference_t hal_uart_driver::threadtx
```
Waiting thread on TX.

### 8.62.2.8 mutex

```cpp
mutex_t hal_uart_driver::mutex
```
Mutex protecting the peripheral.

### 8.63 hal_wspi_config Struct Reference

Driver configuration structure.

```cpp
#include <hal_wspi.h>
```
Collaboration diagram for hal_wspi_config:

![Collaboration diagram for hal_wspi_config](diagram.png)
8.64 hal_wspi_driver Struct Reference

Structure representing an WSPI driver.

#include <hal_wspi.h>

---

Data Fields

- **wspicallback_t end_cb**
  
  *Operation complete callback or NULL.*

- **wspicallback_t error_cb**
  
  *Operation error callback or NULL.*

8.63.1 Detailed Description

Driver configuration structure.

8.63.2 Field Documentation

8.63.2.1 end_cb

`wspicallback_t hal_wspi_config::end_cb`

*Operation complete callback or NULL.*

8.63.2.2 error_cb

`wspicallback_t hal_wspi_config::error_cb`

*Operation error callback or NULL.*
Collaboration diagram for hal_wspi_driver:

Data Fields

- `wspistate_t state`
  Driver state.
- `const WSPIConfig *Config`
  Current configuration data.
- `thread_reference_t thread`
  Waiting thread.
- `mutex_t mutex`
  Mutex protecting the peripheral.

8.64.1 Detailed Description

Structure representing an WSPI driver.

8.64.2 Field Documentation

8.64.2.1 state

`wspistate_t hal_wspi_driver::state`

Driver state.
8.64.2.2 config

const WSPICfg* hal_wspi_driver::config

Current configuration data.

8.64.2.3 thread

thread_reference_t hal_wspi_driver::thread

Waiting thread.

8.64.2.4 mutex

mutex_t hal_wspi_driver::mutex

Mutex protecting the peripheral.

8.65 halclkcfg_t Struct Reference

Type of a clock configuration structure.

#include <hal_lld.h>

Collaboration diagram for halclkcfg_t:

8.65.1 Detailed Description

Type of a clock configuration structure.
8.66 HMACSHA256Context Struct Reference

Type of a HMAC_SHA256 context.

#include <hal_crypto_lld.h>

Collaboration diagram for HMACSHA256Context:

```
+ dummy
   HMACSHA256Context
```

8.66.1 Detailed Description

Type of a HMAC_SHA256 context.

8.67 HMACSHA512Context Struct Reference

Type of a HMAC_SHA512 context.

#include <hal_crypto_lld.h>

Collaboration diagram for HMACSHA512Context:

```
+ dummy
   HMACSHA512Context
```

8.67.1 Detailed Description

Type of a HMAC_SHA512 context.
8.68 ICUConfig Struct Reference

Driver configuration structure.

#include <hal_icu_lld.h>

Collaboration diagram for ICUConfig:

![Collaboration diagram](image)

**Data Fields**

- `icumode_t mode`
  
  Driver mode.

- `icufreq_t frequency`
  
  Timer clock in Hz.

- `icucallback_t width_cb`
  
  Callback for pulse width measurement.

- `icucallback_t period_cb`
  
  Callback for cycle period measurement.

- `icucallback_t overflow_cb`
  
  Callback for timer overflow.

### 8.68.1 Detailed Description

Driver configuration structure.

**Note**

It could be empty on some architectures.
8.68.2 Field Documentation

8.68.2.1 mode

icumode_t ICUConfig::mode

Driver mode.

8.68.2.2 frequency

icufreq_t ICUConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.68.2.3 width_cb

icucallback_t ICUConfig::width_cb

Callback for pulse width measurement.

8.68.2.4 period_cb

icucallback_t ICUConfig::period_cb

Callback for cycle period measurement.

8.68.2.5 overflow_cb

icucallback_t ICUConfig::overflow_cb

Callback for timer overflow.
Structure representing an ICU driver.

```
#include <hal_icu_lld.h>
```

Collaboration diagram for ICUDriver:

```
ICUDriver
+ state

ICUConfig
+ mode
+ frequency
  + period_cb
  + width_cb
  + overflow_cb
  +config
```

### Data Fields

- `icustate_t state`
  
  Driver state.

- `const ICUConfig * config`
  
  Current configuration data.

### 8.69.1 Detailed Description

Structure representing an ICU driver.

### 8.69.2 Field Documentation

#### 8.69.2.1 state

```
icustate_t ICUDriver::state
```

Driver state.
8.69.2.2  config

```c
const ICUConfig* ICUDriver::config
```

Current configuration data.

8.70  io_buffers_queue Struct Reference

Structure of a generic buffers queue.

```c
#include <hal_buffers.h>
```

Collaboration diagram for `io_buffers_queue`:

![Collaboration diagram](image)

**Data Fields**

- `threads_queue_t waiting`
  
  Queue of waiting threads.

- `bool suspended`
  
  Queue suspended state flag.

- `volatile size_t bcounter`
  
  Active buffers counter.

- `uint8_t * bwrptr`
  
  Buffer write pointer.
• uint8_t * brdptr
  Buffer read pointer.
• uint8_t * btop
  Pointer to the buffers boundary.
• size_t bsize
  Size of buffers.
• size_t bn
  Number of buffers.
• uint8_t * buffers
  Queue of buffer objects.
• uint8_t * ptr
  Pointer for R/W sequential access.
• uint8_t * top
  Boundary for R/W sequential access.
• bqnotify_t notify
  Data notification callback.
• void * link
  Application defined field.

8.70.1 Detailed Description
Structure of a generic buffers queue.

8.70.2 Field Documentation

8.70.2.1 waiting
threads_queue_t ioBuffers_queue::waiting
Queue of waiting threads.

8.70.2.2 suspended
bool ioBuffers_queue::suspended
Queue suspended state flag.

8.70.2.3 bcouter
volatile size_t ioBuffers_queue::bcouter
Active buffers counter.
### 8.70.2.4 bwrptr

```
uint8_t* io_buffers_queue::bwrptr
```

Buffer write pointer.

### 8.70.2.5 brdptr

```
uint8_t* io_buffers_queue::brdptr
```

Buffer read pointer.

### 8.70.2.6 btop

```
uint8_t* io_buffers_queue::btop
```

Pointer to the buffers boundary.

### 8.70.2.7 bsize

```
size_t io_buffers_queue::bsize
```

Size of buffers.

**Note**

The buffer size must be not lower than `sizeof(size_t) + 2` because the first bytes are used to store the used size of the buffer.

### 8.70.2.8 bn

```
size_t ioBuffersQueue::bn
```

Number of buffers.
8.70.2.9 buffers

uint8_t* io_buffers_queue::buffers

Queue of buffer objects.

8.70.2.10 ptr

uint8_t* io_buffers_queue::ptr

Pointer for R/W sequential access.

Note

It is NULL if a new buffer must be fetched from the queue.

8.70.2.11 top

uint8_t* io_buffers_queue::top

Boundary for R/W sequential access.

8.70.2.12 notify

bqnotify_t io_buffers_queue::notify

Data notification callback.

8.70.2.13 link

void* io_buffers_queue::link

Application defined field.
8.71 io_queue Struct Reference

Generic I/O queue structure.
#include <hal_queues.h>

Collaboration diagram for io_queue:

![Collaboration diagram](image)

Data Fields

- **threads_queue_t q_waiting**
  Queue of waiting threads.

- **volatile size_t q_counter**
  Resources counter.

- **uint8_t * q_buffer**
  Pointer to the queue buffer.

- **uint8_t * q_top**
  Pointer to the first location after the buffer.

- **uint8_t * q_wrptr**
  Write pointer.

- **uint8_t * q_rdptr**
  Read pointer.

- **qnotify_t q_notify**
  Data notification callback.

- **void * q_link**
  Application defined field.
### 8.71.1 Detailed Description

Generic I/O queue structure.

This structure represents a generic Input or Output asymmetrical queue. The queue is asymmetrical because one end is meant to be accessed from a thread context, and thus can be blocking, the other end is accessible from interrupt handlers or from within a kernel lock zone and is non-blocking.

### 8.71.2 Field Documentation

**8.71.2.1 q_waiting**

```cpp
t threads_queue_t io_queue::q_waiting
```

Queue of waiting threads.

**8.71.2.2 q_counter**

```cpp
volatile size_t io_queue::q_counter
```

Resources counter.

**8.71.2.3 q_buffer**

```cpp
uint8_t* io_queue::q_buffer
```

Pointer to the queue buffer.

**8.71.2.4 q_top**

```cpp
uint8_t* io_queue::q_top
```

Pointer to the first location after the buffer.
8.71.2.5  q_wrptr

`uint8_t* io_queue::q_wrptr`

Write pointer.

8.71.2.6  q_rdptr

`uint8_t* io_queue::q_rdptr`

Read pointer.

8.71.2.7  q_notify

`qnotify_t io_queue::q_notify`

Data notification callback.

8.71.2.8  q_link

`void* io_queue::q_link`

Application defined field.

8.72  IOBus Struct Reference

I/O bus descriptor.

```c
#include <hal_pal.h>
```

Collaboration diagram for IOBus:

```
<table>
<thead>
<tr>
<th>IOBus</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ portid</td>
</tr>
<tr>
<td>+ mask</td>
</tr>
<tr>
<td>+ offset</td>
</tr>
</tbody>
</table>
```
Data Fields

- `ioportid_t portid`
  Port identifier.
- `ioportmask_t mask`
  Bus mask aligned to port bit 0.
- `uint_fast8_t offset`
  Offset, within the port, of the least significant bit of the bus.

8.72.1 Detailed Description

I/O bus descriptor.

This structure describes a group of contiguous digital I/O lines that have to be handled as bus.

Note

I/O operations on a bus do not affect I/O lines on the same port but not belonging to the bus.

8.72.2 Field Documentation

8.72.2.1 portid

`ioportid_t IOBus::portid`

Port identifier.

8.72.2.2 mask

`ioportmask_t IOBus::mask`

Bus mask aligned to port bit 0.

Note

The bus mask implicitly define the bus width. A logic AND is performed on the bus data.

8.72.2.3 offset

`uint_fast8_t IOBus::offset`

Offset, within the port, of the least significant bit of the bus.

ChibiOS/HAL
8.73 MemoryStream Struct Reference

Memory stream object.

#include <memstreams.h>

Inheritance diagram for MemoryStream:
Collaboration diagram for MemoryStream:

Data Fields

- const struct MemStreamVMT * vmt

Virtual Methods Table.

8.73.1 Detailed Description

Memory stream object.

8.73.2 Field Documentation

8.73.2.1 vmt

const struct MemStreamVMT * MemoryStream::vmt

Virtual Methods Table.
8.74 MemStreamVMT Struct Reference

MemStream virtual methods table.

#include <memstreams.h>

Collaboration diagram for MemStreamVMT:

```
MemStreamVMT
```

8.74.1 Detailed Description

MemStream virtual methods table.

8.75 mfs_bank_header_t Union Reference

Type of a bank header.

#include <hal_mfs.h>

Collaboration diagram for mfs_bank_header_t:

```
mfs_bank_header_t
+ magic1
+ magic2
+ counter
+ reserved1
+ crc
+ fields
+ hdr8
+ hdr32
```
8.75.1 Detailed Description

Type of a bank header.

Note
The header resides in the first 16 bytes of a bank.

8.75.2 Field Documentation

8.75.2.1 magic1

uint32_t mfs_bank_header_t::magic1

Bank magic 1.

8.75.2.2 magic2

uint32_t mfs_bank_header_t::magic2

Bank magic 2.

8.75.2.3 counter

uint32_t mfs_bank_header_t::counter

Usage counter of the bank.
This value is increased each time a bank swap is performed. It indicates how much wearing the flash has already endured.

8.75.2.4 reserved1

uint16_t mfs_bank_header_t::reserved1

Reserved field.
8.75.2.5 crc

```c
uint16_t mfs_bank_header_t::crc
```

Header CRC.

### 8.76 mfs_data_header_t Union Reference

Type of a data block header.

```c
#include <hal_mfs.h>
```

Collaboration diagram for mfs_data_header_t:

![Collaboration diagram for mfs_data_header_t](image)

#### 8.76.1 Detailed Description

Type of a data block header.

This structure is placed before each written data block.

#### 8.76.2 Field Documentation

##### 8.76.2.1 magic1

```c
uint32_t mfs_data_header_t::magic1
```

Data header magic 1.
8.76.2.2  magic2

uint32_t mfs_data_header_t::magic2

Data header magic 2.

8.76.2.3  id

uint16_t mfs_data_header_t::id

Record identifier.

8.76.2.4  crc

uint16_t mfs_data_header_t::crc

Data CRC.

8.76.2.5  size

uint32_t mfs_data_header_t::size

Data size.

Note

The next record is located at MFS_ALIGN_NEXT(size).
8.77  mfs_nocache_buffer Union Reference

Type of an non-cacheable MFS buffer.

#include <hal_mfs.h>

Collaboration diagram for mfs_nocache_buffer:

8.77.1  Detailed Description

Type of an non-cacheable MFS buffer.

8.78  mfs_transaction_op_t Struct Reference

Type of a buffered write/erase operation within a transaction.

#include <hal_mfs.h>
Collaboration diagram for mfs_transaction_op_t:

![Collaboration diagram for mfs_transaction_op_t](image)

Data Fields

- `flash_offset_t offset`
  Written header offset.
- `size_t size`
  Written data size.
- `mfs_id_t id`
  Record identifier.

### 8.78.1 Detailed Description

Type of a buffered write/erase operation within a transaction.

### 8.78.2 Field Documentation

#### 8.78.2.1 offset

```c
flash_offset_t mfs_transaction_op_t::offset
```

Written header offset.

#### 8.78.2.2 size

```c
size_t mfs_transaction_op_t::size
```

Written data size.
8.78.2.3 id

```c
mfs_id_t mfs_transaction_op_t::id
```

Record identifier.

8.79 MFSConfig Struct Reference

Type of a MFS configuration structure.

```c
#include <hal_mfs.h>
```

Collaboration diagram for MFSConfig:

![Collaboration diagram for MFSConfig](image)

**Data Fields**

- `BaseFlash * flashp`
  
  *Flash driver associated to this MFS instance.*

- `uint32_t erased`
  
  *Erased value.*
• `flash_offset_t bank_size`
  Banks size.

• `flash_sector_t bank0_start`
  Base sector index for bank 0.

• `flash_sector_t bank0_sectors`
  Number of sectors for bank 0.

• `flash_sector_t bank1_start`
  Base sector index for bank 1.

• `flash_sector_t bank1_sectors`
  Number of sectors for bank 1.

### 8.79.1 Detailed Description

Type of a MFS configuration structure.

### 8.79.2 Field Documentation

#### 8.79.2.1 flashp

`BaseFlash* MFSConfig::flashp`

Flash driver associated to this MFS instance.

#### 8.79.2.2 erased

`uint32_t MFSConfig::erased`

Erased value.

#### 8.79.2.3 bank_size

`flash_offset_t MFSConfig::bank_size`

Banks size.
8.79.2.4 bank0_start

`flash_sector_t MFSConfig::bank0_start`

Base sector index for bank 0.

8.79.2.5 bank0_sectors

`flash_sector_t MFSConfig::bank0_sectors`

Number of sectors for bank 0.

**Note**

The total size of bank0 sectors must be greater or equal to `bank_size`.

8.79.2.6 bank1_start

`flash_sector_t MFSConfig::bank1_start`

Base sector index for bank 1.

8.79.2.7 bank1_sectors

`flash_sector_t MFSConfig::bank1_sectors`

Number of sectors for bank 1.

**Note**

The total size of bank1 sectors must be greater or equal to `bank_size`. 
8.80 MFSDriver Struct Reference

Type of an MFS instance.

#include <hal_mfs.h>

Collaboration diagram for MFSDriver:

Data Fields

- **mfs_state_t state**
  
  Driver state.

- **const MFSConfig * config**
  
  Current configuration data.

- **mfs_bank_t current_bank**
  
  Bank currently in use.

- **uint32_t current_counter**
  
  Usage counter of the current bank.

- **flash_offset_t next_offset**
  
  Pointer to the next free position in the current bank.

- **flash_offset_t used_space**

---

ChibiOS/HAL
Used space in the current bank without considering erased records.

- mfs_record_descriptor_t descriptors [MFS_CFG_MAX_RECORDS]
  Offsets of the most recent instance of the records.
- flash_offset_t tr_next_offset
  Next write offset for current transaction.
- flash_offset_t tr_limit_offset
  Maximum offset for the transaction.
- uint32_t tr_nops
  Number of buffered operations in current transaction.
- mfs_transaction_op_t tr_ops [MFS_CFG_TRANSACTION_MAX]
  Buffered operations in current transaction.
- mfs_nocache_buffer_t *ncbuf
  Associated non-cacheable buffer.

8.80.1 Detailed Description

Type of an MFS instance.

8.80.2 Field Documentation

8.80.2.1 state

mfs_state_t MFSDriver::state

Driver state.

8.80.2.2 config

const MFSConfig* MFSDriver::config

Current configuration data.

8.80.2.3 current_bank

mfs_bank_t MFSDriver::current_bank

Bank currently in use.
8.80.2.4 current_counter

`uint32_t MFSDriver::current_counter`

Usage counter of the current bank.

8.80.2.5 next_offset

`flash_offset_t MFSDriver::next_offset`

Pointer to the next free position in the current bank.

8.80.2.6 used_space

`flash_offset_t MFSDriver::used_space`

Used space in the current bank without considering erased records.

8.80.2.7 descriptors

`mfs_record_descriptor_t MFSDriver::descriptors[MFS_CFG_MAX_RECORDS]`

Offsets of the most recent instance of the records.

Note

Zero means that there is not a record with that id.

8.80.2.8 tr_next_offset

`flash_offset_t MFSDriver::tr_next_offset`

Next write offset for current transaction.

8.80.2.9 tr_limit_offset

`flash_offset_t MFSDriver::tr_limit_offset`

Maximum offset for the transaction.
8.80.2.10  tr_nops

`uint32_t MFSDriver::tr_nops`

Number of buffered operations in current transaction.

8.80.2.11  tr_ops

`mfs_transaction_op_t MFSDriver::tr_ops[MFS_CFG_TRANSACTION_MAX]`

Buffered operations in current transaction.

8.80.2.12  ncbuf

`mfs_nocache_buffer_t* MFSDriver::ncbuf`

Associated non-cacheable buffer.

8.81  mmc_spi_config_t Struct Reference

Type of a MMC/SD over SPI driver configuration structure.

`#include <hal_mmc_spi.h>`
Data Fields

- **SPIDriver * spip**
  
  *SPI driver associated to this MMC driver.*

- **const SPIConfig * lscfg**
  
  *SPI low speed configuration used during initialization.*

- **const SPIConfig * hscfg**
  
  *SPI high speed configuration used during transfers.*

8.81.1 Detailed Description

Type of a MMC/SD over SPI driver configuration structure.
8.81.2 Field Documentation

8.81.2.1 spip

SPI driver associated to this MMC driver.

8.81.2.2 lscfg

SPI low speed configuration used during initialization.

8.81.2.3 hscfg

SPI high speed configuration used during transfers.

8.82 mmc_spi_driver_t Struct Reference

Structure representing a MMC/SD over SPI driver.

#include <hal_mmc_spi.h>
Inheritance diagram for mmc_spi_driver_t:
Collaboration diagram for mmc_spi_driver_t:

Data Fields

- const struct mmc_spi_driver_vmt * vmt
  
  Virtual Methods Table.
- const _mmcsd_block_device_data mmc_spi_config_t * config
  
  Current configuration data.
- bool block_addresses
  
  Addresses use blocks instead of bytes.
- uint8_t * buffer
  
  Pointer to an un-cacheable buffer of size MMC_BUFFER_SIZE.

8.82.1 Detailed Description

Structure representing a MMC/SD over SPI driver.
8.82.2 Field Documentation

8.82.2.1 vmt

const struct mmc_spi_driver_vmt* mmc_spi_driver_t::vmt

Virtual Methods Table.

8.82.2.2 config

const _mmcsd_block_device_data mmc_spi_config_t* mmc_spi_driver_t::config

Current configuration data.

8.82.2.3 block_addresses

bool mmc_spi_driver_t::block_addresses

Addresses use blocks instead of bytes.

8.82.2.4 buffer

uint8_t* mmc_spi_driver_t::buffer

Pointer to an un-cacheable buffer of size MMC_BUFFER_SIZE.
8.83 mmc_spi_driver_vmt Struct Reference

MMCDriver virtual methods table.

#include <hal_mmc_spi.h>

Inheritance diagram for mmc_spi_driver_vmt:
Collaboration diagram for mmc_spi_driver_vmt:

8.83.1 Detailed Description

MMCDriver virtual methods table.

8.84 MMCSDBlockDevice Struct Reference

MCC/SD block device class.

#include <hal_mmcsd.h>

ChibiOS/HAL
Inheritance diagram for MMCSDBlockDevice:
Data Fields

- const struct MMCSDBlockDeviceVMT * vmt
  Virtual Methods Table.

8.84.1 Detailed Description

MCC/SD block device class.

This class represents a, block-accessible, MMC/SD device.

8.84.2 Field Documentation

8.84.2.1 vmt

const struct MMCSDBlockDeviceVMT * MMCSDBlockDevice::vmt

Virtual Methods Table.
8.85  MMCSDBlockDeviceVMT Struct Reference

`MMCSDBlockDevice` virtual methods table.

```c
#include <hal_mmcsd.h>
```

Inheritance diagram for MMCSDBlockDeviceVMT:

![Inheritance Diagram]

Collaboration diagram for MMCSDBlockDeviceVMT:
8.85.1 Detailed Description

nullstream.h

8.86 NullStream Struct Reference

Null stream object.

#include <nullstreams.h>

Inheritance diagram for NullStream:

[Diagram of class inheritance structure]
Collaboration diagram for NullStream:

**Data Fields**

- const struct NullStreamVMT * vmt
  
  *Virtual Methods Table.*

**8.86.1 Detailed Description**

Null stream object.

**8.86.2 Field Documentation**

**8.86.2.1 vmt**

const struct NullStreamVMT* NullStream::vmt

*Virtual Methods Table.*
8.87  NullStreamVMT Struct Reference

NullStream virtual methods table.

#include <nullstreams.h>

Collaboration diagram for NullStreamVMT:

8.87.1  Detailed Description

NullStream virtual methods table.

8.88  PALConfig Struct Reference

Generic I/O ports static initializer.

#include <hal_pal_lld.h>

Collaboration diagram for PALConfig:

8.88.1  Detailed Description

Generic I/O ports static initializer.

An instance of this structure must be passed to palInit() at system startup time in order to initialized the digital I/O subsystem. This represents only the initial setup, specific pads or whole ports can be reprogrammed at later time.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.
8.89  palevent_t Struct Reference

Type of a PAL event record.

```c
#include <hal_pal.h>
```

Collaboration diagram for palevent_t:

![Collaboration diagram](image)

**Data Fields**

- `threads_queue_t threads`  
  Threads queued for an event.
- `palcallback_t cb`  
  Event callback.
- `void * arg`  
  Event callback argument.

8.89.1 Detailed Description

Type of a PAL event record.

8.89.2 Field Documentation

8.89.2.1 threads

```c
threads_queue_t palevent_t::threads
```

Threads queued for an event.
8.89.2 cb

`palcallback_t palevent_t::cb`

Event callback.

8.89.2.3 arg

`void* palevent_t::arg`

Event callback argument.

8.90 PWMChannelConfig Struct Reference

Type of a PWM driver channel configuration structure.

```
#include <hal_pwm_lld.h>
```

Collaboration diagram for PWMChannelConfig:
Data Fields

- `pwmmode_t mode`
  
  Channel active logic level.

- `pwmcallback_t callback`
  
  Channel callback pointer.

8.90.1 Detailed Description

Type of a PWM driver channel configuration structure.

8.90.2 Field Documentation

8.90.2.1 mode

```
pwmmode_t PWMChannelConfig::mode
```

Channel active logic level.

8.90.2.2 callback

```
pwmcallback_t PWMChannelConfig::callback
```

Channel callback pointer.

Note

This callback is invoked on the channel compare event. If set to `NULL` then the callback is disabled.
8.91 PWMConfig Struct Reference

Type of a PWM driver configuration structure.

```c
#include <hal_pwm_lld.h>
```

Collaboration diagram for PWMConfig:

![Collaboration Diagram](image)

**Data Fields**

- `uint32_t frequency`
  - Timer clock in Hz.
- `pwmcnt_t period`
  - PWM period in ticks.
- `pwmcallback_t callback`
  - Periodic callback pointer.
- `PWMChannelConfig channels [PWM_CHANNELS]`
  - Channels configurations.

8.91.1 Detailed Description

Type of a PWM driver configuration structure.
8.91.2 Field Documentation

8.91.2.1 frequency

`uint32_t PWMConfig::frequency`

Timer clock in Hz.

Note
The low level can use assertions in order to catch invalid frequency specifications.

8.91.2.2 period

`pwmcnt_t PWMConfig::period`

PWM period in ticks.

Note
The low level can use assertions in order to catch invalid period specifications.

8.91.2.3 callback

`pwmcallback_t PWMConfig::callback`

Periodic callback pointer.

Note
This callback is invoked on PWM counter reset. If set to `NULL` then the callback is disabled.

8.91.2.4 channels

`PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]`

Channels configurations.
8.92 PWMDriver Struct Reference

Structure representing a PWM driver.

```c
#include <hal_pwm_lld.h>
```

Collaboration diagram for PWMDriver:

![Collaboration Diagram](image)

Data Fields

- `pwmstate_t state`
  - Driver state.
- `const PWMConfig * config`
  - Current driver configuration data.
- `pwmcnt_t period`
  - Current PWM period in ticks.
- `pwmchnmsk_t enabled`
  - Mask of the enabled channels.
- `pwmchannel_t channels`
  - Number of channels in this instance.
8.92.1 Detailed Description

Structure representing a PWM driver.

8.92.2 Field Documentation

8.92.2.1 state

pwmstate_t PWMDriver::state

Driver state.

8.92.2.2 config

const PWMConfig* PWMDriver::config

Current driver configuration data.

8.92.2.3 period

pwmcnt_t PWMDriver::period

Current PWM period in ticks.

8.92.2.4 enabled

pwmchnmask_t PWMDriver::enabled

Mask of the enabled channels.

8.92.2.5 channels

pwmchannel_t PWMDriver::channels

Number of channels in this instance.
8.93 RTCAAlarm Struct Reference

Type of a structure representing an RTC alarm time stamp.

#include <hal_rtc_lld.h>

Collaboration diagram for RTCAAlarm:

8.93.1 Detailed Description

Type of a structure representing an RTC alarm time stamp.

8.94 RTCDatetime Struct Reference

Type of a structure representing an RTC date/time stamp.

#include <hal_rtc.h>

Collaboration diagram for RTCDatetime:
Data Fields

- `uint32_t year`: 8
  Years since 1980.

- `uint32_t month`: 4
  Months 1..12.

- `uint32_t dstflag`: 1
  DST correction flag.

- `uint32_t dayofweek`: 3
  Day of week 1..7.

- `uint32_t day`: 5
  Day of the month 1..31.

- `uint32_t millisecond`: 27
  Milliseconds since midnight.

8.94.1 Detailed Description

Type of a structure representing an RTC date/time stamp.

8.94.2 Field Documentation

8.94.2.1 year

`uint32_t RTCDateTime::year`

Years since 1980.

8.94.2.2 month

`uint32_t RTCDateTime::month`

Months 1..12.
8.94.2.3 dstflag

uint32_t RTCDateTime::dstflag

DST correction flag.

8.94.2.4 dayofweek

uint32_t RTCDateTime::dayofweek

Day of week 1..7.

8.94.2.5 day

uint32_t RTCDateTime::day

Day of the month 1..31.

8.94.2.6 millisecond

uint32_t RTCDateTime::millisecond

Milliseconds since midnight.

8.95 RTCDriver Struct Reference

Structure representing an RTC driver.

#include <hal_rtc.h>
Collaboration diagram for RTCDriver:

Data Fields

- const struct RTCDriverVMT * vmt
  
  Virtual Methods Table.

8.95.1 Detailed Description

Structure representing an RTC driver.

8.95.2 Field Documentation

8.95.2.1 vmt

const struct RTCDriverVMT * RTCDriver::vmt

Virtual Methods Table.
RTCDriver virtual methods table.

#include <hal_rtc.h>

Inheritance diagram for RTCDriverVMT:
Collaboration diagram for RTCDriverVMT:

![Collaboration Diagram]

**Additional Inherited Members**

**8.96.1 Detailed Description**

*RTCDriver* virtual methods table.

**8.97 SDCCConfig Struct Reference**

Driver configuration structure.

```c
#include <hal_sdc_lld.h>
```

Collaboration diagram for SDCCConfig:

![Collaboration Diagram]
Data Fields

- `sdcbusmode_t bus_width`

  Bus width.

8.97.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.97.2 Field Documentation

8.97.2.1 bus_width

`sdcbusmode_t SDCConfig::bus_width`

Bus width.

8.98 SDCDriver Struct Reference

Structure representing an SDC driver.

#include <hal_sdc_lld.h>
Collaboration diagram for SDCDriver:

- const struct SDCDriverVMT * vmt
  Virtual Methods Table.
- const _mmcsd_block_device_data SDCConfig * config
  Current configuration data.
- sdcmode_t cardmode
  Various flags regarding the mounted card.
- sdcflags_t errors
  Errors flags.
- uint32_t rca
  Card RCA.
- uint8_t buf [MMCSD_BLOCK_SIZE]
  Buffer for internal operations.

8.98.1 Detailed Description

Structure representing an SDC driver.
8.98.2 Field Documentation

8.98.2.1 vmt

const struct SDCDriverVMT* SDCDriver::vmt

Virtual Methods Table.

8.98.2.2 config

const _mmcsd_block_device_data SDCConfig* SDCDriver::config

Current configuration data.

8.98.2.3 cardmode

sdcmode_t SDCDriver::cardmode

Various flags regarding the mounted card.

8.98.2.4 errors

sdcflags_t SDCDriver::errors

Errors flags.

8.98.2.5 rca

uint32_t SDCDriver::rca

Card RCA.

8.98.2.6 buf

uint8_t SDCDriver::buf[MMCSD_BLOCK_SIZE]

Buffer for internal operations.
8.99 SDCDriverVMT Struct Reference

SDC Driver virtual methods table.

#include <hal_sdc_lld.h>

Inheritance diagram for SDCDriverVMT:
Collaboration diagram for SDCDriverVMT:

```
SDCDriverVMT
        
MMCSDBlockDeviceVMT
        
BaseBlockDeviceVMT
```

## 8.99.1 Detailed Description

SDCDriver virtual methods table.

## 8.100 SerialDriverVMT Struct Reference

SerialDriver virtual methods table.

```
#include <hal_serial.h>
```

ChibiOS/HAL
Inheritance diagram for SerialDriverVMT:

Collaboration diagram for SerialDriverVMT:
8.100.1 Detailed Description

SerialDriver virtual methods table.

8.101 SerialUSBConfig Struct Reference

Serial over USB Driver configuration structure.

#include <hal_serial_usb.h>

Collaboration diagram for SerialUSBConfig:

Data Fields

- **USBDriver** *usbp
  
  USB driver to use.
- **usbep_t bulk_in**
  
  Bulk IN endpoint used for outgoing data transfer.
- **usbep_t bulk_out**
  
  Bulk OUT endpoint used for incoming data transfer.
- **usbep_t int_in**
  
  Interrupt IN endpoint used for notifications.
8.101.1 Detailed Description

Serial over USB Driver configuration structure.

An instance of this structure must be passed to `sduStart()` in order to configure and start the driver operations.

8.101.2 Field Documentation

8.101.2.1 usbp

```c
USBDriver* SerialUSBConfig::usbp
```

USB driver to use.

8.101.2.2 bulk_in

```c
usbep_t SerialUSBConfig::bulk_in
```

Bulk IN endpoint used for outgoing data transfer.

8.101.2.3 bulk_out

```c
usbep_t SerialUSBConfig::bulk_out
```

Bulk OUT endpoint used for incoming data transfer.

8.101.2.4 int_in

```c
usbep_t SerialUSBConfig::int_in
```

Interrupt IN endpoint used for notifications.

Note

If set to zero then the INT endpoint is assumed to be not present, USB descriptors must be changed accordingly.
Full duplex serial driver class.

#include <hal_serial_usb.h>

Inheritance diagram for SerialUSBDriver:
Collaboration diagram for SerialUSBDriver:

```
Collaboration diagram for SerialUSBDriver:

Data Fields

- const struct SerialUSBDriverVMT * vmt

  Virtual Methods Table.

8.102.1 Detailed Description

Full duplex serial driver class.

This class extends BaseAsynchronousChannel by adding physical I/O queues.

8.102.2 Field Documentation

8.102.2.1 vmt

const struct SerialUSBDriverVMT* SerialUSBDriver::vmt

Virtual Methods Table.
SerialDriver virtual methods table.

#include <hal_serial_usb.h>

Inheritance diagram for SerialUSBDriverVMT:
Collaboration diagram for SerialUSBDriverVMT:

![Collaboration diagram](image)

8.103.1 Detailed Description

SerialDriver virtual methods table.

8.104 SHA1Context Struct Reference

Type of a SHA1 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA1Context:
8.104.1 Detailed Description

Type of a SHA1 context.

8.105 SHA256Context Struct Reference

Type of a SHA256 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA256Context:

![Collaboration diagram for SHA256Context](image)

8.105.1 Detailed Description

Type of a SHA256 context.

8.106 SHA512Context Struct Reference

Type of a SHA512 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA512Context:

![Collaboration diagram for SHA512Context](image)
8.106.1 Detailed Description

Type of a SHA512 context.

8.107 sio_driver_vmt Struct Reference

SIODriver virtual methods table.

#include <hal_sio.h>

Inheritance diagram for sio_driver_vmt:
8.107.1 Detailed Description

SIODriver virtual methods table.

8.108 SNORConfig Struct Reference

Type of a SNOR configuration structure.

#include <hal_serial_nor.h>

Collaboration diagram for SNORConfig:
8.108.1 Detailed Description

Type of a SNOR configuration structure.

8.109 SNORDriver Struct Reference

Type of SNOR flash class.

#include <hal_serial_nor.h>

Inheritance diagram for SNORDriver:
Collaboration diagram for SNORDriver:

Data Fields

- const struct SNORDriverVMT * vmt
  
  *SNORDriver Virtual Methods Table.*
- const _base_flash_data SNORConfig * config
  
  *Current configuration data.*
- snor_nocache_buffer_t * nocache
  
  *Non-cacheable buffer associated to this instance.*
- mutex_t mutex
  
  *Mutex protecting SNOR.*

8.109.1 Detailed Description

Type of SNOR flash class.

8.109.2 Field Documentation
8.109.2.1 vmt

const struct SNORDriverVMT* SNORDriver::vmt

SNORDriver Virtual Methods Table.

8.109.2.2 config

const _base_flash_data SNORConfig* SNORDriver::config

Current configuration data.

8.109.2.3 nocache

snor_nocache_buffer_t* SNORDriver::nocache

Non-cacheable buffer associated to this instance.

8.109.2.4 mutex

mutex_t SNORDriver::mutex

Mutex protecting SNOR.

8.110 SNORDriverVMT Struct Reference

SNOR virtual methods table.

#include <hal_serial_nor.h>

Inheritance diagram for SNORDriverVMT:
8.111 threads_queue_t Struct Reference

Collaboration diagram for SNORDriverVMT:

```
<table>
<thead>
<tr>
<th>BaseFlashVMT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>SNORDriverVMT</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

8.111.1 Detailed Description

Type of a thread queue.

A thread queue is a queue of sleeping threads, queued threads can be dequeued one at time or all together.

Note

If the OSAL is implemented on a bare metal machine without RTOS then the queue can be implemented as a single thread reference.

ChibiOS/HAL
8.112 unpacked_mmc_cid_t Struct Reference

Unpacked CID register from MMC.

```c
#include <hal_mmcsd.h>
```

Collaboration diagram for unpacked_mmc_cid_t:

```
+ mid
+ oid
+ pmn
+ prv_n
+ prv_m
+ psn
+ mdt_m
+ mdt_y
+ crc
```

8.112.1 Detailed Description

Unpacked CID register from MMC.

8.113 unpacked_mmc_csd_t Struct Reference

Unpacked CSD register from MMC.

```c
#include <hal_mmcsd.h>
```

Collaboration diagram for unpacked_mmc_csd_t:

```
+ csd_structure
+ spec_vers
+ taac
+ nsac
+ tran_speed
+ ccc
+ read_bl_len
+ read_bl_partial
+ write_blk_misalign
+ read_blk_misalign
+ and 23 more...
```
8.113.1 Detailed Description

Unpacked CSD register from MMC.

8.114 unpacked_sdc_cid_t Struct Reference

Unpacked CID register from SDC.

#include <hal_mmcsd.h>

Collaboration diagram for unpacked_sdc_cid_t:

```plaintext
table unpacked_sdc_cid_t
+ mid
+ oid
+ pnm
+ prv_n
+ prv_m
+ psn
+ mdt_m
+ mdt_y
+ crc
```

8.114.1 Detailed Description

Unpacked CID register from SDC.

8.115 unpacked_sdc_csd_10_t Struct Reference

Unpacked CSD v1.0 register from SDC.

#include <hal_mmcsd.h>
8.115.1 Detailed Description

Unpacked CSD v1.0 register from SDC.

8.116 unpacked_sdc_csd_20_t Struct Reference

Unpacked CSD v2.0 register from SDC.

#include <hal_mmcscd.h>

Collaboration diagram for unpacked_sdc_csd_20_t:
8.116.1 Detailed Description

Unpacked CSD v2.0 register from SDC.

8.117 USBConfig Struct Reference

Type of an USB driver configuration structure.

```c
#include <hal_usb_lld.h>
```

Collaboration diagram for USBConfig:
Data Fields

- **usbeventcb_t event_cb**
  USB events callback.
- **usbgetdescriptor_t get_descriptor_cb**
  Device GET_DESCRIPTOR request callback.
- **usbreqhandler_t requests_hook_cb**
  Requests hook callback.
- **usbcallback_t sof_cb**
  Start Of Frame callback.

8.117.1 Detailed Description

Type of an USB driver configuration structure.

8.117.2 Field Documentation

8.117.2.1 event_cb

`usbeventcb_t USBConfig:event_cb`

USB events callback.

This callback is invoked when an USB driver event is registered.

8.117.2.2 get_descriptor_cb

`usbgetdescriptor_t USBConfig:get_descriptor_cb`

Device GET_DESCRIPTOR request callback.

**Note**

This callback is mandatory and cannot be set to NULL.

8.117.2.3 requests_hook_cb

`usbreqhandler_t USBConfig:requests_hook_cb`

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.
8.117.4 sof_cb

`usbcallback_t USBConfig::sof_cb`

Start Of Frame callback.

8.118 USBDescriptor Struct Reference

Type of an USB descriptor.

```c
#include <hal_usb.h>
```

Collaboration diagram for USBDescriptor:

```
+ ud_size
+ ud_string
```

**Data Fields**

- `size_t ud_size`
  
  *Descriptor size in unicode characters.*

- `const uint8_t * ud_string`
  
  *Pointer to the descriptor.*

8.118.1 Detailed Description

Type of an USB descriptor.

8.118.2 Field Documentation

8.118.2.1 ud_size

```c
size_t USBDescriptor::ud_size
```

*Descriptor size in unicode characters.*
8.118.2.2 ud_string

```
const uint8_t* USBDescriptor::ud_string
```

Pointer to the descriptor.

8.119 USBDriver Struct Reference

Structure representing an USB driver.

```
#include <hal_usb_lld.h>
```

Collaboration diagram for USBDriver:

![Collaboration diagram for USBDriver](image)

Data Fields

- **usbstate_t state**
  
  Driver state.

- **const USBConfig * config**
  
  Current configuration data.

- **uint16_t transmitting**
  
  Bit map of the transmitting IN endpoints.

- **uint16_t receiving**
  
  Bit map of the receiving OUT endpoints.

- **const USBEndpointConfig * epc [USB_MAX_ENDPOINTS+1]**
  
  Active endpoints configurations.

- **void * in_params [USB_MAX_ENDPOINTS]**
  
  Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.
• **void * out_params [USB_MAX_ENDPOINTS]**  
  Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.

• **usbep0state_t ep0state**  
  Endpoint 0 state.

• **uint8_t * ep0next**  
  Next position in the buffer to be transferred through endpoint 0.

• **size_t ep0n**  
  Number of bytes yet to be transferred through endpoint 0.

• **usbcallback_t ep0endcb**  
  Endpoint 0 end transaction callback.

• **uint8_t setup [8]**  
  Setup packet buffer.

• **uint16_t status**  
  Current USB device status.

• **uint8_t address**  
  Assigned USB address.

• **uint8_t configuration**  
  Current USB device configuration.

• **usbstate_t saved_state**  
  State of the driver when a suspend happened.

### 8.119.1 Detailed Description

Structure representing an USB driver.

### 8.119.2 Field Documentation

#### 8.119.2.1 state

```c
usbstate_t USBDriver::state
```

Driver state.

#### 8.119.2.2 config

```c
const USBConfig* USBDriver::config
```

Current configuration data.
8.119.2.3 transmitting

`uint16_t USBDriver::transmitting`

Bit map of the transmitting IN endpoints.

8.119.2.4 receiving

`uint16_t USBDriver::receiving`

Bit map of the receiving OUT endpoints.

8.119.2.5 epc

`const USBEndpointConfig* USBDriver::epc[USB_MAX_ENDPOINTS+1]`

Active endpoints configurations.

8.119.2.6 in_params

`void* USBDriver::in_params[USB_MAX_ENDPOINTS]`

Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.

Note

The base index is one, the endpoint zero does not have a reserved element in this array.

8.119.2.7 out_params

`void* USBDriver::out_params[USB_MAX_ENDPOINTS]`

Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.

Note

The base index is one, the endpoint zero does not have a reserved element in this array.
8.119.2.8 ep0state

*usbep0state_t* USBDriver::ep0state

Endpoint 0 state.

8.119.2.9 ep0next

*uint8_t* USBDriver::ep0next

Next position in the buffer to be transferred through endpoint 0.

8.119.2.10 ep0n

*size_t* USBDriver::ep0n

Number of bytes yet to be transferred through endpoint 0.

8.119.2.11 ep0endcb

*usbcallback_t* USBDriver::ep0endcb

Endpoint 0 end transaction callback.

8.119.2.12 setup

*uint8_t* USBDriver::setup[8]

Setup packet buffer.

8.119.2.13 status

*uint16_t* USBDriver::status

Current USB device status.
8.119.2.14  address

uint8_t USBDriver::address

Assigned USB address.

8.119.2.15  configuration

uint8_t USBDriver::configuration

Current USB device configuration.

8.119.2.16  saved_state

usbstate_t USBDriver::saved_state

State of the driver when a suspend happened.

8.120  USBEndpointConfig Struct Reference

Type of an USB endpoint configuration structure.

#include <hal_usb_lld.h>
Collaboration diagram for USBEndpointConfig:

Data Fields

- `uint32_t ep_mode`
  
  *Type and mode of the endpoint.*
- `usbepcallback_t setup_cb`
  
  *Setup packet notification callback.*
- `usbepcallback_t in_cb`
  
  *USBInEndpointState*
**IN endpoint notification callback.**
- `usbepcallback_t out_cb`
  **OUT endpoint notification callback.**
- `uint16_t in_maxsize`
  **IN endpoint maximum packet size.**
- `uint16_t out_maxsize`
  **OUT endpoint maximum packet size.**
- `USBInEndpointState * in_state`
  USBEndpointState associated to the IN endpoint.
- `USBOutEndpointState * out_state`
  USBEndpointState associated to the OUT endpoint.

### 8.120.1 Detailed Description

Type of an USB endpoint configuration structure.

**Note**
Platform specific restrictions may apply to endpoints.

### 8.120.2 Field Documentation

#### 8.120.2.1 ep_mode

```
uint32_t USBEndpointConfig::ep_mode
```

Type and mode of the endpoint.

#### 8.120.2.2 setup_cb

```
usbepcallback_t USBEndpointConfig::setup_cb
```

Setup packet notification callback.

This callback is invoked when a setup packet has been received.

**Postcondition**

The application must immediately call `usbReadPacket()` in order to access the received packet.

**Note**

This field is only valid for `USB_EP_MODE_TYPE_CTRL` endpoints, it should be set to `NULL` for other endpoint types.
8.120.2.3 in_cb

`usbepcallback_t USBEndpointConfig::in_cb`

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

8.120.2.4 out_cb

`usbepcallback_t USBEndpointConfig::out_cb`

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

8.120.2.5 in_maxsize

`uint16_t USBEndpointConfig::in_maxsize`

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

8.120.2.6 out_maxsize

`uint16_t USBEndpointConfig::out_maxsize`

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

8.120.2.7 in_state

`USBInEndpointState* USBEndpointConfig::in_state`

USBEndpointState associated to the IN endpoint.

This structure maintains the state of the IN endpoint.

8.120.2.8 out_state

`USBOutEndpointState* USBEndpointConfig::out_state`

USBEndpointState associated to the OUT endpoint.

This structure maintains the state of the OUT endpoint.
**8.121 USBInEndpointState Struct Reference**

Type of an IN endpoint state structure.

```c
#include <hal_usb_lld.h>
```

Collaboration diagram for USBInEndpointState:

![Collaboration diagram for USBInEndpointState](image)

**Data Fields**

- `size_t txsize`
  
  Requested transmit transfer size.
- `size_t txcnt`
  
  Transmitted bytes so far.
- `const uint8_t * txbuf`
  
  Pointer to the transmission linear buffer.
- `thread_reference_t thread`
  
  Waiting thread.

**8.121.1 Detailed Description**

Type of an IN endpoint state structure.

**8.121.2 Field Documentation**

**8.121.2.1 txsize**

```c
size_t USBInEndpointState::txsize
```

Requested transmit transfer size.
8.122 USBOutEndpointState Struct Reference

Type of an OUT endpoint state structure.

#include <hal_usb_lld.h>

Collaboration diagram for USBOutEndpointState:

```
#include <hal_usb_lld.h>

ChibiOS/HAL
```
Data Fields

- `size_t rxsize`
  Requested receive transfer size.
- `size_t rxcnt`
  Received bytes so far.
- `uint8_t * rxbuf`
  Pointer to the receive linear buffer.
- `thread_reference_t thread`
  Waiting thread.

8.122.1 Detailed Description
Type of an OUT endpoint state structure.

8.122.2 Field Documentation

8.122.2.1 rxsize

`size_t USBOutEndpointState::rxsize`
Requested receive transfer size.

8.122.2.2 rxcnt

`size_t USBOutEndpointState::rxcnt`
Received bytes so far.

8.122.2.3 rxbuf

`uint8_t * USBOutEndpointState::rxbuf`
Pointer to the receive linear buffer.

8.122.2.4 thread

`thread_reference_t USBOutEndpointState::thread`
Waiting thread.
8.123 WDGConfig Struct Reference

Driver configuration structure.

#include <hal_wdg_lld.h>

Collaboration diagram for WDGConfig:

![Collaboration diagram for WDGConfig]

8.123.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.124 WDGDriver Struct Reference

Structure representing an WDG driver.

#include <hal_wdg_lld.h>

Collaboration diagram for WDGDriver:

![Collaboration diagram for WDGDriver]
Data Fields

- `wdgstate_t state`
  Driver state.
- `const WDGConfig * config`
  Current configuration data.

8.124.1 Detailed Description

Structure representing an WDG driver.

8.124.2 Field Documentation

8.124.2.1 state

`wdgstate_t WDGDriver::state`

Driver state.

8.124.2.2 config

`const WDGConfig* WDGDriver::config`

Current configuration data.

8.125 wspi_command_t Struct Reference

Type of a WSPI command descriptor.

#include <hal_wspi.h>

Collaboration diagram for wspi_command_t:
### Data Fields

- **uint32_t **\_cfg
  
  Transfer configuration field.

- **uint32_t **\_cmd
  
  Command phase data.

- **uint32_t **\_addr
  
  Address phase data.

- **uint32_t **\_alt
  
  Alternate phase data.

- **uint32_t **\_dummy
  
  Number of dummy cycles to be inserted.

### 8.125.1 Detailed Description

Type of a WSPI command descriptor.

### 8.125.2 Field Documentation

#### 8.125.2.1 \_cfg

```
uint32_t wspi_command_t::cfg
```

Transfer configuration field.

#### 8.125.2.2 \_cmd

```
uint32_t wspi_command_t::cmd
```

Command phase data.

#### 8.125.2.3 \_addr

```
uint32_t wspi_command_t::addr
```

Address phase data.

#### 8.125.2.4 \_alt

```
uint32_t wspi_command_t::alt
```

Alternate phase data.

#### 8.125.2.5 \_dummy

```
uint32_t wspi_command_t::dummy
```

Number of dummy cycles to be inserted.
Chapter 9

File Documentation

9.1 bufstreams.c File Reference

Buffered streams code.

#include <string.h>
#include "hal.h"
#include "bufstreams.h"

Functions

- void bsaObjectInit (BufferedStreamAdapter ∗bsap, BaseSequentialStream ∗bssp)
  
  *Buffered stream adapter object initialization.*

9.1.1 Detailed Description

Buffered streams code.

9.2 bufstreams.h File Reference

Buffered streams structures and macros.

Data Structures

- struct BufferedStreamAdapterVMT
  
  *BufferedStreamAdapter virtual methods table.*
- struct BufferedStreamAdapter
  
  *Buffered stream adapter object.*
Macros

- #define BUFSTREAM_BUFFER_SIZE 1
  Buffer size for unget.
- #define _buffered_stream_adapter_data
  BufferedStreamAdapter specific data.

Functions

- void bsaObjectInit (BufferedStreamAdapter *bsap, BaseSequentialStream *bssp)
  Buffered stream adapter object initialization.

9.2.1 Detailed Description

Buffered streams structures and macros.

9.3 chprintf.c File Reference

Mini printf-like functionality.

#include "hal.h"
#include "chprintf.h"
#include "memstreams.h"

Functions

- int chvprintf (BaseSequentialStream *chp, const char *fmt, va_list ap)
  System formatted output function.
- int chprintf (BaseSequentialStream *chp, const char *fmt,...)
  System formatted output function.
- int chsnprintf (char *str, size_t size, const char *fmt,...)
  System formatted output function.
- int chvsnprintf (char *str, size_t size, const char *fmt, va_list ap)
  System formatted output function.

9.3.1 Detailed Description

Mini printf-like functionality.

9.4 chprintf.h File Reference

Mini printf-like functionality.

#include <stdarg.h>
9.5 chscanf.c File Reference

Macros

• #define CHPRINTF_USE_FLOAT FALSE
  
  Float type support.

Functions

• int chvprintf (BaseSequentialStream *chp, const char *fmt, va_list ap)
  
  System formatted output function.

• int chprintf (BaseSequentialStream *chp, const char *fmt,....)
  
  System formatted output function.

• int chsnprintf (char *str, size_t size, const char *fmt,....)
  
  System formatted output function.

• int chvsnprintf (char *str, size_t size, const char *fmt, va_list ap)
  
  System formatted output function.

9.4.1 Detailed Description

Mini printf-like functionality.

9.5 chscanf.c File Reference

Mini scanf-like functionality.

#include <ctype.h>
#include "hal.h"
#include "chscanf.h"
#include "memstreams.h"

Functions

• int chvscanf (BaseBufferedStream *chp, const char *fmt, va_list ap)
  
  System formatted input function.

• int chscanf (BaseBufferedStream *chp, const char *fmt,....)
  
  System formatted input function.

• int chsnscanf (char *str, size_t size, const char *fmt,....)
  
  System formatted input function.

• int chvsnscanf (char *str, size_t size, const char *fmt, va_list ap)
  
  System formatted input function.

9.5.1 Detailed Description

Mini scanf-like functionality.
9.6 chscanf.h File Reference

Mini scanf-like functionality.

```c
#include <stdarg.h>
#include "bufstreams.h"
```

**Macros**

- `#define CHSCANF_USE_FLOAT FALSE`
  
  Float type support.

**Functions**

- `int chvscanf (BaseBufferedStream *chp, const char *fmt, va_list ap)`
  
  System formatted input function.
- `int chscanf (BaseBufferedStream *chp, const char *fmt,...)`
  
  System formatted input function.
- `int chsnscanf (char *str, size_t size, const char *fmt,...)`
  
  System formatted input function.
- `int chvsnscanf (char *str, size_t size, const char *fmt, va_list ap)`
  
  System formatted input function.

9.6.1 Detailed Description

Mini scanf-like functionality.

# hal.c File Reference

HAL subsystem code.

```c
#include "hal.h"
```

**Functions**

- `void halInit (void)`
  
  HAL initialization.

9.7.1 Detailed Description

HAL subsystem code.
9.8 hal.h File Reference

HAL subsystem header.

```c
#include "ccportab.h"
#include "osal.h"
#include "board.h"
#include "halconf.h"
#include "hal_lld.h"
#include "hal_objects.h"
#include "hal_streams.h"
#include "hal_channels.h"
#include "hal_files.h"
#include "hal_ioblock.h"
#include "hal_mmcasd.h"
#include "hal_persistent.h"
#include "hal_flash.h"
#include "hal_buffers.h"
#include "hal_queues.h"
#include "hal_buffered_serial.h"
#include "hal_pal.h"
#include "hal_adc.h"
#include "hal_can.h"
#include "hal_crypto.h"
#include "hal_dac.h"
#include "hal_efl.h"
#include "hal_gpt.h"
#include "hal_i2c.h"
#include "hal_i2s.h"
#include "hal_icu.h"
#include "hal_mac.h"
#include "hal_pwm.h"
#include "hal_rtc.h"
#include "hal_serial.h"
#include "hal_sdc.h"
#include "hal_sio.h"
#include "hal_spi.h"
#include "hal_trng.h"
#include "hal_uart.h"
#include "hal_usb.h"
#include "hal_wdg.h"
#include "hal_wspi.h"
#include "hal_mmc_spi.h"
#include "hal_serial_usb.h"
#include "hal_community.h"
```

Macros

- `#define __CHIBIOS_HAL__`
  ChibiOS/HAL identification macro.
- `#define CH_HAL_STABLE 1`
  Stable release flag.

ChibiOS/HAL version identification
• #define CH_HAL_VERSION "8.4.0"
  HAL version string.
• #define CH_HAL_MAJOR 8
  HAL version major number.
• #define CH_HAL_MINOR 4
  HAL version minor number.
• #define CH_HAL_PATCH 0
  HAL version patch number.

Return codes

• #define HAL_SUCCESS false
  HAL operation success.
• #define HAL_FAILED true
  HAL operation failed.

Return codes for HAL functions

• #define HAL_RET_SUCCESS MSG_OK
• #define HAL_RET_CONFIG_ERROR (msg_t)-16
  Configuration error.
• #define HAL_RET_NO_RESOURCE (msg_t)-17
  A required resource is not available.
• #define HAL_RET_HW_BUSY (msg_t)-18
  The peripheral is busy.
• #define HAL_RET_HW_FAILURE (msg_t)-19
  Peripheral failure.
• #define HAL_RET_UNKNOWN_CTL (msg_t)-20
  Unknown control code.

Functions

• void halInit (void)
  HAL initialization.
• static bool halClockSwitchMode (const halclkcfg_t *ccp)
  Switches to a different clock configuration.
• static halfreq_t halClockGetPointX (halclkpt_t clkpt)
  Returns the frequency of a clock point in Hz.

9.8.1 Detailed Description

HAL subsystem header.

9.9 hal_adc.c File Reference

ADC Driver code.

#include "hal.h"
Functions

- void adcInit (void)
  ADC Driver initialization.
- void adcObjectInit (ADCDriver *adcp)
  Initializes the standard part of an ADCDriver structure.
- msg_t adcStart (ADCDriver *adcp, const ADCConfig *config)
  Configures and activates the ADC peripheral.
- void adcStop (ADCDriver *adcp)
  Deactivates the ADC peripheral.
- void adcStartConversion (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Starts an ADC conversion.
- void adcStartConversionI (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Starts an ADC conversion.
- void adcStopConversion (ADCDriver *adcp)
  Stops an ongoing conversion.
- void adcStopConversionI (ADCDriver *adcp)
  Stops an ongoing conversion.
- msg_t adcConvert (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Performs an ADC conversion.
- void adcAcquireBus (ADCDriver *adcp)
  Gains exclusive access to the ADC peripheral.
- void adcReleaseBus (ADCDriver *adcp)
  Releases exclusive access to the ADC peripheral.

9.9.1 Detailed Description

ADC Driver code.

9.10 hal_adc.h File Reference

ADC Driver macros and structures.

#include "hal_adc_lld.h"

Data Structures

- struct hal_adc_configuration_group
  Conversion group configuration structure.
- struct hal_adc_config
  Driver configuration structure.
- struct hal_adc_driver
  Structure representing an ADC driver.
Macros

ADC configuration options

- #define ADC_USE_WAIT TRUE
  Enables synchronous APIs.
- #define ADC_USE_MUTUAL_EXCLUSION TRUE
  Enables the adcAcquireBus() and adcReleaseBus() APIs.

Macro Functions

- #define adcIsBufferComplete(adcp) ((bool)((adcp)->state == ADC_COMPLETE))
  Buffer state.

Low level driver helper macros

- #define _adc_reset_i(adcp) osalThreadResumeI(&(adcp)->thread, MSG_RESET)
  Resumes a thread waiting for a conversion completion.
- #define _adc_reset_s(adcp) osalThreadResumeS(&(adcp)->thread, MSG_RESET)
  Resumes a thread waiting for a conversion completion.
- #define _adc_wakeup_isr(adcp)
  Wakes up the waiting thread.
- #define _adc_timeout_isr(adcp)
  Wakes up the waiting thread with a timeout message.
- #define _adc_isr_half_code(adcp)
  Common ISR code, half buffer event.
- #define _adc_isr_full_code(adcp)
  Common ISR code, full buffer event.
- #define _adc_isr_error_code(adcp, err)
  Common ISR code, error event.

Typedefs

- typedef struct hal_adc_driver ADCDriver
  Type of a structure representing an ADC driver.
- typedef struct hal_adc_config ADCConfig
  Type of a structure representing an ADC driver configuration.
- typedef struct hal_adc_configuration_group ADCConversionGroup
  Conversion group configuration structure.
- typedef void(∗adccallback_t)(ADCDriver ∗adcp)
  Type of an ADC notification callback.
- typedef void(∗adcerrorcallback_t)(ADCDriver ∗adcp, adcerror_t err)
  Type of an ADC error callback.

Enumerations

- enum adcstate_t {
  ADC_UNINIT = 0, ADC_STOP = 1, ADC_READY = 2, ADC_ACTIVE = 3,
  ADC_COMPLETE = 4, ADC_ERROR = 5 } 
  Driver state machine possible states.
Functions

- void adcInit (void)
  ADC Driver initialization.
- void adcObjectInit (ADCDriver *adcp)
  Initializes the standard part of a ADCDriver structure.
- msg_t adcStart (ADCDriver *adcp, const ADCConfig *config)
  Configures and activates the ADC peripheral.
- void adcStop (ADCDriver *adcp)
  Deactivates the ADC peripheral.
- void adcStartConversion (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Starts an ADC conversion.
- void adcStartConversionI (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Starts an ADC conversion.
- void adcStopConversion (ADCDriver *adcp)
  Stops an ongoing conversion.
- void adcStopConversionI (ADCDriver *adcp)
  Stops an ongoing conversion.
- msg_t adcConvert (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
  Performs an ADC conversion.
- void adcAcquireBus (ADCDriver *adcp)
  Gains exclusive access to the ADC peripheral.
- void adcReleaseBus (ADCDriver *adcp)
  Releases exclusive access to the ADC peripheral.

9.10.1 Detailed Description

ADC Driver macros and structures.

9.11 hal_adc_lld.c File Reference

PLATFORM ADC subsystem low level driver source.

#include "hal.h"

Functions

- void adc_lld_init (void)
  Low level ADC driver initialization.
- void adc_lld_start (ADCDriver *adcp)
  Configures and activates the ADC peripheral.
- void adc_lld_stop (ADCDriver *adcp)
  Deactivates the ADC peripheral.
- void adc_lld_start_conversion (ADCDriver *adcp)
  Starts an ADC conversion.
- void adc_lld_stop_conversion (ADCDriver *adcp)
  Stops an ongoing conversion.
Variables

- ADCDriver ADCD1
  
  ADC1 driver identifier.

9.11.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.12 hal_adc_lld.h File Reference

PLATFORM ADC subsystem low level driver header.

Macros

- #define adc_lld_driver_fields
  
  Low level fields of the ADC driver structure.
- #define adc_lld_config_fields
  
  Low level fields of the ADC configuration structure.
- #define adc_lld_configuration_group_fields
  
  Low level fields of the ADC configuration structure.

Possible ADC errors mask bits.

- #define ADC_ERR_DMAFAILURE 1U
- #define ADC_ERR_OVERFLOW 2U
- #define ADC_ERR_AWD 4U

PLATFORM configuration options

- #define PLATFORM_ADC_USE_ADC1 FALSE
  
  ADC1 driver enable switch.

Typedefs

- typedef uint16_t adcsample_t
  
  ADC sample data type.
- typedef uint16_t adc_channels_num_t
  
  Channels number in a conversion group.
- typedef uint32_t adcerror_t
  
  Type of an ADC error mask.
9.13 hal_buffered_serial.h File Reference

Buffered Serial Driver code.

Data Structures

• struct BufferedSerialVMT
  
  BufferedSerial virtual methods table.

• struct hal_buffered_serial
  
  Buffered serial channel class.

Macros

• #define __buffered_serial_methods _base_asynchronous_channel_methods
  
  BufferedSerial specific methods.

• #define __buffered_serial_data
  
  BufferedSerial specific data.

Serial status flags (legacy)

• #define SD_PARITY_ERROR CHN_PARITY_ERROR
• #define SD_FRAMING_ERROR CHN_FRAMING_ERROR
• #define SD_OVERRUN_ERROR CHN_OVERRUN_ERROR
• #define SD_NOISE_ERROR CHN_NOISE_ERROR
• #define SD_BREAK_DETECTED CHN_BREAK_DETECTED
• #define SD_QUEUE_FULL_ERROR CHN_BUFFER_FULL_ERROR

Typedefs

• typedef struct hal_buffered_serial BufferedSerial
  
  Structure representing a buffered serial class.
Enumerations

- enum bsstate_t { BS_UNINIT = 0, BS_STOP = 1, BS_READY = 2 }

BufferedSerial state machine states.

Functions

- void bsIncomingDataI (BufferedSerial *bsp, uint8_t b)
  Handles incoming data.
- msg_t bsRequestDataI (BufferedSerial *bsp)
  Handles outgoing data.

Methods implementations

- static CC_FORCE_INLINE void __buffered_serial_objinit_impl (void *ip, const void *vmt, uint8_t *ib, size_t ibsize, qnotify_t inotify, void *iarg, uint8_t *ob, size_t obsize, qnotify_t onotify, void *oarg)
  Object initialization implementation.
- static CC_FORCE_INLINE size_t __buffered_serial_write_impl (void *ip, const uint8_t *bp, size_t n)
- static CC_FORCE_INLINE size_t __buffered_serial_read_impl (void *ip, uint8_t *bp, size_t n)
- static CC_FORCE_INLINE msg_t __buffered_serial_put_impl (void *ip, uint8_t b)
- static CC_FORCE_INLINE msg_t __buffered_serial_get_impl (void *ip)
- static CC_FORCE_INLINE msg_t __buffered_serial_put_timeout_impl (void *ip, uint8_t b, sysinterval_t timeout)
- static CC_FORCE_INLINE msg_t __buffered_serial_get_timeout_impl (void *ip, sysinterval_t timeout)
- static CC_FORCE_INLINE size_t __buffered_serial_write_timeout_impl (void *ip, const uint8_t *bp, size_t n, sysinterval_t timeout)
- static CC_FORCE_INLINE size_t __buffered_serial_read_timeout_impl (void *ip, uint8_t *bp, size_t n, sysinterval_t timeout)
- static CC_FORCE_INLINE msg_t __buffered_serial_ctl_impl (void *ip, unsigned int operation, void *arg)

9.13.1 Detailed Description

Buffered Serial Driver code.

Buffered Serial Driver macros and structures.

9.14 hal_buffers.c File Reference

I/O Buffers code.

#include <string.h>
#include "hal.h"
Functions

- void ibqObjectInit (input_buffers_queue_t *ibqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link)
  Initializes an input buffers queue object.
- void ibqResetI (input_buffers_queue_t *ibqp)
  Resets an input buffers queue.
- uint8_t * ibqGetEmptyBufferI (input_buffers_queue_t *ibqp)
  Gets the next empty buffer from the queue.
- void ibqPostFullBufferI (input_buffers_queue_t *ibqp, size_t size)
  Posts a new filled buffer to the queue.
- msg_t ibqGetFullBufferTimeout (input_buffers_queue_t *ibqp, sysinterval_t timeout)
  Gets the next filled buffer from the queue.
- msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t *ibqp, sysinterval_t timeout)
  Gets the next filled buffer from the queue.
- void ibqReleaseEmptyBuffer (input_buffers_queue_t *ibqp)
  Releases the buffer back in the queue.
- void ibqReleaseEmptyBufferS (input_buffers_queue_t *ibqp)
  Releases the buffer back in the queue.
- msg_t ibqGetTimeout (input_buffers_queue_t *ibqp, sysinterval_t timeout)
  Input queue read with timeout.
- size_t ibqReadTimeout (inputBuffersQueue_t *ibqp, uint8_t *bp, size_t n, sysinterval_t timeout)
  Input queue read with timeout.
- void obqObjectInit (outputBuffersQueue_t *obqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t onfy, void *link)
  Initializes an output buffers queue object.
- void obqResetI (outputBuffersQueue_t *obqp)
  Resets an output buffers queue.
- uint8_t * obqGetFullBufferI (outputBuffersQueue_t *obqp, size_t *sizep)
  Gets the next filled buffer from the queue.
- void obqReleaseEmptyBufferI (outputBuffersQueue_t *obqp)
  Releases the next filled buffer back in the queue.
- msg_t obqGetEmptyBufferTimeout (outputBuffersQueue_t *obqp, sysinterval_t timeout)
  Gets the next empty buffer from the queue.
- msg_t obqGetEmptyBufferTimeoutS (outputBuffersQueue_t *obqp, sysinterval_t timeout)
  Gets the next empty buffer from the queue.
- void obqPostFullBuffer (outputBuffersQueue_t *obqp, size_t size)
  Posts a new filled buffer to the queue.
- void obqPostFullBufferS (outputBuffersQueue_t *obqp, size_t size)
  Posts a new filled buffer to the queue.
- msg_t obqPutTimeout (outputBuffersQueue_t *obqp, uint8_t b, sysinterval_t timeout)
  Output queue write with timeout.
- size_t obqWriteTimeout (outputBuffersQueue_t *obqp, const uint8_t *bp, size_t n, sysinterval_t timeout)
  Output queue write with timeout.
- bool obqTryFlushI (outputBuffersQueue_t *obqp)
  Flushes the current, partially filled, buffer to the queue.
- void obqFlush (outputBuffersQueue_t *obqp)
  Flushes the current, partially filled, buffer to the queue.
9.14.1 Detailed Description

I/O Buffers code.

9.15 hal_buffers.h File Reference

I/O Buffers macros and structures.

Data Structures

- struct io_buffers_queue
  
  Structure of a generic buffers queue.

Macros

- #define BUFFERS_CHUNKS_SIZE 64
  
  Maximum size of blocks copied in critical sections.

- #define BQ_BUFFER_SIZE(n, size) (((size_t)(size) + sizeof (size_t)) * (size_t)(n))
  
  Computes the size of a buffers queue buffer size.

Macro Functions

- #define bqSizeX(bqp) ((bqp)->bn)
  
  Returns the queue's number of buffers.

- #define bqSpaceI(bqp) ((bqp)->bcounter)
  
  Return the ready buffers number.

- #define bqGetLinkX(bqp) ((bqp)->link)
  
  Returns the queue application-defined link.

- #define bqSetLinkX(bqp, lk) ((bqp)->link = lk)
  
  Sets the queue application-defined link.

- #define bqIsSuspendedX(bqp) ((bqp)->suspended)
  
  Return the suspended state of the queue.

- #define bqSuspendI(bqp)
  
  Puts the queue in suspended state.

- #define bqResumeX(bqp)
  
  Resumes normal queue operations.

- #define ibqIsEmptyI(ibqp) ((bool)(bqSpaceI(ibqp) == 0U))
  
  Evaluates to true if the specified input buffers queue is empty.

- #define ibqIsFullI(ibqp)
  
  Evaluates to true if the specified input buffers queue is full.

- #define obqIsEmptyI(obqp)
  
  Evaluates to true if the specified output buffers queue is empty.

- #define obqIsFullI(obqp) ((bool)(bqSpaceI(obqp) == 0U))
  
  Evaluates to true if the specified output buffers queue is full.
Typedefs

- `typedef struct io_buffers_queue io_buffers_queue_t`
  Type of a generic queue of buffers.
- `typedef void(*) bqnotify_t (io_buffers_queue_t *bqp)`
  Double buffer notification callback type.
- `typedef io_buffers_queue_t input_buffers_queue_t`
  Type of an input buffers queue.
- `typedef io_buffers_queue_t output_buffers_queue_t`
  Type of an output buffers queue.

Functions

- `void ibqObjectInit (input_buffers_queue_t *ibqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link)`
  Initializes an input buffers queue object.
- `void ibqResetI (input_buffers_queue_t *ibqp)`
  Resets an input buffers queue.
- `uint8_t * ibqGetEmptyBufferI (input_buffers_queue_t *ibqp)`
  Gets the next empty buffer from the queue.
- `void ibqPostFullBufferI (input_buffers_queue_t *ibqp, size_t size)`
  Posts a new filled buffer to the queue.
- `msg_t ibqGetFullBufferTimeout (input_buffers_queue_t *ibqp, sysinterval_t timeout)`
  Gets the next filled buffer from the queue.
- `msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t *ibqp, sysinterval_t timeout)`
  Gets the next filled buffer from the queue.
- `void ibqReleaseEmptyBuffer (input_buffers_queue_t *ibqp)`
  Releases the buffer back in the queue.
- `void ibqReleaseEmptyBufferS (input_buffers_queue_t *ibqp)`
  Releases the buffer back in the queue.
- `msg_t ibqGetTimeout (input_buffers_queue_t *ibqp, sysinterval_t timeout)`
  Input queue read with timeout.
- `size_t ibqReadTimeout (input_buffers_queue_t *ibqp, uint8_t *bp, size_t n, sysinterval_t timeout)`
  Input queue read with timeout.
- `void obqObjectInit (output_buffers_queue_t *obqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t onfy, void *link)`
  Initializes an output buffers queue object.
- `void obqResetI (output_buffers_queue_t *obqp)`
  Resets an output buffers queue.
- `uint8_t * obqGetFullBufferI (output_buffers_queue_t *obqp, size_t *sizep)`
  Gets the next filled buffer from the queue.
- `void obqReleaseEmptyBufferI (output_buffers_queue_t *obqp)`
  Releases the next filled buffer back in the queue.
- `msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t *obqp, sysinterval_t timeout)`
  Gets the next empty buffer from the queue.
- `msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t *obqp, sysinterval_t timeout)`
  Gets the next empty buffer from the queue.
- `void obqPostFullBuffer (output_buffers_queue_t *obqp, size_t size)`
  Posts a new filled buffer to the queue.
- `void obqPostFullBufferS (output_buffers_queue_t *obqp, size_t size)`
Posts a new filled buffer to the queue.

- `msg_t obqPutTimeout (output_buffers_queue_t *obqp, uint8_t b, sysinterval_t timeout)`
  Output queue write with timeout.

- `size_t obqWriteTimeout (output_buffers_queue_t *obqp, const uint8_t *bp, size_t n, sysinterval_t timeout)`
  Output queue write with timeout.

- `bool obqTryFlushI (outputBuffers_queue_t *obqp)`
  Flushes the current, partially filled, buffer to the queue.

- `void obqFlush (outputBuffers_queue_t *obqp)`
  Flushes the current, partially filled, buffer to the queue.

### 9.15.1 Detailed Description

I/O Buffers macros and structures.

### 9.16 hal_can.c File Reference

CAN Driver code.

```c
#include "hal.h"
```

**Functions**

- `void canInit (void)`
  CAN Driver initialization.

- `void canObjectInit (CANDriver *canp)`
  Initializes the standard part of a CANDriver structure.

- `msg_t canStart (CANDriver *canp, const CANConfig *config)`
  Configures and activates the CAN peripheral.

- `void canStop (CANDriver *canp)`
  Deactivates the CAN peripheral.

- `bool canTryTransmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)`
  Can frame transmission attempt.

- `bool canTryReceive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)`
  Can frame receive attempt.

- `void canTryAbortX (CANDriver *canp, canmbx_t mailbox)`
  Tries to abort an ongoing transmission.

- `msg_t canTransmitTimeout (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, sysinterval_t timeout)`
  Can frame transmission.

- `msg_t canReceiveTimeout (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, sysinterval_t timeout)`
  Can frame receive.

- `void canSleep (CANDriver *canp)`
  Enters the sleep mode.

- `void canWakeup (CANDriver *canp)`
  Enforces leaving the sleep mode.
9.16.1 Detailed Description

CAN Driver code.

9.17 hal_can.h File Reference

CAN Driver macros and structures.

#include "hal_can_lld.h"

Macros

• #define CAN_ANY_MAILBOX 0U
  Special mailbox identifier.

CAN status flags

• #define CAN_LIMIT_WARNING 1U
  Errors rate warning.
• #define CAN_LIMIT_ERROR 2U
  Errors rate error.
• #define CAN_BUS_OFF_ERROR 4U
  Bus off condition reached.
• #define CAN_FRAMING_ERROR 8U
  Framing error of some kind on the CAN bus.
• #define CAN_OVERFLOW_ERROR 16U
  Overflow in receive queue.

CAN configuration options

• #define CAN_USE_SLEEP_MODE TRUE
  Sleep mode related APIs inclusion switch.
• #define CAN_ENFORCE_USE_CALLBACKS FALSE
  Enforces the driver to use direct callbacks rather than OSAL events.

Macro Functions

• #define CAN_MAILBOX_TO_MASK(mbx) (1U << ((mbx) - 1U))
  Converts a mailbox index to a bit mask.
• #define canTransmit(canp, mailbox, ctp, timeout) canTransmitTimeout(canp, mailbox, ctp, timeout)
  Legacy name for canTransmitTimeout().
• #define canReceive(canp, mailbox, crf, timeout) canReceiveTimeout(canp, mailbox, crf, timeout)
  Legacy name for canReceiveTimeout().

Low level driver helper macros

• #define __can_tx_empty_isr(canp, flags)
  TX mailbox empty event.
• #define __can_rx_full_isr(canp, flags)
  RX mailbox empty full event.
• #define __can_wakeup_isr(canp)
  Wakeup event.
• #define __can_error_isr(canp, flags)
  Error event.
Enumerations

- `enum canstate_t {
  CAN_UNINIT = 0, CAN_STOP = 1, CAN_STARTING = 2, CAN_STOPPING = 3,
  CAN_READY = 4, CAN_SLEEP = 5 }

  Driver state machine possible states.

Functions

- `void canInit (void)`
  CAN Driver initialization.
- `void canObjectInit (CANDriver *canp)`
  Initializes the standard part of a CANDriver structure.
- `msg_t canStart (CANDriver *canp, const CANConfig *config)`
  Configures and activates the CAN peripheral.
- `void canStop (CANDriver *canp)`
  Deactivates the CAN peripheral.
- `bool canTryTransmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)`
  Can frame transmission attempt.
- `bool canTryReceive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)`
  Can frame receive attempt.
- `void canTryAbortX (CANDriver *canp, canmbx_t mailbox)`
  Tries to abort an ongoing transmission.
- `msg_t canTransmitTimeout (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, sysinterval_t timeout)`
  Can frame transmission.
- `msg_t canReceiveTimeout (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, sysinterval_t timeout)`
  Can frame receive.

9.17.1 Detailed Description

CAN Driver macros and structures.

9.18 hal_can_lld.c File Reference

PLATFORM CAN subsystem low level driver source.

#include "hal.h"
9.19 hal_can_lld.h File Reference

Functions

- void can_lld_init (void)
  
  Low level CAN driver initialization.
- void can_lld_start (CANDriver *canp)
  
  Configures and activates the CAN peripheral.
- void can_lld_stop (CANDriver *canp)
  
  Deactivates the CAN peripheral.
- bool can_lld_is_tx_empty (CANDriver *canp, canmbx_t mailbox)
  
  Determines whether a frame can be transmitted.
- void can_lld_transmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)
  
  Inserts a frame into the transmit queue.
- bool can_lld_is_rx_nonempty (CANDriver *canp, canmbx_t mailbox)
  
  Determines whether a frame has been received.
- void can_lld_receive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)
  
  Receives a frame from the input queue.
- void can_lld_abort (CANDriver *canp, canmbx_t mailbox)
  
  Tries to abort an ongoing transmission.
- void can_lld_sleep (CANDriver *canp)
  
  Enters the sleep mode.
- void can_lld_wakeup (CANDriver *canp)
  
  Enforces leaving the sleep mode.

Variables

- CANDriver CAND1
  
  CAN1 driver identifier.

9.18.1 Detailed Description

PLATFORM CAN subsystem low level driver source.

9.19 hal_can_lld.h File Reference

PLATFORM CAN subsystem low level driver header.

Data Structures

- struct CANTxFrame
  
  CAN transmission frame.
- struct CANRxFrame
  
  CAN received frame.
- struct hal_can_config
  
  Type of a CAN configuration structure.
- struct hal_can_driver
  
  Structure representing an CAN driver.
Macros

• #define CAN_TX_MAILBOXES 1  
  Number of transmit mailboxes.
• #define CAN_RX_MAILBOXES 1  
  Number of receive mailboxes.

PLATFORM configuration options

• #define PLATFORM_CAN_USE_CAN1 FALSE  
  CAN1 driver enable switch.

Typedefs

• typedef struct hal_can_driver CANDriver  
  Type of a structure representing an CAN driver.
• typedef uint32_t canmbx_t  
  Type of a transmission mailbox index.
• typedef void(* can_callback_t) (CANDriver *canp, uint32_t flags)  
  Type of a CAN notification callback.
• typedef struct hal_can_config CANConfig  
  Type of a CAN configuration structure.

Functions

• void can_lld_init (void)  
  Low level CAN driver initialization.
• void can_lld_start (CANDriver *canp)  
  Configures and activates the CAN peripheral.
• void can_lld_stop (CANDriver *canp)  
  Deactivates the CAN peripheral.
• bool can_lld_is_tx_empty (CANDriver *canp, canmbx_t mailbox)  
  Determines whether a frame can be transmitted.
• void can_lld_transmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)  
  Inserts a frame into the transmit queue.
• bool can_lld_is_rx_nonempty (CANDriver *canp, canmbx_t mailbox)  
  Determines whether a frame has been received.
• void can_lld_receive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)  
  Receives a frame from the input queue.
• void can_lld_abort (CANDriver *canp, canmbx_t mailbox)  
  Tries to abort an ongoing transmission.
• void can_lld_sleep (CANDriver *canp)  
  Enters the sleep mode.
• void can_lld_wakeup (CANDriver *canp)  
  Enforces leaving the sleep mode.

9.19.1 Detailed Description

PLATFORM CAN subsystem low level driver header.
9.20  hal_channels.h File Reference

I/O channels access.

Data Structures

- struct BaseChannelVMT
  BaseChannel virtual methods table.
- struct BaseChannel
  Base channel class.
- struct BaseAsynchronousChannelVMT
  BaseAsynchronousChannel virtual methods table.
- struct BaseAsynchronousChannel
  Base asynchronous channel class.

Macros

- #define _base_channel_methods
  BaseChannel specific methods.
- #define _base_channel_data _base_sequential_stream_data
  BaseChannel specific data.
- #define _base_asynchronous_channel_methods _base_channel_methods \n  BaseAsynchronousChannel specific methods.
- #define _base_asynchronous_channel_data
  BaseAsynchronousChannel specific data.

Default control operation codes.

- #define CHN_CTL_INVALID 0
  Invalid operation code.
- #define CHN_CTL_NOP 1
  Does nothing.
- #define CHN_CTL_TX_WAIT 2
  Wait for TX completion.

Macro Functions (BaseChannel)

- #define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))
  Channel blocking byte write with timeout.
- #define chnGetTimeout(ip, time) ((ip)->vmt->gett(ip, time))
  Channel blocking byte read with timeout.
- #define chnWrite(ip, bp, n) streamWrite(ip, bp, n)
  Channel blocking write.
- #define chnWriteTimeout(ip, bp, n, time) ((ip)->vmt->writet(ip, bp, n, time))
  Channel blocking write with timeout.
- #define chnRead(ip, bp, n) streamRead(ip, bp, n)
  Channel blocking read.
- #define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))
  Channel blocking read with timeout.
I/O status flags added to the event listener

- #define CHN_NO_ERROR (eventflags_t)0
  No pending conditions.
- #define CHN_CONNECTED (eventflags_t)1
  Connection happened.
- #define CHN_DISCONNECTED (eventflags_t)2
  Disconnection happened.
- #define CHN_INPUT_AVAILABLE (eventflags_t)4
  Data available in the input queue.
- #define CHN_OUTPUT_EMPTY (eventflags_t)8
  Output queue empty.
- #define CHN_TRANSMISSION_END (eventflags_t)16
  Transmission end.
- #define CHN_PARITY_ERROR (eventflags_t)32
  Parity error.
- #define CHN_FRAMING_ERROR (eventflags_t)64
  Framing error.
- #define CHN_NOISE_ERROR (eventflags_t)128
  Line noise error.
- #define CHN_OVERRUN_ERROR (eventflags_t)256
  Overflow error.
- #define CHN_IDLE_DETECTED (eventflags_t)512
  RX line idle.
- #define CHN_BREAK_DETECTED (eventflags_t)1024
  LIN Break.
- #define CHN_BUFFER_FULL_ERROR (eventflags_t)2048

Macro Functions (BaseAsynchronousChannel)

- #define chnGetEventSource(ip) (&((ip)->->event))
  Returns the I/O condition event source.
- #define chnAddFlagsI(ip, flags)
  Adds status flags to the listener's flags mask.

9.20.1 Detailed Description

I/O channels access.

This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

9.21 hal_crypto.c File Reference

Cryptographic Driver code.

#include "hal.h"
Functions

- **void cryInit (void)**
  Cryptographic Driver initialization.

- **void cryObjectInit (CRYDriver ∗cryp)**
  Initializes the standard part of a CRYDriver structure.

- **msg_t cryStart (CRYDriver ∗cryp, const CRYConfig ∗config)**
  Configures and activates the cryptographic peripheral.

- **void cryStop (CRYDriver ∗cryp)**
  Deactivates the cryptographic peripheral.

- **cryerror_t cryLoadAESTransientKey (CRYDriver ∗cryp, size_t size, const uint8_t ∗keyp)**
  Initializes the AES transient key.

- **cryerror_t cryEncryptAES (CRYDriver ∗cryp, crykey_t key_id, const uint8_t ∗in, uint8_t ∗out)**
  Encryption of a single block using AES.

- **cryerror_t cryDecryptAES (CRYDriver ∗cryp, crykey_t key_id, const uint8_t ∗in, uint8_t ∗out)**
  Decryption of a single block using AES.

- **cryerror_t cryEncryptAES_ECB (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)**
  Encryption operation using AES-ECB.

- **cryerror_t cryDecryptAES_ECB (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)**
  Decryption operation using AES-ECB.

- **cryerror_t cryEncryptAES_CBC (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)**
  Encryption operation using AES-CBC.

- **cryerror_t cryDecryptAES_CBC (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)**
  Decryption operation using AES-CBC.

- **cryerror_t cryEncryptAES_CFB (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)**
  Encryption operation using AES-CFB.

- **cryerror_t cryDecryptAES_CFB (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)**
  Decryption operation using AES-CTR.

- **cryerror_t cryEncryptAES_GCM (CRYDriver ∗cryp, crykey_t key_id, size_t auth_size, const uint8_t ∗auth_in, size_t text_size, const uint8_t ∗text_in, uint8_t ∗text_out, const uint8_t ∗iv, size_t tag_size, uint8_t ∗tag_out)**
  Encryption operation using AES-GCM.

- **cryerror_t cryDecryptAES_GCM (CRYDriver ∗cryp, crykey_t key_id, size_t auth_size, const uint8_t ∗auth_in, size_t text_size, const uint8_t ∗text_in, uint8_t ∗text_out, const uint8_t ∗iv, size_t tag_size, const uint8_t ∗tag_out)**
  Decryption operation using AES-GCM.

- **cryerror_t cryLoadDESTransientKey (CRYDriver ∗cryp, size_t size, const uint8_t ∗keyp)**
  Initializes the DES transient key.

- **cryerror_t cryEncryptDES (CRYDriver ∗cryp, crykey_t key_id, const uint8_t ∗in, uint8_t ∗out)**
  Encryption of a single block using (T)DES.

- **cryerror_t cryDecryptDES (CRYDriver ∗cryp, crykey_t key_id, const uint8_t ∗in, uint8_t ∗out)**
  Decryption of a single block using (T)DES.
Decryption of a single block using (T)DES.

- cryerror_t cryEncryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)
  Encryption operation using (T)DES-ECB.

- cryerror_t cryDecryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)
  Decryption operation using (T)DES-ECB.

- cryerror_t cryEncryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
  Encryption operation using (T)DES-CBC.

- cryerror_t cryDecryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
  Decryption operation using (T)DES-CBC.

- cryerror_t crySHA1Init (CRYDriver *cryp, SHA1Context *sha1ctxp)
  Hash initialization using SHA1.

- cryerror_t crySHA1Update (CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)
  Hash update using SHA1.

- cryerror_t crySHA1Final (CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)
  Hash finalization using SHA1.

- cryerror_t crySHA256Init (CRYDriver *cryp, SHA256Context *sha256ctxp)
  Hash initialization using SHA256.

- cryerror_t crySHA256Update (CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)
  Hash update using SHA256.

- cryerror_t crySHA256Final (CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)
  Hash finalization using SHA256.

- cryerror_t crySHA512Init (CRYDriver *cryp, SHA512Context *sha512ctxp)
  Hash initialization using SHA512.

- cryerror_t crySHA512Update (CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)
  Hash update using SHA512.

- cryerror_t crySHA512Final (CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)
  Hash finalization using SHA512.

- cryerror_t cryLoadHMACTransientKey (CRYDriver *cryp, size_t size, const uint8_t *keyp)
  Initializes the HMAC transient key.

- cryerror_t cryHMACSHA256Init (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp)
  Hash initialization using HMAC_SHA256.

- cryerror_t cryHMACSHA256Update (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, size_t size, const uint8_t *in)
  Hash update using HMAC.

- cryerror_t cryHMACSHA256Final (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, uint8_t *out)
  Hash finalization using HMAC.

- cryerror_t cryHMACSHA512Init (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp)
  Hash initialization using HMAC_SHA512.

- cryerror_t cryHMACSHA512Update (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, size_t size, const uint8_t *in)
  Hash update using HMAC.

- cryerror_t cryHMACSHA512Final (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, uint8_t *out)
  Hash finalization using HMAC.

9.21.1 Detailed Description

Cryptographic Driver code.
9.22 hal_crypto.h File Reference

Cryptographic Driver macros and structures.

#include "hal_crypto_lld.h"
#include "hal_crypto_fallback.h"

Data Structures

- struct SHA1Context
  Type of a SHA1 context.
- struct SHA256Context
  Type of a SHA256 context.
- struct SHA512Context
  Type of a SHA512 context.
- struct HMACSHA256Context
  Type of a HMAC_SHA256 context.
- struct HMACSHA512Context
  Type of a HMAC_SHA512 context.

Macros

- #define HAL_CRY_USE_FALLBACK FALSE
  Enables the SW fall-back of the cryptographic driver.
- #define HAL_CRY_ENFORCE_FALLBACK FALSE
  Makes the driver forcibly use the fall-back implementations.
- #define HAL_CRY_USE_FALLBACK TRUE
  Enables the SW fall-back of the cryptographic driver.

Typedefs

- typedef size_t bitsize_t
  Size, in bits, of a crypto field or message.

Enumerations

- enum crystate_t { CRY_UNINIT = 0, CRY_STOP = 1, CRY_READY = 2 }
  Driver state machine possible states.
- enum cryerror_t {
  CRY_NOERROR = 0, CRY_ERR_INV_ALGO = 1, CRY_ERR_INV_KEY_SIZE = 2, CRY_ERR_INV_KEY_TYPE = 3,
  CRY_ERR_INV_KEY_ID = 4, CRY_ERR_AUTH_FAILED = 5, CRY_ERR_OP_FAILURE = 6 }
  Driver error codes.
- enum cryalgorithm_t { , cry_algo_aes, cry_algo_des, cry_algo_hmac }
  Type of an algorithm identifier.
Functions

- `void cryInit (void)`
  Cryptographic Driver initialization.
- `void cryObjectInit (CRYDriver *cryp)`
  Initializes the standard part of a `CRYDriver` structure.
- `msg_t cryStart (CRYDriver *cryp, const CRYConfig *config)`
  Configures and activates the cryptographic peripheral.
- `void cryStop (CRYDriver *cryp)`
  Deactivates the cryptographic peripheral.
- `cryerror_t cryLoadAESTransientKey (CRYDriver *cryp, size_t size, const uint8_t *keyp)`
  Initializes the AES transient key.
- `cryerror_t cryEncryptAES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`
  Encryption of a single block using AES.
- `cryerror_t cryDecryptAES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`
  Decryption of a single block using AES.
- `cryerror_t cryEncryptAES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`
  Encryption operation using AES-ECB.
- `cryerror_t cryDecryptAES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`
  Decryption operation using AES-ECB.
- `cryerror_t cryEncryptAES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Encryption operation using AES-CBC.
- `cryerror_t cryDecryptAES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Decryption operation using AES-CBC.
- `cryerror_t cryEncryptAES_CFB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Encryption operation using AES-CFB.
- `cryerror_t cryDecryptAES_CFB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Decryption operation using AES-CFB.
- `cryerror_t cryEncryptAES_GCM (CRYDriver *cryp, crykey_t key_id, size_t auth_size, const uint8_t *auth_in, size_t text_size, const uint8_t *text_in, uint8_t *text_out, const uint8_t *iv, size_t tag_size, uint8_t *tag_out)`
  Encryption operation using AES-GCM.
- `cryerror_t cryDecryptAES_GCM (CRYDriver *cryp, crykey_t key_id, size_t auth_size, const uint8_t *auth_in, size_t text_size, const uint8_t *text_in, uint8_t *text_out, const uint8_t *iv, size_t tag_size, const uint8_t *tag_in)`
  Decryption operation using AES-GCM.
- `cryerror_t cryLoadDESTransientKey (CRYDriver *cryp, size_t size, const uint8_t *keyp)`
  Initializes the DES transient key.
- `cryerror_t cryEncryptDES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`
  Encryption of a single block using (T)DES.
- `cryerror_t cryDecryptDES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`
  Decryption of a single block using (T)DES.
Decryption of a single block using (T)DES.

- cryerror_t cryDecryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)

Encryption operation using (T)DES-ECB.

- cryerror_t cryEncryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)

Decryption operation using (T)DES-ECB.

- cryerror_t cryDecryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Encryption operation using (T)DES-CBC.

- cryerror_t cryEncryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)

Decryption operation using (T)DES-CBC.

- cryerror_t crySHA1Init (CRYDriver *cryp, SHA1Context *sha1ctxp)

Hash initialization using SHA1.

- cryerror_t crySHA1Update (CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)

Hash update using SHA1.

- cryerror_t crySHA1Final (CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)

Hash finalization using SHA1.

- cryerror_t crySHA256Init (CRYDriver *cryp, SHA256Context *sha256ctxp)

Hash initialization using SHA256.

- cryerror_t crySHA256Update (CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)

Hash update using SHA256.

- cryerror_t crySHA256Final (CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)

Hash finalization using SHA256.

- cryerror_t crySHA512Init (CRYDriver *cryp, SHA512Context *sha512ctxp)

Hash initialization using SHA512.

- cryerror_t crySHA512Update (CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)

Hash update using SHA512.

- cryerror_t crySHA512Final (CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)

Hash finalization using SHA512.

- cryerror_t cryLoadHMACTransientKey (CRYDriver *cryp, size_t size, const uint8_t *keyp)

Initializes the HMAC transient key.

- cryerror_t cryHMACSHA256Init (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp)

Hash initialization using HMAC_SHA256.

- cryerror_t cryHMACSHA256Update (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, size_t size, const uint8_t *in)

Hash update using HMAC.

- cryerror_t cryHMACSHA256Final (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, uint8_t *out)

Hash finalization using HMAC.

- cryerror_t cryHMACSHA512Init (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp)

Hash initialization using HMAC_SHA512.

- cryerror_t cryHMACSHA512Update (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, size_t size, const uint8_t *in)

Hash update using HMAC.

- cryerror_t cryHMACSHA512Final (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, uint8_t *out)

Hash finalization using HMAC.

9.22.1 Detailed Description

Cryptographic Driver macros and structures.
9.23  hal_crypto_lld.c File Reference

PLATFORM cryptographic subsystem low level driver source.

#include "hal.h"

Functions

- void cry_lld_init (void)
  
  Low level crypto driver initialization.

- void cry_lld_start (CRYDriver ∗cryp)
  
  Configures and activates the crypto peripheral.

- void cry_lld_stop (CRYDriver ∗cryp)
  
  Deactivates the crypto peripheral.

- cryerror_t cry_lld_aes_loadkey (CRYDriver ∗cryp, size_t size, const uint8_t ∗keyp)
  
  Initializes the AES transient key.

- cryerror_t cry_lld_encrypt_AES (CRYDriver ∗cryp, crykey_t key_id, const uint8_t ∗in, uint8_t ∗out)
  
  Encryption of a single block using AES.

- cryerror_t cry_lld_decrypt_AES (CRYDriver ∗cryp, crykey_t key_id, const uint8_t ∗in, uint8_t ∗out)
  
  Decryption of a single block using AES.

- cryerror_t cry_lld_encrypt_AES_ECB (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out)
  
  Encryption operation using AES-ECB.

- cryerror_t cry_lld_decrypt_AES_ECB (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out)
  
  Decryption operation using AES-ECB.

- cryerror_t cry_lld_encrypt_AES_CBC (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)
  
  Encryption operation using AES-CBC.

- cryerror_t cry_lld_decrypt_AES_CBC (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)
  
  Decryption operation using AES-CBC.

- cryerror_t cry_lld_encrypt_AES_CFB (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)
  
  Encryption operation using AES-CFB.

- cryerror_t cry_lld_decrypt_AES_CFB (CRYDriver ∗cryp, crykey_t key_id, size_t size, const uint8_t ∗in, uint8_t ∗out, const uint8_t ∗iv)
  
  Decryption operation using AES-CFB.

- cryerror_t cry_lld_encrypt_AES_CTR (CRYDriver ∗cryp, crykey_t key_id, size_t size, uint8_t ∗text_in, const uint8_t ∗iv, size_t tag_size, uint8_t ∗tag_out)
  
  Encryption operation using AES-CTR.

- cryerror_t cry_lld_decrypt_AES_CTR (CRYDriver ∗cryp, crykey_t key_id, size_t size, uint8_t ∗text_in, const uint8_t ∗iv, size_t tag_size, const uint8_t ∗tag_in)
Decryption operation using AES-GCM.

- `cryerror_t cry_lld_des_loadkey (CRYDriver *cryp, size_t size, const uint8_t *keyp)`
  Initializes the DES transient key.

- `cryerror_t cry_lld_encryptDES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`
  Encryption of a single block using (T)DES.

- `cryerror_t cry_lld_decryptDES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`
  Decryption of a single block using (T)DES.

- `cryerror_t cry_lld_encryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`
  Encryption operation using (T)DES-ECB.

- `cryerror_t cry_lld_decryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`
  Decryption operation using (T)DES-ECB.

- `cryerror_t cry_lld_encryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Encryption operation using (T)DES-CBC.

- `cryerror_t cry_lld_decryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  Decryption operation using (T)DES-CBC.

- `cryerror_t cry_lld_SHA1_init (CRYDriver *cryp, SHA1Context *sha1ctxp)`
  Hash initialization using SHA1.

- `cryerror_t cry_lld_SHA1_update (CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)`
  Hash update using SHA1.

- `cryerror_t cry_lld_SHA1_final (CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)`
  Hash finalization using SHA1.

- `cryerror_t cry_lld_SHA256_init (CRYDriver *cryp, SHA256Context *sha256ctxp)`
  Hash initialization using SHA256.

- `cryerror_t cry_lld_SHA256_update (CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)`
  Hash update using SHA256.

- `cryerror_t cry_lld_SHA256_final (CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)`
  Hash finalization using SHA256.

- `cryerror_t cry_lld_SHA512_init (CRYDriver *cryp, SHA512Context *sha512ctxp)`
  Hash initialization using SHA512.

- `cryerror_t cry_lld_SHA512_update (CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)`
  Hash update using SHA512.

- `cryerror_t cry_lld_SHA512_final (CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)`
  Hash finalization using SHA512.

- `cryerror_t cry_lld_hmac_loadkey (CRYDriver *cryp, size_t size, const uint8_t *keyp)`
  Initializes the HMAC transient key.

- `cryerror_t cry_lld_HMACSHA256_init (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp)`
  Hash initialization using HMAC_SHA256.

- `cryerror_t cry_lld_HMACSHA256_update (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, size_t size, const uint8_t *in)`
  Hash update using HMAC.

- `cryerror_t cry_lld_HMACSHA256_final (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, uint8_t *out)`
  Hash finalization using HMAC.
• cryerror_t cry_lld_HMACSHA512_update (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, size_t size, const uint8_t *in)
  Hash update using HMAC.

• cryerror_t cry_lld_HMACSHA512_final (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, uint8_t *out)
  Hash finalization using HMAC.

Variables

• CRYDriver CRYD1
  CRY1 driver identifier.

9.23.1 Detailed Description

PLATFORM cryptographic subsystem low level driver source.

9.24 hal_crypto_lld.h File Reference

PLATFORM cryptographic subsystem low level driver header.

Data Structures

• struct CRYConfig
  Driver configuration structure.
• struct CRYDriver
  Structure representing an CRY driver.
• struct SHA1Context
  Type of a SHA1 context.
• struct SHA256Context
  Type of a SHA256 context.
• struct SHA512Context
  Type of a SHA512 context.
• struct HMACSHA256Context
  Type of a HMAC_SHA256 context.
• struct HMACSHA512Context
  Type of a HMAC_SHA512 context.
Macros

Driver capability switches

- #define CRY_LLD_SUPPORTS_AES TRUE
- #define CRY_LLD_SUPPORTS_AES_ECB TRUE
- #define CRY_LLD_SUPPORTS_AES_CBC TRUE
- #define CRY_LLD_SUPPORTS_AES_CFB TRUE
- #define CRY_LLD_SUPPORTS_AES_CTR TRUE
- #define CRY_LLD_SUPPORTS_AES_GCM TRUE
- #define CRY_LLD_SUPPORTS_DES TRUE
- #define CRY_LLD_SUPPORTS_DES_ECB TRUE
- #define CRY_LLD_SUPPORTS_DES_CBC TRUE
- #define CRY_LLD_SUPPORTS_SHA1 TRUE
- #define CRY_LLD_SUPPORTS_SHA256 TRUE
- #define CRY_LLD_SUPPORTS_SHA512 TRUE
- #define CRY_LLD_SUPPORTS_HMAC_SHA256 TRUE
- #define CRY_LLD_SUPPORTS_HMAC_SHA512 TRUE

PLATFORM configuration options

- #define PLATFORM_CRY_USE_CRY1 FALSE
  CRY1 driver enable switch.

Typedefs

- typedef uint32_t crykey_t
  CRY key identifier type.
- typedef struct CRYDriver CRYDriver
  Type of a structure representing an CRY driver.

Functions

- void cry_lld_init (void)
  Low level crypto driver initialization.
- void cry_lld_start (CRYDriver *cryp)
  Configures and activates the crypto peripheral.
- void cry_lld_stop (CRYDriver *cryp)
  Deactivates the crypto peripheral.
- cryerror_t cry_lld_aes_loadkey (CRYDriver *cryp, size_t size, const uint8_t *keyp)
  Initializes the AES transient key.
- cryerror_t cry_lld_encrypt_AES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)
  Encryption of a single block using AES.
- cryerror_t cry_lld_decrypt_AES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)
  Decryption of a single block using AES.
- cryerror_t cry_lld_encrypt_AES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)
  Encryption operation using AES-ECB.
- cryerror_t cry_lld_decrypt_AES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)
  Decryption operation using AES-ECB.
- cryerror_t cry_lld_encrypt_AES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)
  Encryption operation using AES-CBC.
Encryption operation using AES-CBC.

- `cryerror_t cry_lld_decrypt_AES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`

Decryption operation using AES-CBC.

- `cryerror_t cry_lld_encrypt_AES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`

Encryption operation using AES-CFB.

- `cryerror_t cry_lld_decrypt_AES_CFB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`

Decryption operation using AES-CFB.

- `cryerror_t cry_lld_encrypt_AES_CFB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`

Encryption operation using AES-CTR.

- `cryerror_t cry_lld_decrypt_AES_CTR (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`

Decryption operation using AES-CTR.

- `cryerror_t cry_lld_encrypt_AES_CTR (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`

Encryption operation using AES-GCM.

- `cryerror_t cry_lld_decrypt_AES_GCM (CRYDriver *cryp, crykey_t key_id, size_t auth_size, const uint8_t *auth_in, size_t text_size, const uint8_t *text_in, uint8_t *text_out, const uint8_t *iv, size_t tag_size, uint8_t *tag_out)`

Decryption operation using AES-GCM.

- `cryerror_t cry_lld_decrypt_AES_CTR (CRYDriver *cryp, crykey_t key_id, size_t auth_size, const uint8_t *auth_in, size_t text_size, const uint8_t *text_in, uint8_t *text_out, const uint8_t *iv, size_t tag_size, const uint8_t *tag_in)`

Encryption operation using AES-GCM.

- `cryerror_t cry_lld_des_loadkey (CRYDriver *cryp, size_t size, const uint8_t *keyp)`

Initializes the DES transient key.

- `cryerror_t cry_lld_encrypt_DES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`

Encryption of a single block using (T)DES.

- `cryerror_t cry_lld_decrypt_DES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)`

Decryption of a single block using (T)DES.

- `cryerror_t cry_lld_encrypt_DES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`

Encryption operation using (T)DES-ECB.

- `cryerror_t cry_lld_decrypt_DES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`

Decryption operation using (T)DES-ECB.

- `cryerror_t cry_lld_encrypt_DES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`

Encryption operation using (T)DES-CBC.

- `cryerror_t cry_lld_decrypt_DES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`

Decryption operation using (T)DES-CBC.

- `cryerror_t cry_lld_SHA1_init (CRYDriver *cryp, SHA1Context *sha1ctxp)`

Hash initialization using SHA1.

- `cryerror_t cry_lld_SHA1_update (CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)`

Hash update using SHA1.

- `cryerror_t cry_lld_SHA1_final (CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)`

Hash finalization using SHA1.

- `cryerror_t cry_lld_SHA256_init (CRYDriver *cryp, SHA256Context *sha256ctxp)`

Hash initialization using SHA256.

- `cryerror_t cry_lld_SHA256_update (CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)`

Hash update using SHA256.
Hash update using SHA256:
• cryerror_t cry_lld_SHA256_final (CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)
  Hash finalization using SHA256.
• cryerror_t cry_lld_SHA512_init (CRYDriver *cryp, SHA512Context *sha512ctxp)
  Hash initialization using SHA512.
• cryerror_t cry_lld_SHA512_update (CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)
  Hash update using SHA512.
• cryerror_t cry_lld_SHA512_final (CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)
  Hash finalization using SHA512.
• cryerror_t cry_lld_hmac_loadkey (CRYDriver *cryp, size_t size, const uint8_t *keyp)
  Initializes the HMAC transient key.
• cryerror_t cry_lld_HMACSHA256_init (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp)
  Hash initialization using HMAC_SHA256.
• cryerror_t cry_lld_HMACSHA256_update (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, size_t size, const uint8_t *in)
  Hash update using HMAC.
• cryerror_t cry_lld_HMACSHA256_final (CRYDriver *cryp, HMACSHA256Context *hmacsha256ctxp, uint8_t *out)
  Hash finalization using HMAC.
• cryerror_t cry_lld_HMACSHA512_init (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp)
  Hash initialization using HMAC_SHA512.
• cryerror_t cry_lld_HMACSHA512_update (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, size_t size, const uint8_t *in)
  Hash update using HMAC.
• cryerror_t cry_lld_HMACSHA512_final (CRYDriver *cryp, HMACSHA512Context *hmacsha512ctxp, uint8_t *out)
  Hash finalization using HMAC.

9.24.1 Detailed Description

PLATFORM cryptographic subsystem low level driver header.

9.25 hal_dac.c File Reference

DAC Driver code.

#include "hal.h"
Functions

- void dacInit (void)
  
  DAC Driver initialization.

- void dacObjectInit (DACDriver *dcp)
  
  Initializes the standard part of a DACDriver structure.

- msg_t dacStart (DACDriver *dcp, const DACConfig *config)
  
  Configures and activates the DAC peripheral.

- void dacStop (DACDriver *dcp)
  
  Deactivates the DAC peripheral.

- void dacPutChannelX (DACDriver *dcp, dacchannel_t channel, dacsamples_t sample)
  
  Outputs a value directly on a DAC channel.

- void dacStartConversion (DACDriver *dcp, const DACConversionGroup *grpp, dacsamples_t *samples, size_t depth)
  
  Starts a DAC conversion.

- void dacStartConversionI (DACDriver *dcp, const DACConversionGroup *grpp, dacsamples_t *samples, size_t depth)
  
  Starts a DAC conversion.

- void dacStopConversion (DACDriver *dcp)
  
  Stops an ongoing conversion.

- void dacStopConversionI (DACDriver *dcp)
  
  Stops an ongoing conversion.

- msg_t dacConvert (DACDriver *dcp, const DACConversionGroup *grpp, dacsamples_t *samples, size_t depth)
  
  Performs a DAC conversion.

- void dacAcquireBus (DACDriver *dcp)
  
  Gains exclusive access to the DAC bus.

- void dacReleaseBus (DACDriver *dcp)
  
  Releases exclusive access to the DAC bus.

9.25.1 Detailed Description

DAC Driver code.

9.26 hal_dac.h File Reference

DAC Driver macros and structures.

#include "hal_dac_lld.h"

Data Structures

- struct hal_dac_conversion_group
  
  DAC Conversion group structure.

- struct hal_dac_config
  
  Driver configuration structure.

- struct hal_dac_driver
  
  Structure representing a DAC driver.
Macros

**DAC configuration options**

- `#define DAC_USE_WAIT TRUE`
  Enables synchronous APIs.
- `#define DAC_USE_MUTUAL_EXCLUSION TRUE`
  Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

**Low level driver helper macros**

- `#define dacIsBufferComplete(dacp) ((bool)((dacp)->state == DAC_COMPLETE))`
  Buffer state.
- `#define _dac_wait_s(dacp) osalThreadSuspendS(&(dacp)->thread)`
  Waits for operation completion.
- `#define _dac_reset_i(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)`
  Resumes a thread waiting for a conversion completion.
- `#define _dac_reset_s(dacp) osalThreadResumeS(&(dacp)->thread, MSG_RESET)`
  Resumes a thread waiting for a conversion completion.
- `#define _dac_wakeup_isr(dacp)`
  Wakes up the waiting thread.
- `#define _dac_timeout_isr(dacp)`
  Wakes up the waiting thread with a timeout message.
- `#define _dac_isr_half_code(dacp)`
  Common ISR code, half buffer event.
- `#define _dac_isr_full_code(dacp)`
  Common ISR code, full buffer event.
- `#define _dac_isr_error_code(dacp, err)`
  Common ISR code, error event.

**Typedefs**

- `typedef struct hal_dac_driver DACDriver`
  Type of a structure representing an DAC driver.
- `typedef struct hal_dac_config DACConfig`
  Type of a structure representing an DAC driver configuration.
- `typedef struct hal_dac_conversion_group DACConversionGroup`
  Type of a DAC conversion group.
- `typedef void (∗daccallback_t) (DACDriver ∗dacp)`
  DAC notification callback type.
- `typedef void (∗dacerrorcallback_t) (DACDriver ∗dacp, dacerror_t err)`
  DAC error callback type.

**Enumerations**

- `enum dacstate_t {`
  - `DAC_UNINIT = 0, DAC_STOP = 1, DAC_READY = 2, DAC_ACTIVE = 3,`
  - `DAC_COMPLETE = 4, DAC_ERROR = 5 }`
  Driver state machine possible states.
Functions

- void dacInit (void)
  
  DAC Driver initialization.

- void dacObjectInit (DACDriver *dcp)

  Initializes the standard part of a DACDriver structure.

- msg_t dacStart (DACDriver *dcp, const DACConfig *config)

  Configures and activates the DAC peripheral.

- void dacStop (DACDriver *dcp)

  Deactivates the DAC peripheral.

- void dacPutChannelX (DACDriver *dcp, dacchannel_t channel, dacsample_t sample)

  Outputs a value directly on a DAC channel.

- void dacStartConversion (DACDriver *dcp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)

  Starts a DAC conversion.

- void dacStartConversionI (DACDriver *dcp, const DACConversionGroup *grpp, dacsample_t *samples, size_t depth)

  Starts a DAC conversion.

- void dacStopConversion (DACDriver *dcp)

  Stops an ongoing conversion.

- void dacStopConversionI (DACDriver *dcp)

  Stops an ongoing conversion.

9.26.1 Detailed Description

DAC Driver macros and structures.

9.27 hal_dac_lld.c File Reference

PLATFORM DAC subsystem low level driver source.

#include "hal.h"

Functions

- void dac_lld_init (void)

  Low level DAC driver initialization.

- void dac_lld_start (DACDriver *dcp)

  Configures and activates the DAC peripheral.

- void dac_lld_stop (DACDriver *dcp)

  Deactivates the DAC peripheral.

- void dac_lld_put_channel (DACDriver *dcp, dacchannel_t channel, dacsample_t sample)

  Outputs a value directly on a DAC channel.

- void dac_lld_start_conVERSION (DACDriver *dcp)

  Starts a DAC conversion.

- void dac_lld_stop_conVERSION (DACDriver *dcp)

  Stops an ongoing conversion.
Variables

- DACDriver DACD1
  
  DAC1 driver identifier.

9.27.1 Detailed Description

PLATFORM DAC subsystem low level driver source.

9.28 hal_dac_lld.h File Reference

PLATFORM DAC subsystem low level driver header.

Macros

- #define DAC_MAX_CHANNELS 2
  
  Maximum number of DAC channels per unit.
- #define dac_lld_driver_fields
  
  Low level fields of the DAC driver structure.
- #define dac_lld_config_fields
  
  Low level fields of the DAC configuration structure.
- #define dac_lld_conversion_group_fields
  
  Low level fields of the DAC group configuration structure.

Configuration options

- #define PLATFORM_DAC_USE_DAC1 FALSE
  
  DAC1 CH1 driver enable switch.

Typedefs

- typedef uint32_t dacchannel_t
  
  Type of a DAC channel index.
- typedef uint16_t dacsample_t
  
  Type representing a DAC sample.

Enumerations

- enum dacerror_t { DAC_ERR_DMAFAILURE = 0, DAC_ERR_UNDERFLOW = 1 }
  
  Possible DAC failure causes.
Functions

- void dac_lld_init (void)
  
  Low level DAC driver initialization.

- void dac_lld_start (DACDriver *dacp)
  
  Configures and activates the DAC peripheral.

- void dac_lld_stop (DACDriver *dacp)
  
  Deactivates the DAC peripheral.

- void dac_lld_put_channel (DACDriver *dacp, dacchannel_t channel, dacsample_t sample)
  
  Outputs a value directly on a DAC channel.

- void dac_lld_start_conversion (DACDriver *dacp)
  
  Starts a DAC conversion.

- void dac_lld_stop_conversion (DACDriver *dacp)
  
  Stops an ongoing conversion.

9.28.1 Detailed Description

PLATFORM DAC subsystem low level driver header.

9.29 hal_efl.c File Reference

Embedded Flash Driver code.

#include "hal.h"

Functions

- void eflInit (void)
  
  Embedded Flash Driver initialization.

- void eflObjectInit (EFlashDriver *eflp)
  
  Initializes a generic EFlashDriver object.

- msg_t eflStart (EFlashDriver *eflp, const EFlashConfig *config)
  
  Configures and starts the driver.

- void eflStop (EFlashDriver *eflp)
  
  Stops the driver.

9.29.1 Detailed Description

Embedded Flash Driver code.

9.30 hal_efl.h File Reference

Embedded Flash Driver macros and structures.

#include "hal_efl_lld.h"
Data Structures

• struct EFlashDriverVMT
  EFlash virtual methods table.

• struct hal_efl_config
  Type of a structure representing a flash driver configuration.

• struct hal_efl_driver
  Structure representing an embedded flash driver.

Macros

• #define EFL_USE_MUTUAL_EXCLUSION TRUE
  Enables the eflAcquireUse() and eflReleaseUse() APIs.

• #define _efl_flash_methods_alone
  EFlashDriver specific methods.

• #define _efl_flash_methods
  EFlashDriver specific methods with inherited ones.

• #define _efl_driver_data
  EFlashDriver specific data.

Typedefs

• typedef struct hal_efl_driver EFlashDriver
  Type of external flash driver class.

• typedef struct hal_efl_config EFlashConfig
  Type of a structure representing a flash driver configuration.

Functions

• void eflInit (void)
  Embedded Flash Driver initialization.

• void eflObjectInit (EFlashDriver *eflp)
  Initializes a generic EFlashDriver object.

• msg_t eflStart (EFlashDriver *eflp, const EFlashConfig *config)
  Configures and starts the driver.

• void eflStop (EFlashDriver *eflp)
  Stops the driver.

9.30.1 Detailed Description

Embedded Flash Driver macros and structures.

9.31 hal_efl_lld.c File Reference

PLATFORM Embedded Flash subsystem low level driver source.

#include "hal.h"
Functions

- void efl_lld_init (void)
  
  Low level Embedded Flash driver initialization.
- void efl_lld_start (EFlashDriver *eflp)
  
  Configures and activates the Embedded Flash peripheral.
- void efl_lld_stop (EFlashDriver *eflp)
  
  Deactivates the Embedded Flash peripheral.
- const flash_descriptor_t * efl_lld_get_descriptor (void *instance)
  
  Gets the flash descriptor structure.
- flash_error_t efl_lld_read (void *instance, flash_offset_t offset, size_t n, uint8_t *rp)
  
  Read operation.
- flash_error_t efl_lld_program (void *instance, flash_offset_t offset, size_t n, const uint8_t *pp)
  
  Program operation.
- flash_error_t efl_lld_start_erase_all (void *instance)
  
  Starts a whole-device erase operation.
- flash_error_t efl_lld_start_erase_sector (void *instance, flash_sector_t sector)
  
  Starts an sector erase operation.
- flash_error_t efl_lld_query_erase (void *instance, uint32_t *msec)
  
  Queries the driver for erase operation progress.
- flash_error_t efl_lld_verify_erase (void *instance, flash_sector_t sector)
  
  Returns the erase state of a sector.

Variables

- EFlashDriver EFLD1
  
  EFL1 driver identifier.

9.31.1 Detailed Description

PLATFORM Embedded Flash subsystem low level driver source.

9.32 hal_efl_lld.h File Reference

PLATFORM Embedded Flash subsystem low level driver header.

Macros

- #define efl_lld_driver_fields
  
  Low level fields of the embedded flash driver structure.
- #define efl_lld_config_fields
  
  Low level fields of the embedded flash configuration structure.

PLATFORM configuration options

- #define PLATFORM_EFL_USE_EFL1 FALSE
  
  EFL1 driver enable switch.
Functions

- void **efl_lld_init**(void)
  
  *Low level Embedded Flash driver initialization.*

- void **efl_lld_start**(EFlashDriver *eflp)
  
  *Configures and activates the Embedded Flash peripheral.*

- void **efl_lld_stop**(EFlashDriver *eflp)
  
  *Deactivates the Embedded Flash peripheral.*

- const flash_descriptor_t **efl_lld_get_descriptor**(void *instance)
  
  *Gets the flash descriptor structure.*

- flash_error_t **efl_lld_read**(void *instance, flash_offset_t offset, size_t n, uint8_t *rp)
  
  *Read operation.*

- flash_error_t **efl_lld_program**(void *instance, flash_offset_t offset, size_t n, const uint8_t *pp)
  
  *Program operation.*

- flash_error_t **efl_lld_start_erase_all**(void *instance)
  
  *Starts a whole-device erase operation.*

- flash_error_t **efl_lld_start_erase_sector**(void *instance, flash_sector_t sector)
  
  *Starts an sector erase operation.*

- flash_error_t **efl_lld_query_erase**(void *instance, uint32_t *msec)
  
  *Queries the driver for erase operation progress.*

- flash_error_t **efl_lld_verify_erase**(void *instance, flash_sector_t sector)
  
  *Returns the erase state of a sector.*

9.32.1 Detailed Description

PLATFORM Embedded Flash subsystem low level driver header.

9.33 hal_files.h File Reference

Data files.

Data Structures

- struct **FileStreamVMT**
  
  *FileStream virtual methods table.*

- struct **FileStream**
  
  *Base file stream class.*
Macros

- `#define _file_stream_methods`  
  FileStream specific methods.
- `#define _file_stream_data_base_sequential_stream_data`  
  FileStream specific data.

Files return codes

- `#define FILE_OK STM_OK`  
  No error return code.
- `#define FILE_ERROR STM_TIMEOUT`  
  Error code from the file stream methods.
- `#define FILE_EOF STM_RESET`  
  End-of-file condition for file get/put methods.

Macro Functions (FileStream)

- `#define fileStreamWrite(ip, bp, n) streamWrite(ip, bp, n)`  
  File stream write.
- `#define fileStreamRead(ip, bp, n) streamRead(ip, bp, n)`  
  File stream read.
- `#define fileStreamPut(ip, b) streamPut(ip, b)`  
  File stream blocking byte write.
- `#define fileStreamGet(ip) streamGet(ip)`  
  File stream blocking byte read.
- `#define fileStreamClose(ip) ((ip)->vmt->close(ip))`  
  File Stream close.
- `#define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))`  
  Returns an implementation dependent error code.
- `#define fileStreamGetSize(ip, offset) ((ip)->vmt->getsize(ip, offset))`  
  Returns the current file size.
- `#define fileStreamGetPosition(ip, offset) ((ip)->vmt->getposition(ip, offset))`  
  Returns the current file pointer position.
- `#define fileStreamSetPosition(ip, offset) ((ip)->vmt->setposition(ip, offset))`  
  Moves the file current pointer to an absolute position.

Typedefs

- `typedef uint32_t fileoffset_t`  
  File offset type.

9.33.1 Detailed Description

Data files.

This header defines abstract interfaces useful to access generic data files in a standardized way.

9.34 hal_flash.c File Reference

Generic flash driver class code.

```c
#include "hal.h"
#include "hal_flash.h"
```
Functions

- `flash_error_t flashWaitErase (BaseFlash *devp)`
  Waits until the current erase operation is finished.
- `flash_offset_t flashGetSectorOffset (BaseFlash *devp, flash_sector_t sector)`
  Returns the offset of a sector.
- `uint32_t flashGetSectorSize (BaseFlash *devp, flash_sector_t sector)`
  Returns the size of a sector.
- `flash_sector_t flashGetOffsetSector (BaseFlash *devp, flash_offset_t offset)`
  Returns the sector of an offset.
- `void *flashGetOffsetAddress (BaseFlash *devp, flash_offset_t offset)`
  Get absolute address from offset.
- `flash_offset_t flashGetAddressOffset (BaseFlash *devp, void *addr)`
  Get offset from absolute address.

9.34.1 Detailed Description

Generic flash driver class code.

9.35 hal_flash.h File Reference

Generic flash driver class header.

Data Structures

- `struct flash_sector_descriptor_t`
  Flash sector descriptor.
- `struct flash_descriptor_t`
  Type of a flash device descriptor.
- `struct BaseFlashVMT`
  BaseFlash virtual methods table.
- `struct BaseFlash`
  Base flash class.

Macros

- `#define _base_flash_methods_alone`
  BaseFlash specific methods.
- `#define _base_flash_methods`
  BaseFlash specific methods with inherited ones.
- `#define _base_flash_data`
  BaseFlash specific data.

Flash attributes

- `#define FLASH_ATTR_ERASED_IS_ONE 0x00000001U`
  Defines one as the erased bit state.
• #define FLASH_ATTR_MEMORY_MAPPED 0x00000002U
  The memory is accessible in a memory mapped mode.
• #define FLASH_ATTR_REWRITABLE 0x00000004U
  Programmed pages can be programmed again.
• #define FLASH_ATTR_ECC_CAPABLE 0x00000008U
  The memory is protected by an ECC mechanism.
• #define FLASH_ATTR_ECC_ZERO_LINE_CAPABLE 0x00000010U
  The device is able to overwrite zero to a line.
• #define FLASH_ATTR_SUSPEND_ERASE_CAPABLE 0x00000020U
  The device is able to suspend erase operations.

Macro Functions (BaseFlash)

• #define getBaseFlash(ip) ((BaseFlash *)&(ip)->vmt)
  Instance getter.
• #define flashGetDescriptor(ip) (ip)->vmt->get_descriptor(ip)
  Gets the flash descriptor structure.
• #define flashRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)
  Read operation.
• #define flashProgram(ip, offset, n, pp) (ip)->vmt->program(ip, offset, n, pp)
  Program operation.
• #define flashStartEraseAll(ip) (ip)->vmt->start_erase_all(ip)
  Starts a whole-device erase operation.
• #define flashStartEraseSector(ip, sector) (ip)->vmt->start_erase_sector(ip, sector)
  Starts an sector erase operation.
• #define flashQueryErase(ip, msec) (ip)->vmt->query_erase(ip, msec)
  Queries the driver for erase operation progress.
• #define flashVerifyErase(ip, sector) (ip)->vmt->verify_erase(ip, sector)
  Returns the erase state of a sector.
• #define flashAcquireExclusive(ip) (ip)->vmt->acquire_exclusive(ip)
  Acquires exclusive access to flash.
• #define flashReleaseExclusive(ip) (ip)->vmt->release_exclusive(ip)
  Releases exclusive access to flash.

Typedefs

• typedef uint32_t flash_offset_t
  Type of a flash offset.
• typedef uint32_t flash_sector_t
  Type of a flash sector number.

Enumerations

• enum flash_state_t
  Driver state machine possible states.
• enum flash_error_t
  Type of a flash error code.
Functions

- `flash_error_t flashWaitErase (BaseFlash *devp)`
  Waits until the current erase operation is finished.
- `flash_offset_t flashGetSectorOffset (BaseFlash *devp, flash_sector_t sector)`
  Returns the offset of a sector.
- `uint32_t flashGetSectorSize (BaseFlash *devp, flash_sector_t sector)`
  Returns the size of a sector.
- `flash_sector_t flashGetOffsetSector (BaseFlash *devp, flash_offset_t offset)`
  Returns the sector of an offset.
- `void * flashGetOffsetAddress (BaseFlash *devp, flash_offset_t offset)`
  Get absolute address from offset.
- `flash_offset_t flashGetAddressOffset (BaseFlash *devp, void *addr)`
  Get offset from absolute address.

9.35.1 Detailed Description

Generic flash driver class header.

9.36 hal_gpt.c File Reference

GPT Driver code.

#include "hal.h"

Functions

- `void gptInit (void)`
  GPT Driver initialization.
- `void gptObjectInit (GPTDriver *gptp)`
  Initiates the standard part of a GPTDriver structure.
- `msg_t gptStart (GPTDriver *gptp, const GPTConfig *config)`
  Configures and activates the GPT peripheral.
- `void gptStop (GPTDriver *gptp)`
  Deactivates the GPT peripheral.
- `void gptChangeInterval (GPTDriver *gptp, gptcnt_t interval)`
  Changes the interval of GPT peripheral.
- `void gptStartContinuous (GPTDriver *gptp, gptcnt_t interval)`
  Starts the timer in continuous mode.
- `void gptStartContinuousI (GPTDriver *gptp, gptcnt_t interval)`
  Starts the timer in continuous mode.
- `void gptStartOneShot (GPTDriver *gptp, gptcnt_t interval)`
  Starts the timer in one shot mode.
- `void gptStartOneShotI (GPTDriver *gptp, gptcnt_t interval)`
  Starts the timer in one shot mode.
- `void gptStopTimer (GPTDriver *gptp)`
  Stops the timer.
- `void gptStopTimerI (GPTDriver *gptp)`
  Stops the timer.
- `void gptPolledDelay (GPTDriver *gptp, gptcnt_t interval)`
  Starts the timer in one shot mode and waits for completion.
9.36.1 Detailed Description

GPT Driver code.

9.37 hal_gpt.h File Reference

GPT Driver macros and structures.

#include "hal_gpt_lld.h"

Macros

- #define gptChangeInterval(gptp, interval)
  Changes the interval of GPT peripheral.
- #define gptGetIntervalX(gptp) gpt_lld_get_interval(gptp)
  Returns the interval of GPT peripheral.
- #define gptGetCounterX(gptp) gpt_lld_get_counter(gptp)
  Returns the counter value of GPT peripheral.
- #define _gpt_isr_invoke_cb(gptp)
  Common ISR code, GPT period event.

Typedefs

- typedef struct GPTDriver GPTDriver
  Type of a structure representing a GPT driver.
- typedef void(*gptcallback_t)(GPTDriver *gptp)
  GPT notification callback type.

Enumerations

- enum gptstate_t {
  GPT_UNINIT = 0, GPT_STOP = 1, GPT_READY = 2, GPT_CONTINUOUS = 3,
  GPT_ONESHOT = 4
}  
  Driver state machine possible states.
9.38 hal_gpt_lld.c File Reference

Functions

• void gptInit (void)
  GPT Driver initialization.

• void gptObjectInit (GPTDriver *gptp)
  Initializes the standard part of a GPTDriver structure.

• msg_t gptStart (GPTDriver *gptp, const GPTConfig *config)
  Configures and activates the GPT peripheral.

• void gptStop (GPTDriver *gptp)
  Deactivates the GPT peripheral.

• void gptStartContinuous (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in continuous mode.

• void gptStartContinuousI (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in continuous mode.

• void gptChangeInterval (GPTDriver *gptp, gptcnt_t interval)
  Changes the interval of GPT peripheral.

• void gptStartOneShot (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in one shot mode.

• void gptStartOneShotI (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in one shot mode.

• void gptStopTimer (GPTDriver *gptp)
  Stops the timer.

• void gptStopTimerI (GPTDriver *gptp)
  Stops the timer.

• void gptPolledDelay (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in one shot mode and waits for completion.

9.37.1 Detailed Description

GPT Driver macros and structures.

9.38 hal_gpt_lld.c File Reference

PLATFORM GPT subsystem low level driver source.

#include "hal.h"

Functions

• void gpt_lld_init (void)
  Low level GPT driver initialization.

• void gpt_lld_start (GPTDriver *gptp)
  Configures and activates the GPT peripheral.

• void gpt_lld_stop (GPTDriver *gptp)
  Deactivates the GPT peripheral.

• void gpt_lld_start_timer (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in continuous mode.

• void gpt_lld_stop_timer (GPTDriver *gptp)
  Stops the timer.

• void gpt_lld_polled_delay (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in one shot mode and waits for completion.
Variables

- GPTDriver GPTD1
  GPTD1 driver identifier.

9.38.1 Detailed Description

PLATFORM GPT subsystem low level driver source.

9.39 hal_gpt_lld.h File Reference

PLATFORM GPT subsystem low level driver header.

Data Structures

- struct GPTConfig
  Driver configuration structure.
- struct GPTDriver
  Structure representing a GPT driver.

Macros

- #define gpt_lld_change_interval(gptp, interval)
  Changes the interval of GPT peripheral.

PLATFORM configuration options

- #define PLATFORM_GPT_USE_GPT1 FALSE
  GPTD1 driver enable switch.

Typedefs

- typedef uint32_t gptfreq_t
  GPT frequency type.
- typedef uint16_t gptcnt_t
  GPT counter type.

Functions

- void gpt_lld_init (void)
  Low level GPT driver initialization.
- void gpt_lld_start (GPTDriver *gptp)
  Configures and activates the GPT peripheral.
- void gpt_lld_stop (GPTDriver *gptp)
  Deactivates the GPT peripheral.
- void gpt_lld_start_timer (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in continuous mode.
- void gpt_lld_stop_timer (GPTDriver *gptp)
  Stops the timer.
- void gpt_lld_polled_delay (GPTDriver *gptp, gptcnt_t interval)
  Starts the timer in one shot mode and waits for completion.
9.39.1 Detailed Description

PLATFORM GPT subsystem low level driver header.

9.40 hal_i2c.c File Reference

I2C Driver code.

#include "hal.h"

Functions

- void i2cInit (void)
  I2C Driver initialization.
- void i2cObjectInit (I2CDriver *i2cp)
  Initializes the standard part of a I2CDriver structure.
- msg_t i2cStart (I2CDriver *i2cp, const I2CConfig *config)
  Configures and activates the I2C peripheral.
- void i2cStop (I2CDriver *i2cp)
  Deactivates the I2C peripheral.
- i2cflags_t i2cGetErrors (I2CDriver *i2cp)
  Returns the errors mask associated to the previous operation.
- msg_t i2cMasterTransmitTimeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  Sends data via the I2C bus.
- msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  Receives data from the I2C bus.
- void i2cAcquireBus (I2CDriver *i2cp)
  Gains exclusive access to the I2C bus.
- void i2cReleaseBus (I2CDriver *i2cp)
  Releases exclusive access to the I2C bus.

9.40.1 Detailed Description

I2C Driver code.

9.41 hal_i2c.h File Reference

I2C Driver macros and structures.

#include "hal_i2c_lld.h"
Macros

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`
  Enables the mutual exclusion APIs on the I2C bus.

- `#define _i2c_wakeup_isr(i2cp)`
  Wakes up the waiting thread notifying no errors.

- `#define _i2c_wakeup_error_isr(i2cp)`
  Wakes up the waiting thread notifying errors.

- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)`
  Wrap `i2cMasterTransmit` function with `TIME_INFINITE` timeout.

- `#define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes)`
  Wrap `i2cMasterReceiveTimeout` function with `TIME_INFINITE` timeout.

I2C bus error conditions

- `#define I2C_NO_ERROR 0x00`
  No error.

- `#define I2C_BUS_ERROR 0x01`
  Bus Error.

- `#define I2C_ARBITRATION_LOST 0x02`
  Arbitration Lost.

- `#define I2C_ACK_FAILURE 0x04`
  Acknowledge Failure.

- `#define I2C_OVERRUN 0x08`
  Overrun/Underrun.

- `#define I2C_PEC_ERROR 0x10`
  PEC Error in reception.

- `#define I2C_TIMEOUT 0x20`
  Hardware timeout.

- `#define I2C_SMB_ALERT 0x40`
  SMBus Alert.

Enumerations

- `enum i2cstate_t {
  I2C_UNINIT = 0, I2C_STOP = 1, I2C_READY = 2, I2C_ACTIVE_TX = 3,
  I2C_ACTIVE_RX = 4, I2C_LOCKED = 5 }
  Driver state machine possible states.`
9.42 hal_i2c_lld.c File Reference

Platform I2C subsystem low level driver source.

```c
#include "hal.h"
```

Functions

- `void i2c_lld_init (void)`
  - **Description**: Low level I2C driver initialization.
  - **Parameters**: `void` (empty)
- `void i2c_lld_start (I2CDriver *i2cp)`
  - **Description**: Configures and activates the I2C peripheral.
  - **Parameters**: I2CDriver structure pointer (`*i2cp`)
- `msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)`
  - **Description**: Receives data via the I2C bus as master.
  - **Parameters**: I2CDriver structure pointer (`*i2cp`), I2C address (`addr`), receive buffer pointer (`*rxbuf`), receive buffer size (`rxbytes`), timeout (`sysinterval_t timeout`)
- `msg_t i2c_lld_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)`
  - **Description**: Transmits data via the I2C bus as master.
  - **Parameters**: I2CDriver structure pointer (`*i2cp`), I2C address (`addr`), transmission buffer pointer (`*txbuf`), transmission buffer size (`txbytes`), receive buffer pointer (`*rxbuf`), receive buffer size (`rxbytes`), timeout (`sysinterval_t timeout`)

---

ChibiOS/HAL
Variables

- **I2CDriver I2CD1**
  
  I2C1 driver identifier.

### 9.42.1 Detailed Description

PLATFORM I2C subsystem low level driver source.

### 9.43 hal_i2c_lld.h File Reference

PLATFORM I2C subsystem low level driver header.

### Data Structures

- **struct hal_i2c_config**
  
  I2C driver configuration structure.

- **struct hal_i2c_driver**
  
  Structure representing an I2C driver.

### Macros

- **#define i2c_lld_get_errors(i2cp) ((i2cp)->errors)**

  Get errors from I2C driver.

### PLATFORM configuration options

- **#define PLATFORM_I2C_USE_I2C1 FALSE**

  I2C1 driver enable switch.

### Typedefs

- **typedef uint16_t i2caddr_t**

  Type representing an I2C address.

- **typedef uint32_t i2cflags_t**

  Type of I2C Driver condition flags.

- **typedef struct hal_i2c_config I2CConfig**

  Type of a structure representing an I2C configuration.

- **typedef struct hal_i2c_driver I2CDriver**

  Type of a structure representing an I2C driver.
Functions

- void i2c_lld_init (void)
  Low level I2C driver initialization.
- void i2c_lld_start (I2CDriver *i2cp)
  Configures and activates the I2C peripheral.
- void i2c_lld_stop (I2CDriver *i2cp)
  Deactivates the I2C peripheral.
- msg_t i2c_lld_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  Transmits data via the I2C bus as master.
- msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  Receives data via the I2C bus as master.

9.43.1 Detailed Description

PLATFORM I2C subsystem low level driver header.

9.44 hal_i2s.c File Reference

I2S Driver code.

#include "hal.h"

Functions

- void i2sInit (void)
  I2S Driver initialization.
- void i2sObjectInit (I2SDriver *i2sp)
  Initializes the standard part of a I2SDriver structure.
- msg_t i2sStart (I2SDriver *i2sp, const I2SConfig *config)
  Configures and activates the I2S peripheral.
- void i2sStop (I2SDriver *i2sp)
  Deactivates the I2S peripheral.
- void i2sStartExchange (I2SDriver *i2sp)
  Starts a I2S data exchange.
- void i2sStopExchange (I2SDriver *i2sp)
  Stops the ongoing data exchange.

9.44.1 Detailed Description

I2S Driver code.
# I2S Driver Macros and Structures

I2S Driver macros and structures.

```c
#include "hal_i2s_lld.h"
```

## Data Structures

- **struct hal_i2s_driver**
  - Structure representing an I2S driver.
- **struct hal_i2s_config**
  - Driver configuration structure.

## Macros

### I2S modes

- `#define I2S_MODE_SLAVE 0`  
- `#define I2S_MODE_MASTER 1`

### Macro Functions

- `#define i2sIsBufferComplete(i2sp) ((bool)((i2sp)->state == I2S_COMPLETE))`
  - Buffer state.
- `#define i2sStartExchangeI(i2sp)`  
  - Starts a I2S data exchange.
- `#define i2sStopExchangeI(i2sp)`  
  - Stops the ongoing data exchange.
- `#define _i2s_isr_half_code(i2sp)`  
  - Common ISR code, half buffer event.
- `#define _i2s_isr_full_code(i2sp)`  
  - Common ISR code.

## Typedefs

- `typedef struct hal_i2s_driver I2SDriver`
  - Type of a structure representing an I2S driver.
- `typedef struct hal_i2s_config I2SConfig`
  - Type of a structure representing an I2S driver configuration.
- `typedef void(*i2scallback_t)(I2SDriver *i2sp)`
  - I2S notification callback type.

## Enumerations

```c
enum i2sstate_t {
  I2S_UNINIT = 0, I2S_STOP = 1, I2S_READY = 2, I2S_ACTIVE = 3, I2S_COMPLETE = 4
}
```

- Driver state machine possible states.
Functions

• void i2sInit (void)
  I2S Driver initialization.
• void i2sObjectInit (I2SDriver *i2sp)
  Initializes the standard part of a I2SDriver structure.
• msg_t i2sStart (I2SDriver *i2sp, const I2SConfig *config)
  Configures and activates the I2S peripheral.
• void i2sStop (I2SDriver *i2sp)
  Deactivates the I2S peripheral.
• void i2sStartExchange (I2SDriver *i2sp)
  Starts a I2S data exchange.
• void i2sStopExchange (I2SDriver *i2sp)
  Stops the ongoing data exchange.

9.45.1 Detailed Description

I2S Driver macros and structures.

9.46 hal_i2s_lld.c File Reference

PLATFORM I2S subsystem low level driver source.

#include "hal.h"

Functions

• void i2s_lld_init (void)
  Low level I2S driver initialization.
• void i2s_lld_start (I2SDriver *i2sp)
  Configures and activates the I2S peripheral.
• void i2s_lld_stop (I2SDriver *i2sp)
  Deactivates the I2S peripheral.
• void i2s_lld_start_exchange (I2SDriver *i2sp)
  Starts a I2S data exchange.
• void i2s_lld_stop_exchange (I2SDriver *i2sp)
  Stops the ongoing data exchange.

Variables

• I2SDriver I2S1
  I2S2 driver identifier.

9.46.1 Detailed Description

PLATFORM I2S subsystem low level driver source.
9.47 hal_i2s_lld.h File Reference

PLATFORM I2S subsystem low level driver header.

**Macros**

- #define i2s_lld_driver_fields
  
  *Low level fields of the I2S driver structure.*

- #define i2s_lld_config_fields
  
  *Low level fields of the I2S configuration structure.*

**PLATFORM configuration options**

- #define PLATFORM_I2S_USE_I2S1 FALSE
  
  I2SD1 driver enable switch.

**Functions**

- void i2s_lld_init (void)
  
  *Low level I2S driver initialization.*

- void i2s_lld_start (I2SDriver *i2sp)
  
  *Configures and activates the I2S peripheral.*

- void i2s_lld_stop (I2SDriver *i2sp)
  
  *Deactivates the I2S peripheral.*

- void i2s_lld_start_exchange (I2SDriver *i2sp)
  
  *Starts an I2S data exchange.*

- void i2s_lld_stop_exchange (I2SDriver *i2sp)
  
  *Stops the ongoing data exchange.*

9.47.1 Detailed Description

PLATFORM I2S subsystem low level driver header.

9.48 hal_icu.c File Reference

ICU Driver code.

#include "hal.h"
Functions

- `void icuInit (void)`
  ICU Driver initialization.
- `void icuObjectInit (ICUDriver *icup)`
  Initializes the standard part of a `ICUDriver` structure.
- `msg_t icuStart (ICUDriver *icup, const ICUConfig *config)`
  Configures and activates the ICU peripheral.
- `void icuStop (ICUDriver *icup)`
  Deactivates the ICU peripheral.
- `void icuStartCapture (ICUDriver *icup)`
  Starts the input capture.
- `bool icuWaitCapture (ICUDriver *icup)`
  Waits for a completed capture.
- `void icuStopCapture (ICUDriver *icup)`
  Stops the input capture.
- `void icuEnableNotifications (ICUDriver *icup)`
  Enables notifications.
- `void icuDisableNotifications (ICUDriver *icup)`
  Disables notifications.

9.48.1 Detailed Description

ICU Driver code.

9.49 hal_icu.h File Reference

ICU Driver macros and structures.

```c
#include "hal_icu_lld.h"
```

Macros

Macro Functions

- `#define icuStartCapture(icup)`
  Starts the input capture.
- `#define icuStopCapture(icup)`
  Stops the input capture.
- `#define icuEnableNotifications(icup) icu_lld_enable_notifications(icup)`
  Enables notifications.
- `#define icuDisableNotifications(icup) icu_lld_disable_notifications(icup)`
  Disables notifications.
- `#define icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)`
  Check on notifications status.
- `#define icuGetWidthX(icup) icu_lld_get_width(icup)`
  Returns the width of the latest pulse.
- `#define icuGetPeriodX(icup) icu_lld_get_period(icup)`
  Returns the width of the latest cycle.
Low level driver helper macros

- \#define _icu_isr_invoke_width_cb(icup)
  Common ISR code, ICU width event.
- \#define _icu_isr_invoke_period_cb(icup)
  Common ISR code, ICU period event.
- \#define _icu_isr_invoke_overflow_cb(icup)
  Common ISR code, ICU timer overflow event.

Typedefs

- typedef struct ICUDriver ICUDriver
  Type of a structure representing an ICU driver.
- typedef void(icucallback_t)(ICUDriver *icup)
  ICU notification callback type.

Enumerations

- enum icustate_t {
  ICU_UNINIT = 0, ICU_STOP = 1, ICU_READY = 2, ICU_WAITING = 3, 
  ICU_ACTIVE = 4 }
  Driver state machine possible states.

Functions

- void icuInit (void)
  ICU Driver initialization.
- void icuObjectInit (ICUDriver *icup)
  Initializes the standard part of a ICUDriver structure.
- msg_t icuStart (ICUDriver *icup, const ICUConfig *config)
  Configures and activates the ICU peripheral.
- void icuStop (ICUDriver *icup)
  Deactivates the ICU peripheral.
- void icuStartCapture (ICUDriver *icup)
  Starts the input capture.
- bool icuWaitCapture (ICUDriver *icup)
  Waits for a completed capture.
- void icuStopCapture (ICUDriver *icup)
  Stops the input capture.
- void icuEnableNotifications (ICUDriver *icup)
  Enables notifications.
- void icuDisableNotifications (ICUDriver *icup)
  Disables notifications.

9.49.1 Detailed Description

ICU Driver macros and structures.
Functions

• void icu_lld_init (void)
  
  Low level ICU driver initialization.
• void icu_lld_start (ICUDriver *icup)
  
  Configures and activates the ICU peripheral.
• void icu_lld_stop (ICUDriver *icup)
  
  Deactivates the ICU peripheral.
• void icu_lld_start_capture (ICUDriver *icup)
  
  Starts the input capture.
• bool icu_lld_wait_capture (ICUDriver *icup)
  
  Waits for a completed capture.
• void icu_lld_stop_capture (ICUDriver *icup)
  
  Stops the input capture.
• void icu_lld_enable_notifications (ICUDriver *icup)
  
  Enables notifications.
• void icu_lld_disable_notifications (ICUDriver *icup)
  
  Disables notifications.

Variables

• ICUDriver ICUD1
  
  ICUD1 driver identifier.

9.50.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.51 hal_icu_lld.h File Reference

PLATFORM ICU subsystem low level driver header.

Data Structures

• struct ICUConfig
  
  Driver configuration structure.
• struct ICUDriver
  
  Structure representing an ICU driver.
Macros

• #define icu_lld_get_width(icup) 0
  Returns the width of the latest pulse.
• #define icu_lld_get_period(icup) 0
  Returns the width of the latest cycle.
• #define icu_lld_are_notifications_enabled(icup) false
  Check on notifications status.

PLATFORM configuration options

• #define PLATFORM_ICU_USE_ICU1 FALSE
  ICU1 driver enable switch.

Typedefs

• typedef uint32_t icufreq_t
  ICU frequency type.
• typedef uint32_t icucnt_t
  ICU counter type.

Enumerations

• enum icumode_t { ICU_INPUT_ACTIVE_HIGH = 0, ICU_INPUT_ACTIVE_LOW = 1 }
  ICU driver mode.

Functions

• void icu_lld_init (void)
  Low level ICU driver initialization.
• void icu_lld_start (ICUDriver *icup)
  Configures and activates the ICU peripheral.
• void icu_lld_stop (ICUDriver *icup)
  Deactivates the ICU peripheral.
• void icu_lld_start_capture (ICUDriver *icup)
  Starts the input capture.
• bool icu_lld_wait_capture (ICUDriver *icup)
  Waits for a completed capture.
• void icu_lld_stop_capture (ICUDriver *icup)
  Stops the input capture.
• void icu_lld_enable_notifications (ICUDriver *icup)
  Enables notifications.
• void icu_lld_disable_notifications (ICUDriver *icup)
  Disables notifications.

9.51.1 Detailed Description

PLATFORM ICU subsystem low level driver header.
9.52 hal_ioblock.h File Reference

I/O block devices access.

Data Structures

- struct BlockDeviceInfo
  
  Block device info.

- struct BaseBlockDeviceVMT
  
  BaseBlockDevice virtual methods table.

- struct BaseBlockDevice
  
  Base block device class.

Macros

- #define _base_block_device_methods
  
  BaseBlockDevice specific methods.

- #define _base_block_device_data
  
  BaseBlockDevice specific data.

Macro Functions (BaseBlockDevice)

- #define blkGetDriverState(ip) ((ip)->state)
  
  Returns the driver state.

- #define blkIsTransferring(ip)
  
  Determines if the device is transferring data.

- #define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))
  
  Returns the media insertion status.

- #define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))
  
  Returns the media write protection status.

- #define blkConnect(ip) ((ip)->vmt->connect(ip))
  
  Performs the initialization procedure on the block device.

- #define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))
  
  Terminates operations on the block device.

- #define blkRead(ip, startblk, buf, n) ((ip)->vmt->read(ip, startblk, buf, n))
  
  Reads one or more blocks.

- #define blkWrite(ip, startblk, buf, n) ((ip)->vmt->write(ip, startblk, buf, n))
  
  Writes one or more blocks.

- #define blkSync(ip) ((ip)->vmt->sync(ip))
  
  Ensures write synchronization.

- #define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))
  
  Returns a media information structure.

Enumerations

- enum blkstate_t {
    BLK_UNINIT = 0, BLK_STOP = 1, BLK_ACTIVE = 2, BLK_CONNECTING = 3,
    BLK_DISCONNECTING = 4, BLK_READY = 5, BLK_READING = 6, BLK_WRITING = 7,
    BLKSYNCING = 8 }

  Driver state machine possible states.
9.52.1 Detailed Description

I/O block devices access.

This header defines an abstract interface useful to access generic I/O block devices in a standardized way.

9.53 hal_lld.c File Reference

PLATFORM HAL subsystem low level driver source.

#include "hal.h"

Functions

- void hal_lld_init (void)
  Low level HAL driver initialization.
- bool hal_lld_clock_switch_mode (const halclkcfg_t *ccp)
  Switches to a different clock configuration.
- halfreq_t hal_lld_get_clock_point (halclkpt_t clkpt)
  Returns the frequency of a clock point in Hz.

9.53.1 Detailed Description

PLATFORM HAL subsystem low level driver source.

9.54 hal_lld.h File Reference

PLATFORM HAL subsystem low level driver header.

Data Structures

- struct halclkcfg_t
  Type of a clock configuration structure.

Macros

- #define HAL_LLD_SELECT_SPI_V2 TRUE
  Requires use of SPIv2 driver model.
- #define HAL_LLD_USE_CLOCK_MANAGEMENT
  Specifies implementation of dynamic clock management.

Platform identification macros

- #define PLATFORM_NAME "templates"
9.55 hal_mac.c File Reference

Typedefs

- **typedef unsigned halclkpt_t**
  
  Type of a clock point identifier.

- **typedef uint32_t halfreq_t**
  
  Type of a clock point frequency in Hz.

Functions

- **void hal_lld_init (void)**
  
  Low level HAL driver initialization.

- **bool hal_lld_clock_switch_mode (const halclkcfg_t *ccp)**
  
  Switches to a different clock configuration.

- **halfreq_t hal_lld_get_clock_point (halclkpt_t clkpt)**
  
  Returns the frequency of a clock point in Hz.

9.54.1 Detailed Description

PLATFORM HAL subsystem low level driver header.

9.55 hal_mac.c File Reference

MAC Driver code.

#include "hal.h"

Functions

- **void macInit (void)**
  
  MAC Driver initialization.

- **void macObjectInit (MACDriver *macp)**
  
  Initialize the standard part of a MACDriver structure.

- **msg_t macStart (MACDriver *macp, const MACConfig *config)**
  
  Configures and activates the MAC peripheral.

- **void macStop (MACDriver *macp)**
  
  Deactivates the MAC peripheral.

- **eventflags_t macGetAndClearEventsI (MACDriver *macp)**
  
  Get and clears SIO event flags.

- **msg_t macWaitTransmitDescriptor (MACDriver *macp, MACTransmitDescriptor *tdp, sysinterval_t timeout)**
  
  Allocates a transmission descriptor.

- **msg_t macWaitReceiveDescriptor (MACDriver *macp, MACReceiveDescriptor *rdp, sysinterval_t timeout)**
  
  Waits for a received frame.

- **bool macPollLinkStatus (MACDriver *macp)**
  
  Updates and returns the link status.
9.55.1 Detailed Description

MAC Driver code.

9.56 hal_mac.h File Reference

MAC Driver macros and structures.

#include "hal_mac_lld.h"

Data Structures

• struct hal_mac_config
  Driver configuration structure.
• struct hal_mac_driver
  Structure representing a MAC driver.
• struct hal_mac_transmit_descriptor
  Structure representing a MAC transmit descriptor.
• struct hal_mac_receive_descriptor
  Structure representing a MAC receive descriptor.

Macros

MAC event flags

• #define MAC_FLAGS_TX (1U << 0)
• #define MAC_FLAGS_RX (1U << 1)

MAC configuration options

• #define MAC_USE_ZERO_COPY FALSE
  Enables an event sources for incoming packets.
• #define MAC_USE_EVENTS TRUE
  Enables an event sources for incoming packets.

Low level driver helper macros

• #define __mac_callback(macp)
  MAC callback.
• #define __mac_tx_wakeup(macp)
  MAC TX wakeup and event.
• #define __mac_rx_wakeup(macp)
  MAC RX wakeup and event.

Macro Functions

• #define macSetCallbackX(macp, f) (siop)->cb = (f)
  Associates a callback to the MAC instance.
• #define macGetEventSource(macp) (&(macp)->es)
Returns the driver events source.

- `#define macGetTransmitDescriptorX(macp, tdp) mac_lld_get_transmit_descriptor(macp, tdp)`
  Returns a transmission descriptor.
- `#define macReleaseTransmitDescriptorX(tdp) mac_lld_release_transmit_descriptor(tdp)`
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- `#define macGetReceiveDescriptorX(macp, rdp) mac_lld_get_receive_descriptor(macp, rdp)`
  Returns a receive descriptor.
- `#define macReleaseReceiveDescriptorX(rdp) mac_lld_release_receive_descriptor(rdp)`
  Releases a receive descriptor.
- `#define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)`
  Writes to a transmit descriptor's stream.
- `#define macReadReceiveDescriptor(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)`
  Reads from a receive descriptor's stream.
- `#define macGetNextTransmitBuffer(tdp, size, sizep) mac_lld_get_next_transmit_buffer(tdp, size, sizep)`
  Returns a pointer to the next transmit buffer in the descriptor chain.
- `#define macGetNextReceiveBuffer(rdp, sizep) mac_lld_get_next_receive_buffer(rdp, sizep)`
  Returns a pointer to the next receive buffer in the descriptor chain.

### Typedefs

- `typedef struct hal_mac_driver MACDriver`  
  Type of a structure representing a MAC driver.
- `typedef struct hal_mac_config MACConfig`  
  Type of structure representing a MAC configuration.
- `typedef struct hal_mac_transmit_descriptor MACTransmitDescriptor`  
  Type of structure representing a MAC transmit descriptor.
- `typedef struct hal_mac_receive_descriptor MACReceiveDescriptor`  
  Type of structure representing a MAC receive descriptor.
- `typedef void(*maccb_t)(MACDriver *macp)`  
  Generic ETH notification callback type.

### Enumerations

- `enum macstate_t { MAC_UNINIT = 0, MAC_STOP = 1, MAC_ACTIVE = 2 }`  
  Driver state machine possible states.

### Functions

- `void macInit (void)`  
  MAC Driver initialization.
- `void macObjectInit (MACDriver *macp)`  
  Initialize the standard part of a MACDriver structure.
- `msg_t macStart (MACDriver *macp, const MACConfig *config)`  
  Configures and activates the MAC peripheral.
- `void macStop (MACDriver *macp)`  
  Deactivates the MAC peripheral.
- `eventflags_t macGetAndClearEventsI (MACDriver *macp)`  
  Get and clears SIO event flags.
- `msg_t macWaitTransmitDescriptor (MACDriver *macp, MACTransmitDescriptor *tdp, sysinterval_t timeout)`  
  Allocates a transmission descriptor.
- `msg_t macWaitReceiveDescriptor (MACDriver *macp, MACReceiveDescriptor *rdp, sysinterval_t timeout)`  
  Waits for a received frame.
- `bool macPollLinkStatus (MACDriver *macp)`  
  Updates and returns the link status.
9.56.1 Detailed Description

MAC Driver macros and structures.

9.57 hal_mac_lld.c File Reference

PLATFORM MAC subsystem low level driver source.

```
#include <string.h>
#include "hal.h"
#include "hal_mii.h"
```

Functions

- `void mac_lld_init (void)`
  Low level MAC initialization.
- `void mac_lld_start (MACDriver *macp)`
  Configures and activates the MAC peripheral.
- `void mac_lld_stop (MACDriver *macp)`
  Deactivates the MAC peripheral.
- `msg_t mac_lld_get_transmit_descriptor (MACDriver *macp, MACTransmitDescriptor *tdp)`
  Returns a transmission descriptor.
- `void mac_lld_release_transmit_descriptor (MACTransmitDescriptor *tdp)`
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- `msg_t mac_lld_get_receive_descriptor (MACDriver *macp, MACReceiveDescriptor *rdp)`
  Returns a receive descriptor.
- `void mac_lld_release_receive_descriptor (MACReceiveDescriptor *rdp)`
  Releases a receive descriptor.
- `bool mac_lld_poll_link_status (MACDriver *macp)`
  Updates and returns the link status.
- `size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)`
  Writes to a transmit descriptor's stream.
- `size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)`
  Reads from a receive descriptor's stream.
- `uint8_t * mac_lld_get_next_transmit_buffer (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)`
  Returns a pointer to the next transmit buffer in the descriptor chain.
- `const uint8_t * mac_lld_get_next_receive_buffer (MACReceiveDescriptor *rdp, size_t size, size_t *sizep)`
  Returns a pointer to the next receive buffer in the descriptor chain.

Variables

- `MACDriver ETHD1`
  MAC1 driver identifier.

9.57.1 Detailed Description

PLATFORM MAC subsystem low level driver source.
PLATEFORM MAC subsystem low level driver header.

**Macros**

- ```c
#define MAC_SUPPORTS_ZERO_COPY TRUE
```
  This implementation supports the zero-copy mode API.
- ```c
#define mac_lld_driver_fields
```
  Low level fields of the MAC driver structure.
- ```c
#define mac_lld_config_fields
```
  Low level fields of the MAC configuration structure.
- ```c
#define mac_lld_transmit_descriptor_fields
```
  Low level fields of the MAC transmit descriptor structure.
- ```c
#define mac_lld_receive_descriptor_fields
```
  Low level fields of the MAC receive descriptor structure.

**PLATFORM configuration options**

- ```c
#define PLATFORM_MAC_USE_MAC1 FALSE
```
  MAC driver enable switch.

**Functions**

- ```c
void mac_lld_init (void)
```
  Low level MAC initialization.
- ```c
void mac_lld_start (MACDriver *macp)
```
  Configures and activates the MAC peripheral.
- ```c
void mac_lld_stop (MACDriver *macp)
```
  Deactivates the MAC peripheral.
- ```c
msg_t mac_lld_get_transmit_descriptor (MACDriver *macp, MACTransmitDescriptor *tdp)
```
  Returns a transmission descriptor.
- ```c
void mac_lld_release_transmit_descriptor (MACTransmitDescriptor *tdp)
```
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- ```c
msg_t mac_lld_get_receive_descriptor (MACDriver *macp, MACReceiveDescriptor *rdp)
```
  Returns a receive descriptor.
- ```c
void mac_lld_release_receive_descriptor (MACReceiveDescriptor *rdp)
```
  Releases a receive descriptor.
- ```c
bool mac_lld_poll_link_status (MACDriver *macp)
```
  Updates and returns the link status.
- ```c
size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)
```
  Writes to a transmit descriptor's stream.
- ```c
size_t mac_lld_readReceive_descriptor (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)
```
  Reads from a receive descriptor's stream.
- ```c
uint8_t * mac_lld_get_next_transmit_buffer (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)
```
  Returns a pointer to the next transmit buffer in the descriptor chain.
- ```c
const uint8_t * mac_lld_get_next_receive_buffer (MACReceiveDescriptor *rdp, size_t *sizep)
```
  Returns a pointer to the next receive buffer in the descriptor chain.
9.58.1 Detailed Description

PLATFORM MAC subsystem low level driver header.

9.59 hal_mfs.c File Reference

Managed Flash Storage module code.

#include <string.h>
#include "hal.h"
#include "hal_mfs.h"

Macros

- #define ALIGNED_REC_SIZE(n) (flash_offset_t)MFS_ALIGN_NEXT(sizeof (mfs_data_header_t) + (size_t)(n))
  Data record size aligned.
- #define ALIGNED_DHDR_SIZE ALIGNED_REC_SIZE(0)
  Data record header size aligned.
- #define ALIGNED_SIZEOF(t) (((sizeof (t) - 1U) | MFS_ALIGN_MASK) + 1U)
  Aligned size of a type.
- #define PAIR(a, b) (((unsigned)(a)<<2U) | (unsigned)(b))
  Combines two values (0..3) in one (0..15).
- #define RET_ON_ERROR(err)
  Error check helper.

Functions

- static mfs_error_t mfs_flash_read (MFSDriver *mfsp, flash_offset_t offset, size_t n, uint8_t *rp)
  Flash read.
- static mfs_error_t mfs_flash_write (MFSDriver *mfsp, flash_offset_t offset, size_t n, const uint8_t *wp)
  Flash write.
- static mfs_error_t mfs_flash_copy (MFSDriver *mfsp, flash_offset_t doffset, flash_offset_t soffset, uint32_t n)
  Flash copy.
- static mfs_error_t mfs_bank_erase (MFSDriver *mfsp, mfs_bank_t bank)
  Erases and verifies all sectors belonging to a bank.
- static mfs_error_t mfs_bank_verify_erase (MFSDriver *mfsp, mfs_bank_t bank)
  Erases and verifies all sectors belonging to a bank.
- static mfs_error_t mfs_bank_write_header (MFSDriver *mfsp, mfs_bank_t bank, uint32_t cnt)
  Writes the validation header in a bank.
- static mfs_bank_state_t mfs_bank_check_header (MFSDriver *mfsp)
  Checks integrity of the header in the shared buffer.
- static mfs_error_t mfs_bank_scan_records (MFSDriver *mfsp, mfs_bank_t bank, bool wflagp)
  Scans blocks searching for records.
- static mfs_error_t mfs_bank_get_state (MFSDriver *mfsp, mfs_bank_t bank, mfs_bank_state_t *statep, uint32_t *cntp)
  Determines the state of a bank.
9.59.1 Detailed Description

Managed Flash Storage module code.

This module manages a flash partition as a generic storage where arbitrary data records can be created, updated, deleted and retrieved.
A managed partition is composed of two banks of equal size, a bank is composed of one or more erasable sectors, a sector is divided in writable pages.
The module handles flash wear leveling and recovery of damaged banks (where possible) caused by power loss during operations. Both operations are transparent to the user.

9.60 hal_mfs.h File Reference

Managed Flash Storage module header.

#include "hal_flash.h"
Data Structures

- union mfs_bank_header_t
  Type of a bank header.
- union mfs_data_header_t
  Type of a data block header.
- struct MFSCfg
  Type of a MFS configuration structure.
- struct mfs_transaction_op_t
  Type of a buffered write/erase operation within a transaction.
- union mfs_nocache_buffer
  Type of a non-cacheable MFS buffer.
- struct MFSDriver
  Type of an MFS instance.

Macros

Configuration options

- #define MFS_CFG_MAX_RECORDS 32
  Maximum number of indexed records in the managed storage.
- #define MFS_CFG_MAX_REPAIR_ATTEMPTS 3
  Maximum number of repair attempts on partition mount.
- #define MFS_CFG_WRITE_VERIFY TRUE
  Verify written data.
- #define MFS_CFG_STRONG_CHECKING TRUE
  Enables a stronger and slower check procedure on mount.
- #define MFS_CFG_BUFFER_SIZE 32
  Size of the buffer used for data copying.
- #define MFS_CFG_MEMORY_ALIGNMENT 2
  Enforced memory alignment.
- #define MFS_CFG_TRANSACTION_MAX 16
  Maximum number of objects writable in a single transaction.

Error codes handling macros

- #define MFS_IS_ERROR(err) ((err) < MFS_NO_ERROR)
- #define MFS_IS_WARNING(err) ((err) > MFS_NO_ERROR)

Alignment macros

- #define MFS_ALIGN_MASK ((uint32_t)MFS_CFG_MEMORY_ALIGNMENT - 1U)
- #define MFS_IS_ALIGNED(v) (((uint32_t)(v) & MFS_ALIGN_MASK) == 0U)
- #define MFS_ALIGN_PREV(v) ((uint32_t)(v) & ~MFS_ALIGN_MASK)
- #define MFS_ALIGN_NEXT(v)

Typedefs

- typedef uint32_t mfs_id_t
  Type of a record identifier.
- typedef union mfs_nocache_buffer mfs_nocache_buffer_t
  Type of an non-cacheable MFS buffer.
Enumerations

- enum mfs_bank_t
  Type of a flash bank.
- enum mfs_state_t
  Type of driver state machine states.
- enum mfs_error_t
  Type of an MFS error code.
- enum mfs_bank_state_t
  Type of a bank state assessment.

Functions

- void mfsObjectInit (MFSDriver *mfsp, mfs_nocache_buffer_t *ncbuf)
  Initializes an instance.
- mfs_error_t mfsStart (MFSDriver *mfsp, const MFSCfg *config)
  Configures and activates a MFS driver.
- void mfsStop (MFSDriver *mfsp)
  Deactivates a MFS driver.
- mfs_error_t mfsErase (MFSDriver *mfsp)
  Destroys the state of the managed storage by erasing the flash.
- mfs_error_t mfsReadRecord (MFSDriver *mfsp, mfs_id_t id, size_t *np, uint8_t *buffer)
  Retrieves and reads a data record.
- mfs_error_t mfsWriteRecord (MFSDriver *mfsp, mfs_id_t id, size_t n, const uint8_t *buffer)
  Creates or updates a data record.
- mfs_error_t mfsEraseRecord (MFSDriver *mfsp, mfs_id_t id)
  Erases a data record.
- mfs_error_t mfsPerformGarbageCollection (MFSDriver *mfsp)
  Enforces a garbage collection operation.
- mfs_error_t mfsStartTransaction (MFSDriver *mfsp, size_t size)
  Puts the driver in transaction mode.
- mfs_error_t mfsCommitTransaction (MFSDriver *mfsp)
  A transaction is committed and finalized atomically.
- mfs_error_t mfsRollbackTransaction (MFSDriver *mfsp)
  A transaction is rolled back atomically.

9.60.1 Detailed Description

Managed Flash Storage module header.

9.61 hal_mii.h File Reference

MII macros and structures.
Macros

**Generic MII registers**

- `#define MII_BMCR 0x00`
- `#define MII_BMSR 0x01`
- `#define MII_PHYSID1 0x02`
- `#define MII_PHYSID2 0x03`
- `#define MII_ADVERTISE 0x04`
- `#define MII_LPA 0x05`
- `#define MII_EXPANSION 0x06`
- `#define MII_ANNPTR 0x07`
- `#define MII_CTRL1000 0x09`
- `#define MII_STAT1000 0x0a`
- `#define MII_ESTATUS 0x0f`
- `#define MII_PHYSTS 0x10`
- `#define MII_MICR 0x11`
- `#define MII_DCOUNT 0x12`
- `#define MII_FCSCOUNT 0x13`
- `#define MII_NWAYTEST 0x14`
- `#define MII_RERRCOUNTER 0x15`
- `#define MII_SREVISION 0x16`
- `#define MII_RESV1 0x17`
- `#define MII_LBRError 0x18`
- `#define MII_PHYADDR 0x19`
- `#define MII_RESV2 0x1a`
- `#define MII_TPISTATUS 0x1b`
- `#define MII_NCONFIG 0x1c`

**Basic mode control register**

- `#define BMCR_RESV 0x007f`
- `#define BMCR_CTST 0x0080`
- `#define BMCR_FULLDPLX 0x0100`
- `#define BMCR_ANRESTART 0x0200`
- `#define BMCR_ISOLATE 0x0400`
- `#define BMCR_PDOWN 0x0800`
- `#define BMCR_ANENABLE 0x1000`
- `#define BMCR_SPEED100 0x2000`
- `#define BMCR_LOOPBACK 0x4000`
- `#define BMCR_RESET 0x8000`

**Basic mode status register**

- `#define BMSR_ERCAP 0x0001`
- `#define BMSR_JCD 0x0002`
- `#define BMSR_LSTATUS 0x0004`
- `#define BMSR_ANEGCAPABLE 0x0008`
- `#define BMSR_RFAULT 0x0010`
- `#define BMSR_ANEGCOMPLETE 0x0020`
- `#define BMSR_MFPRESUPPCAP 0x0040`
- `#define BMSR_RESV 0x0780`
- `#define BMSR_10HALF 0x0800`
- `#define BMSR_10FULL 0x1000`
- `#define BMSR_100HALF 0x2000`
- `#define BMSR_100FULL 0x4000`
- `#define BMSR_100BASE4 0x8000`

**Advertisement control register**

- `#define ADVERTISE_SLCT 0x001f`
• #define ADVERTISE_CSMA 0x0001
• #define ADVERTISE_10HALF 0x0020
• #define ADVERTISE_10FULL 0x0040
• #define ADVERTISE_100HALF 0x0080
• #define ADVERTISE_100FULL 0x0100
• #define ADVERTISE_100BASE4 0x0200
• #define ADVERTISE_PAUSE_CAP 0x0400
• #define ADVERTISE_PAUSE_ASYM 0x0800
• #define ADVERTISE_RESV 0x1000
• #define ADVERTISE_RFAULT 0x2000
• #define ADVERTISE_LPACK 0x4000
• #define ADVERTISE_NPAGE 0x8000
• #define ADVERTISE_FULL
• #define ADVERTISE_ALL

Link partner ability register

• #define LPA_SLCT 0x001f
• #define LPA_10HALF 0x0020
• #define LPA_10FULL 0x0040
• #define LPA_100HALF 0x0080
• #define LPA_100FULL 0x0100
• #define LPA_100BASE4 0x0200
• #define LPA_PAUSE_CAP 0x0400
• #define LPA_PAUSE_ASYM 0x0800
• #define LPA_RESV 0x1000
• #define LPA_RFAULT 0x2000
• #define LPA_LPACK 0x4000
• #define LPA_NPAGE 0x8000
• #define LPA_DUPLEX (LPA_10FULL | LPA_100FULL)
• #define LPA_100 (LPA_100FULL | LPA_100HALF | LPA_100BASE4)

Expansion register for auto-negotiation

• #define EXPANSION_NWAY 0x0001
• #define EXPANSION_LCWP 0x0002
• #define EXPANSION_ENABLENPAGE 0x0004
• #define EXPANSION_NPCAPABLE 0x0008
• #define EXPANSION_MFAULTS 0x0010
• #define EXPANSION_RESV 0xffe0

N-way test register

• #define NWAYTEST_RESV1 0x00ff
• #define NWAYTEST_LOOPBACK 0x0100
• #define NWAYTEST_RESV2 0xfee0

PHY identifiers

• #define MII_DM9161_ID 0x0181b8a0
• #define MII_AM79C875_ID 0x00225540
• #define MII_KSZ8081_ID 0x00221560
• #define MII_DS8721_ID 0x00221610
• #define MII_STE101P_ID 0x00061C50
• #define MII_DP83848I_ID 0x20005C90
• #define MII_LAN8710A_ID 0x0007C0F1
• #define MII_LAN8720_ID 0x0007C0F0
• #define MII_LAN8742A_ID 0x0007C130
• #define MII_LAN8740A_ID 0x0007C110
9.61.1 Detailed Description

MII macros and structures.

9.62 hal_mmc_spi.c File Reference

MMC over SPI driver code.

```c
#include <string.h>
#include "hal.h"
```

Functions

- static uint8_t mmc_crc7 (uint8_t crc, const uint8_t *buffer, size_t len)
  
  Calculate the MMC standard CRC-7 based on a lookup table.

- static bool mmc_wait_idle (MMCDriver *mmcp)
  
  Waits an idle condition.

- static bool mmc_send_hdr (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)
  
  Sends a command header.

- static bool mmc_recvr1 (MMCDriver *mmcp, uint8_t *r1p)
  
  Receives a single byte response.

- static bool mmc_recvr3 (MMCDriver *mmcp, uint8_t *r1p)
  
  Receives a three byte response.

- static bool mmc_send_command_R1 (MMCDriver *mmcp, uint8_t cmd, uint32_t arg, uint8_t *r1p)
  
  Sends a command an returns a single byte response.

- static bool mmc_send_command_R3 (MMCDriver *mmcp, uint8_t cmd, uint32_t arg, uint8_t *r1p)
  
  Sends a command which returns a five bytes response (R3).

- static bool mmc_read_CxD (MMCDriver *mmcp, uint8_t cmd, uint32_t cxd[4])
  
  Reads the CSD.

- void mmcInit (void)
  
  MMC over SPI driver initialization.

- void mmcObjectInit (MMCDriver *mmcp, uint8_t *buffer)
  
  Initializes an instance.

- msg_t mmcStart (MMCDriver *mmcp, const MMCConfig *config)
  
  Configures and activates the MMC peripheral.

- void mmcStop (MMCDriver *mmcp)
  
  Disables the MMC peripheral.

- bool mmcConnect (MMCDriver *mmcp)
  
  Performs the initialization procedure on the inserted card.

- bool mmcDisconnect (MMCDriver *mmcp)
  
  Brings the driver in a state safe for card removal.

- bool mmcStartSequentialRead (MMCDriver *mmcp, uint32_t startblk)
  
  Starts a sequential read.

- bool mmcSequentialRead (MMCDriver *mmcp, uint8_t *buffer)
  
  Reads a block within a sequential read operation.

- bool mmcStopSequentialRead (MMCDriver *mmcp)
  
  Stops a sequential read gracefully.

- bool mmcStartSequentialWrite (MMCDriver *mmcp, uint32_t startblk)
Starts a sequential write.

- **bool mmcSequentialWrite (MMCDriver *mmcp, const uint8_t *buffer)**
  
  Writes a block within a sequential write operation.

- **bool mmcStopSequentialWrite (MMCDriver *mmcp)**
  
  Stops a sequential write gracefully.

- **bool mmcSync (MMCDriver *mmcp)**
  
  Waits for card idle condition.

- **bool mmcGetInfo (MMCDriver *mmcp, BlockDeviceInfo *bdip)**
  
  Returns the media info.

- **bool mmcErase (MMCDriver *mmcp, uint32_t startblk, uint32_t endblk)**
  
  Erases blocks.

### Variables

- **static const struct mmc_spi_driver_vmt mmc_vmt**
  
  Virtual methods table.

- **static const uint8_t mmc_crc7_lookup_table [256]**
  
  Lookup table for CRC-7 (based on polynomial $x^7 + x^3 + 1$).

### 9.62.1 Detailed Description

MMC over SPI driver code.

### 9.63 hal_mmc_spi.h File Reference

MMC over SPI driver header.

### Data Structures

- **struct mmc_spi_config_t**
  
  Type of a MMC/SD over SPI driver configuration structure.

- **struct mmc_spi_driver_vmt**
  
  MMCDriver virtual methods table.

- **struct mmc_spi_driver_t**
  
  Structure representing a MMC/SD over SPI driver.
Macros

- `#define MMC_BUFFER_SIZE 16U`
  Size of the buffer to be supplied to the driver.
- `#define __mmc_driver_methods _mmcsd_block_device_methods`
  MMCDriver specific methods.

MMC_SPI configuration options

- `#define MMC_IDLE_TIMEOUT_MS 1000`
  Timeout before assuming a failure while waiting for card idle.
- `#define MMC_USE_MUTUAL_EXCLUSION TRUE`
  Mutual exclusion on the SPI bus.

Macro Functions

- `#define mmcIsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)`
  Returns the card insertion status.
- `#define mmcIsWriteProtected(mmcp) mmc_lld_is_write_protected(mmcp)`
  Returns the write protect status.

Typedefs

- `typedef mmc_spi_config_t MMConfig`
  Legacy name for compatibility.
- `typedef mmc_spi_driver_t MMCDriver`
  Legacy name for compatibility.

Functions

- `void mmcInit (void)`
  MMC over SPI driver initialization.
- `void mmcObjectInit (MMCDriver *mmcp, uint8_t *buffer)`
  Initializes an instance.
- `msg_t mmcStart (MMCDriver *mmcp, const MMConfig *config)`
  Configures and activates the MMC peripheral.
- `void mmcStop (MMCDriver *mmcp)`
  Disables the MMC peripheral.
- `bool mmcConnect (MMCDriver *mmcp)`
  Performs the initialization procedure on the inserted card.
- `bool mmcDisconnect (MMCDriver *mmcp)`
  Brings the driver in a state safe for card removal.
- `bool mmcStartSequentialRead (MMCDriver *mmcp, uint32_t startblk)`
  Starts a sequential read.
- `bool mmcSequentialRead (MMCDriver *mmcp, uint8_t *buffer)`
  Reads a block within a sequential read operation.
- `bool mmcStopSequentialRead (MMCDriver *mmcp)`
  Stops a sequential read gracefully.
- `bool mmcStartSequentialWrite (MMCDriver *mmcp, uint32_t startblk)`
  Starts a sequential write.
9.64.1 Detailed Description

MMC/SD cards common code.

#include "hal.h"

Functions

- `uint32_t _mmcsd_get_slice (const uint32_t *data, uint32_t end, uint32_t start)`
  
  Gets a bit field from a words array.

- `uint32_t _mmcsd_get_capacity (const uint32_t *csd)`
  
  Extract card capacity from a CSD.

- `uint32_t _mmcsd_get_capacity_ext (const uint8_t *ext_csd)`
  
  Extract MMC card capacity from EXT_CSD.

- `void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice *sdcp, unpacked_sdc_cid_t *cidsdc)`
  
  Unpacks SDC CID array in structure.

- `void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice *sdcp, unpacked_mmc_cid_t *cidmmc)`
  
  Unpacks MMC CID array in structure.

- `void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice *sdcp, unpacked_mmc_csd_t *csdmmc)`
  
  Unpacks MMC CSD array in structure.

- `void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_10_t *csd10)`
  
  Unpacks SDC CSD v1.0 array in structure.

- `void _mmcsd_unpack_csd_v20 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_20_t *csd20)`
  
  Unpacks SDC CSD v2.0 array in structure.

9.64 hal_mmcsd.c File Reference

MMC/SD cards common code.
9.65  hal_mmcsd.h File Reference

MMC/SD cards common header.

Data Structures

- struct MMCSDBlockDeviceVMT
  MMCSDBlockDevice virtual methods table.
- struct MMCSDBlockDevice
  MCC/SD block device class.
- struct unpacked_sdc_cid_t
  Unpacked CID register from SDC.
- struct unpacked_mmc_cid_t
  Unpacked CID register from MMC.
- struct unpacked_sdc_csd_10_t
  Unpacked CSD v1.0 register from SDC.
- struct unpacked_sdc_csd_20_t
  Unpacked CSD v2.0 register from SDC.
- struct unpacked_mmc_csd_t
  Unpacked CSD register from MMC.

Macros

- #define MMCSD_BLOCK_SIZE 512U
  Fixed block size for MMC/SD block devices.
- #define MMCSD_R1_ERROR_MASK 0xFDFFE008U
  Mask of error bits in R1 responses.
- #define MMCSD_CMD8_PATTERN 0x000001AAU
  Fixed pattern for CMD8.
- #define _mmcsd_block_device_methods _base_block_device_methods
  MMCSDBlockDevice specific methods.
- #define _mmcsd_block_device_data
  MMCSDBlockDevice specific data.

SD/MMC status conditions

- #define MMCSD_STS_IDLE 0U
- #define MMCSD_STS_READY 1U
- #define MMCSD_STS_IDENT 2U
- #define MMCSD_STS_STBY 3U
- #define MMCSD_STS_TRAN 4U
- #define MMCSD_STS_DATA 5U
- #define MMCSD_STS_RCV 6U
- #define MMCSD_STS_PRG 7U
- #define MMCSD_STS_DIS 8U

SD/MMC commands

- #define MMCSD_CMD_GO_IDLE_STATE 0U
- #define MMCSD_CMD_INIT 1U
- #define MMCSD_CMD_ALL_SEND_CID 2U
• #define MMCSD_CMD_SEND_RELATIVE_ADDR 3U
• #define MMCSD_CMD_SET_RELATIVE_ADDR MMCSD_CMD_SEND_RELATIVE_ADDR
• #define MMCSD_CMD_SET_BUS_WIDTH 6U
• #define MMCSD_CMD_SWITCH MMCSD_CMD_SET_BUS_WIDTH
• #define MMCSD_CMD_SEL_DESEL_CARD 7U
• #define MMCSD_CMD_SEND_IF_COND 8U
• #define MMCSD_CMD_SEND_EXT_CSD MMCSD_CMD_SEND_IF_COND
• #define MMCSD_CMD_SEND_CSD 9U
• #define MMCSD_CMD_SEND_CID 10U
• #define MMCSD_CMD_STOP_TRANSMISSION 12U
• #define MMCSD_CMD_SEND_STATUS 13U
• #define MMCSD_CMD_SET_BLOCKLEN 16U
• #define MMCSD_CMD_READ_SINGLE_BLOCK 17U
• #define MMCSD_CMD_READ_MULTIPLE_BLOCK 18U
• #define MMCSD_CMD_SET_BLOCK_COUNT 23U
• #define MMCSD_CMD_WRITE_BLOCK 24U
• #define MMCSD_CMD_WRITE_MULTIPLE_BLOCK 25U
• #define MMCSD_CMD_ERASE_RW_BLK_START 32U
• #define MMCSD_CMD_ERASE_RW_BLK_END 33U
• #define MMCSD_CMD_ERASE 38U
• #define MMCSD_CMD_APP_OP_COND 41U
• #define MMCSD_CMD_LOCK_UNLOCK 42U
• #define MMCSD_CMD_APP_CMD 55U
• #define MMCSD_CMD_READ_OCR 58U

CSD record offsets

• #define MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE 127U, 126U
• #define MMCSD_CSD_MMC_SPEC_VERS_SLICE 125U, 122U
• #define MMCSD_CSD_MMC_TAAC_SLICE 119U, 112U
• #define MMCSD_CSD_MMC_NSAC_SLICE 111U, 104U
• #define MMCSD_CSD_MMC_TRAN_SPEED_SLICE 103U, 96U
• #define MMCSD_CSD_MMC_CCC_SLICE 95U, 84U
• #define MMCSD_CSD_MMC_READ_BL_LEN_SLICE 83U, 80U
• #define MMCSD_CSD_MMC_READ_BL_PARTIAL_SLICE 79U, 79U
• #define MMCSD_CSD_MMC_WRITE_BLK_MISALIGN_SLICE 78U, 78U
• #define MMCSD_CSD_MMC_READ_BLK_MISALIGN_SLICE 77U, 77U
• #define MMCSD_CSD_MMC_DSR_IMP_SLICE 76U, 76U
• #define MMCSD_CSD_MMC_C_SIZE_SLICE 73U, 62U
• #define MMCSD_CSD_MMC_VDD_R_CURR_MIN_SLICE 61U, 59U
• #define MMCSD_CSD_MMC_VDD_R_CURR_MAX_SLICE 58U, 56U
• #define MMCSD_CSD_MMC_VDD_W_CURR_MIN_SLICE 55U, 53U
• #define MMCSD_CSD_MMC_VDD_W_CURR_MAX_SLICE 52U, 50U
• #define MMCSD_CSD_MMC_C_SIZE_MULT_SLICE 49U, 47U
• #define MMCSD_CSD_MMC_ERASE_GRP_SIZE_SLICE 46U, 42U
• #define MMCSD_CSD_MMC_ERASE_GRP_MULT_SLICE 41U, 37U
• #define MMCSD_CSD_MMC_WP_GRP_SIZE_SLICE 36U, 32U
• #define MMCSD_CSD_MMC_WP_GRP_ENABLE_SLICE 31U, 31U
• #define MMCSD_CSD_MMC_DEFAULT_ECC_SLICE 30U, 29U
• #define MMCSD_CSD_MMC_R2W_FACTOR_SLICE 28U, 26U
• #define MMCSD_CSD_MMC_WRITE_BL_LEN_SLICE 25U, 22U
• #define MMCSD_CSD_MMC_WRITE_BL_PARTIAL_SLICE 21U, 21U
• #define MMCSD_CSD_MMC_CONTENT_PROT_APP_SLICE 16U, 16U
• #define MMCSD_CSD_MMC_FILE_FORMAT_GRP_SLICE 15U, 15U
• #define MMCSD_CSD_MMC_COPY_SLICE 14U, 14U
• #define MMCSD_CSD_MMC_PERM_WRITE_PROTECT_SLICE 13U, 13U
• #define MMCSD_CSD_MMC_TMP_WRITE_PROTECT_SLICE 12U, 12U
• #define MMCSD_CSD_MMC_FILE_FORMAT_SLICE 11U, 10U
• #define MMCSD_CSD_MMC_CRC_SLICE 9U, 8U
• #define MMCSD_CSD_MMC_ECC_SLICE 7U, 8U
• #define MMCSD_CSD_20_CRC_SLICE 7U, 1U
• #define MMCSD_CSD_20_FILE_FORMAT_SLICE 11U, 10U
• #define MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE 12U, 12U
• #define MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE 13U, 13U
• #define MMCSD_CSD_20_COPY_SLICE 14U, 14U
• #define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 15U, 15U
• #define MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE 21U, 21U
• #define MMCSD_CSD_20_WRITE_BL_LEN_SLICE 25U, 12U
• #define MMCSD_CSD_20_R2W_FACTOR_SLICE 28U, 26U
• #define MMCSD_CSD_20_WP_GRP_ENABLE_SLICE 31U, 31U
• #define MMCSD_CSD_20_WP_GRP_SIZE_SLICE 38U, 32U
• #define MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE 45U, 39U
• #define MMCSD_CSD_20_ERASE_BLK_EN_SLICE 46U, 46U
• #define MMCSD_CSD_20_C_SIZE_SLICE 69U, 48U
• #define MMCSD_CSD_20_DSR_IMP_SLICE 76U, 76U
• #define MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE 77U, 77U
• #define MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE 78U, 78U
• #define MMCSD_CSD_20_READ_BL_PARTIAL_SLICE 79U, 79U
• #define MMCSD_CSD_20_READ_BL_LEN_SLICE 83U, 80U
• #define MMCSD_CSD_20_CCC_SLICE 95U, 84U
• #define MMCSD_CSD_20_TRANS_SPEED_SLICE 103U, 96U
• #define MMCSD_CSD_20_NSAC_SLICE 111U, 104U
• #define MMCSD_CSD_20_TAAC_SLICE 119U, 112U
• #define MMCSD_CSD_20_CSD_STRUCTURE_SLICE 127U, 126U
• #define MMCSD_CSD_10_CRC_SLICE MMCSD_CSD_20_CRC_SLICE
• #define MMCSD_CSD_10_FILE_FORMAT_SLICE MMCSD_CSD_20_FILE_FORMAT_SLICE
• #define MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE
• #define MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE
• #define MMCSD_CSD_10_COPY_SLICE MMCSD_CSD_20_COPY_SLICE
• #define MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE
• #define MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE
• #define MMCSD_CSD_10_WRITE_BL_LEN_SLICE MMCSD_CSD_20_WRITE_BL_LEN_SLICE
• #define MMCSD_CSD_10_R2W_FACTOR_SLICE MMCSD_CSD_20_R2W_FACTOR_SLICE
• #define MMCSD_CSD_10_WP_GRP_ENABLE_SLICE MMCSD_CSD_20_WP_GRP_ENABLE_SLICE
• #define MMCSD_CSD_10_WP_GRP_SIZE_SLICE MMCSD_CSD_20_WP_GRP_SIZE_SLICE
• #define MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE
• #define MMCSD_CSD_10_C_SIZE_MULT_SLICE 49U, 47U
• #define MMCSD_CSD_10_VDD_W_CURR_MAX_SLICE 52U, 50U
• #define MMCSD_CSD_10_VDD_W_CURR_MIN_SLICE 55U, 53U
• #define MMCSD_CSD_10_VDD_R_CURR_MAX_SLICE 58U, 56U
• #define MMCSD_CSD_10_VDD_R_CURR_MIX_SLICE 61U, 59U
• #define MMCSD_CSD_10_C_SIZE_SLICE 73U, 62U
• #define MMCSD_CSD_10_DSR_IMP_SLICE MMCSD_CSD_20_DSR_IMP_SLICE
• #define MMCSD_CSD_10_READ_BLK_MISALIGN_SLICE MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE
• #define MMCSD_CSD_10_WRITE_BLK_MISALIGN_SLICE MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE
• #define MMCSD_CSD_10_READ_BL_PARTIAL_SLICE MMCSD_CSD_20_READ_BL_PARTIAL_SLICE
• #define MMCSD_CSD_10_READ_BL_LEN_SLICE 83U, 80U
• #define MMCSD_CSD_10_CCC_SLICE MMCSD_CSD_20_CCC_SLICE
• #define MMCSD_CSD_10_TRANS_SPEED_SLICE MMCSD_CSD_20_TRANS_SPEED_SLICE
• #define MMCSD_CSD_10_NSAC_SLICE MMCSD_CSD_20_NSAC_SLICE
• #define MMCSD_CSD_10_TAAC_SLICE MMCSD_CSD_20_TAAC_SLICE
• #define MMCSD_CSD_10_CSD_STRUCTURE_SLICE MMCSD_CSD_20_CSD_STRUCTURE_SLICE

CID record offsets

• #define MMCSD_CID_SDC_CRC_SLICE 7U, 1U
• #define MMCSD_CID_SDC_MDT_M_SLICE 11U, 8U
• #define MMCSD_CID_SDC_MDT_Y_SLICE 19U, 12U
• #define MMCSD_CID_SDC_PSN_SLICE 55U, 24U
• #define MMCSD_CID_SDC_PRV_M_SLICE 59U, 56U
• #define MMCSD_CID_SDC_PRV_N_SLICE 63U, 60U
• #define MMCSD_CID_SDC_PNM0_SLICE 71U, 64U
• #define MMCSD_CID_SDC_PNM1_SLICE 79U, 72U
• #define MMCSD_CID_SDC_PNM2_SLICE 87U, 80U
• #define MMCSD_CID_SDC_PNM3_SLICE 95U, 88U
• #define MMCSD_CID_SDC_PNM4_SLICE 103U, 96U
• #define MMCSD_CID_SDC_OID_SLICE 119U, 104U
• #define MMCSD_CID_SDC_MID_SLICE 127U, 120U
• #define MMCSD_CID_MMC_CRC_SLICE 7U, 1U
• #define MMCSD_CID_MMC_MDT_Y_SLICE 11U, 8U
• #define MMCSD_CID_MMC_MDT_M_SLICE 15U, 12U
• #define MMCSD_CID_MMC_PSN_SLICE 47U, 16U
• #define MMCSD_CID_MMC_PRV_M_SLICE 51U, 48U
• #define MMCSD_CID_MMC_PRV_N_SLICE 55U, 52U
• #define MMCSD_CID_MMC_PNM0_SLICE 63U, 56U
• #define MMCSD_CID_MMC_PNM1_SLICE 71U, 64U
• #define MMCSD_CID_MMC_PNM2_SLICE 79U, 72U
• #define MMCSD_CID_MMC_PNM3_SLICE 87U, 80U
• #define MMCSD_CID_MMC_PNM4_SLICE 95U, 88U
• #define MMCSD_CID_MMC_PNM5_SLICE 103U, 96U
• #define MMCSD_CID_MMC_OID_SLICE 119U, 104U
• #define MMCSD_CID_MMC_MID_SLICE 127U, 120U

R1 response utilities

• #define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)
  Evaluates to true if the R1 response contains error flags.
• #define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)
  Returns the status field of an R1 response.
• #define MMCSD_R1_IS_CARD_LOCKED(r1) (((((r1) >> 21U) & 1U) != 0U)
  Evaluates to true if the R1 response indicates a locked card.

Macro Functions

• #define mmcsdGetCardCapacity(ip) ((ip)->capacity)
  Returns the card capacity in blocks.

Functions

• uint32_t _mmcsd_get_slice (const uint32_t *data, uint32_t end, uint32_t start)
  Gets a bit field from a words array.
• uint32_t _mmcsd_get_capacity (const uint32_t *csd)
  Extract card capacity from a CSD.
• uint32_t _mmcsd_get_capacity_ext (const uint8_t *ext_csd)
  Extract MMC card capacity from EXT_CSD.
• void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice *sdcp, unpacked_sdc_cid_t *cidsdc)
  Unpacks SDC CID array in structure.
• void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice *sdcp, unpacked_mmc_cid_t *cidmmc)
  Unpacks MMC CID array in structure.
• void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice *sdcp, unpacked_mmc_csd_t *csdmmmc)
  Unpacks MMC CSD array in structure.
• void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_10_t *csd10)
  Unpacks SDC CSD v1.0 array in structure.
• void _mmcsd_unpack_csd_v20 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_20_t *csd20)
  Unpacks SDC CSD v2.0 array in structure.
9.65.1 Detailed Description

MMC/SD cards common header.

This header defines an abstract interface useful to access MMC/SD I/O block devices in a standardized way.

9.66 hal_objects.h File Reference

Base object.

Data Structures

- struct BaseObjectVMT
  BaseObject virtual methods table.
- struct BaseObject
  Base object class.

Macros

- #define _base_object_methods
  BaseObject specific methods.
- #define _base_object_data
  BaseObject specific data.

Macro Functions (BaseObject)

- #define objGetInstance(type, ip) ((type)(((size_t)(ip)) - (ip)->vmt->instance_offset))
  Returns the instance pointer starting from an interface pointer.

9.66.1 Detailed Description

Base object.

This header defines a base object that is the root for the inheritance system.

9.67 hal_pal.c File Reference

I/O Ports Abstraction Layer code.

#include "hal.h"
### Functions

- **ioportmask_t palReadBus** (const IOBus *bus)
  
  Read from an I/O bus.

- **void palWriteBus** (const IOBus *bus, ioportmask_t bits)
  
  Write to an I/O bus.

- **void palSetBusMode** (const IOBus *bus, iomode_t mode)
  
  Programs a bus with the specified mode.

- **void palSetPadCallbackI** (ioportid_t port, iopadid_t pad, palcallback_t cb, void *arg)
  
  Associates a callback to a port/pad.

- **void palSetLineCallbackI** (ioline_t line, palcallback_t cb, void *arg)
  
  Associates a callback to a line.

- **msg_t palWaitPadTimeoutS** (ioportid_t port, iopadid_t pad, sysinterval_t timeout)
  
  Waits for an edge on the specified port/pad.

- **msg_t palWaitPadTimeout** (ioportid_t port, iopadid_t pad, sysinterval_t timeout)
  
  Waits for an edge on the specified port/pad.

- **msg_t palWaitLineTimeoutS** (ioline_t line, sysinterval_t timeout)
  
  Waits for an edge on the specified line.

- **msg_t palWaitLineTimeout** (ioline_t line, sysinterval_t timeout)
  
  Waits for an edge on the specified line.

### 9.67.1 Detailed Description

I/O Ports Abstraction Layer code.

### 9.68 hal_pal.h File Reference

I/O Ports Abstraction Layer macros, types and structures.

```c
#include "hal_pal_lld.h"
```

### Data Structures

- **struct palevent_t**
  
  Type of a PAL event record.

- **struct IOBus**
  
  I/O bus descriptor.
Macros

- `#define PAL_PORT_BIT(n) ((ioportmask_t)(1U << (n)))`
  Port bit helper macro.
- `#define PAL_GROUP_MASK(width) ((ioportmask_t)(1U << (width)) - 1U)`
  Bits group mask helper.
- `#define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}`
  Data part of a static I/O bus initializer.
- `#define IOBUS_DECL(name, port, width, offset) IOBus name = _IOBUS_DATA(name, port, width, offset)`
  Static I/O bus initializer.

Pads mode constants

- `#define PAL_MODE_RESET 0U`  
  After reset state.
- `#define PAL_MODE_UNCONNECTED 1U`  
  Safe state for unconnected pads.
- `#define PAL_MODE_INPUT 2U`  
  Regular input high-Z pad.
- `#define PAL_MODE_INPUT_PULLUP 3U`  
  Input pad with weak pull up resistor.
- `#define PAL_MODE_INPUT_PULLDOWN 4U`  
  Input pad with weak pull down resistor.
- `#define PAL_MODE_INPUT_ANALOG 5U`  
  Analog input mode.
- `#define PAL_MODE_OUTPUT_PUSHPULL 6U`  
  Push-pull output pad.
- `#define PAL_MODE_OUTPUT_OPENDRAIN 7U`  
  Open-drain output pad.

Logic level constants

- `#define PAL_LOW 0U`  
  Logical low state.
- `#define PAL_HIGH 1U`  
  Logical high state.

PAL event modes

- `#define PAL_EVENT_MODE_EDGES_MASK 3U`  
  Mask of edges field.
- `#define PAL_EVENT_MODE_DISABLED 0U`  
  Channel disabled.
- `#define PAL_EVENT_MODE_RISING_EDGE 1U`  
  Rising edge callback.
- `#define PAL_EVENT_MODE_FALLING_EDGE 2U`  
  Falling edge callback.
- `#define PAL_EVENT_MODE_BOTH_EDGES 3U`  
  Both edges callback.
PAL configuration options

- #define PAL_USE_CALLBACKS TRUE
  Enables synchronous APIs.
- #define PAL_USE_WAIT TRUE
  Enables synchronous APIs.

Low level driver helper macros

- #define _pal_init_event(e)
  Initializes a PAL event object.
- #define _pal_init_event(e)
  Initializes a PAL event object.
- #define _pal_init_event(e)
  Initializes a PAL event object.
- #define _pal_clear_event(e)
  Clears a PAL event object.
- #define _pal_clear_event(e)
  Clears a PAL event object.
- #define _pal_clear_event(e)
  Clears a PAL event object.
- #define _pal_isr_code(e)
  Common ISR code.
- #define _pal_isr_code(e)
  Common ISR code.
- #define _pal_isr_code(e)
  Common ISR code.

Macro Functions

- #define palInit() pal_lld_init()
  PAL subsystem initialization.
- #define palReadPort(port) ((void)(port), 0U)
  Reads the physical I/O port states.
- #define palReadLatch(port) ((void)(port), 0U)
  Reads the output latch.
- #define palWritePort(port, bits) ((void)(port), (void)(bits))
  Writes a bits mask on a I/O port.
- #define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))
  Sets a bits mask on a I/O port.
- #define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ~ (bits))
  Clears a bits mask on a I/O port.
- #define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))
  Toggles a bits mask on a I/O port.
- #define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))
  Reads a group of bits.
- #define palReadGroupLatch(port, mask, offset) ((palReadLatch(port) >> (offset)) & (mask))
  Reads the group latch.
- #define palWriteGroup(port, mask, offset, bits)
  Writes a group of bits.
- #define palSetGroupMode(port, mask, offset, mode)
  Pads group mode setup.
- #define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)
  Reads an input pad logic state.
- #define palWritePad(port, pad, bit)
  Writes a logic state on an output pad.
- #define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))
  Sets a pad logic state to PAL_HIGH.
• #define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))
  Clears a pad logic state to PAL_LOW.
• #define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))
  Toggles a pad logic state.
• #define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)
  Pad mode setup.
• #define palReadLine(line) palReadPad(PAL_PORT(line), PAL_PAD(line))
  Reads an input line logic state.
• #define palWriteLine(line, bit) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)
  Writes a logic state on an output line.
• #define palSetLine(line) palSetPad(PAL_PORT(line), PAL_PAD(line))
  Sets a line logic state to PAL_HIGH.
• #define palClearLine(line) palClearPad(PAL_PORT(line), PAL_PAD(line))
  Clears a line logic state to PAL_LOW.
• #define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))
  Toggles a line logic state.
• #define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)
  Line mode setup.
• #define palEnablePadEventI(port, pad, mode) palEnablePadEvent(port, pad, mode)
  Pad event enable.
• #define palDisablePadEventI(port, pad) palDisablePadEvent(port, pad)
  Pad event disable.
• #define palEnablePadEvent(port, pad, mode) palEnablePadEventI(PAL_PORT(port), PAL_PAD(port), mode)
  Pad event enable.
• #define palDisablePadEvent(port, pad) palDisablePadEventI(PAL_PORT(port), PAL_PAD(port))
  Pad event disable.
• #define palEnableLineEventI(line, mode) palEnableLineEventI(PAL_PORT(line), PAL_PAD(line), mode)
  Line event enable.
• #define palDisableLineEventI(line) palDisableLineEventI(PAL_PORT(line), PAL_PAD(line))
  Line event disable.
• #define palEnableLineEvent(line, mode) palEnableLineEventI(PAL_PORT(line), PAL_PAD(line), mode)
  Line event enable.
• #define palDisableLineEvent(line) palDisableLineEventI(PAL_PORT(line), PAL_PAD(line))
  Line event disable.
• #define palIsPadEventEnabledX(port, pad) false
  Pad event enable check.
• #define palIsLineEventEnabledX(line) pal_lld_ispadeventenabled(PAL_PORT(line), PAL_PAD(line))
  Line event enable check.
• #define palSetPadCallback(port, pad, cb, arg) palcallback_t
  Associates a callback to a pad.
• #define palSetLineCallback(line, cb, arg) palcallback_t
  Associates a callback to a line.

Typedefs

• typedef void(*palcallback_t)(void *arg)
  Type of a PAL event callback.

Functions

• ioportmask_t palReadBus (const IOBus *bus)
  Read from an I/O bus.
• void palWriteBus (const IOBus *bus, ioportmask_t bits)
  Write to an I/O bus.
• void palSetBusMode (const IOBus *bus, iomode_t mode)
Programs a bus with the specified mode.

- void palSetPadCallbackI (ioportid_t port, iopadid_t pad, palcallback_t cb, void *arg)
  Associates a callback to a port/pad.
- void palSetLineCallbackI (ioline_t line, palcallback_t cb, void *arg)
  Associates a callback to a line.
- msg_t palWaitPadTimeoutS (ioportid_t port, iopadid_t pad, sysinterval_t timeout)
  Waits for an edge on the specified port/pad.
- msg_t palWaitPadTimeout (ioportid_t port, iopadid_t pad, sysinterval_t timeout)
  Waits for an edge on the specified port/pad.
- msg_t palWaitLineTimeoutS (ioline_t line, sysinterval_t timeout)
  Waits for an edge on the specified line.
- msg_t palWaitLineTimeout (ioline_t line, sysinterval_t timeout)
  Waits for an edge on the specified line.

9.68.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures.

9.69 hal_pal_lld.c File Reference

PLATFORM PAL subsystem low level driver source.

#include "hal.h"

Functions

- void _pal_lld_init (void)
  STM32 I/O ports configuration.
- void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)
  Pads mode setup.

9.69.1 Detailed Description

PLATFORM PAL subsystem low level driver source.

9.70 hal_pal_lld.h File Reference

PLATFORM PAL subsystem low level driver header.

Data Structures

- struct PALConfig
  Generic I/O ports static initializer.
Macros

- `#define IPORT1 0`
  First I/O port identifier.
- `#define pal_lld_init() _pal_lld_init()`
  Low level PAL subsystem initialization.
- `#define pal_lld_readport(port) 0U`
  Reads the physical I/O port states.
- `#define pal_lld_readlatch(port) 0U`
  Reads the output latch.
- `#define pal_lld_writeport(port, bits)`
  Writes a bits mask on a I/O port.
- `#define pal_lld_setport(port, bits)`
  Sets a bits mask on a I/O port.
- `#define pal_lld_clearport(port, bits)`
  Clears a bits mask on a I/O port.
- `#define pal_lld_toggleport(port, bits)`
  Toggles a bits mask on a I/O port.
- `#define pal_lld_readgroup(port, mask, offset) 0U`
  Reads a group of bits.
- `#define pal_lld_writegroup(port, mask, offset, bits)`
  Writes a group of bits.
- `#define pal_lld_setgroupmode(port, mask, offset, mode) _pal_lld_setgroupmode(port, mask << offset, mode)`
  Pads group mode setup.
- `#define pal_lld_readpad(port, pad) PAL_LOW`
  Reads a logical state from an I/O pad.
- `#define pal_lld_writepad(port, pad, bit)`
  Writes a logical state on an output pad.
- `#define pal_lld_setpad(port, pad)`
  Sets a pad logical state to `PAL_HIGH`.
- `#define pal_lld_clearpad(port, pad)`
  Clears a pad logical state to `PAL_LOW`.
- `#define pal_lld_togglepad(port, pad)`
  Toggles a pad logical state.
- `#define pal_lld_setpadmode(port, pad, mode)`
  Pad mode setup.
- `#define pal_lld_get_pad_event(port, pad) &_pal_events[0]; (void)(port); (void)pad`
  Returns a PAL event structure associated to a pad.
- `#define pal_lld_get_line_event(line) &_pal_events[0]; (void)line`
  Returns a PAL event structure associated to a line.

Port related definitions

- `#define PAL_IOPORTS_WIDTH 16U`
  Width, in bits, of an I/O port.
- `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)`
  Whole port mask.

Line handling macros

- `#define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad)))`
Forms a line identifier.
• #define PAL_PORT(line) ((stm32_gpio_t *)((uint32_t)(line)) & 0xFFFFFFF0U))
  Decodes a port identifier from a line identifier.
• #define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))
  Decodes a pad identifier from a line identifier.
• #define PAL_NOLINE 0U
  Value identifying an invalid line.

Typedefs

• typedef uint32_t ioportmask_t
  Digital I/O port sized unsigned type.
• typedef uint32_t iomode_t
  Digital I/O modes.
• typedef uint32_t ioline_t
  Type of an I/O line.
• typedef uint32_t ioportid_t
  Port Identifier.
• typedef uint32_t iopadid_t
  Type of an pad identifier.

Functions

• void _pal_lld_init (void)
  STM32 I/O ports configuration.
• void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)
  Pads mode setup.

9.70.1 Detailed Description

PLATFORM PAL subsystem low level driver header.

9.71 hal_persistent.h File Reference

Generic persistent storage class header.

Data Structures

• struct BasePersistentStorageVMT
  BasePersistentStorage virtual methods table.
• struct BasePersistentStorage
  Base persistent storage class.
Macros

- `#define _base_pers_storage_methods_alone`
  BasePersistentStorage specific methods.
- `#define _base_pers_storage_methods`
  BasePersistentStorage specific methods with inherited ones.
- `#define _base_persistent_storage_data _base_object_data`
  BasePersistentStorage specific data.

Macro Functions (BasePersistentStorage)

- `#define getBasePersistentStorage(ip) ((BasePersistentStorage ∗)(ip)->vmt)`
  Instance getter.
- `#define psGetStorageSize(ip) (ip)->vmt->getsize(ip)`
  Get storage size.
- `#define psRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)`
  Read operation.
- `#define psWrite(ip, offset, n, wp) (ip)->vmt->write(ip, offset, n, wp)`
  Write operation.

Typedefs

- `typedef uint32_t ps_offset_t`
  Type of a persistent storage offset.

Enumerations

- `enum ps_error_t`
  Type of a persistent storage error code.

9.71.1 Detailed Description

Generic persistent storage class header.

9.72 hal_pwm.c File Reference

PWM Driver code.

#include "hal.h"
9.73 hal_pwm.h File Reference

Functions

- void pwmInit (void)
  PWM Driver initialization.

- void pwmObjectInit (PWMDriver *pwmp)
  Initializes the standard part of a PWMDriver structure.

- msg_t pwmStart (PWMDriver *pwmp, const PWMConfig *config)
  Configures and activates the PWM peripheral.

- void pwmStop (PWMDriver *pwmp)
  Deactivates the PWM peripheral.

- void pwmChangePeriod (PWMDriver *pwmp, pwmcnt_t period)
  Changes the period the PWM peripheral.

- void pwmEnableChannel (PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)
  Enables a PWM channel.

- void pwmDisableChannel (PWMDriver *pwmp, pwmchannel_t channel)
  Disables a PWM channel and its notification.

- void pwmEnablePeriodicNotification (PWMDriver *pwmp)
  Enables the periodic activation edge notification.

- void pwmDisablePeriodicNotification (PWMDriver *pwmp)
  Disables the periodic activation edge notification.

- void pwmEnableChannelNotification (PWMDriver *pwmp, pwmchannel_t channel)
  Enables a channel de-activation edge notification.

- void pwmDisableChannelNotification (PWMDriver *pwmp, pwmchannel_t channel)
  Disables a channel de-activation edge notification.

9.72.1 Detailed Description

PWM Driver code.

9.73 hal_pwm.h File Reference

PWM Driver macros and structures.

#include "hal_pwm_lld.h"

Macros

PWM output mode macros

- #define PWM_OUTPUT_MASK 0x0FU
  Standard output modes mask.

- #define PWM_OUTPUT_DISABLED 0x00U
  Output not driven, callback only.

- #define PWM_OUTPUT_ACTIVE_HIGH 0x01U
  Positive PWM logic, active is logic level one.

- #define PWM_OUTPUT_ACTIVE_LOW 0x02U
  Inverse PWM logic, active is logic level zero.
PWM duty cycle conversion

- #define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)
  Converts from fraction to pulse width.
- #define PWM_DEGREES_TO_WIDTH(pwmp, degrees) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)
  Converts from degrees to pulse width.
- #define PWM_PERCENTAGE_TO_WIDTH(pwmp, percentage) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)
  Converts from percentage to pulse width.

Macro Functions

- #define pwmChangePeriodI(pwmp, value)
  Changes the period the PWM peripheral.
- #define pwmEnableChannelI(pwmp, channel, width)
  Enables a PWM channel.
- #define pwmDisableChannelI(pwmp, channel)
  Disables a PWM channel.
- #define pwmIsChannelEnabledI(pwmp, channel) (((pwmp)>
  Returns a PWM channel status.
- #define pwmEnablePeriodicNotificationI(pwmp) pwm_lld_enable_periodic_notification(pwmp)
  Enables the periodic activation edge notification.
- #define pwmDisablePeriodicNotificationI(pwmp) pwm_lld_disable_periodic_notification(pwmp)
  Disables the periodic activation edge notification.
- #define pwmEnableChannelNotificationI(pwmp, channel) pwm_lld_enable_channel_notification(pwmp,
  Enables a channel de-activation edge notification.
- #define pwmDisableChannelNotificationI(pwmp, channel) pwm_lld_disable_channel_notification(pwmp,
  Disables a channel de-activation edge notification.

Typedefs

- typedef struct PWMDriver PWMDriver
  Type of a structure representing a PWM driver.
- typedef void(∗pwmcallback_t)(PWMDriver∗pwmp)
  Type of a PWM notification callback.

Enumerations

- enum pwmstate_t { PWM_UNINIT = 0, PWM_STOP = 1, PWM_READY = 2 }
  Driver state machine possible states.
9.74 hal_pwm_lld.c File Reference

Functions

• void `pwmInit` (void)
  PWM Driver initialization.

• void `pwmObjectInit` (PWMDriver ∗pwmp)
  Initializes the standard part of a PWMDriver structure.

• `msg_t pwmStart` (PWMDriver ∗pwmp, const PWMConfig ∗config)
  Configures and activates the PWM peripheral.

• void `pwmStop` (PWMDriver ∗pwmp)
  Deactivates the PWM peripheral.

• void `pwmChangePeriod` (PWMDriver ∗pwmp, pwmcnt_t period)
  Changes the period the PWM peripheral.

• `msg_t pwmEnableChannel` (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)
  Enables a PWM channel.

• `msg_t pwmDisableChannel` (PWMDriver ∗pwmp, pwmchannel_t channel)
  Disables a PWM channel and its notification.

• void `pwmEnablePeriodicNotification` (PWMDriver ∗pwmp)
  Enables the periodic activation edge notification.

• void `pwmDisablePeriodicNotification` (PWMDriver ∗pwmp)
  Disables the periodic activation edge notification.

• void `pwmEnableChannelNotification` (PWMDriver ∗pwmp, pwmchannel_t channel)
  Enables a channel de-activation edge notification.

• void `pwmDisableChannelNotification` (PWMDriver ∗pwmp, pwmchannel_t channel)
  Disables a channel de-activation edge notification.

9.73.1 Detailed Description

PWM Driver macros and structures.

9.74 hal_pwm_lld.c File Reference

PLATFORM PWM subsystem low level driver source.

#include "hal.h"

Functions

• void `pwm_lld_init` (void)
  Low level PWM driver initialization.

• void `pwm_lld_start` (PWMDriver ∗pwmp)
  Configures and activates the PWM peripheral.

• void `pwm_lld_stop` (PWMDriver ∗pwmp)
  Deactivates the PWM peripheral.

• void `pwm_lld_enable_channel` (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)
  Enables a PWM channel.

• void `pwm_lld_disable_channel` (PWMDriver ∗pwmp, pwmchannel_t channel)
  Disables a PWM channel and its notification.
• void pwm_lld_enable_periodic_notification (PWMDriver *pwmp)
   Enables the periodic activation edge notification.
• void pwm_lld_disable_periodic_notification (PWMDriver *pwmp)
   Disables the periodic activation edge notification.
• void pwm_lld_enable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)
   Enables a channel de-activation edge notification.
• void pwm_lld_disable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)
   Disables a channel de-activation edge notification.

Variables

• PWMDriver PWMD1
  PWMD1 driver identifier.

9.74.1 Detailed Description

PLATFORM PWM subsystem low level driver source.

9.75 hal_pwm_lld.h File Reference

PLATFORM PWM subsystem low level driver header.

Data Structures

• struct PWMChannelConfig
  Type of a PWM driver channel configuration structure.
• struct PWMDriver
  Structure representing a PWM driver.

Macros

• #define PWM_CHANNELS 4
  Number of PWM channels per PWM driver.
• #define pwm_lld_change_period(pwmp, period)
  Changes the period the PWM peripheral.

PLATFORM configuration options

• #define PLATFORM_PWM_USE_PWM1 FALSE
  PWMD1 driver enable switch.
Typedefs

- typedef uint32_t pwmmode_t
  Type of a PWM mode.
- typedef uint8_t pwmchannel_t
  Type of a PWM channel.
- typedef uint32_t pwmchnmsk_t
  Type of a channels mask.
- typedef uint32_t pwmcnt_t
  Type of a PWM counter.

Functions

- void pwm_lld_init (void)
  Low level PWM driver initialization.
- void pwm_lld_start (PWMDriver *pwmp)
  Configures and activates the PWM peripheral.
- void pwm_lld_stop (PWMDriver *pwmp)
  Deactivates the PWM peripheral.
- void pwm_lld_enable_channel (PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)
  Enables a PWM channel.
- void pwm_lld_disable_channel (PWMDriver *pwmp, pwmchannel_t channel)
  Disables a PWM channel and its notification.
- void pwm_lld_enable_periodic_notification (PWMDriver *pwmp)
  Enables the periodic activation edge notification.
- void pwm_lld_disable_periodic_notification (PWMDriver *pwmp)
  Disables the periodic activation edge notification.
- void pwm_lld_enable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)
  Enables a channel de-activation edge notification.
- void pwm_lld_disable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)
  Disables a channel de-activation edge notification.

9.75.1 Detailed Description

PLATFORM PWM subsystem low level driver header.

9.76 hal_queues.c File Reference

I/O Queues code.

#include <string.h>
#include "hal.h"
Functions

- `static size_t iq_read (input_queue_t *iqp, uint8_t *bp, size_t n)`
  Non-blocking input queue read.

- `static size_t oq_write (output_queue_t *oqp, const uint8_t *bp, size_t n)`
  Non-blocking output queue write.

- `void iqObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)`
  Initializes an input queue.

- `void iqResetI (input_queue_t *iqp)`
  Resets an input queue.

- `msg_t iqPutI (input_queue_t *iqp, uint8_t b)`
  Input queue write.

- `msg_t iqGetI (input_queue_t *iqp)`
  Input queue non-blocking read.

- `msg_t iqGetTimeout (input_queue_t *iqp, sysinterval_t timeout)`
  Input queue read with timeout.

- `size_t iqReadI (input_queue_t *iqp, uint8_t *bp, size_t n)`
  Input queue non-blocking read.

- `size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, sysinterval_t timeout)`
  Input queue read with timeout.

- `void oqObjectInit (output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)`
  Initializes an output queue.

- `void oqResetI (output_queue_t *oqp)`
  Resets an output queue.

- `msg_t oqPutI (output_queue_t *oqp, uint8_t b)`
  Output queue non-blocking write.

- `msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, sysinterval_t timeout)`
  Output queue write with timeout.

- `msg_t oqGetI (output_queue_t *oqp)`
  Output queue read.

- `size_t oqWriteI (output_queue_t *oqp, const uint8_t *bp, size_t n)`
  Output queue non-blocking write.

- `size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, sysinterval_t timeout)`
  Output queue write with timeout.

9.76.1 Detailed Description

I/O Queues code.

9.77 hal_queues.h File Reference

I/O Queues macros and structures.

Data Structures

- `struct io_queue`
  Generic I/O queue structure.
Macros

Queue functions returned status value

- `#define Q_OK MSG_OK`
  Operation successful.

- `#define Q_TIMEOUT MSG_TIMEOUT`
  Timeout condition.

- `#define Q_RESET MSG_RESET`
  Queue has been reset.

- `#define Q_EMPTY MSG_TIMEOUT`
  Queue empty.

- `#define Q_FULL MSG_TIMEOUT`
  Queue full.

Macro Functions

- `#define qSizeX(qp)`
  Returns the queue’s buffer size.

- `#define qSpaceI(qp) ((qp)->q_counter)`
  Queue space.

- `#define qGetLink(qp) ((qp)->q_link)`
  Returns the queue application-defined link.

- `#define qSetLink(qp, lk) ((qp)->q_link = lk)`
  Sets the queue application-defined link.

- `#define iqGetFullI(iqp) qSpaceI(iqp)`
  Returns the filled space into an input queue.

- `#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))`
  Returns the empty space into an input queue.

- `#define iqIsEmptyI(iqp) ((bool)(qSpaceI(iqp) == 0U))`
  Evaluates to true if the specified input queue is empty.

- `#define iqIsFullI(iqp)`
  Evaluates to true if the specified input queue is full.

- `#define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)`
  Input queue read.

- `#define oqGetFullI(oqp) (qSizeX(oqp) - qSpaceI(oqp))`
  Returns the filled space into an output queue.

- `#define oqGetEmptyI(oqp) qSpaceI(oqp)`
  Returns the empty space into an output queue.

- `#define oqIsEmptyI(oqp)`
  Evaluates to true if the specified output queue is empty.

- `#define oqIsFullI(oqp) ((bool)(qSpaceI(oqp) == 0U))`
  Evaluates to true if the specified output queue is full.

- `#define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)`
  Output queue write.

Typedefs

- `typedef struct io_queue io_queue_t`  
  Type of a generic I/O queue structure.

- `typedef void (qnotify_t) (io_queue_t *qp)`  
  Queue notification callback type.

- `typedef io_queue_t input_queue_t`  
  Type of an input queue structure.

- `typedef io_queue_t output_queue_t`  
  Type of an output queue structure.
Functions

- void iqObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)
  
  Initializes an input queue.
- void iqResetI (input_queue_t *iqp)
  
  Resets an input queue.
- msg_t iqPutI (input_queue_t *iqp, uint8_t b)
  
  Input queue write.
- msg_t iqGetI (input_queue_t *iqp)
  
  Input queue non-blocking read.
- msg_t iqGetTimeout (input_queue_t *iqp, sysinterval_t timeout)
  
  Input queue read with timeout.
- size_t iqReadI (input_queue_t *iqp, uint8_t *bp, size_t n)
  
  Input queue non-blocking read.
- size_t iqReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, sysinterval_t timeout)
  
  Input queue read with timeout.
- void oqObjectInit (output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)
  
  Initializes an output queue.
- void oqResetI (output_queue_t *oqp)
  
  Resets an output queue.
- msg_t oqPutI (output_queue_t *oqp, uint8_t b)
  
  Output queue non-blocking write.
- msg_t oqPutTimeout (output_queue_t *oqp, uint8_t b, sysinterval_t timeout)
  
  Output queue write with timeout.
- msg_t oqGetI (output_queue_t *oqp)
  
  Output queue read.
- size_t oqWriteI (output_queue_t *oqp, const uint8_t *bp, size_t n)
  
  Output queue non-blocking write.
- size_t oqWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, sysinterval_t timeout)
  
  Output queue write with timeout.

9.77.1 Detailed Description

I/O Queues macros and structures.

9.78 hal_rtc.c File Reference

RTC Driver code.

#include "hal.h"
Functions

- `void rtcInit (void)`
  RTC Driver initialization.
- `void rtcObjectInit (RTCDriver *rtcp)`
  Initializes a generic RTC driver object.
- `void rtcSetTime (RTCDriver *rtcp, const RTCDateTime *timespec)`
  Set current time.
- `void rtcGetTime (RTCDriver *rtcp, RTCDateTime *timespec)`
  Get current time.
- `void rtcSetAlarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)`
  Set alarm time.
- `void rtcGetAlarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)`
  Get current alarm.
- `void rtcSetCallback (RTCDriver *rtcp, rtccb_t callback)`
  Enables or disables RTC callbacks.
- `void rtcConvertDateTimeToStructTm (const RTCDateTime *timespec, struct tm *timp, uint32_t *tv_msec)`
  Convert RTCDateTime to broken-down time structure.
- `void rtcConvertStructTmToDateTime (const struct tm *timp, uint32_t tv_msec, RTCDateTime *timespec)`
  Convert broken-down time structure to RTCDateTime.
- `uint32_t rtcConvertDateTimeToFAT (const RTCDateTime *timespec)`
  Get current time in format suitable for usage in FAT file system.

9.78.1 Detailed Description

RTC Driver code.

9.79 hal_rtc.h File Reference

RTC Driver macros and structures.

```c
#include <time.h>
#include "hal_rtc_lld.h"
```

Data Structures

- `struct RTCDateTime`
  Type of a structure representing an RTC date/time stamp.
- `struct RTCDriverVMT`
  RTCDriver virtual methods table.
- `struct RTCDriver`
  Structure representing an RTC driver.
Macros

- #define RTC_BASE_YEAR 1980U  
  Base year of the calendar.
- #define _rtc_driver_methods _base_pers_storage_methods  
  BasePersistentStorage specific methods.

Date/Time bit masks for FAT format

- #define RTC_FAT_TIME_SECONDS_MASK 0x0000001FU
- #define RTC_FAT_TIME_MINUTES_MASK 0x000007E0U
- #define RTC_FAT_TIME_HOURS_MASK 0x0000F800U
- #define RTC_FAT_DATE_DAYS_MASK 0x001F0000U
- #define RTC_FAT_DATE_MONTHS_MASK 0x01E00000U
- #define RTC_FAT_DATE_YEARS_MASK 0xFE000000U

Day of week encoding

- #define RTC_DAY_SUNDAY 7U
- #define RTC_DAY_SATURDAY 6U
- #define RTC_DAY_FRIDAY 5U
- #define RTC_DAY_THURSDAY 4U
- #define RTC_DAY_WEDNESDAY 3U
- #define RTC_DAY_TUESDAY 2U
- #define RTC_DAY_MONDAY 1U
- #define RTC_DAY_CATURDAY 0U

Typedefs

- typedef struct RTCDriver RTCDriver  
  Type of a structure representing an RTC driver.
- typedef unsigned int rtcalarm_t  
  Type of an RTC alarm number.

Functions

- void rtcInit (void)  
  RTC Driver initialization.
- void rtcObjectInit (RTCDriver *rtcp)  
  Initializes a generic RTC driver object.
- void rtcSetTime (RTCDriver *rtcp, const RTCDateTime *timespec)  
  Set current time.
- void rtcGetTime (RTCDriver *rtcp, RTCDateTime *timespec)  
  Get current time.
- void rtcSetCallback (RTCDriver *rtcp, rtccb_t callback)  
  Enables or disables RTC callbacks.
- void rtcConvertDateTimeToStructTm (const RTCDateTime *timespec, struct tm *timp, uint32_t *tv_msec)  
  Convert RTCDateTime to broken-down time structure.
- void rtcConvertStructTmToDateTme (const struct tm *timp, uint32_t tv_msec, RTCDateTime *timespec)  
  Convert broken-down time structure to RTCDateTime.
- uint32_t rtcConvertDateTimeToFAT (const RTCDateTime *timespec)  
  Get current time in format suitable for usage in FAT file system.
9.79.1 Detailed Description

RTC Driver macros and structures.

9.80 hal_rtc_lld.c File Reference

PLATFORM RTC subsystem low level driver source.

#include "hal.h"

Functions

- void rtc_lld_init (void)
  RTC driver identifier.
- void rtc_lld_set_time (RTCDriver *rtcp, const RTCDateTime *timespec)
  Set current time.
- void rtc_lld_get_time (RTCDriver *rtcp, RTCDateTime *timespec)
  Get current time.
- void rtc_lld_set_alarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)
  Set alarm time.
- void rtc_lld_get_alarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)
  Get alarm time.

9.80.1 Detailed Description

PLATFORM RTC subsystem low level driver source.

9.81 hal_rtc_lld.h File Reference

PLATFORM RTC subsystem low level driver header.

Data Structures

- struct RTCAlarm
  Type of a structure representing an RTC alarm time stamp.
Macros

- `#define rtc_lld_driver_fields uint32_t dummy`
  Implementation-specific `RTCDriver` fields.

Implementation capabilities

- `#define RTC_SUPPORTS_CALLBACKS TRUE`
  Callback support in the driver.
- `#define RTC_ALARMS 2`
  Number of alarms available.
- `#define RTC_HAS_STORAGE FALSE`
  Presence of a local persistent storage.

PLATFORM configuration options

- `#define PLATFORM_RTC_USE_RTC1 FALSE`
  RTCD1 driver enable switch.

Typedefs

- `typedef void(∗ rtccb_t) (RTCDriver ∗rtcp, rtcevent_t event)`
  Type of a generic RTC callback.

Enumerations

- `enum rtcevent_t`
  Type of an RTC event.

Functions

- `void rtc_lld_init (void)`
  RTC driver identifier.
- `void rtc_lld_set_time (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)`
  Set current time.
- `void rtc_lld_get_time (RTCDriver ∗rtcp, RTCDateTime ∗timespec)`
  Get current time.
- `void rtc_lld_set_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAAlarm ∗alarmspec)`
  Set alarm time.
- `void rtc_lld_get_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAAlarm ∗alarmspec)`
  Get alarm time.

9.81.1 Detailed Description

PLATFORM RTC subsystem low level driver header.
9.82  hal_sdc.c File Reference

SDC Driver code.

#include <string.h>
#include "hal.h"

Enumerations

• enum mmc_switch_t
  MMC switch mode.
• enum sd_switch_t
  SDC switch mode.
• enum sd_switch_function_t
  SDC switch function.

Functions

• static bool mode_detect (SDCDriver *sdcp)
  Detects card mode.
• static bool mmc_init (SDCDriver *sdcp)
  Init procedure for MMC.
• static bool sdc_init (SDCDriver *sdcp)
  Init procedure for SDC.
• static uint32_t mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)
  Constructs CMD6 argument for MMC.
• static uint32_t sdc_cmd6_construct (sd_switch_t mode, sd_switch_function_t function, uint32_t value)
  Constructs CMD6 argument for SDC.
• static uint16_t sdc_cmd6_extract_info (sd_switch_function_t function, const uint8_t *buf)
  Extracts information from CMD6 answer.
• static bool sdc_cmd6_check_status (sd_switch_function_t function, const uint8_t *buf)
  Checks status after switching using CMD6.
• static bool sdc_detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)
  Reads supported bus clock and switch SDC to appropriate mode.
• static bool mmc_detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)
  Reads supported bus clock and switch MMC to appropriate mode.
• static bool detect_bus_clk (SDCDriver *sdcp, sdcbusclk_t *clk)
  Reads supported bus clock and switch card to appropriate mode.
• static bool sdc_set_bus_width (SDCDriver *sdcp)
  Sets bus width for SDC.
• static bool mmc_set_bus_width (SDCDriver *sdcp)
  Sets bus width for MMC.
• bool _sdc_wait_for_transfer_state (SDCDriver *sdcp)
  Wait for the card to complete pending operations.
• void sdcInit (void)
  SDC Driver initialization.
• void sdcObjectInit (SDCDriver *sdcp)
  Initializes the standard part of a SDCDriver structure.
• `msg_t sdcStart (SDCDriver *sdcp, const SDConfig *config)`
  Configures and activates the SDC peripheral.
• `void sdcStop (SDCDriver *sdcp)`
  Deactivates the SDC peripheral.
• `bool sdcConnect (SDCDriver *sdcp)`
  Performs the initialization procedure on the inserted card.
• `bool sdcDisconnect (SDCDriver *sdcp)`
  Brings the driver in a state safe for card removal.
• `bool sdcRead (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`
  Reads one or more blocks.
• `bool sdcWrite (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`
  Writes one or more blocks.
• `sdcflags_t sdcGetAndClearErrors (SDCDriver *sdcp)`
  Returns the errors mask associated to the previous operation.
• `bool sdcSync (SDCDriver *sdcp)`
  Waits for card idle condition.
• `bool sdcGetInfo (SDCDriver *sdcp, BlockDeviceInfo *bdip)`
  Returns the media info.
• `bool sdcErase (SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)`
  Erases the supplied blocks.

**Variables**

• `static const struct SDCDriverVMT sdc_vmt`
  Virtual methods table.

**9.82.1 Detailed Description**

SDC Driver code.

**9.83 hal_sdc.h File Reference**

SDC Driver macros and structures.

```
#include "hal_sdc_lld.h"
```
Macros

SD card types

- \#define SDC_MODE_CARDTYPE_MASK 0xFU
- \#define SDC_MODE_CARDTYPE_SDV11 0U
- \#define SDC_MODE_CARDTYPE_SDV20 1U
- \#define SDC_MODE_CARDTYPE_MMC 2U
- \#define SDC_MODE_HIGH_CAPACITY 0x10U

SDC bus error conditions

- \#define SDC_NO_ERROR 0U
- \#define SDC_CMD_CRC_ERROR 1U
- \#define SDC_DATA_CRC_ERROR 2U
- \#define SDC_DATA_TIMEOUT 4U
- \#define SDC_COMMAND_TIMEOUT 8U
- \#define SDC_TX_UNDERRUN 16U
- \#define SDC_RX_OVERRUN 32U
- \#define SDC_STARTBIT_ERROR 64U
- \#define SDC_OVERFLOW_ERROR 128U
- \#define SDC_UNHANDLED_ERROR 0xFFFFFFFFU

SDC configuration options

- \#define SDC_INIT_RETRY 100  
  Number of initialization attempts before rejecting the card.
- \#define SDC_MMC_SUPPORT FALSE  
  Include support for MMC cards.
- \#define SDC_NICE_WAITING TRUE  
  Delays insertions.
- \#define SDC_INIT_OCR_V20 0x50FF8000U  
  OCR initialization constant for V20 cards.
- \#define SDC_INIT_OCR 0x80100000U  
  OCR initialization constant for non-V20 cards.

Macro Functions

- \#define sdcIsCardInserted(sdcp) (sdc_lld_is_card_inserted(sdcp))  
  Returns the card insertion status.
- \#define sdcIsWriteProtected(sdcp) (sdc_lld_is_write_protected(sdcp))  
  Returns the write protect status.

Enumerations

- enum sdcbusmode_t  
  Type of SDIO bus mode.
- enum sdcbusclk_t  
  Max supported clock.
Functions

- **void sdcInit (void)**
  
  *SDC Driver initialization.*

- **void sdcObjectInit (SDCDriver **sdcp)**
  
  *Initializes the standard part of a SCDriver structure.*

- **msg_t sdcStart (SDCDriver **sdcp, const SDCCConfig **config)**
  
  *Configures and activates the SDC peripheral.*

- **void sdcStop (SDCDriver **sdcp)**
  
  *Deactivates the SDC peripheral.*

- **bool sdcConnect (SDCDriver **sdcp)**
  
  *Performs the initialization procedure on the inserted card.*

- **bool sdcDisconnect (SDCDriver **sdcp)**
  
  *Brings the driver in a state safe for card removal.*

- **bool sdcRead (SDCDriver **sdcp, uint32_t startblk, uint8_t **buf, uint32_t n)**
  
  *Reads one or more blocks.*

- **bool sdcWrite (SDCDriver **sdcp, uint32_t startblk, const uint8_t **buf, uint32_t n)**
  
  *Writes one or more blocks.*

- **sdcflags_t sdcGetAndClearErrors (SDCDriver **sdcp)**
  
  *Returns the errors mask associated to the previous operation.*

- **bool sdcSync (SDCDriver **sdcp)**
  
  *Waits for card idle condition.*

- **bool sdcGetInfo (SDCDriver **sdcp, BlockDeviceInfo **bdip)**
  
  *Returns the media info.*

- **bool sdcErase (SDCDriver **sdcp, uint32_t startblk, uint32_t endblk)**
  
  *Erases the supplied blocks.*

- **bool _sdc_wait_for_transfer_state (SDCDriver **sdcp)**
  
  *Wait for the card to complete pending operations.*

9.83.1 Detailed Description

SDC Driver macros and structures.

9.84 hal_sdc_lld.c File Reference

PLATFORM SDC subsystem low level driver source.

#include "hal.h"
Functions

- **void sdc_lld_init (void)**
  
  Low level SDC driver initialization.

- **void sdc_lld_start (SDCDriver *sdc)**
  
  Configures and activates the SDC peripheral.

- **void sdc_lld_stop (SDCDriver *sdc)**
  
  Deactivates the SDC peripheral.

- **void sdc_lld_start_clk (SDCDriver *sdc)**
  
  Starts the SDIO clock and sets it to init mode (400kHz or less).

- **void sdc_lld_set_data_clk (SDCDriver *sdc, sdcbusclk_t clk)**
  
  Sets the SDIO clock to data mode (25MHz or less).

- **void sdc_lld_stop_clk (SDCDriver *sdc)**
  
  Stops the SDIO clock.

- **void sdc_lld_set_bus_mode (SDCDriver *sdc, sdcbusmode_t mode)**
  
  Switches the bus to 4 bits mode.

- **void sdc_lld_send_cmd_none (SDCDriver *sdc, uint8_t cmd, uint32_t arg)**
  
  Sends an SDIO command with no response expected.

- **bool sdc_lld_send_cmd_short (SDCDriver *sdc, uint8_t cmd, uint32_t arg, uint32_t *resp)**
  
  Sends an SDIO command with a short response expected.

- **bool sdc_lld_send_cmd_short_crc (SDCDriver *sdc, uint8_t cmd, uint32_t arg, uint32_t *resp)**
  
  Sends an SDIO command with a short response expected and CRC.

- **bool sdc_lld_send_cmd_long_crc (SDCDriver *sdc, uint8_t cmd, uint32_t arg, uint32_t *resp)**
  
  Sends an SDIO command with a long response expected and CRC.

- **bool sdc_lld_read (SDCDriver *sdc, uint32_t startblk, uint8_t *buf, uint32_t n)**
  
  Reads one or more blocks.

- **bool sdc_lld_write (SDCDriver *sdc, uint32_t startblk, const uint8_t *buf, uint32_t n)**
  
  Writes one or more blocks.

- **bool sdc_lld_sync (SDCDriver *sdc)**
  
  Waits for card idle condition.

Variables

- **SDCDriver SDCD1**
  
  SDCD1 driver identifier.

9.84.1 Detailed Description

PLATFORM SDC subsystem low level driver source.

9.85 hal_sdc_lld.h File Reference

PLATFORM SDC subsystem low level driver header.
Data Structures

- struct SDCConfig
  
  Driver configuration structure.

- struct SDCDriverVMT
  
  SDCDriver virtual methods table.

- struct SDCDriver
  
  Structure representing an SDC driver.

Macros

- #define _sdc_driver_methods _mmcsd_block_device_methods
  
  SDCDriver specific methods.

PLATFORM configuration options

- #define PLATFORM_SDC_USE_SDC1 FALSE
  
  PWMD1 driver enable switch.

Typedefs

- typedef uint32_t sdcmode_t
  
  Type of card flags.

- typedef uint32_t sdcflags_t
  
  SDC Driver condition flags type.

- typedef struct SDCDriver SDCDriver
  
  Type of a structure representing an SDC driver.

Functions

- void sdc_lld_init (void)
  
  Low level SDC driver initialization.

- void sdc_lld_start (SDCDriver *sdcp)
  
  Configures and activates the SDC peripheral.

- void sdc_lld_stop (SDCDriver *sdcp)
  
  Deactivates the SDC peripheral.

- void sdc_lld_start_clk (SDCDriver *sdcp)
  
  Starts the SDIO clock and sets it to init mode (400kHz or less).

- void sdc_lld_set_data_clk (SDCDriver *sdcp, sdcbusclk_t clk)
  
  Sets the SDIO clock to data mode (25MHz or less).

- void sdc_lld_stop_clk (SDCDriver *sdcp)
  
  Stops the SDIO clock.

- void sdc_lld_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)
  
  Switches the bus to 4 bits mode.

- void sdc_lld_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)
  
  Sends an SDIO command with no response expected.

- bool sdc_lld_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)
  
  Sends an SDIO command with a short response expected.

- bool sdc_lld_send_cmd_short_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)
Sends an SDIO command with a short response expected and CRC.

- `bool sdc_lld_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`

  Sends an SDIO command with a long response expected and CRC.

- `bool sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`

  Reads one or more blocks.

- `bool sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`

  Writes one or more blocks.

- `bool sdc_lld_sync (SDCDriver *sdcp)`

  Waits for card idle condition.

### 9.85.1 Detailed Description

PLATFORM SDC subsystem low level driver header.

### 9.86 hal_serial.c File Reference

Serial Driver code.

```c
#include "hal.h"
```

### Functions

- `void sdInit (void)`

  Serial Driver initialization.

- `void sdObjectInit (SerialDriver *sdp, qnotify_t inotify, qnotify_t onotify)`

  Initializes a generic serial driver object.

- `msg_t sdStart (SerialDriver *sdp, const SerialConfig *config)`

  Configures and starts the driver.

- `void sdStop (SerialDriver *sdp)`

  Stops the driver.

- `void sdIncomingDataI (SerialDriver *sdp, uint8_t b)`

  Handles incoming data.

- `msg_t sdRequestDataI (SerialDriver *sdp)`

  Handles outgoing data.

- `bool sdPutWouldBlock (SerialDriver *sdp)`

  Direct output check on a SerialDriver.

- `bool sdGetWouldBlock (SerialDriver *sdp)`

  Direct input check on a SerialDriver.

- `msg_t sdControl (SerialDriver *sdp, unsigned int operation, void *arg)`

  Control operation on a serial port.

### 9.86.1 Detailed Description

Serial Driver code.
9.87  `hal_serial.h` File Reference

Serial Driver macros and structures.

```c
#include "hal_serial_lld.h"
```

**Data Structures**

- `struct SerialDriverVMT`  
  `SerialDriver virtual methods table.`
- `struct hal_serial_driver`  
  `Full duplex serial driver class.`

**Macros**

- `#define _serial_driver_methods_base_asynchronous_channel_methods`  
  `SerialDriver specific methods.`

**Serial status flags (legacy)**

- `#define SD_PARITY_ERROR CHN_PARITY_ERROR`
- `#define SD_FRAMING_ERROR CHN_FRAMING_ERROR`
- `#define SD_OVERRUN_ERROR CHN_OVERRUN_ERROR`
- `#define SD_NOISE_ERROR CHN_NOISE_ERROR`
- `#define SD_BREAK_DETECTED CHN_BREAK_DETECTED`
- `#define SD_QUEUE_FULL_ERROR CHN_BUFFER_FULL_ERROR`

**Serial configuration options**

- `#define SERIAL_DEFAULT_BITRATE 38400`  
  `Default bit rate.`
- `#define SERIAL_BUFFERS_SIZE 16`  
  `Serial buffers size.`

**Macro Functions**

- `#define sdPut(sdp, b) oqPut(&(sdp)->oqueue, b)`  
  `Direct write to a SerialDriver.`
- `#define sdPutI(sdp, b) oqPutI(&(sdp)->oqueue, b)`  
  `Direct write to a SerialDriver.`
- `#define sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->oqueue, b, t)`  
  `Direct write to a SerialDriver with timeout specification.`
- `#define sdGetI(sdp) iqGetI(&(sdp)->iqueue)`  
  `Direct read from a SerialDriver.`
- `#define sdGet(sdp) iqGet(&(sdp)->iqueue)`  
  `Direct read from a SerialDriver.`
- `#define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->iqueue, t)`  
  `Direct read from a SerialDriver with timeout specification.`
- `#define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)`  
  `Direct blocking write to a SerialDriver.`
- `#define sdWriteI(sdp, b, n) oqWriteI(&(sdp)->oqueue, b, n)`  
  `Direct non-blocking write to a SerialDriver.`
9.87.1 Detailed Description

Serial Driver macros and structures.
#include "hal.h"

## Functions

- **void sd_lld_init (void)**
  
  *Low level serial driver initialization.*

- **void sd_lld_start (SerialDriver *sdp, const SerialConfig *config)**
  
  *Low level serial driver configuration and (re)start.*

- **void sd_lld_stop (SerialDriver *sdp)**
  
  *Low level serial driver stop.*

## Variables

- **SerialDriver SD1**
  
  *USART1 serial driver identifier.*

- **static const SerialConfig default_config**
  
  *Driver default configuration.*

## Detailed Description

PLATFORM serial subsystem low level driver source.

## 9.89 hal_serial_lld.h File Reference

PLATFORM serial subsystem low level driver header.

## Data Structures

- **struct hal_serial_config**
  
  *PLATFORM Serial Driver configuration structure.*

## Macros

- **#define _serial_driver_data**
  
  *SerialDriver specific data.*

## PLATFORM configuration options

- **#define PLATFORM_SERIAL_USE_USART1 FALSE**
  
  *USART1 driver enable switch.*
Typedefs

- typedef struct hal_serial_config SerialConfig
  
  PLATFORM Serial Driver configuration structure.

Functions

- void sd_lld_init (void)
  
  Low level serial driver initialization.

- void sd_lld_start (SerialDriver *sdp, const SerialConfig *config)
  
  Low level serial driver configuration and (re)start.

- void sd_lld_stop (SerialDriver *sdp)
  
  Low level serial driver stop.

9.89.1 Detailed Description

PLATFORM serial subsystem low level driver header.

9.90 hal_serial_nor.c File Reference

Serial NOR serial flash driver code.

#include "hal.h"
#include "hal_serial_nor.h"

Functions

- static const flash_descriptor_t * snor_get_descriptor (void *instance)
  
  Returns a pointer to the device descriptor.

- void bus_acquire (BUSDriver *busp, const BUSConfig *config)
  
  Bus acquisition and lock.

- void bus_release (BUSDriver *busp)
  
  Bus release.

- void bus_stop (BUSDriver *busp)
  
  Stops the underlying bus driver.

- void bus_cmd (BUSDriver *busp, uint32_t cmd)
  
  Sends a naked command.

- void bus_cmd_send (BUSDriver *busp, uint32_t cmd, size_t n, const uint8_t *p)
  
  Sends a command followed by a data transmit phase.

- void bus_cmd_receive (BUSDriver *busp, uint32_t cmd, size_t n, uint8_t *p)
  
  Sends a command followed by a data receive phase.

- void bus_cmd_addr (BUSDriver *busp, uint32_t cmd, flash_offset_t offset)
  
  Sends a command followed by a flash address.

- void bus_cmd_addr_send (BUSDriver *busp, uint32_t cmd, flash_offset_t offset, size_t n, const uint8_t *p)
  
  Sends a command followed by a flash address and a data transmit phase.

- void bus_cmd_addr_receive (BUSDriver *busp, uint32_t cmd, flash_offset_t offset, size_t n, uint8_t *p)
Sends a command followed by a flash address and a data receive phase.

- void bus_cmd_dummy_receive (BUSDriver ∗busp, uint32_t cmd, uint32_t dummy, size_t n, uint8_t ∗p)
  Sends a command followed by dummy cycles and a data receive phase.

- void bus_cmd_addr_dummy_receive (BUSDriver ∗busp, uint32_t cmd, flash_offset_t offset, uint32_t dummy, size_t n, uint8_t ∗p)
  Sends a command followed by a flash address, dummy cycles and a data receive phase.

- void snorObjectInit (SNORDriver ∗devp, snor_nocache_buffer_t ∗nocache)
  Initializes an instance.

- void snorStart (SNORDriver ∗devp, const SNORConfig ∗config)
  Configures and activates SNOR driver.

- void snorStop (SNORDriver ∗devp)
  Deactivates the SNOR driver.

- void snorMemoryMap (SNORDriver ∗devp, uint8_t ∗∗addrp)
  Enters the memory Mapping mode.

- void snorMemoryUnmap (SNORDriver ∗devp)
  Leaves the memory Mapping mode.

Variables

- static const struct SNORDriverVMT snor_vmt
  Virtual methods table.

9.90.1 Detailed Description

Serial NOR serial flash driver code.

9.91 hal_serial_nor.h File Reference

Serial NOR driver header.

#include "hal_flash.h"
#include "hal_flash_device.h"

Data Structures

- struct SNORConfig
  Type of a SNOR configuration structure.

- struct SNORDriverVMT
  SNOR virtual methods table.

- struct SNORDriver
  Type of SNOR flash class.
Macros

- `#define SNOR_BUFFER_SIZE 32`
  
  Size of the buffer for internal operations.

- `#define _snor.flash.methods.alone`
  
  SNORDriver specific methods.

- `#define _snor.flash.methods`
  
  SNORDriver specific methods with inherited ones.

Bus interface modes.

- `#define SNOR_BUS_DRIVER_SPI 0U`
- `#define SNOR_BUS_DRIVER_WSPI 1U`

Configuration options

- `#define SNOR_BUS_DRIVER_SNOR_BUS_DRIVER_WSPI`
  
  Physical transport interface.

- `#define SNOR_SHARED_BUS TRUE`
  
  Shared bus switch.

- `#define SNOR_USE_MUTUAL_EXCLUSION TRUE`
  
  Exclusive access control.

- `#define SNOR_SPI_4BYTES_ADDRESS FALSE`
  
  SPI 4-bytes address switch.

Functions

- `void bus_acquire (BUSDriver *busp, const BUSConfig *config)`
  
  Bus acquisition and lock.

- `void bus_release (BUSDriver *busp)`
  
  Bus release.

- `void bus_cmd (BUSDriver *busp, uint32_t cmd)`
  
  Sends a naked command.

- `void bus_cmd_send (BUSDriver *busp, uint32_t cmd, size_t n, const uint8_t *p)`
  
  Sends a command followed by a data transmit phase.

- `void bus_cmd_receive (BUSDriver *busp, uint32_t cmd, size_t n, uint8_t *p)`
  
  Sends a command followed by a data receive phase.

- `void bus_cmd_addr (BUSDriver *busp, uint32_t cmd, flash_offset_t offset)`
  
  Sends a command followed by a flash address.

- `void bus_cmd_addr_send (BUSDriver *busp, uint32_t cmd, flash_offset_t offset, size_t n, const uint8_t *p)`
  
  Sends a command followed by a flash address and a data transmit phase.

- `void bus_cmd_addr_receive (BUSDriver *busp, uint32_t cmd, flash_offset_t offset, size_t n, uint8_t *p)`
  
  Sends a command followed by a flash address and a data receive phase.

- `void bus_cmd_dummy_receive (BUSDriver *busp, uint32_t cmd, uint32_t dummy, size_t n, uint8_t *p)`
  
  Sends a command followed by dummy cycles and a data receive phase.

- `void bus_cmd_addr_dummy_receive (BUSDriver *busp, uint32_t cmd, flash_offset_t offset, uint32_t dummy, size_t n, uint8_t *p)`
  
  Sends a command followed by a flash address, dummy cycles and a data receive phase.

- `void snorObjectInit (SNORDriver *devp, snor_nocache_buffer_t *nocache)`
  
  Initializes an instance.

- `void snorStart (SNORDriver *devp, const SNORConfig *config)`
  
  Configures and activates SNOR driver.

- `void snorStop (SNORDriver *devp)`
  
  Deactivates the SNOR driver.

- `void snorMemoryMap (SNORDriver *devp, uint8_t **addrp)`
  
  Enters the memory Mapping mode.

- `void snorMemoryUnmap (SNORDriver *devp)`
  
  Leaves the memory Mapping mode.
9.91.1 Detailed Description

Serial NOR driver header.

9.92 hal_serial_usb.c File Reference

Serial over USB Driver code.

#include "hal.h"

Functions

- static void ibnotify (io_buffers_queue_t *bqp)
  Notification of empty buffer released into the input buffers queue.
- static void obnotify (io_buffers_queue_t *bqp)
  Notification of filled buffer inserted into the output buffers queue.
- void sduInit (void)
  Serial Driver initialization.
- void sduObjectInit (SerialUSBDriver *dup)
  Initializes a generic full duplex driver object.
- msg_t sduStart (SerialUSBDriver *dup, const SerialUSBConfig *config)
  Configures and starts the driver.
- void sduStop (SerialUSBDriver *dup)
  Stops the driver.
- void sduSuspendHookI (SerialUSBDriver *dup)
  USB device suspend handler.
- void sduWakeupHookI (SerialUSBDriver *dup)
  USB device wakeup handler.
- void sduConfigureHookI (SerialUSBDriver *dup)
  USB device configured handler.
- bool sduRequestsHook (USBDriver *usbp)
  Default requests hook.
- void sduSOFHookI (SerialUSBDriver *dup)
  SOF handler.
- void sduDataTransmitted (USBDriver *usbp, usbep_t ep)
  Default data transmitted callback.
- void sduDataReceived (USBDriver *usbp, usbep_t ep)
  Default data received callback.
- void sduInterruptTransmitted (USBDriver *usbp, usbep_t ep)
  Default data received callback.
- msg_t sduControl (USBDriver *usbp, unsigned int operation, void *arg)
  Control operation on a serial USB port.

9.92.1 Detailed Description

Serial over USB Driver code.
Serial over USB Driver macros and structures.

#include "hal_usb_cdc.h"

Data Structures

- struct SerialUSBConfig
  Serial over USB Driver configuration structure.
- struct SerialUSBDriverVMT
  SerialDriver virtual methods table.
- struct SerialUSBDriver
  Full duplex serial driver class.

Macros

- #define _serial_usb_driver_data
  SerialDriver specific data.
- #define _serial_usb_driver_methods _base_asynchronous_channel_methods
  SerialUSBDriver specific methods.

SERIAL_USB configuration options

- #define SERIAL_USB_BUFFERS_SIZE 256
  Serial over USB buffers size.
- #define SERIAL_USB_BUFFERS_NUMBER 2
  Serial over USB number of buffers.

Typedefs

- typedef struct SerialUSBDriver SerialUSBDriver
  Structure representing a serial over USB driver.

Enumerations

- enum sdustate_t { SDU_UNINIT = 0, SDU_STOP = 1, SDU_READY = 2 }
  Driver state machine possible states.
Functions

- **void sduInit (void)**
  
  Serial Driver initialization.

- **void sduObjectInit (SerialUSBDriver *sdup)**
  
  Initializes a generic full duplex driver object.

- **msg_t sduStart (SerialUSBDriver *sdup, const SerialUSBConfig *config)**
  
  Configures and starts the driver.

- **void sduStop (SerialUSBDriver *sdup)**
  
  Stops the driver.

- **void sduSuspendHookI (SerialUSBDriver *sdup)**
  
  USB device suspend handler.

- **void sduWakeupHookI (SerialUSBDriver *sdup)**
  
  USB device wakeup handler.

- **void sduConfigureHookI (SerialUSBDriver *sdup)**
  
  USB device configured handler.

- **bool sduRequestsHook (USBDriver *usbp)**
  
  Default requests hook.

- **void sduSOFHookI (SerialUSBDriver *sdup)**
  
  SOF handler.

- **void sduDataTransmitted (USBDriver *usbp, usbep_t ep)**
  
  Default data transmitted callback.

- **void sduDataReceived (USBDriver *usbp, usbep_t ep)**
  
  Default data received callback.

- **void sduInterruptTransmitted (USBDriver *usbp, usbep_t ep)**
  
  Default data received callback.

- **msg_t sduControl (USBDriver *usbp, unsigned int operation, void *arg)**
  
  Control operation on a serial USB port.

### 9.93.1 Detailed Description

Serial over USB Driver macros and structures.

### 9.94 hal_sio.c File Reference

SIO Driver code.

```c
#include "hal.h"
```
Functions

- **void sioInit (void)**
  SIO Driver initialization.

- **void sioObjectInit (SIODriver *siop)**
  Initializes the standard part of a SIODriver structure.

- **msg_t sioStart (SIODriver *siop, const SIOConfig *config)**
  Configures and activates the SIO peripheral.

- **void sioStop (SIODriver *siop)**
  Deactivates the SIO peripheral.

- **void sioWriteEnableFlags (SIODriver *siop, sioevents_t mask)**
  Writes the enabled events flags mask.

- **void sioSetEnableFlags (SIODriver *siop, sioevents_t mask)**
  Sets flags into the enabled events flags mask.

- **void sioClearEnableFlags (SIODriver *siop, sioevents_t mask)**
  Clears flags from the enabled events flags mask.

- **sioevents_t sioGetAndClearErrors (SIODriver *siop)**
  Get and clears SIO error event flags.

- **sioevents_t sioGetAndClearEvents (SIODriver *siop)**
  Get and clears SIO event flags.

- **sioevents_t sioGetEvents (SIODriver *siop)**
  Returns the pending SIO event flags.

- **size_t sioAsyncRead (SIODriver *siop, uint8_t *buffer, size_t n)**
  Reads data from the RX FIFO.

- **size_t sioAsyncWrite (SIODriver *siop, const uint8_t *buffer, size_t n)**
  Writes data into the TX FIFO.

- **msg_t sioSynchronizeRX (SIODriver *siop, sysinterval_t timeout)**
  Synchronizes with RX FIFO data availability.

- **msg_t sioSynchronizeRXIdle (SIODriver *siop, sysinterval_t timeout)**
  Synchronizes with RX going idle.

- **msg_t sioSynchronizeTX (SIODriver *siop, sysinterval_t timeout)**
  Synchronizes with TX FIFO space availability.

- **msg_t sioSynchronizeTXEnd (SIODriver *siop, sysinterval_t timeout)**
  Synchronizes with TX completion.

9.94.1 Detailed Description

SIO Driver code.

9.95 hal_sio.h File Reference

SIO Driver macros and structures.

#include "hal_sio_lld.h"
Data Structures

- struct hal_sio_config
  Driver configuration structure.
- struct sio_driver_vmt
  SIODriver virtual methods table.
- struct hal_sio_driver
  Structure representing a SIO driver.

Macros

- #define _sio_driver_methods _base_channel_methods
  SIODriver specific methods.
- #define sioSetCallbackX(siop, f) (siop)->cb = (f)
  Associates a callback to the SIO instance.
- #define sioIsRXEmptyX(siop) sio_lld_is_rx_empty(siop)
  Determines the state of the RX FIFO.
- #define sioIsRXIdleX(siop) sio_lld_is_rx_idle(siop)
  Determines the activity state of the receiver.
- #define sioHasRXErrorsX(siop) sio_lld_has_rx_errors(siop)
  Determines if RX has pending errors to be read and cleared.
- #define sioIsTXFullX(siop) sio_lld_is_tx_full(siop)
  Determines the state of the TX FIFO.
- #define sioIsTXOngoingX(siop) sio_lld_is_tx_ongoing(siop)
  Determines the transmission state.
- #define sioWriteEnableFlagsX(siop, mask)
  Writes the enabled events mask.
- #define sioSetEnableFlagsX(siop, mask)
  Sets flags into the enabled events flags mask.
- #define sioClearEnableFlagsX(siop, mask)
  Clears flags from the enabled events flags mask.
- #define sioGetEnableFlagsX(siop) (siop)->enabled
  Return the enabled condition flags mask.
- #define sioGetAndClearErrorsX(siop) sio_lld_get_and_clear_errors(siop)
  Get and clears SIO error event flags.
- #define sioGetAndClearEventsX(siop) sio_lld_get_and_clear_events(siop)
  Get and clears SIO event flags.
- #define sioGetEventsX(siop) sio_lld_get_events(siop)
  Returns the pending SIO event flags.
- #define sioGetX(siop) sio_lld_get(siop)
  Returns one frame from the RX FIFO.
- #define sioPutX(siop, data) sio_lld_put(siop, data)
  Pushes one frame into the TX FIFO.
- #define sioAsyncReadX(siop, size, buffer) sio_lld_read(siop, size, buffer)
  Reads data from the RX FIFO.
- #define sioAsyncWriteX(siop, size, buffer) sio_lld_write(siop, size, buffer)
  Writes data into the TX FIFO.
- #define sioControlX(siop, operation, arg) sio_lld_control(siop, operation, arg)
  Control operation on a serial port.
- #define __sio_reloc_field(v, m, s, d) ((((v) & m) >>> (s)) <<< (d))
Relocates a bit field.

SIO masks offsets

- `#define SIO_EV_RXNOTEMPTY_POS 2 /* CHN_INPUT_AVAILABLE */`
- `#define SIO_EV_TXNOTFULL_POS 3 /* CHN_OUTPUT_EMPTY */`
- `#define SIO_EV_TXDONE_POS 4 /* CHN_TRANSMISSION_END */`
- `#define SIO_EV_ALL_ERRORS_POS SIO_EV_PARITY_ERR_POS`
- `#define SIO_EV_PARITY_ERR_Pos 5 /* CHN_PARITY_ERROR */`
- `#define SIO_EV_FRAMING_ERR_Pos 6 /* CHN_FRAMING_ERROR */`
- `#define SIO_EV_NOISE_ERR_Pos 7 /* CHN_NOISE_ERROR */`
- `#define SIO_EV_OVERRUN_ERR_Pos 8 /* CHN_OVERRUN_ERROR */`
- `#define SIO_EV_RXIDLE_Pos 9 /* CHN_IDLE_DETECTED */`
- `#define SIO_EV_RXBREAK_Pos 10 /* CHN_BREAK_DETECTED */`

Event flags (compatible with channel and serial events)

- `#define SIO_EV_NONE 0U`
- `#define SIO_EV_RXNOTEMPTY (1U << SIO_EV_RXNOTEMPTY_POS)`
- `#define SIO_EV_TXNOTFULL (1U << SIO_EV_TXNOTFULL_POS)`
- `#define SIO_EV_ALL_DATA (SIO_EV_RXNOTEMPTY | SIO_EV_TXNOTFULL)`
- `#define SIO_EV_TXDONE (1U << SIO_EV_TXDONE_POS)`
- `#define SIO_EV_PARITY_ERR (1U << SIO_EV_PARITY_ERR_POS)`
- `#define SIO_EV_FRAMING_ERR (1U << SIO_EV_FRAMING_ERR_POS)`
- `#define SIO_EV_NOISE_ERR (1U << SIO_EV_NOISE_ERR_POS)`
- `#define SIO_EV_OVERRUN_ERR (1U << SIO_EV_OVERRUN_ERR_POS)`
- `#define SIO_EV_RXIDLE (1U << SIO_EV_RXIDLE_POS)`
- `#define SIO_EV_RXBREAK (1U << SIO_EV_RXBREAK_POS)`
- `#define SIO_EV_ALL_ERRORS`
- `#define SIO_EV_ALL_EVENTS`

Additional messages

- `#define SIO_MSG_ERRORS 1`

SIO configuration options

- `#define SIO_DEFAULT_BITRATE 38400`
  Default bit rate.
- `#define SIO_USE_SYNCHRONIZATION TRUE`
  Support for thread synchronization API.
- `#define SIO_USE_STREAMS_INTERFACE SIO_USE_SYNCHRONIZATION`
  Support for streams interface.

Low level driver helper macros

- `#define __sio_callback(siop)`
  SIO callback.
- `#define __sio_wakeup_errors(siop)`
  Wakes up because RX errors.
- `#define __sio_wakeup_rx(siop)`
  Wakes up the RX-waiting thread.
- `#define __sio_wakeup_rxidle(siop)`
  Wakes up the RX-idle-waiting thread.
- `#define __sio_wakeup_tx(siop)`
  Wakes up the TX-waiting thread.
- `#define __sio_wakeup_txend(siop)`
  Wakes up the TXend-waiting thread.
Typedefs

- **typedef eventflags_t sioevents_t**
  Type of event flags.
- **typedef struct hal_sio_driver SIODriver**
  Type of structure representing a SIO driver.
- **typedef struct hal_sio_config SIOConfig**
  Type of structure representing a SIO configuration.
- **typedef void(siocb_t)(SIODriver *siop)**
  Generic SIO notification callback type.

Enumerations

- **enum siostate_t { SIO_UNINIT = 0, SIO_STOP = 1, SIO_READY = 2 }**
  Driver state machine possible states.

Functions

- **void sioInit (void)**
  SIO Driver initialization.
- **void sioObjectInit (SIODriver *siop)**
  Initializes the standard part of a SIODriver structure.
- **msg_t sioStart (SIODriver *siop, const SIOConfig *config)**
  Configures and activates the SIO peripheral.
- **void sioStop (SIODriver *siop)**
  Deactivates the SIO peripheral.
- **void sioWriteEnableFlags (SIODriver *siop, sioevents_t mask)**
  Writes the enabled events flags mask.
- **void sioSetEnableFlags (SIODriver *siop, sioevents_t mask)**
  Sets flags into the enabled events flags mask.
- **void sioClearEnableFlags (SIODriver *siop, sioevents_t mask)**
  Clears flags from the enabled events flags mask.
- **sioevents_t sioGetAndClearErrors (SIODriver *siop)**
  Get and clears SIO error event flags.
- **sioevents_t sioGetAndClearEvents (SIODriver *siop)**
  Get and clears SIO event flags.
- **sioevents_t sioGetEvents (SIODriver *siop)**
  Returns the pending SIO event flags.
- **size_t sioAsyncRead (SIODriver *siop, uint8_t *buffer, size_t n)**
  Reads data from the RX FIFO.
- **size_t sioAsyncWrite (SIODriver *siop, const uint8_t *buffer, size_t n)**
  Writes data into the TX FIFO.
- **msg_t sioSynchronizeRX (SIODriver *siop, sysinterval_t timeout)**
  Synchronizes with RX FIFO data availability.
- **msg_t sioSynchronizeRXIdle (SIODriver *siop, sysinterval_t timeout)**
  Synchronizes with RX going idle.
- **msg_t sioSynchronizeTX (SIODriver *siop, sysinterval_t timeout)**
  Synchronizes with TX FIFO space availability.
- **msg_t sioSynchronizeTXEnd (SIODriver *siop, sysinterval_t timeout)**
  Synchronizes with TX completion.
9.95.1 Detailed Description

SIO Driver macros and structures.

9.96 hal_sio_lld.c File Reference

PLATFORM SIO subsystem low level driver source.

#include "hal.h"

Functions

- void sio_lld_init (void)
  Low level SIO driver initialization.
- msg_t sio_lld_start (SIODriver *siop)
  Configures and activates the SIO peripheral.
- void sio_lld_stop (SIODriver *siop)
  Deactivates the SIO peripheral.
- void sio_lld_update_enable_flags (SIODriver *siop)
  Enable flags change notification.
- sioevents_t sio_lld_get_and_clear_errors (SIODriver *siop)
  Get and clears SIO error event flags.
- sioevents_t sio_lld_get_and_clear_events (SIODriver *siop)
  Get and clears SIO event flags.
- sioevents_t sio_lld_get_events (SIODriver *siop)
  Returns the pending SIO event flags.
- size_t sio_lld_read (SIODriver *siop, uint8_t *buffer, size_t n)
  Reads data from the RX FIFO.
- size_t sio_lld_write (SIODriver *siop, const uint8_t *buffer, size_t n)
  Writes data into the TX FIFO.
- msg_t sio_lld_get (SIODriver *siop)
  Returns one frame from the RX FIFO.
- void sio_lld_put (SIODriver *siop, uint_fast16_t data)
  Pushes one frame into the TX FIFO.
- msg_t sio_lld_control (SIODriver *siop, unsigned int operation, void *arg)
  Control operation on a serial port.
- void sio_lld_serve_interrupt (SIODriver *siop)
  Serves an UART interrupt.

Variables

- SIODriver SIOD1
  SIO1 driver identifier.

9.96.1 Detailed Description

PLATFORM SIO subsystem low level driver source.
9.97  hal_sio_lld.h File Reference

PLATFORM SIO subsystem low level driver header.

Macros

• #define sio_lld_driver_fields uint32_t dummy
  Low level fields of the SIO driver structure.
• #define sio_lld_config_fields uint32_t dummy
  Low level fields of the SIO configuration structure.
• #define sio_lld_is_rx_empty(siop) false
  Determines the state of the RX FIFO.
• #define sio_lld_is_rx_idle(siop) false
  Determines the activity state of the receiver.
• #define sio_lld_has_rx_errors(siop) false
  Determines if RX has pending error events to be read and cleared.
• #define sio_lld_is_tx_full(siop) false
  Determines the state of the TX FIFO.
• #define sio_lld_is_tx_ongoing(siop) false
  Determines the transmission state.

PLATFORM configuration options

• #define PLATFORM_SIO_USE_SIO1 FALSE
  SIO driver enable switch.

Functions

• void sio_lld_init (void)
  Low level SIO driver initialization.
• msg_t sio_lld_start (SIODriver *siop)
  Configures and activates the SIO peripheral.
• void sio_lld_stop (SIODriver *siop)
  Deactivates the SIO peripheral.
• void sio_lld_update_enable_flags (SIODriver *siop)
  Enable flags change notification.
• sioevents_t sio_lld_get_and_clear_errors (SIODriver *siop)
  Get and clears SIO error event flags.
• sioevents_t sio_lld_get_and_clear_events (SIODriver *siop)
  Get and clears SIO event flags.
• sioevents_t sio_lld_get_events (SIODriver *siop)
  Returns the pending SIO event flags.
• size_t sio_lld_read (SIODriver *siop, uint8_t *buffer, size_t n)
  Reads data from the RX FIFO.
• size_t sio_lld_write (SIODriver *siop, const uint8_t *buffer, size_t n)
  Writes data into the TX FIFO.
• msg_t sio_lld_get (SIODriver *siop)
  Returns one frame from the RX FIFO.
• void sio_lld_put (SIODriver *siop, uint_fast16_t data)
  Pushes one frame into the TX FIFO.
• msg_t sio_lld_control (SIODriver *siop, unsigned int operation, void *arg)
  Control operation on a serial port.
• void sio_lld_serve_interrupt (SIODriver *siop)
  Serves an UART interrupt.
9.97.1 Detailed Description

PLATFORM SIO subsystem low level driver header.

9.98 hal_spi.c File Reference

SPI Driver selector code.

#include "hal.h"
#include "hal_spi_v2.inc"
#include "hal_spi_v1.inc"

9.98.1 Detailed Description

SPI Driver selector code.

9.99 hal_spi.h File Reference

SPI Driver selector macros and structures.

#include "hal_spi_v2.h"
#include "hal_spi_v1.h"

9.99.1 Detailed Description

SPI Driver selector macros and structures.

9.100 hal_spi_lld.c File Reference

PLATFORM SPI (v1) subsystem low level driver source.

#include "hal.h"
Functions

- void spi_lld_init (void)
  
  Low level SPI driver initialization.
- void spi_lld_start (SPIDriver *spip)
  
  Configures and activates the SPI peripheral.
- void spi_lld_stop (SPIDriver *spip)
  
  Deactivates the SPI peripheral.
- void spi_lld_select (SPIDriver *spip)
  
  Asserts the slave select signal and prepares for transfers.
- void spi_lld_unselect (SPIDriver *spip)
  
  Deasserts the slave select signal.
- void spi_lld_ignore (SPIDriver *spip, size_t n)
  
  Ignores data on the SPI bus.
- void spi_lld_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  
  Exchanges data on the SPI bus.
- void spi_lld_send (SPIDriver *spip, size_t n, const void *txbuf)
  
  Sends data over the SPI bus.
- void spi_lld_receive (SPIDriver *spip, size_t n, void *rxbuf)
  
  Receives data from the SPI bus.
- void spi_lld_abort (SPIDriver *spip)
  
  Aborts the ongoing SPI operation, if any.
- uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)
  
  Exchanges one frame using a polled wait.

Variables

- SPIDriver SPID1
  
  SPI1 driver identifier.

9.100.1 Detailed Description

PLATFORM SPI (v1) subsystem low level driver source.

9.101 hal_spi_lld.h File Reference

PLATFORM SPI (v1) subsystem low level driver header.

Macros

- #define SPI_SUPPORTS_CIRCULAR TRUE
  
  Circular mode support flag.
- #define spi_lld_driver_fields
  
  Low level fields of the SPI driver structure.
- #define spi_lld_config_fields
  
  Low level fields of the SPI configuration structure.

PLATFORM configuration options

- #define PLATFORM_SPI_USE_SPI1 FALSE
  
  SPI1 driver enable switch.
Functions

- void spi_lld_init (void)
  Low level SPI driver initialization.
- void spi_lld_start (SPIDriver *spip)
  Configures and activates the SPI peripheral.
- void spi_lld_stop (SPIDriver *spip)
  Deactivates the SPI peripheral.
- void spi_lld_select (SPIDriver *spip)
  Asserts the slave select signal and prepares for transfers.
- void spi_lld_unselect (SPIDriver *spip)
  Deasserts the slave select signal.
- void spi_lld_ignore (SPIDriver *spip, size_t n)
  Ignores data on the SPI bus.
- void spi_lld_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  Exchanges data on the SPI bus.
- void spi_lld_send (SPIDriver *spip, size_t n, const void *txbuf)
  Sends data over the SPI bus.
- void spi_lld_receive (SPIDriver *spip, size_t n, void *rxbuf)
  Receives data from the SPI bus.
- void spi_lld_abort (SPIDriver *spip)
  Aborts the ongoing SPI operation, if any.
- uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)
  Exchanges one frame using a polled wait.

9.101.1 Detailed Description

PLATFORM SPI (v1) subsystem low level driver header.

9.102 hal_spi_v1.h File Reference

SPI (v1) Driver macros and structures.

#include "hal_spi_lld.h"

Data Structures

- struct hal_spi_config
  Driver configuration structure.
- struct hal_spi_driver
  Structure representing an SPI driver.
Macros

Chip Select modes

- `#define SPI_SELECT_MODE_NONE`
- `#define SPI_SELECT_MODE_PAD` 1 /** @brief Legacy mode. */
- `#define SPI_SELECT_MODE_PORT` 2 /** @brief Fastest mode. */
- `#define SPI_SELECT_MODE_LINE` 3 /** @brief Packed mode. */
- `#define SPI_SELECT_MODE_LLD` 4 /** @brief LLD-defined mode. */

SPI configuration options

- `#define SPI_USE_WAIT TRUE`
  Enables synchronous APIs.
- `#define SPI_USE_MUTUAL EXCLUSION TRUE`
  Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.
- `#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD`
  Handling method for SPI CS line.

Macro Functions

- `#define spiIsBufferComplete(spip) ((bool)((spip)->state == SPI_COMPLETE))`
  Buffer state.
- `#define spiSelectI(spip)`
  Asserts the slave select signal and prepares for transfers.
- `#define spiUnselectI(spip)`
  Deasserts the slave select signal.
- `#define spiStartIgnoreI(spip, n)`
  Ignores data on the SPI bus.
- `#define spiStartExchangeI(spip, n, txbuf, rxbuf)`
  Exchanges data on the SPI bus.
- `#define spiStartSendI(spip, n, txbuf)`
  Sends data over the SPI bus.
- `#define spiStartReceiveI(spip, n, rxbuf)`
  Receives data from the SPI bus.
- `#define spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)`
  Exchanges one frame using a polled wait.

Low level driver helper macros

- `#define spi_wakeup_isr(spip)`
  Wakes up the waiting thread.
- `#define spi_isr_code(spip)`
  Common ISR code when circular mode is not supported.
- `#define spi_isr_half_code(spip)`
  Half buffer filled ISR code in circular mode.
- `#define spi_isr_full_code(spip)`
  Full buffer filled ISR code in circular mode.

Typedefs

- `typedef struct hal_spi_driver SPIDriver`
  Type of a structure representing an SPI driver.
- `typedef struct hal_spi_config SPIConfig`
  Type of a SPI driver configuration structure.
- `typedef void(*spicallback_t)(SPIDriver *spip)`
  SPI notification callback type.
Enumerations

- enum spistate_t {
  SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,
  SPI_COMPLETE = 4, SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2,
  SPI_ACTIVE = 3, SPI_COMPLETE = 4 }

  Driver state machine possible states.

9.102.1 Detailed Description

SPI (v1) Driver macros and structures.

9.103 hal_spi_v2.h File Reference

SPI (v2) Driver macros and structures.

#include "hal_spi_v2_lld.h"

Data Structures

- struct hal_spi_config
  Driver configuration structure.
- struct hal_spi_driver
  Structure representing an SPI driver.

Macros

Chip Select modes

- #define SPI_SELECT_MODE_NONE
- #define SPI_SELECT_MODE_PAD 1 /**< @brief Legacy mode. */
- #define SPI_SELECT_MODE_PORT 2 /**< @brief Fastest mode. */
- #define SPI_SELECT_MODE_LINE 3 /**< @brief Packed mode. */
- #define SPI_SELECT_MODE_LLD 4 /**< @brief LLD-defined mode. */

SPI configuration options

- #define SPI_USE_SYNCHRONIZATION FALSE
  Support for thread synchronization API.
- #define SPI_USE_ASSERT_ON_ERROR TRUE
  Inserts an assertion on function errors before returning.
- #define SPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the spiAcquireBus() and spiReleaseBus() APIs.
- #define SPI_SELECT_MODE SPI_SELECT_MODE_PAD
  Handling method for SPI CS line.

Macro Functions

- #define spIsBufferComplete(spip) ((bool)((spip)->state == SPI_COMPLETE))
Buffer state.
• #define spiSelectI(spip)
  Asserts the slave select signal and prepares for transfers.
• #define spiUnselectI(spip)
  Deasserts the slave select signal.
• #define spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)
  Exchanges one frame using a polled wait.
• #define spiAbortI(spip) spiStopTransferI(spip, NULL)
  Compatibility API with SPI driver v1.
• #define spiAbort(spip) spiStopTransfer(spip, NULL)
  Compatibility API with SPI driver v1.

Low level driver helper macros
• #define __spi_wakeup_isr(spip, msg)
  Wakes up the waiting thread.
• #define __spi_isr_complete_code(spip)
  Common ISR code in linear mode.
• #define __spi_isr_half_code(spip)
  Half buffer filled ISR code in circular mode.
• #define __spi_isr_full_code(spip)
  Full buffer filled ISR code in circular mode.
• #define __spi_isr_error_code(spip, msg)
  ISR error reporting code.

Typedefs
• typedef struct hal_spi_driver SPIDriver
  Type of a structure representing an SPI driver.
• typedef struct hal_spi_config SPIConfig
  Type of a SPI driver configuration structure.
• typedef void(* spicb_t)(SPIDriver *spip)
  SPI notification callback type.

Enumerations
• enum spistate_t {
  SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,
  SPI_COMPLETE = 4, SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2,
  SPI_ACTIVE = 3, SPI_COMPLETE = 4 }
  Driver state machine possible states.

9.103.1 Detailed Description
SPI (v2) Driver macros and structures.

9.104 hal_spi_v2_lld.c File Reference
PLATFORM SPI (v2) subsystem low level driver source.
#include "hal.h"
Functions

- void spi_lld_init (void)
  Low level SPI driver initialization.
- msg_t spi_lld_start (SPIDriver *spip)
  Configures and activates the SPI peripheral.
- void spi_lld_stop (SPIDriver *spip)
  Deactivates the SPI peripheral.
- void spi_lld_select (SPIDriver *spip)
  Asserts the slave select signal and prepares for transfers.
- void spi_lld_unselect (SPIDriver *spip)
  Deasserts the slave select signal.
- msg_t spi_lld_ignore (SPIDriver *spip, size_t n)
  Ignores data on the SPI bus.
- msg_t spi_lld_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  Exchanges data on the SPI bus.
- msg_t spi_lld_send (SPIDriver *spip, size_t n, const void *txbuf)
  Sends data over the SPI bus.
- msg_t spi_lld_receive (SPIDriver *spip, size_t n, void *rxbuf)
  Receives data from the SPI bus.
- msg_t spi_lld_stop_transfer (SPIDriver *spip, size_t *sizep)
  Aborts the ongoing SPI operation, if any.
- uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)
  Exchanges one frame using a polled wait.

Variables

- SPIDriver SPID1
  SPI1 driver identifier.

9.104.1 Detailed Description

PLATFORM SPI (v2) subsystem low level driver source.

9.105 hal_spi_v2_lld.h File Reference

PLATFORM SPI (v2) subsystem low level driver header.

Macros

- #define SPI_SUPPORTS_CIRCULAR TRUE
  Circular mode support flag.
- #define SPI_SUPPORTS_SLAVE_MODE TRUE
  Slave mode support flag.
- #define spi_lld_driver_fields
  Low level fields of the SPI driver structure.
- #define spi_lld_config_fields
  Low level fields of the SPI configuration structure.

PLATFORM configuration options

- #define PLATFORM_SPI_USE_SPI1 FALSE
  SPI1 driver enable switch.
Functions

- void spi_lld_init (void)
  
  Low level SPI driver initialization.

- msg_t spi_lld_start (SPIDriver *spip)
  
  Configures and activates the SPI peripheral.

- void spi_lld_stop (SPIDriver *spip)
  
  Deactivates the SPI peripheral.

- void spi_lld_select (SPIDriver *spip)
  Asserts the slave select signal and prepares for transfers.

- void spi_lld_unselect (SPIDriver *spip)
  Deasserts the slave select signal.

- msg_t spi_lld_ignore (SPIDriver *spip, size_t n)
  Ignores data on the SPI bus.

- msg_t spi_lld_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  Exchanges data on the SPI bus.

- msg_t spi_lld_send (SPIDriver *spip, size_t n, const void *txbuf)
  Sends data over the SPI bus.

- msg_t spi_lld_receive (SPIDriver *spip, size_t n, void *rxbuf)
  Receives data from the SPI bus.

- msg_t spi_lld_stop_transfer (SPIDriver *spip, size_t *sizep)
  aborts the ongoing SPI operation, if any.

- uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)
  Exchanges one frame using a polled wait.

9.105.1 Detailed Description

PLATFORM SPI (v2) subsystem low level driver header.

9.106 hal_st.c File Reference

ST Driver code.

#include "hal.h"

Functions

- void stInit (void)
  
  ST Driver initialization.

- void stBind (void)
  Enables an alarm interrupt on the invoking core.

- systime_t stGetCounter (void)
  Returns the time counter value.

- void stStartAlarm (systime_t abstime)
  Starts the alarm zero.

- void stStopAlarm (void)
  Stops the alarm zero interrupt.
9.107 hal_st.h File Reference

- void stSetAlarm (systime_t abstime)
  *Sets the alarm zero time.*
- systime_t stGetAlarm (void)
  *Returns the alarm zero current time.*
- bool stsAlarmActive (void)
  *Determines if the alarm zero is active.*
- void stSetCallback (unsigned alarm, st_callback_t cb)
  *Associates a callback to an alarm.*
- void stBindAlarmN (unsigned alarm)
  *Enables an alarm interrupt on the invoking core.*
- void stStartAlarmN (unsigned alarm, systime_t abstime)
  *Starts an additional alarm.*
- void stStopAlarmN (unsigned alarm)
  *Stops an additional alarm.*
- void stSetAlarmN (unsigned alarm, systime_t abstime)
  *Sets an additional alarm time.*
- systime_t stGetAlarmN (unsigned alarm)
  *Returns an additional alarm current time.*
- bool stsAlarmActiveN (unsigned alarm)
  *Determines if the specified alarm is active.*

Variables

- st_callback_t st_callbacks [ST_LLD_NUM_ALARMS]
  *Callback pointers for each alarm.*

9.106.1 Detailed Description

ST Driver code.

9.107 hal_st.h File Reference

ST Driver macros and structures.

#include "hal_st_lld.h"

Functions

- void stInit (void)
  *ST Driver initialization.*
- systime_t stGetCounter (void)
  *Returns the time counter value.*
- void stStartAlarm (systime_t abstime)
  *Starts the alarm zero.*
- void stStopAlarm (void)
  *Stops the alarm zero interrupt.*
- void stSetAlarm (systime_t abstime)
  *Sets the alarm zero time.*
- systime_t stGetAlarm (void)
  *Returns the alarm zero current time.*
- bool stsAlarmActive (void)
  *Determines if the alarm zero is active.*
9.107.1 Detailed Description

ST Driver macros and structures.

This header is designed to be include-able without having to include other files from the HAL.

9.108 hal_st_lld.c File Reference

PLATFORM ST subsystem low level driver source.

```c
#include "hal.h"
```

Functions

- void st_lld_init (void)
  
  *Low level ST driver initialization.*

9.108.1 Detailed Description

PLATFORM ST subsystem low level driver source.

9.109 hal_st_lld.h File Reference

PLATFORM ST subsystem low level driver header.

Functions

- void st_lld_init (void)
  
  *Low level ST driver initialization.*
- static systime_t st_lld_get_counter (void)
  
  *Returns the time counter value.*
- static void st_lld_start_alarm (systime_t abstime)
  
  *Starts the alarm.*
- static void st_lld_stop_alarm (void)
  
  *Stops the alarm interrupt.*
- static void st_lld_set_alarm (systime_t abstime)
  
  *Sets the alarm time.*
- static systime_t st_lld_get_alarm (void)
  
  *Returns the current alarm time.*
- static bool st_lld_is_alarm_active (void)
  
  *Determines if the alarm is active.*
9.109.1 Detailed Description

PLATFORM ST subsystem low level driver header.

This header is designed to be include-able without having to include other files from the HAL.

9.110 hal_streams.h File Reference

Data streams.

Data Structures

- struct BaseSequentialStreamVMT
  BaseSequentialStream virtual methods table.
- struct BaseSequentialStream
  Base stream class.
- struct BaseBufferedStreamVMT
  BaseBufferedStream virtual methods table.
- struct BaseBufferedStream
  Buffered stream class.

Macros

- #define _base_sequential_stream_methods
  BaseSequentialStream specific methods.
- #define _base_sequential_stream_data _base_object_data
  BaseSequentialStream specific data.
- #define _base_buffered_stream_methods
  BaseBufferedStream specific methods.
- #define _base_buffered_stream_data _base_sequential_stream_data
  BaseBufferedStream specific data.

Streams return codes

- #define STM_OK MSG_OK
- #define STM_TIMEOUT MSG_TIMEOUT
- #define STM_RESET MSG_RESET

Macro Functions (BaseSequentialStream)

- #define streamWrite(ip, bp, n) ((ip)->vmt->write(ip, bp, n))
  Sequential Stream write.
- #define streamRead(ip, bp, n) ((ip)->vmt->read(ip, bp, n))
  Sequential Stream read.
- #define streamPut(ip, b) ((ip)->vmt->put(ip, b))
  Sequential Stream blocking byte write.
- #define streamGet(ip) ((ip)->vmt->get(ip))
  Sequential Stream blocking byte read.

Macro Functions (BaseBufferedStream)

- #define streamUnget(ip, b) ((ip)->vmt->unget(ip, b))
  Buffered Stream unget.
9.110.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

9.111 hal_trng.c File Reference

TRNG Driver code.

#include "hal.h"

Functions

• void trngInit (void)
  TRNG Driver initialization.
• void trngObjectInit (TRNGDriver ∗trngp)
  Initializes the standard part of a TRNGDriver structure.
• msg_t trngStart (TRNGDriver ∗trngp, const TRNGConfig ∗config)
  Configures and activates the TRNG peripheral.
• void trngStop (TRNGDriver ∗trngp)
  Deactivates the TRNG peripheral.
• bool trngGenerate (TRNGDriver ∗trngp, size_t size, uint8_t ∗out)
  True random numbers generator.

9.111.1 Detailed Description

TRNG Driver code.

9.112 hal_trng.h File Reference

TRNG Driver macros and structures.

#include "hal_trng_lld.h"

Data Structures

• struct hal_trng_config
  Driver configuration structure.
• struct hal_trng_driver
  Structure representing a TRNG driver.
Typedefs

- typedef struct hal_trng_driver TRNGDriver
  
  Type of a structure representing a TRNG driver.

- typedef struct hal_trng_config TRNGConfig
  
  Driver configuration structure.

Enumerations

- enum trngstate_t { TRNG_UNINIT = 0, TRNG_STOP = 1, TRNG_READY = 2, TRNG_RUNNING = 3 }
  
  Driver state machine possible states.

Functions

- void trngInit (void)
  
  TRNG Driver initialization.

- void trngObjectInit (TRNGDriver *trngp)
  
  Initializes the standard part of a TRNGDriver structure.

- msg_t trngStart (TRNGDriver *trngp, const TRNGConfig *config)
  
  Configures and activates the TRNG peripheral.

- void trngStop (TRNGDriver *trngp)
  
  Deactivates the TRNG peripheral.

- bool trngGenerate (TRNGDriver *trngp, size_t size, uint8_t *out)
  
  True random numbers generator.

9.112.1 Detailed Description

TRNG Driver macros and structures.

9.113 hal_trng_lld.c File Reference

PLATFORM TRNG subsystem low level driver source.

#include "hal.h"

Functions

- void trng_lld_init (void)
  
  Low level TRNG driver initialization.

- void trng_lld_start (TRNGDriver *trngp)
  
  Configures and activates the TRNG peripheral.

- void trng_lld_stop (TRNGDriver *trngp)
  
  Deactivates the TRNG peripheral.

- bool trng_lld_generate (TRNGDriver *trngp, size_t size, uint8_t *out)
  
  True random numbers generator.
Variables

- TRNGDriver TRNGD1
  TRNGD1 driver identifier.

9.113.1 Detailed Description

PLATFORM TRNG subsystem low level driver source.

9.114 hal_trng_lld.h File Reference

PLATFORM TRNG subsystem low level driver header.

Macros

- #define trng_lld_driver_fields
  Low level fields of the TRNG driver structure.
- #define trng_lld_config_fields
  Low level fields of the TRNG configuration structure.

PLATFORM configuration options

- #define PLATFORM_TRNG_USE_TRNG1 FALSE
  TRNGD1 driver enable switch.

Functions

- void trng_lld_init (void)
  Low level TRNG driver initialization.
- void trng_lld_start (TRNGDriver *trngp)
  Configures and activates the TRNG peripheral.
- void trng_lld_stop (TRNGDriver *trngp)
  Deactivates the TRNG peripheral.
- bool trng_lld_generate (TRNGDriver *trngp, size_t size, uint8_t *out)
  True random numbers generator.

9.114.1 Detailed Description

PLATFORM TRNG subsystem low level driver header.

9.115 hal_uart.c File Reference

UART Driver code.

#include "hal.h"
Functions

- void uartInit (void)
  UART Driver initialization.
- void uartObjectInit (UARTDriver *uartp)
  Initializes the standard part of a UARTDriver structure.
- msg_t uartStart (UARTDriver *uartp, const UARTConfig *config)
  Configures and activates the UART peripheral.
- void uartStop (UARTDriver *uartp)
  Deactivates the UART peripheral.
- void uartStartSend (UARTDriver *uartp, size_t n, const void *txbuf)
  Starts a transmission on the UART peripheral.
- void uartStartSendI (UARTDriver *uartp, size_t n, const void *txbuf)
  Starts a transmission on the UART peripheral.
- size_t uartStopSend (UARTDriver *uartp)
  Stops any ongoing transmission.
- size_t uartStopSendI (UARTDriver *uartp)
  Stops any ongoing transmission.
- void uartStartReceive (UARTDriver *uartp, size_t n, void *rxbuf)
  Starts a receive operation on the UART peripheral.
- void uartStartReceiveI (UARTDriver *uartp, size_t n, void *rxbuf)
  Starts a receive operation on the UART peripheral.
- size_t uartStopReceive (UARTDriver *uartp)
  Stops any ongoing receive operation.
- size_t uartStopReceiveI (UARTDriver *uartp)
  Stops any ongoing receive operation.
- msg_t uartSendTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)
  Performs a transmission on the UART peripheral.
- msg_t uartSendFullTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)
  Performs a transmission on the UART peripheral.
- msg_t uartReceiveTimeout (UARTDriver *uartp, size_t *np, void *rxbuf, sysinterval_t timeout)
  Performs a receive operation on the UART peripheral.
- void uartAcquireBus (UARTDriver *uartp)
  Gains exclusive access to the UART bus.
- void uartReleaseBus (UARTDriver *uartp)
  Releases exclusive access to the UART bus.

9.115.1 Detailed Description

UART Driver code.

9.116 hal_uart.h File Reference

UART Driver macros and structures.

#include "hal_uart_lld.h"
Macros

UART status flags

• #define UART_NO_ERROR 0
  No pending conditions.

• #define UART_PARITY_ERROR 4
  Parity error happened.

• #define UART_FRAMING_ERROR 8
  Framing error happened.

• #define UART_OVERRUN_ERROR 16
  Overflow happened.

• #define UART_NOISE_ERROR 32
  Noise on the line.

• #define UART_BREAK_DETECTED 64
  Break detected.

UART error conditions

• #define UART_ERR_NOT_ACTIVE (size_t)-1

UART configuration options

• #define UART_USE_WAIT FALSE
  Enables synchronous APIs.

• #define UART_USE_MUTUAL_EXCLUSION FALSE
  Enables the uartAcquireBus() and uartReleaseBus() APIs.

Low level driver helper macros

• #define _uart_wakeup_tx1_isr(uartp)
  Wakes up the waiting thread in case of early TX complete.

• #define _uart_wakeup_tx2_isr(uartp)
  Wakes up the waiting thread in case of late TX complete.

• #define _uart_wakeup_rx_complete_isr(uartp)
  Wakes up the waiting thread in case of RX complete.

• #define _uart_wakeup_rx_error_isr(uartp)
  Wakes up the waiting thread in case of RX error.

• #define _uart_wakeup_rx_cm_isr(uartp)
  Wakes up the waiting thread in case of RX character match.

• #define _uart_wakeup_rx_timeout_isr(uartp)
  Wakes up the waiting thread in case of RX timeout.

• #define _uart_tx1_isr_code(uartp)
  Common ISR code for early TX.

• #define _uart_tx2_isr_code(uartp)
  Common ISR code for late TX.

• #define _uart_rx_complete_isr_code(uartp)
  Common ISR code for RX complete.

• #define _uart_rx_error_isr_code(uartp, errors)
  Common ISR code for RX error.

• #define _uart_rx_idle_code(uartp)
  Common ISR code for RX on idle.

• #define _uart_timeout_isr_code(uartp)
  Timeout ISR code for receiver.

• #define _uart_rx_char_match_isr_code(uartp)
  Character match ISR code for receiver.
Enumerations

- `enum uartstate_t` { UART_UNINIT = 0, UART_STOP = 1, UART_READY = 2 }
  Driver state machine possible states.
- `enum uartxstate_t` { UART_TX_IDLE = 0, UART_TX_ACTIVE = 1, UART_TX_COMPLETE = 2 }
  Transmitter state machine states.
- `enum uartrxstate_t` { UART_RX_IDLE = 0, UART_RX_ACTIVE = 1, UART_RX_COMPLETE = 2 }
  Receiver state machine states.

Functions

- `void uartInit (void)`
  UART Driver initialization.
- `void uartObjectInit (UARTDriver *uartp)`
  Initializes the standard part of a UARTDriver structure.
- `msg_t uartStart (UARTDriver *uartp, const UARTConfig *config)`
  Configures and activates the UART peripheral.
- `void uartStop (UARTDriver *uartp)`
  Deactivates the UART peripheral.
- `void uartStartSend (UARTDriver *uartp, size_t n, const void *txbuf)`
  Starts a transmission on the UART peripheral.
- `void uartStartSendI (UARTDriver *uartp, size_t n, const void *txbuf)`
  Starts a transmission on the UART peripheral.
- `size_t uartStopSend (UARTDriver *uartp)`
  Stops any ongoing transmission.
- `size_t uartStopSendI (UARTDriver *uartp)`
  Stops any ongoing transmission.
- `void uartStartReceive (UARTDriver *uartp, size_t n, void *rxbuf)`
  Starts a receive operation on the UART peripheral.
- `void uartStartReceiveI (UARTDriver *uartp, size_t n, void *rxbuf)`
  Starts a receive operation on the UART peripheral.
- `size_t uartStopReceive (UARTDriver *uartp)`
  Stops any ongoing receive operation.
- `size_t uartStopReceiveI (UARTDriver *uartp)`
  Stops any ongoing receive operation.
- `msg_t uartSendTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)`
  Performs a transmission on the UART peripheral.
- `msg_t uartSendFullTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)`
  Performs a transmission on the UART peripheral.
- `msg_t uartReceiveTimeout (UARTDriver *uartp, size_t *np, void *rxbuf, sysinterval_t timeout)`
  Performs a receive operation on the UART peripheral.
- `void uartAcquireBus (UARTDriver *uartp)`
  Gains exclusive access to the UART bus.
- `void uartReleaseBus (UARTDriver *uartp)`
  Releases exclusive access to the UART bus.

9.116.1 Detailed Description

UART Driver macros and structures.
9.117  hal_uart_lld.c File Reference

PLATFORM UART subsystem low level driver source.

#include "hal.h"

Functions

• void uart_lld_init (void)
  Low level UART driver initialization.
• void uart_lld_start (UARTDriver *uartp)
  Configures and activates the UART peripheral.
• void uart_lld_stop (UARTDriver *uartp)
  Deactivates the UART peripheral.
• void uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)
  Starts a transmission on the UART peripheral.
• size_t uart_lld_stop_send (UARTDriver *uartp)
  Stops any ongoing transmission.
• void uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rxbuf)
  Starts a receive operation on the UART peripheral.
• size_t uart_lld_stop_receive (UARTDriver *uartp)
  Stops any ongoing receive operation.

Variables

• UARTDriver UARTD1
  UART1 driver identifier.

9.117.1  Detailed Description

PLATFORM UART subsystem low level driver source.

9.118  hal_uart_lld.h File Reference

PLATFORM UART subsystem low level driver header.

Data Structures

• struct hal_uart_config
  Driver configuration structure.
• struct hal_uart_driver
  Structure representing an UART driver.
Macros

PLATFORM configuration options

- \#define PLATFORM_UART_USE_UART1 FALSE
  UART driver enable switch.

TypeDefs

- typedef uint32_t uartflags_t
  UART driver condition flags type.
- typedef struct hal_uart_driver UARTDriver
  Type of structure representing an UART driver.
- typedef void (UARTDriver \* uartp)
  Generic UART notification callback type.
- typedef void (UARTDriver \* uartp, uint16_t c)
  Character received UART notification callback type.
- typedef void (UARTDriver \* uartp, uartflags_t e)
  Receive error UART notification callback type.
- typedef struct hal_uart_config UARTConfig
  Driver configuration structure.

Functions

- void uart_lld_init (void)
  Low level UART driver initialization.
- void uart_lld_start (UARTDriver \* uartp)
  Configures and activates the UART peripheral.
- void uart_lld_stop (UARTDriver \* uartp)
  Deactivates the UART peripheral.
- void uart_lld_start_send (UARTDriver \* uartp, size_t n, const void \* txbuf)
  Starts a transmission on the UART peripheral.
- size_t uart_lld_stop_send (UARTDriver \* uartp)
  Stops any ongoing transmission.
- void uart_lld_start_receive (UARTDriver \* uartp, size_t n, void \* rxbuf)
  Starts a receive operation on the UART peripheral.
- size_t uart_lld_stop_receive (UARTDriver \* uartp)
  Stops any ongoing receive operation.

9.118.1 Detailed Description

PLATFORM UART subsystem low level driver header.

9.119 hal_usb.c File Reference

USB Driver code.

#include <string.h>
#include "hal.h"
Functions

- static void set_address (USBDriver *usbp)
  SET ADDRESS transaction callback.
- static bool default_handler (USBDriver *usbp)
  Standard requests handler.
- void usbInit (void)
  USB Driver initialization.
- void usbObjectInit (USBDriver *usbp)
  Initializes the standard part of a USBDriver structure.
- msg_t usbStart (USBDriver *usbp, const USBCfg *config)
  Configures and activates the USB peripheral.
- void usbStop (USBDriver *usbp)
  Deactivates the USB peripheral.
- void usbInitEndpointI (USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)
  Enables an endpoint.
- void usbDisableEndpointsI (USBDriver *usbp)
  Disables all the active endpoints.
- void usbStartReceiveI (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)
  Starts a receive transaction on an OUT endpoint.
- void usbStartTransmitI (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)
  Starts a transmit transaction on an IN endpoint.
- msg_t usbReceive (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)
  Performs a receive transaction on an OUT endpoint.
- msg_t usbTransmit (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)
  Performs a transmit transaction on an IN endpoint.
- bool usbStallReceiveI (USBDriver *usbp, usbep_t ep)
  Stalls an OUT endpoint.
- bool usbStallTransmitI (USBDriver *usbp, usbep_t ep)
  Stalls an IN endpoint.
- void usbWakeupHost (USBDriver *usbp)
  Host wake-up procedure.
- void _usb_reset (USBDriver *usbp)
  USB reset routine.
- void _usb_suspend (USBDriver *usbp)
  USB suspend routine.
- void _usb_wakeup (USBDriver *usbp)
  USB wake-up routine.
- void _usb_ep0setup (USBDriver *usbp, usbep_t ep)
  Default EP0 SETUP callback.
- void _usb_ep0in (USBDriver *usbp, usbep_t ep)
  Default EP0 IN callback.
- void _usb_ep0out (USBDriver *usbp, usbep_t ep)
  Default EP0 OUT callback.

9.119.1 Detailed Description

USB Driver code.
9.120 hal_usb.h File Reference

USB Driver macros and structures.

```c
#include "hal_usb_lld.h"
```

**Data Structures**

- `struct USBDescriptor`
  Type of an USB descriptor.

**Macros**

- `#define USB_USE_WAIT FALSE`
  Enables synchronous APIs.

**Helper macros for USB descriptors**

- `#define USB_DESC_INDEX(i) ((uint8_t)i)`
  Helper macro for index values into descriptor strings.
- `#define USB_DESC_BYTE(b) ((uint8_t)b)`
  Helper macro for byte values into descriptor strings.
- `#define USB_DESC_WORD(w)`
  Helper macro for word values into descriptor strings.
- `#define USB_DESC_BCD(bcd)`
  Helper macro for BCD values into descriptor strings.
- `#define USB_DESC_DEVICE_SIZE 18U`
  Device Descriptor helper macro.
- `#define USB_DESC_CONFIGURATION_SIZE 9U`
  Configuration Descriptor size.
- `#define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)`
  Configuration Descriptor helper macro.
- `#define USB_DESC_INTERFACE_SIZE 9U`
  Interface Descriptor size.
- `#define USB_DESC_INTERFACE(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)`
  Interface Descriptor helper macro.
- `#define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U`
  Interface Association Descriptor size.
- `#define USB_DESC_INTERFACE_ASSOCIATION(bFirstInterface, bInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, iInterface)`
  Interface Association Descriptor helper macro.
- `#define USB_DESC_ENDPOINT_SIZE 7U`
  Endpoint Descriptor size.
- `#define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)`
  Endpoint Descriptor helper macro.

**Endpoint types and settings**

ChibiOS/HAL
• #define USB_EP_MODE_TYPE 0x0003U
• #define USB_EP_MODE_TYPE_CTRL 0x0000U
• #define USB_EP_MODE_TYPE_ISO 0x0001U
• #define USB_EP_MODE_TYPE_BULK 0x0002U
• #define USB_EP_MODE_TYPE_INTR 0x0003U

Macro Functions

• #define usbGetDriverStateI(usbp) (((usbp)->state))
  Returns the driver state.
• #define usbConnectBus(usbp) usb_lld_connect_bus(usbp)
  Connects the USB device.
• #define usbDisconnectBus(usbp) usb_lld_disconnect_bus(usbp)
  Disconnect the USB device.
• #define usbGetFrameNumberX(usbp) usb_lld_get_frame_number(usbp)
  Returns the current frame number.
• #define usbGetTransmitStatusI(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)
  Returns the status of an IN endpoint.
• #define usbGetReceiveStatusI(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)
  Returns the status of an OUT endpoint.
• #define usbGetReceiveTransactionSizeX(usbp, ep) usb_lld_get_transaction_size(usbp, ep)
  Returns the exact size of a receive transaction.
• #define usbSetupTransfer(usbp, buf, n, endcb)
  Request transfer setup.
• #define usbReadSetup(usbp, ep, buf) usb_lld_read_setup(usbp, ep, buf)
  Reads a setup packet from the dedicated packet buffer.

Low level driver helper macros

• #define _usb_isr_invoke_event_cb(usbp, evt)
  Common ISR code, usb event callback.
• #define _usb_isr_invoke_sof_cb(usbp)
  Common ISR code, SOF callback.
• #define _usb_isr_invoke_setup_cb(usbp, ep)
  Common ISR code, setup packet callback.
• #define _usb_isr_invoke_in_cb(usbp, ep)
  Common ISR code, IN endpoint callback.
• #define _usb_isr_invoke_out_cb(usbp, ep)
  Common ISR code, OUT endpoint event.

Typedefs

• typedef struct USBDriver USBDriver
  Type of a structure representing an USB driver.
• typedef uint8_t usbep_t
  Type of an endpoint identifier.
• typedef void(* usbcallback_t) (USBDriver *usbp)
  Type of an USB generic notification callback.
• typedef void(* usbepcallback_t) (USBDriver *usbp, usbep_t ep)
  Type of an USB endpoint callback.
• typedef void(* usbeventcb_t) (USBDriver *usbp, usbevent_t event)
  Type of an USB event notification callback.
• typedef bool(* usbreqhandler_t) (USBDriver *usbp)
  Type of a requests handler callback.
Enumerations

- enum usbstate_t {
    USB_UNINIT = 0, USB_STOP = 1, USB_READY = 2, USB_SELECTED = 3,
    USB_ACTIVE = 4, USB_SUSPENDED = 5
}
Type of a driver state machine possible states.

- enum usbevent_t {
    USB_EVENT_RESET = 0, USB_EVENT_ADDRESS = 1, USB_EVENT_CONFIGURED = 2,
    USB_EVENT_UNCONFIGURED = 3,
    USB_EVENT_SUSPEND = 4, USB_EVENT_WAKEUP = 5, USB_EVENT_STALLED = 6
}
Type of an enumeration of the possible USB events.

- enum usbevent_t {
    USB_EP0_STOP_WAITING = 0U, USB_EP0_IN_TX = USB_IN_STATE | 1U, USB_EP0_IN_WAITING_TX0 =
    USB_IN_STATE | 2U, USB_EP0_IN_SENDING_STS = USB_IN_STATE | 3U,
    USB_EP0_OUT_WAITING_STS = USB_OUT_STATE | 4U, USB_EP0_OUT_RX = USB_OUT_STATE | 5U,
    USB_EP0_ERROR = 6U
}
Type of an endpoint zero state machine states.

Functions

- void usbInit (void)
    USB Driver initialization.

- void usbevent_t {
    USB_EP0_STOP_WAITING = 0U, USB_EP0_IN_TX = USB_IN_STATE | 1U, USB_EP0_IN_WAITING_TX0 =
    USB_IN_STATE | 2U, USB_EP0_IN_SENDING_STS = USB_IN_STATE | 3U,
    USB_EP0_OUT_WAITING_STS = USB_OUT_STATE | 4U, USB_EP0_OUT_RX = USB_OUT_STATE | 5U,
    USB_EP0_ERROR = 6U
}
Type of an endpoint status.

- void usbevent_t {
    USB_EVENT_RESET = 0, USB_EVENT_ADDRESS = 1, USB_EVENT_CONFIGURED = 2, USB_EVENT_UNCONFIGURED
    = 3,
    USB_EVENT_SUSPEND = 4, USB_EVENT_WAKEUP = 5, USB_EVENT_STALLED = 6
}
Type of an endpoint status.

- void usbObjectInit (USBDriver *usbp)
    Initializes the standard part of a USBDriver structure.

- void usbStart (USBDriver *usbp, const USBConfig *config)
    Configures and activates the USB peripheral.

- void usbevent_t {
    USB_EP0_STOP_WAITING = 0U, USB_EP0_IN_TX = USB_IN_STATE | 1U, USB_EP0_IN_WAITING_TX0 =
    USB_IN_STATE | 2U, USB_EP0_IN_SENDING_STS = USB_IN_STATE | 3U,
    USB_EP0_OUT_WAITING_STS = USB_OUT_STATE | 4U, USB_EP0_OUT_RX = USB_OUT_STATE | 5U,
    USB_EP0_ERROR = 6U
}
Type of an endpoint status.

- void usbStop (USBDriver *usbp)
    Deactivates the USB peripheral.

- void usbInitEndpointI (USBDriver *usbp, usbep_t ep, const USBEndpointConfig *epcp)
    Enables an endpoint.

- void usbDisableEndpointsI (USBDriver *usbp)
    Disables all the active endpoints.

- void usbStartReceiveI (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)
    Starts a receive transaction on an OUT endpoint.

- void usbStartTransmitI (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)
    Starts a transmit transaction on an IN endpoint.

- void usbReceive (USBDriver *usbp, usbep_t ep, uint8_t *buf, size_t n)
    Performs a receive transaction on an OUT endpoint.

- void usbTransmit (USBDriver *usbp, usbep_t ep, const uint8_t *buf, size_t n)
    Performs a transmit transaction on an IN endpoint.

- void usbStallReceiveI (USBDriver *usbp, usbep_t ep)
    Stalls an OUT endpoint.

- void usbStallTransmitI (USBDriver *usbp, usbep_t ep)
    Stalls an IN endpoint.

- void usbWakeupHost (USBDriver *usbp)
    Host wake-up procedure.

- void usb_reset (USBDriver *usbp)
    USB reset routine.

- void usb_suspend (USBDriver *usbp)
USB suspend routine.

- void _usb_wakeup (USBDriver *usbp)

USB wake-up routine.

- void _usb_ep0setup (USBDriver *usbp, usbep_t ep)
  Default EP0 SETUP callback.

- void _usb_ep0in (USBDriver *usbp, usbep_t ep)
  Default EP0 IN callback.

- void _usb_ep0out (USBDriver *usbp, usbep_t ep)
  Default EP0 OUT callback.

Variables

- const typedef USBDescriptor *(*usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)

  Type of an USB descriptor-retrieving callback.

9.120.1 Detailed Description

USB Driver macros and structures.

9.121 hal_usb_cdc.h File Reference

USB CDC macros and structures.

Data Structures

- struct cdc_linecoding_t

  Type of Line Coding structure.

Macros

CDC specific messages.

- #define CDC_SEND_ENCAPSULATED_COMMAND 0x00U
- #define CDC_GET_ENCAPSULATED_RESPONSE 0x01U
- #define CDC_SET_COMM_FEATURE 0x02U
- #define CDC_GET_COMM_FEATURE 0x03U
- #define CDC_CLEAR_COMM_FEATURE 0x04U
- #define CDC_SET_AUX_LINE_STATE 0x10U
- #define CDC_SET_HOOK_STATE 0x11U
- #define CDC_PULSE_SETUP 0x12U
- #define CDC_SEND_PULSE 0x13U
- #define CDC_SET_PULSE_TIME 0x14U
- #define CDC_RING_AUX_JACK 0x15U
- #define CDC_SET_LINE_CODING 0x20U
- #define CDC_GET_LINE_CODING 0x21U
- #define CDC_SET_CONTROL_LINE_STATE 0x22U
- #define CDC_SEND_BREAK 0x23U
- #define CDC_SET_RINGER_PARMS 0x30U
- #define CDC_GET_RINGER_PARMS 0x31U
• #define CDC_SET_OPERATION_PARMS 0x32U
• #define CDC_GET_OPERATION_PARMS 0x33U

CDC classes
• #define CDC_COMMUNICATION_INTERFACE_CLASS 0x02U
• #define CDC_DATA_INTERFACE_CLASS 0x0AU

CDC subclasses
• #define CDC_ABSTRACT_CONTROL_MODEL 0x02U

CDC descriptors
• #define CDC_CS_INTERFACE 0x24U

CDC subdescriptors
• #define CDC_HEADER 0x00U
• #define CDC_CALL_MANAGEMENT 0x01U
• #define CDC_ABSTRACT_CONTROL_MANAGEMENT 0x02U
• #define CDC_UNION 0x06U

Line Control bit definitions.
• #define LC_STOP_1 0U
• #define LC_STOP_1P5 1U
• #define LC_STOP_2 2U
• #define LC_PARITY_NONE 0U
• #define LC_PARITY_ODD 1U
• #define LC_PARITY_EVEN 2U
• #define LC_PARITY_MARK 3U
• #define LC_PARITY_SPACE 4U

9.121.1 Detailed Description
USB CDC macros and structures.

9.122 hal_usb_lld.c File Reference
PLATFORM USB subsystem low level driver source.

#include "hal.h"
Functions

- **void usb_lld_init (void)**
  Low level USB driver initialization.
- **void usb_lld_start (USBDriver ∗usbp)**
  Configures and activates the USB peripheral.
- **void usb_lld_stop (USBDriver ∗usbp)**
  Deactivates the USB peripheral.
- **void usb_lld_reset (USBDriver ∗usbp)**
  USB low level reset routine.
- **void usb_lld_set_address (USBDriver ∗usbp)**
  Sets the USB address.
- **void usb_lld_init_endpoint (USBDriver ∗usbp, usbep_t ep)**
  Enables an endpoint.
- **void usb_lld_disable_endpoints (USBDriver ∗usbp)**
  Disables all the active endpoints except the endpoint zero.
- **usbepstatus_t usb_lld_get_status_out (USBDriver ∗usbp, usbep_t ep)**
  Returns the status of an OUT endpoint.
- **usbepstatus_t usb_lld_get_status_in (USBDriver ∗usbp, usbep_t ep)**
  Returns the status of an IN endpoint.
- **void usb_lld_read_setup (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf)**
  Reads a setup packet from the dedicated packet buffer.
- **void usb_lld_prepare_receive (USBDriver ∗usbp, usbep_t ep)**
  Prepares for a receive operation.
- **void usb_lld_prepare Transmit (USBDriver ∗usbp, usbep_t ep)**
  Prepares for a transmit operation.
- **void usb_lld_start_out (USBDriver ∗usbp, usbep_t ep)**
  Starts a receive operation on an OUT endpoint.
- **void usb_lld_start_in (USBDriver ∗usbp, usbep_t ep)**
  Starts a transmit operation on an IN endpoint.
- **void usb_lld_stall_out (USBDriver ∗usbp, usbep_t ep)**
  Brings an OUT endpoint in the stalled state.
- **void usb_lld_stall_in (USBDriver ∗usbp, usbep_t ep)**
  Brings an IN endpoint in the stalled state.
- **void usb_lld_clear_out (USBDriver ∗usbp, usbep_t ep)**
  Brings an OUT endpoint in the active state.
- **void usb_lld_clear_in (USBDriver ∗usbp, usbep_t ep)**
  Brings an IN endpoint in the active state.

Variables

- **USBDriver USBD1**
  USB1 driver identifier.
- **union {**
  **USBInEndpointState in**
  IN EP0 state.
  **USBOutEndpointState out**
  OUT EP0 state.
  **} ep0_state**

  EP0 state.
- **static const USBEndpointConfig ep0config**
  EP0 initialization structure.
9.122.1 Detailed Description

PLATFORM USB subsystem low level driver source.

9.122.2 Variable Documentation

9.122.2.1 in

USBInEndpointState in

IN EP0 state.

9.122.2.2 out

USBOutEndpointState out

OUT EP0 state.

9.123 hal_usb_lld.h File Reference

PLATFORM USB subsystem low level driver header.

Data Structures

- struct USBInEndpointState
  Type of an IN endpoint state structure.
- struct USBOutEndpointState
  Type of an OUT endpoint state structure.
- struct USBEndpointConfig
  Type of an USB endpoint configuration structure.
- struct USBConfig
  Type of an USB driver configuration structure.
- struct USBDriver
  Structure representing an USB driver.
Macros

- `#define USB_MAX_ENDPOINTS 4`
  Maximum endpoint address.

- `#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW`
  Status stage handling method.

- `#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS`
  The address can be changed immediately upon packet reception.

- `#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW`
  Method for set address acknowledge.

- `#define usb_lld_get_frame_number(usbp) 0`
  Returns the current frame number.

- `#define usb_lld_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`
  Returns the exact size of a receive transaction.

- `#define usb_lld_connect_bus(usbp)`
  Connects the USB device.

- `#define usb_lld_disconnect_bus(usbp)`
  Disconnect the USB device.

- `#define usb_lld_wakeup_host(usbp)`
  Start of host wake-up procedure.

PLATFORM configuration options

- `#define PLATFORM_USB_USE_USB1 FALSE`
  USB driver enable switch.

Functions

- `void usb_lld_init (void)`
  Low level USB driver initialization.

- `void usb_lld_start (USBDriver *usbp)`
  Configures and activates the USB peripheral.

- `void usb_lld_stop (USBDriver *usbp)`
  Deactivates the USB peripheral.

- `void usb_lld_reset (USBDriver *usbp)`
  USB low level reset routine.

- `void usb_lld_set_address (USBDriver *usbp)`
  Sets the USB address.

- `void usb_lld_set_endpoint (USBDriver *usbp, usbep_t ep)`
  Enables an endpoint.

- `void usb_lld_disable_endpoints (USBDriver *usbp)`
  Disables all the active endpoints except the endpoint zero.

- `usbepstatus_t usb_lld_get_status_in (USBDriver *usbp, usbep_t ep)`
  Returns the status of an IN endpoint.

- `usbepstatus_t usb_lld_get_status_out (USBDriver *usbp, usbep_t ep)`
  Returns the status of an OUT endpoint.

- `void usb_lld_read_setup (USBDriver *usbp, usbep_t ep, uint8_t *buf)`
  Reads a setup packet from the dedicated packet buffer.

- `void usb_lld_prepare_receive (USBDriver *usbp, usbep_t ep)`
  Prepares for a receive operation.
• void usb_lld_prepare_transmit (USBDriver *usbp, usbep_t ep)
  Prepares for a transmit operation.
• void usb_lld_start_out (USBDriver *usbp, usbep_t ep)
  Starts a receive operation on an OUT endpoint.
• void usb_lld_start_in (USBDriver *usbp, usbep_t ep)
  Starts a transmit operation on an IN endpoint.
• void usb_lld_stall_out (USBDriver *usbp, usbep_t ep)
  Brings an OUT endpoint in the stalled state.
• void usb_lld_stall_in (USBDriver *usbp, usbep_t ep)
  Brings an IN endpoint in the stalled state.
• void usb_lld_clear_out (USBDriver *usbp, usbep_t ep)
  Brings an OUT endpoint in the active state.
• void usb_lld_clear_in (USBDriver *usbp, usbep_t ep)
  Brings an IN endpoint in the active state.

9.124 hal_wdg.c File Reference

WDG Driver code.

#include "hal.h"

Functions

• void wdgInit (void)
  WDG Driver initialization.
• msg_t wdgStart (WDGDriver *wdgp, const WDGConfig *config)
  Configures and activates the WDG peripheral.
• void wdgStop (WDGDriver *wdgp)
  Deactivates the WDG peripheral.
• void wdgReset (WDGDriver *wdgp)
  Resets WDG's counter.

9.124.1 Detailed Description

WDG Driver code.

9.125 hal_wdg.h File Reference

WDG Driver macros and structures.

#include "hal_wdg_lld.h"
Macros

- `#define wdgResetI(wdgp) wdg_lld_reset(wdgp)`
  Resets WDG's counter.

Enumerations

- `enum wdgstate_t { WDG_UNINIT = 0, WDG_STOP = 1, WDG_READY = 2 }`
  Driver state machine possible states.

Functions

- `void wdgInit (void)`
  WDG Driver initialization.
- `msg_t wdgStart (WDGDriver ∗wdgp, const WDGConfig ∗config)`
  Configures and activates the WDG peripheral.
- `void wdgStop (WDGDriver ∗wdgp)`
  Deactivates the WDG peripheral.
- `void wdgReset (WDGDriver ∗wdgp)`
  Resets WDG's counter.

9.125.1 Detailed Description

WDG Driver macros and structures.

9.126 hal_wdg_lld.c File Reference

WDG Driver subsystem low level driver source template.

#include "hal.h"

Functions

- `void wdg_lld_init (void)`
  Low level WDG driver initialization.
- `void wdg_lld_start (WDGDriver ∗wdgp)`
  Configures and activates the WDG peripheral.
- `void wdg_lld_stop (WDGDriver ∗wdgp)`
  Deactivates the WDG peripheral.
- `void wdg_lld_reset (WDGDriver ∗wdgp)`
  Reloads WDG's counter.

9.126.1 Detailed Description

WDG Driver subsystem low level driver source template.
9.127 hal_wdg_lld.h File Reference

WDG Driver subsystem low level driver header template.

Data Structures

- struct WDGConfig
  
  Driver configuration structure.

- struct WDGDriver
  
  Structure representing an WDG driver.

Macros

Configuration options

- #define PLATFORM_WDG_USE_WDG1 FALSE
  
  WDG1 driver enable switch.

Typedefs

- typedef struct WDGDriver WDGDriver
  
  Type of a structure representing an WDG driver.

Functions

- void wdg_lld_init (void)
  
  Low level WDG driver initialization.

- void wdg_lld_start (WDGDriver *wdgp)
  
  Configures and activates the WDG peripheral.

- void wdg_lld_stop (WDGDriver *wdgp)
  
  Deactivates the WDG peripheral.

- void wdg_lld_reset (WDGDriver *wdgp)
  
  Reloads WDG's counter.

9.127.1 Detailed Description

WDG Driver subsystem low level driver header template.

9.128 hal_wspi.c File Reference

WSPI Driver code.

#include "hal.h"
Functions

- void wspiInit (void)
  WSPI Driver initialization.
- void wspiObjectInit (WSPIDriver *wspip)
  Initializes the standard part of a WSPIDriver structure.
- msg_t wspiStart (WSPIDriver *wspip, const WSPICfg *config)
  Configures and activates the WSPI peripheral.
- void wspiStop (WSPIDriver *wspip)
  Deactivates the WSPI peripheral.
- void wspiStartCommand (WSPIDriver *wspip, const wspi_command_t *cmdp)
  Sends a command without data phase.
- void wspiStartSend (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
  Sends a command with data over the WSPI bus.
- void wspiStartReceive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
  Sends a command then receives data over the WSPI bus.
- bool wspiCommand (WSPIDriver *wspip, const wspi_command_t *cmdp)
  Sends a command without data phase.
- bool wspiSend (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
  Sends a command with data over the WSPI bus.
- bool wspiReceive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
  Sends a command then receives data over the WSPI bus.
- void wspiMapFlash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp)
  Maps in memory space a WSPI flash device.
- void wspiUnmapFlash (WSPIDriver *wspip)
  Unmaps from memory space a WSPI flash device.
- void wspiAcquireBus (WSPIDriver *wspip)
  Gains exclusive access to the WSPI bus.
- void wspiReleaseBus (WSPIDriver *wspip)
  Releases exclusive access to the WSPI bus.

9.128.1 Detailed Description

WSPI Driver code.

9.129 hal_wspi.h File Reference

WSPI Driver macros and structures.

#include "hal_wspi_lld.h"

Data Structures

- struct wspi_command_t
  Type of a WSPI command descriptor.
- struct hal_wspi_config
  Driver configuration structure.
- struct hal_wspi_driver
  Structure representing an WSPI driver.
Macros

WSPI configuration options

- define WSPI_USE_WAIT TRUE
  Enables synchronous APIs.
- define WSPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the wspiAcquireBus() and wspiReleaseBus() APIs.

Transfer options

Note
The low level driver has the option to override the following definitions and use its own ones. In must take care to use the same name for the same function or compatibility is not ensured.

- define WSPI_CFG_CMD_MODE_MASK (7LU << 0LU)
- define WSPI_CFG_CMD_MODE_NONE (0LU << 0LU)
- define WSPI_CFG_CMD_MODE_ONE_LINE (1LU << 0LU)
- define WSPI_CFG_CMD_MODE_TWO_LINES (2LU << 0LU)
- define WSPI_CFG_CMD_MODE_FOUR_LINES (3LU << 0LU)
- define WSPI_CFG_CMD_MODE_EIGHT_LINES (4LU << 0LU)
- define WSPI_CFG_CMD_DTR (1LU << 3LU)
- define WSPI_CFG_CMD_SIZE_MASK (3LU << 4LU)
- define WSPI_CFG_CMD_SIZE_8 (0LU << 4LU)
- define WSPI_CFG_CMD_SIZE_16 (1LU << 4LU)
- define WSPI_CFG_CMD_SIZE_24 (2LU << 4LU)
- define WSPI_CFG_CMD_SIZE_32 (3LU << 4LU)
- define WSPI_CFG_ADDR_MODE_MASK (7LU << 8LU)
- define WSPI_CFG_ADDR_MODE_NONE (0LU << 8LU)
- define WSPI_CFG_ADDR_MODE_ONE_LINE (1LU << 8LU)
- define WSPI_CFG_ADDR_MODE_TWO_LINES (2LU << 8LU)
- define WSPI_CFG_ADDR_MODE_FOUR_LINES (3LU << 8LU)
- define WSPI_CFG_ADDR_MODE_EIGHT_LINES (4LU << 8LU)
- define WSPI_CFG_ADDR_DTR (1LU << 11LU)
- define WSPI_CFG_ADDR_SIZE_MASK (3LU << 12LU)
- define WSPI_CFG_ADDR_SIZE_8 (0LU << 12LU)
- define WSPI_CFG_ADDR_SIZE_16 (1LU << 12LU)
- define WSPI_CFG_ADDR_SIZE_24 (2LU << 12LU)
- define WSPI_CFG_ADDR_SIZE_32 (3LU << 12LU)
- define WSPI_CFG_ALT_MODE_MASK (7LU << 16LU)
- define WSPI_CFG_ALT_MODE_NONE (0LU << 16LU)
- define WSPI_CFG_ALT_MODE_ONE_LINE (1LU << 16LU)
- define WSPI_CFG_ALT_MODE_TWO_LINES (2LU << 16LU)
- define WSPI_CFG_ALT_MODE_FOUR_LINES (3LU << 16LU)
- define WSPI_CFG_ALT_MODE_EIGHT_LINES (4LU << 16LU)
- define WSPI_CFG_ALT_DTR (1LU << 19LU)
- define WSPI_CFG_ALT_SIZE_MASK (3LU << 20LU)
- define WSPI_CFG_ALT_SIZE_8 (0LU << 20LU)
- define WSPI_CFG_ALT_SIZE_16 (1LU << 20LU)
- define WSPI_CFG_ALT_SIZE_24 (2LU << 20LU)
- define WSPI_CFG_ALT_SIZE_32 (3LU << 20LU)
- define WSPI_CFG_DATA_MODE_MASK (7LU << 24LU)
- define WSPI_CFG_DATA_MODE_NONE (0LU << 24LU)
- define WSPI_CFG_DATA_MODE_ONE_LINE (1LU << 24LU)
- define WSPI_CFG_DATA_MODE_TWO_LINES (2LU << 24LU)
- define WSPI_CFG_DATA_MODE_FOUR_LINES (3LU << 24LU)
- define WSPI_CFG_DATA_MODE_EIGHT_LINES (4LU << 24LU)
- define WSPI_CFG_DATA_DTR (1LU << 27LU)
- define WSPI_CFG_DQS_ENABLE (1LU << 29LU)
- define WSPI_CFG_SIOO (1LU << 31LU)
- define WSPI_CFG_ALL_DTR
Macro Functions

- `#define wspiStartCommandI(wspip, cmdp)`
  Sends a command without data phase.
- `#define wspiStartSendI(wspip, cmdp, n, txbuf)`
  Sends data over the WSPI bus.
- `#define wspiStartReceiveI(wspip, cmdp, n, rxbuf)`
  Receives data from the WSPI bus.
- `#define wspiMapFlashI(wspip, cmdp, addrp) wspi_lld_map_flash(wspip, cmdp, addrp)`
  Maps in memory space a WSPI flash device.
- `#define wspiUnmapFlashI(wspip)`
  Maps in memory space a WSPI flash device.

Low level driver helper macros

- `#define _wspi_wakeup_isr(wspip, msg)`
  Wakes up the waiting thread.
- `#define _wspi_isr_code(wspip)`
  Common ISR code.
- `#define _wspi_error_code(wspip)`
  Common error ISR code.

Typedefs

- `typedef struct hal_wspi_driver WSPIDriver`
  Type of a structure representing a WSPI driver.
- `typedef struct hal_wspi_config WSPIConfig`
  Type of a structure representing a WSPI driver configuration.
- `typedef void(wspicallback_t)(WSPIDriver *wspip)`
  Type of a WSPI notification callback.

Enumerations

- `enum wspistate_t {
  WSPI_UNINIT = 0, WSPI_STOP = 1, WSPI_READY = 2, WSPI_SEND = 3,
  WSPI_RECEIVE = 4, WSPI_COMPLETE = 5, WSPI_MEMMAP = 6 }
  Driver state machine possible states.

Functions

- `void wspiInit (void)`
  WSPI Driver initialization.
- `void wspiObjectInit (WSPIDriver *wspip)`
  Initializes the standard part of a WSPIDriver structure.
- `msg_t wspiStart (WSPIDriver *wspip, const WSPIConfig *config)`
  Configures and activates the WSPI peripheral.
- `void wspiStop (WSPIDriver *wspip)`
  Deactivates the WSPI peripheral.
- `void wspiStartCommand (WSPIDriver *wspip, const wspi_command_t *cmdp)`
  Sends a command without data phase.
- `void wspiStartSend (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)`
  Sends data over the WSPI bus.
Sends a command with data over the WSPI bus.

- void wspiStartReceive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rbuf)
  Sends a command then receives data over the WSPI bus.

- bool wspiCommand (WSPIDriver *wspip, const wspi_command_t *cmdp)
  Sends a command without data phase.

- bool wspiSend (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
  Sends a command with data over the WSPI bus.

- bool wspiReceive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rbuf)
  Sends a command then receives data over the WSPI bus.

- void wspiMapFlash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp)
  Maps in memory space a WSPI flash device.

- void wspiUnmapFlash (WSPIDriver *wspip)
  Unmaps from memory space a WSPI flash device.

- void wspiAcquireBus (WSPIDriver *wspip)
  Gains exclusive access to the WSPI bus.

- void wspiReleaseBus (WSPIDriver *wspip)
  Releases exclusive access to the WSPI bus.

9.129.1 Detailed Description

WSPI Driver macros and structures.

9.130 hal_wspi_lld.c File Reference

PLATFORM WSPI subsystem low level driver source.

#include "hal.h"

Functions

- void wspi_lld_init (void)
  Low level WSPI driver initialization.

- void wspi_lld_start (WSPIDriver *wspip)
  Configures and activates the WSPI peripheral.

- void wspi_lld_stop (WSPIDriver *wspip)
  Deactivates the WSPI peripheral.

- void wspi_lld_command (WSPIDriver *wspip, const wspi_command_t *cmdp)
  Sends a command without data phase.

- void wspi_lld_send (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
  Sends a command with data over the WSPI bus.

- void wspi_lld_receive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rbuf)
  Sends a command then receives data over the WSPI bus.

- void wspi_lld_map_flash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp)
  Maps in memory space a WSPI flash device.

- void wspi_lld_unmap_flash (WSPIDriver *wspip)
  Unmaps from memory space a WSPI flash device.
Variables

- **WSPIDriver WSPID1**
  
  WSPID1 driver identifier.

### 9.130.1 Detailed Description

PLATFORM WSPI subsystem low level driver source.

### 9.131 hal_wspi_lld.h File Reference

PLATFORM WSPI subsystem low level driver header.

#### Macros

- `#define wspi_lld_driver_fields`
  
  Low level fields of the WSPI driver structure.

- `#define wspi_lld_config_fields`
  
  Low level fields of the WSPI configuration structure.

#### WSPI implementation capabilities

- `#define WSPI_SUPPORTS_MEMMAP TRUE`
- `#define WSPI_DEFAULT_CFG_MASKS TRUE`

#### Configuration options

- `#define PLATFORM_WSPI_USE_WSPI1 FALSE`
  
  WSPID1 driver enable switch.

#### Functions

- `void wspi_lld_init (void)`
  
  Low level WSPI driver initialization.

- `void wspi_lld_start (WSPIDriver *wspip)`
  
  Configures and activates the WSPI peripheral.

- `void wspi_lld_stop (WSPIDriver *wspip)`
  
  Deactivates the WSPI peripheral.

- `void wspi_lld_command (WSPIDriver *wspip, const wspi_command_t *cmdp)`
  
  Sends a command without data phase.

- `void wspi_lld_send (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)`
  
  Sends a command with data over the WSPI bus.

- `void wspi_lld_receive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf)`
  
  Sends a command then receives data over the WSPI bus.

- `void wspi_lld_map_flash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp)`
  
  Maps in memory space a WSPI flash device.

- `void wspi_lld_unmap_flash (WSPIDriver *wspip)`
  
  Unmaps from memory space a WSPI flash device.
9.131.1 Detailed Description

PLATFORM WSPI subsystem low level driver header.

9.132 halconf.h File Reference

HAL configuration header.

#include "mcuconf.h"

Macros

• #define HAL_USE_PAL TRUE
  Enables the PAL subsystem.
• #define HAL_USE_ADC TRUE
  Enables the ADC subsystem.
• #define HAL_USE_CAN TRUE
  Enables the CAN subsystem.
• #define HAL_USE_CRY TRUE
  Enables the cryptographic subsystem.
• #define HAL_USE_DAC TRUE
  Enables the DAC subsystem.
• #define HAL_USE_EFL TRUE
  Enables the EFlash subsystem.
• #define HAL_USE_GPT TRUE
  Enables the GPT subsystem.
• #define HAL_USE_I2C TRUE
  Enables the I2C subsystem.
• #define HAL_USE_I2S TRUE
  Enables the I2S subsystem.
• #define HAL_USE_ICU TRUE
  Enables the ICU subsystem.
• #define HAL_USE_MAC TRUE
  Enables the MAC subsystem.
• #define HAL_USE_MMC_SPI TRUE
  Enables the MMC_SPI subsystem.
• #define HAL_USE_PWM TRUE
  Enables the PWM subsystem.
• #define HAL_USE_RTC TRUE
  Enables the RTC subsystem.
• #define HAL_USE_SDC TRUE
  Enables the SDC subsystem.
• #define HAL_USE_SERIAL_TRUE
  Enables the SERIAL subsystem.
• #define HAL_USE_SERIAL_USB TRUE
  Enables the SERIAL over USB subsystem.
• #define HAL_USE_SIO TRUE
Enables the SIO subsystem.

- `#define HAL_USE_SPI TRUE`
  Enables the SPI subsystem.

- `#define HAL_USE_TRNG TRUE`
  Enables the TRNG subsystem.

- `#define HAL_USE_UART TRUE`
  Enables the UART subsystem.

- `#define HAL_USE_USB TRUE`
  Enables the USB subsystem.

- `#define HAL_USE_WDG TRUE`
  Enables the WDG subsystem.

- `#define HAL_USE_WSPI TRUE`
  Enables the WSPI subsystem.

- `#define PAL_USE_CALLBACKS FALSE`
  Enables synchronous APIs.

- `#define PAL_USE_WAIT FALSE`
  Enables synchronous APIs.

- `#define ADC_USE_WAIT TRUE`
  Enables synchronous APIs.

- `#define ADC_USE_MUTUAL_EXCLUSION TRUE`
  Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

- `#define CAN_USE_SLEEP_MODE TRUE`
  Sleep mode related APIs inclusion switch.

- `#define CAN_ENFORCE_USE_CALLBACKS FALSE`
  Enforces the driver to use direct callbacks rather than OSAL events.

- `#define HAL_CRY_USE_FALLBACK FALSE`
  Enables the SW fall-back of the cryptographic driver.

- `#define HAL_CRY_ENFORCE_FALLBACK FALSE`
  Makes the driver forcibly use the fall-back implementations.

- `#define DAC_USE_WAIT TRUE`
  Enables synchronous APIs.

- `#define DAC_USE_MUTUAL_EXCLUSION TRUE`
  Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`
  Enables the mutual exclusion APIs on the I2C bus.

- `#define MAC_USE_ZERO_COPY TRUE`
  Enables the zero-copy API.

- `#define MAC_USE_EVENTS TRUE`
  Enables an event sources for incoming packets.

- `#define MMC_IDLE_TIMEOUT_MS 1000`
  Timeout before assuming a failure while waiting for card idle.

- `#define MMC_USE_MUTUAL_EXCLUSION TRUE`
  Mutual exclusion on the SPI bus.

- `#define SDC_INIT_RETRY 100`
  Number of initialization attempts before rejecting the card.

- `#define SDC_MMC_SUPPORT TRUE`
  Include support for MMC cards.

- `#define SDC_NICE_WAITING TRUE`
  Delays insertions.

- `#define SDC_INIT_OCR_V20 0x50FF8000U`
  OCR initialization constant for V20 cards.
9.133 memstreams.c File Reference

Memory streams code.

#include <string.h>
#include "hal.h"
#include "memstreams.h"
Functions

- void msObjectInit (MemoryStream *msp, uint8_t *buffer, size_t size, size_t eos)
  
  Memory stream object initialization.

9.133.1 Detailed Description

Memory streams code.

9.134 memstreams.h File Reference

Memory streams structures and macros.

Data Structures

- struct MemStreamVMT
  
  MemStream virtual methods table.

- struct MemoryStream
  
  Memory stream object.

Macros

- #define _memory_stream_data
  
  MemStream specific data.

Functions

- void msObjectInit (MemoryStream *msp, uint8_t *buffer, size_t size, size_t eos)
  
  Memory stream object initialization.

9.134.1 Detailed Description

Memory streams structures and macros.

9.135 nullstreams.c File Reference

Null streams code.

#include "hal.h"
#include "nullstreams.h"
Nullstreams.h File Reference

Functions

• void nullObjectInit (NullStream *nsp)
  
  Null stream object initialization.

9.135.1 Detailed Description

Null streams code.

9.136 nullstreams.h File Reference

Null streams structures and macros.

Data Structures

• struct NullStreamVMT
  
  NullStream virtual methods table.

• struct NullStream
  
  Null stream object.

Macros

• #define _null_stream_data _base_buffered_stream_data
  
  NullStream specific data.

Functions

• void nullObjectInit (NullStream *nsp)
  
  Null stream object initialization.

9.136.1 Detailed Description

Null streams structures and macros.

9.137 osal.c File Reference

OSAL module code.

#include "osal.h"
Functions

- **void osalInit** (void)
  OSAL module initialization.
- **void osalSysHalt** (const char *reason)
  System halt with error message.
- **void osalSysPolledDelayX** (rtcnt_t cycles)
  Polled delay.
- **void osalOsTimerHandlerI** (void)
  System timer handler.
- **void osalOsRescheduleS** (void)
  Checks if a reschedule is required and performs it.
- **systime_t osalOsGetSystemTimeX** (void)
  Current system time.
- **void osalThreadIdSleepS** (sysinterval_t time)
  Suspends the invoking thread for the specified time.
- **void osalThreadIdSleep** (sysinterval_t time)
  Suspends the invoking thread for the specified time.
- **msg_t osalThreadIdSuspendS** (thread_reference_t *trp)
  Sends the current thread sleeping and sets a reference variable.
- **msg_t osalThreadIdSuspendTimeoutS** (thread_reference_t *trp, sysinterval_t timeout)
  Sends the current thread sleeping and sets a reference variable.
- **void osalThreadIdResumeI** (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.
- **void osalThreadIdResumeS** (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.
- **msg_t osalThreadIdEnqueueTimeoutS** (threads_queue_t *tqp, sysinterval_t timeout)
  Enqueues the caller thread.
- **void osalThreadIdDequeueNextI** (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up one thread from the queue, if any.
- **void osalThreadIdDequeueAllI** (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up all threads from the queue.
- **void osalEventBroadcastFlagsI** (event_source_t *esp, eventflags_t flags)
  Add flags to an event source object.
- **void osalEventBroadcastFlags** (event_source_t *esp, eventflags_t flags)
  Add flags to an event source object.
- **void osalEventSetCallback** (event_source_t *esp, eventcallback_t cb, void *param)
  Event callback setup.
- **void osalMutexLock** (mutex_t *mp)
  Locks the specified mutex.
- **void osalMutexUnlock** (mutex_t *mp)
  Unlocks the specified mutex.

Variables

- **const char * osal_halt_msg**
  Pointer to a halt error message.
9.137.1 Detailed Description

OSAL module code.

9.138 osal.h File Reference

OSAL module header.

```c
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

Data Structures

- `struct event_source`
  
  Events source object.

- `struct threads_queue_t`
  
  Type of a thread queue.

Macros

- `#define OSAL_DBG_ENABLE_ASSERTS FALSE`
  
  Enables OSAL assertions.

- `#define OSAL_DBG_ENABLE_CHECKS FALSE`
  
  Enables OSAL functions parameters checks.

Common constants

- `#define FALSE 0`

- `#define TRUE 1`

- `#define OSAL_SUCCESS false`

- `#define OSAL_FAILED true`

Messages

- `#define MSG_OK (msg_t)0`

- `#define MSG_TIMEOUT (msg_t)-1`

- `#define MSG_RESET (msg_t)-2`

Special time constants

- `#define TIME_IMMEDIATE ((sysinterval_t)0)`

- `#define TIME_INFINITE ((sysinterval_t)-1)`

Systick modes.

- `#define OSAL_ST_MODE_NONE 0`

- `#define OSAL_ST_MODE_PERIODIC 1`

- `#define OSAL_ST_MODE_FREERUNNING 2`
Systick parameters.

- #define OSAL_ST_RESOLUTION 32
  Size in bits of the systick_t type.
- #define OSAL_ST_FREQUENCY 1000
  Required systick frequency or resolution.
- #define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC
  Systick mode required by the underlying OS.

IRQ-related constants

- #define OSAL_IRQ_PRIORITY_LEVELS 16U
  Total priority levels.
- #define OSAL_IRQ_MAXIMUM_PRIORITY 0U
  Highest IRQ priority for HAL drivers.

Debug related macros

- #define osalDbgAssert(c, remark)
  Condition assertion.
- #define osalDbgCheck(c)
  Function parameters check.
- #define osalDbgCheckClassI()
  I-Class state check.
- #define osalDbgCheckClassS()
  S-Class state check.

IRQ service routines wrappers

- #define OSAL_IRQ_IS_VALID_PRIORITY(n) (((n) >= OSAL_IRQ_MAXIMUM_PRIORITY) && ((n) < OSAL_IRQ_PRIORITY_LEVELS))
  Priority level verification macro.
- #define OSAL_IRQ_PROLOGUE()
  IRQ prologue code.
- #define OSAL_IRQ_EPILOGUE()
  IRQ epilogue code.
- #define OSAL_IRQ_HANDLER(id) void id(void)
  IRQ handler function declaration.

Time conversion utilities

- #define OSAL_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) ∗ (time_conv_t)OSAL_ST_FREQUENCY))
  Seconds to time interval.
- #define OSAL_MS2I(msecs)
  Milliseconds to time interval.
- #define OSAL_US2I(usecs)
  Microseconds to time interval.
- #define OSAL_I2S(interval)
  Time interval to seconds.
- #define OSAL_I2MS(interval)
  Time interval to milliseconds.
- #define OSAL_I2US(interval)
  Time interval to microseconds.

Time conversion utilities for the realtime counter
• #define OSAL_S2RTC(freq, sec) ((freq) * (sec))
  Seconds to realtime counter.
• #define OSAL_MS2RTC(freq, msec) ((rtcnt_t)(((freq) + 999UL) / 1000UL) * (msec))
  Milliseconds to realtime counter.
• #define OSAL_US2RTC(freq, usec) ((rtcnt_t)(((freq) + 999999UL) / 1000000UL) * (usec))
  Microseconds to realtime counter.

Sleep macros using absolute time

• #define osalThreadSleepSeconds(secs) osalThreadSleep(OSAL_S2I(secs))
  Delays the invoking thread for the specified number of seconds.
• #define osalThreadSleepMilliseconds(msecs) osalThreadSleep(OSAL_MS2I(msecs))
  Delays the invoking thread for the specified number of milliseconds.
• #define osalThreadSleepMicroseconds(usecs) osalThreadSleep(OSAL_US2I(usecs))
  Delays the invoking thread for the specified number of microseconds.

Typedefs

• typedef uint32_t syssts_t
  Type of a system status word.
• typedef int32_t msg_t
  Type of a message.
• typedef uint32_t systime_t
  Type of system time counter.
• typedef uint32_t sysinterval_t
  Type of system time interval.
• typedef uint64_t time_conv_t
  Type of time conversion variable.
• typedef uint32_t rtcnt_t
  Type of realtime counter.
• typedef void * thread_reference_t
  Type of a thread reference.
• typedef uint32_t eventflags_t
  Type of an event flags mask.
• typedef struct event_source event_source_t
  Type of an event flags object.
• typedef void (*eventcallback_t)(event_source_t *esp)
  Type of an event source callback.
• typedef uint32_t mutex_t
  Type of a mutex.

Functions

• void osalInit (void)
  OSAL module initialization.
• void osalSysHalt (const char *reason)
  System halt with error message.
• void osalSysPolledDelayX (rtcnt_t cycles)
  Polled delay.
• void osalOsTimerHandlerI (void)
  System timer handler.
• void osalOsRescheduleS (void)
  Checks if a reschedule is required and performs it.

• systime_t osalOsGetSystemTimeX (void)
  Current system time.

• void osalThreadSleepS (sysinterval_t time)
  Suspends the invoking thread for the specified time.

• void osalThreadSleep (sysinterval_t time)
  Suspends the invoking thread for the specified time.

• msg_t osalThreadSuspendS (thread_reference_t * trp)
  Sends the current thread sleeping and sets a reference variable.

• msg_t osalThreadSuspendTimeoutS (thread_reference_t * trp, sysinterval_t timeout)
  Sends the current thread sleeping and sets a reference variable.

• void osalThreadResumeI (thread_reference_t * trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.

• void osalThreadResumeS (thread_reference_t * trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.

• msg_t osalThreadEnqueueTimeoutS (threads_queue_t * tqp, sysinterval_t timeout)
  Enqueues the caller thread.

• void osalThreadDequeueNextI (threads_queue_t * tqp, msg_t msg)
  Dequeues and wakes up one thread from the queue, if any.

• void osalThreadDequeueAllI (threads_queue_t * tqp, msg_t msg)
  Dequeues and wakes up all threads from the queue.

• void osalEventBroadcastFlagsI (event_source_t * esp, eventflags_t flags)
  Add flags to an event source object.

• void osalEventBroadcastFlags (event_source_t * esp, eventflags_t flags)
  Add flags to an event source object.

• void osalEventSetCallback (event_source_t * esp, eventcallback_t cb, void * param)
  Event callback setup.

• void osalMutexLock (mutex_t * mp)
  Locks the specified mutex.

• void osalMutexUnlock (mutex_t * mp)
  Unlocks the specified mutex.

• static void osalSysDisable (void)
  Disables interrupts globally.

• static void osalSysEnable (void)
  Enables interrupts globally.

• static void osalSysLock (void)
  Enters a critical zone from thread context.

• static void osalSysUnlock (void)
  Leaves a critical zone from thread context.

• static void osalSysLockFromISR (void)
  Enters a critical zone from ISR context.

• static void osalSysUnlockFromISR (void)
  Leaves a critical zone from ISR context.

• static syssts_t osalSysGetStatusAndLockX (void)
  Returns the execution status and enters a critical zone.

• static void osalSysRestoreStatusX (syssts_t sts)
  Restores the specified execution status and leaves a critical zone.

• static systime_t osalTimeAddX (systime_t systime, sysinterval_t interval)
  Adds an interval to a system time returning a system time.

• static sysinterval_t osalTimeDiffX (systime_t start, systime_t end)
Subtracts two system times returning an interval.

• static bool osalTimeInRangeX (systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time window.

• static void osalThreadQueueObjectInit (threads_queue_t *tqp)
 Initializes a threads queue object.

• static void osalEventObjectInit (event_source_t *esp)
  Initializes an event source object.

• static void osalMutexObjectInit (mutex_t *mp)
  Initializes a mutex_t object.

Variables

• const char * osal_halt_msg
  Pointer to a halt error message.

9.138.1 Detailed Description

OSAL module header.
Index

__IOBUS_DATA
   PAL Driver, 492
__CHIBIOS_HAL__
   HAL Driver, 25
__buffered_serial_data
   Buffered Serial Class, 347
__buffered_serial_methods
   Buffered Serial Class, 347
   Buffered Serial_objinit_impl
   Buffered Serial Class, 349
__mac_callback
   MAC Driver, 377
__mac_rx_wakeup
   MAC Driver, 378
__mac_tx_wakeup
   MAC Driver, 377
__mmc_driver_methods
   MMC over SPI Driver, 438
__sio_callback
   SIO Driver, 683
__sio_reloc_field
   SIO Driver, 686
__sio_wakeup_errors
   SIO Driver, 684
__sio_wakeup_rx
   SIO Driver, 684
__sio_wakeup_rxidle
   SIO Driver, 685
__sio_wakeup_tx
   SIO Driver, 685
__sio_wakeup_txend
   SIO Driver, 686
__spi_isr_complete_code
   SPIv2 Driver, 739
__spi_isr_error_code
   SPIv2 Driver, 741
__spi_isr_full_code
   SPIv2 Driver, 740
__spi_isr_half_code
   SPIv2 Driver, 740
__spi_wakeup_isr
   SPIv2 Driver, 738
__adc_isr_error_code
   ADC Driver, 41
__adc_isr_full_code
   ADC Driver, 40
__adc_isr_half_code
   ADC Driver, 39
__adc_reset_i
   ADC Driver, 38
__adc_reset_s
   ADC Driver, 38
__adc_timeout_isr
   ADC Driver, 39
__adc_wakeup_isr
   ADC Driver, 38
__base_asynchronous_channel_data
   Abstract I/O Channel Interface, 327
__base_asynchronous_channel_methods
   Abstract I/O Channel Interface, 327
__base_block_device_data
   Abstract I/O Block Device, 366
__base_block_device_methods
   Abstract I/O Block Device, 366
__base_buffered_stream_data
   Abstract Streams Interface, 316
__base_buffered_stream_methods
   Abstract Streams Interface, 316
__base_channel_data
   Abstract I/O Channel Interface, 320
__base_channel_methods
   Abstract I/O Channel Interface, 320
__base_flash_data
   Generic Flash Interface, 227
__base_flash_methods
   Generic Flash Interface, 227
__base_flash_methods_alone
   Generic Flash Interface, 227
__base_object_data
   Base Object, 480
__base_object_methods
   Base Object, 480
__base_pers_storage_methods
   Abstract Persistent Storage Interface, 338
__base_pers_storage_methods_alone
   Abstract Persistent Storage Interface, 338
__base_persistent_storage_data
   Abstract Persistent Storage Interface, 338
__base_sequential_stream_data
   Abstract Streams Interface, 313
__base_sequential_stream_methods
   Abstract Streams Interface, 313
__buffered_stream_adapter_data
   Buffered Streams Class, 351
__can_error_isr
   CAN Driver, 91
__can_rx_full_isr
   CAN Driver, 90
_base_pers_storage_methods_alone, 338
_base_persistent_storage_data, 338
getBasePersistentStorage, 338
ps_error_t, 340
ps_offset_t, 340
psGetStorageSize, 338
psRead, 339
psWrite, 339
Abstract Streams Interface, 312
_base_buffered_stream_data, 316
_base_buffered_stream_methods, 316
_base_sequential_stream_data, 313
_base_sequential_stream_methods, 313
streamGet, 315
streamPut, 314
streamRead, 314
streamUnget, 316
streamWrite, 313
ADC Driver, 33
_adc_isr_error_code, 41
_adc_isr_full_code, 40
_adc_isr_half_code, 39
_adc_reset_i, 38
_adc_reset_s, 38
_adc_timeout_isr, 39
_adc_wakeup_isr, 38
ADC_ACTIVE, 45
adc_channels_num_t, 44
ADC_COMPLETE, 45
ADC_ERR_AWD, 42
ADC_ERR_DMAFAILURE, 42
ADC_ERR_OVERFLOW, 42
ADC_ERROR, 45
adc_lld_config_fields, 43
adc_lld_configuration_group_fields, 43
adc_lld_driver_fields, 42
adc_lld_init, 54
adc_lld_start, 54
adc_lld_start_conversion, 55
adc_lld_stop, 54
adc_lld_stop_conversion, 55
ADC_READY, 45
ADC_STOP, 45
ADC_UNINIT, 45
ADC_USE_MUTUAL_EXCLUSION, 37
ADC_USE_WAIT, 37
adcAcquireBus, 52
adccallback_t, 44
ADCCConfig, 43
ADCConversionGroup, 43
adcConvert, 51
ADC1, 55
ADCDriver, 43
adcerrror_t, 45
adcerrorcallback_t, 44
adclnt, 45
adcIsBufferComplete, 37
adcObjectInit, 46
adcReleaseBus, 53
adcSample_t, 44
adcStart, 46
adcStartConversion, 48
adcStartConversionI, 49
adcstate_t, 45
adcStop, 47
adcStopConversion, 50
adcStopConversionI, 50
PLATFORM_ADC_USE_ADC1, 42
ADC_ACTIVE
ADC Driver, 45
adc_channels_num_t
ADC Driver, 44
ADC_COMPLETE
ADC Driver, 45
ADC_ERR_AWD
ADC Driver, 42
ADC_ERR_DMAFAILURE
ADC Driver, 42
ADC_ERR_OVERFLOW
ADC Driver, 42
ADC_ERROR
ADC Driver, 45
adc_lld_config_fields
ADC Driver, 43
adc_lld_configuration_group_fields
ADC Driver, 43
adc_lld_driver_fields
ADC Driver, 42
adc_lld_init
ADC Driver, 54
adc_lld_start
ADC Driver, 54
adc_lld_start_conversion
ADC Driver, 55
adc_lld_stop
ADC Driver, 54
adc_lld_stop_conversion
ADC Driver, 55
ADC_READY
ADC Driver, 45
ADC_STOP
ADC Driver, 45
ADC_UNINIT
ADC Driver, 45
ADC_USE_MUTUAL_EXCLUSION
ADC Driver, 37
Configuration, 905
ADC_USE_WAIT
ADC Driver, 37
Configuration, 905
adcAcquireBus
ADC Driver, 52
adccallback_t
ADC Driver, 44
ADCCConfig
ADC Driver, 43
BaseSequentialStream, 976
vmt, 977
BaseSequentialStreamVMT, 978
bcounter
  io_buffers_queue, 1059
bitsize_t
  Crypto Driver, 115
BLK_ACTIVE
  Abstract I/O Block Device, 373
BLK_CONNECTING
  Abstract I/O Block Device, 373
BLK_DISCONNECTING
  Abstract I/O Block Device, 373
blk_num
  BlockDeviceInfo, 979
BLK_READING
  Abstract I/O Block Device, 373
BLK_READY
  Abstract I/O Block Device, 373
blk_size
  BlockDeviceInfo, 979
BLK_STOP
  Abstract I/O Block Device, 373
BLK_SYNCING
  Abstract I/O Block Device, 373
BLK_UNINIT
  Abstract I/O Block Device, 373
BLK_WRITEING
  Abstract I/O Block Device, 373
blkConnect
  Abstract I/O Block Device, 369
blkDisconnect
  Abstract I/O Block Device, 370
blkGetDriverState
  Abstract I/O Block Device, 367
blkGetInfo
  Abstract I/O Block Device, 372
blkIsInserted
  Abstract I/O Block Device, 368
blkIsTransferring
  Abstract I/O Block Device, 367
blkIsWriteProtected
  Abstract I/O Block Device, 368
blkRead
  Abstract I/O Block Device, 370
blkstate_t
  Abstract I/O Block Device, 373
blkSync
  Abstract I/O Block Device, 371
blkWrite
  Abstract I/O Block Device, 371
block_addresses
  mmc_spi_driver_t, 1085
BlockDeviceInfo, 979
  blk_num, 979
  blk_size, 979
Blocks, 310
BMCR_ANENABLE
  MII/RMI Header, 426
BMCR_ANRESTART
  MII/RMI Header, 426
BMCR_CTST
  MII/RMI Header, 426
BMCR_FULLDPLX
  MII/RMI Header, 426
BMCR_ISOLATE
  MII/RMI Header, 426
BMCR_LOOPBACK
  MII/RMI Header, 427
BMCR_PDOWN
  MII/RMI Header, 426
BMCR_RESET
  MII/RMI Header, 427
BMCR_RESV
  MII/RMI Header, 426
BMCR_SPEED100
  MII/RMI Header, 427
BMSR_100BASE4
  MII/RMI Header, 429
BMSR_100FULL
  MII/RMI Header, 429
BMSR_100HALF
  MII/RMI Header, 428
BMSR_10FULL
  MII/RMI Header, 428
BMSR_10HALF
  MII/RMI Header, 428
BMSR_ANEGCAPABLE
  MII/RMI Header, 427
BMSR_ANEGCOMPLETE
  MII/RMI Header, 428
BMSR_ERCAP
  MII/RMI Header, 427
BMSR_JCD
  MII/RMI Header, 427
BMSR_LSTATUS
  MII/RMI Header, 427
BMSR_MFPRESUPPCAP
  MII/RMI Header, 428
BMSR_RESV
  MII/RMI Header, 428
BMSR_RFAULT
  MII/RMI Header, 428
bn
  io_buffers_queue, 1060
BQ_BUFFER_SIZE
  I/O Buffers Queues, 60
bqGetLinkX
  I/O Buffers Queues, 61
bqlsSuspendedX
  I/O Buffers Queues, 62
bqnotify_t
  I/O Buffers Queues, 66
bqResumeX
  I/O Buffers Queues, 63
bqSetLinkX
INDEX

I/O Buffers Queues, 62
bqSizeX
I/O Buffers Queues, 60
bqSpaceI
I/O Buffers Queues, 61
bqSuspendI
I/O Buffers Queues, 63
brdptr
io_buffers_queue, 1060
BS_READY
Buffered Serial Class, 348
BS_STOP
Buffered Serial Class, 348
BS_UNINIT
Buffered Serial Class, 348
bsaObjectInit
Buffered Streams Class, 352
bsIncomingDataI
Buffered Serial Class, 348
bsize
io_buffers_queue, 1060
bsRequestDataI
Buffered Serial Class, 348
bsstate_t
Buffered Serial Class, 347
btop
io_buffers_queue, 1060
buf
SDCDriver, 1109
buffer
mmc_spi_driver_t, 1085
Buffered Serial Class, 346
__buffered_serial_data, 347
__buffered_serial_methods, 347
__buffered_serial_objinit_impl, 349
BS_READY, 348
BS_STOP, 348
BS_UNINIT, 348
bsIncomingDataI, 348
bsRequestDataI, 348
bsstate_t, 347
BufferedSerial, 347
Buffered SIO Driver, 57
Buffered Streams Class, 351
__buffered_stream_adapter_data, 351
bsaObjectInit, 352
BUFSIZE_STREAM_BUFFER_SIZE, 351
BufferedSerial
Buffered Serial Class, 347
BufferedSerialVMT, 980
BufferedStreamAdapter, 981
vmt, 982
BufferedStreamAdapterVMT, 983
buffers
io_buffers_queue, 1060
BUFFERS_CHUNKS_SIZE
I/O Buffers Queues, 60
BUFSIZE_STREAM_BUFFER_SIZE
Buffered Streams Class, 351
bufstreams.c, 1147
bufstreams.h, 1147
bulk_in
SerialUSBConfig, 1114
bulk_out
SerialUSBConfig, 1114
bus_acquire
Serial NOR Flash Driver, 472
bus_cmd
Serial NOR Flash Driver, 473
bus_cmd_addr
Serial NOR Flash Driver, 474
bus_cmd_addr_dummy_receive
Serial NOR Flash Driver, 476
bus_cmd_addr_receive
Serial NOR Flash Driver, 475
bus_cmd_addr_send
Serial NOR Flash Driver, 474
bus_cmd_dummy_receive
Serial NOR Flash Driver, 475
bus_cmd_receive
Serial NOR Flash Driver, 474
bus_cmd_send
Serial NOR Flash Driver, 473
bus_release
Serial NOR Flash Driver, 472
bus_stop
Serial NOR Flash Driver, 472
bus_width
SDCConfig, 1107
bwrptr
io_buffers_queue, 1059
callback
GPTConfig, 1001
PWMChannelConfig, 1096
PWMConfig, 1098
CAN Driver, 85
__can_error_isr, 91
__can_rx_full_isr, 90
__can_tx_empty_isr, 90
__can_wakeup_isr, 91
CAN_ANY_MAILBOX, 89
CAN_BUS_OFF_ERROR, 88
can_callback_t, 92
CAN_ENFORCE_USE_CALLBACKS, 89
CAN_FRAMING_ERROR, 89
CAN_LIMIT_ERROR, 88
CAN_LIMIT_WARNING, 88
can_lld_abort, 107
can_lld_init, 103
can_lld_is_rx_nonempty, 106
can_lld_is_tx_empty, 105
can_lld_receive, 107
can_lld_sleep, 107
can_lld_start, 103
can_lld_stop, 105
can_lld_transmit, 106

ChibiOS/HAL
chvprintf  
Output Formatter Utility, 353

chvsprintf  
Output Formatter Utility, 357

circular  
hal_adc_configuration_group, 1005  
hal_spi_config, 1040

cmd  
wspi_command_t, 1145

Complex Drivers, 915

config  
CRYDriver, 991  
GPTDriver, 1002  
hal_adc_driver, 1007  
hal_can_driver, 1011  
hal_dac_driver, 1017  
hal_i2c_driver, 1022  
hal_i2s_driver, 1025  
hal_mac_driver, 1028  
hal_sio_driver, 1037  
hal_spi_driver, 1043  
hal_trng_driver, 1045  
hal_uart_driver, 1049  
hal_wspi_driver, 1052  
ICUDriver, 1057  
MFSDriver, 1078  
mmc_spi_driver_t, 1085  
PWMDriver, 1100  
SDCDriver, 1109  
SNORDriver, 1124  
USBDriver, 1133  
WDCDriver, 1144  

Configuration, 898  
ADC_USE_MUTUAL_EXCLUSION, 905  
ADC_USE_WAIT, 905  
CAN_ENFORCE_USE_CALLBACKS, 905  
CAN_USE_SLEEP_MODE, 905  
DAC_USE_MUTUAL_EXCLUSION, 906  
DAC_USE_WAIT, 906  
HAL_CRY_ENFORCE_FALLBACK, 906  
HAL_CRY_USE_FALLBACK, 905  
HAL_USE_ADC, 900  
HAL_USE_CAN, 901  
HAL_USE_CRY, 901  
HAL_USE_DAC, 901  
HAL_USE_EFL, 901  
HAL_USE_GPT, 901  
HAL_USE_I2C, 901  
HAL_USE_I2S, 902  
HAL_USE_ICU, 902  
HAL_USE_ICU, 902  
HAL_USE_MAC, 902  
HAL_USE_MMC_SPI, 902  
HAL_USE_PAL, 900  
HAL_USE_PWM, 902  
HAL_USE_RTC, 902  
HAL_USE_SDC, 903  
HAL_USE_SERIAL, 903  
HAL_USE_SERIAL_USB, 903  
HAL_USE_SIO, 903  
HAL_USE_SPI, 903  
HAL_USE_TRNG, 903  
HAL_USE_UART, 904  
HAL_USE_USB, 904  
HAL_USE_WDG, 904  
HAL_USE_WSPI, 904  
I2C_USE_MUTUAL_EXCLUSION, 906  
MAC_USE_EVENTS, 907  
MAC_USE_ZERO_COPY, 907  
MMC_IDLE_TIMEOUT_MS, 907  
MMC_USE_MUTUAL_EXCLUSION, 907  
PAL_USE_CALLBACKS, 904  
PAL_USE_WAIT, 904  
SDC_INIT_OCR, 908  
SDC_INIT_OCR_V20, 908  
SDC_INIT_RETRY, 907  
SDC_MMC_SUPPORT, 908  
SDC_NICE_WAITING, 908  
SERIAL_BUFFERS_SIZE, 909  
SERIAL_DEFAULT_BITRATE, 909  
SERIAL_USB_BUFFERS_NUMBER, 909  
SERIAL_USB_BUFFERS_SIZE, 909  
SIO_DEFAULT_BITRATE, 909  
SIO_USE_SYNCHRONIZATION, 909  
SPI_SELECT_MODE, 910  
SPI_USE_ASSERT_ON_ERROR, 910  
SPI_USE_MUTUAL_EXCLUSION, 910  
SPI_USE_WAIT, 910  
UART_USE_MUTUAL_EXCLUSION, 911  
UART_USE_WAIT, 911  
USB_USE_WAIT, 911  
WSP_USE_MUTUAL_EXCLUSION, 911  
WSP_USE_WAIT, 911  

configuration  
USBDriver, 1136  
counter  
mfs_bank_header_t, 1069  
crc  
mfs_bank_header_t, 1069  
mfs_data_header_t, 1071  
cry_algo_aes  
Crypto Driver, 117  
cry_algo_des  
Crypto Driver, 117  
cry_algo_hmac  
Crypto Driver, 117  
CRY_ERR_AUTH_FAILED  
Crypto Driver, 117  
CRY_ERR_INV_ALGO  
Crypto Driver, 116  
CRY_ERR_INV_KEY_ID  
Crypto Driver, 117  
CRY_ERR_INV_KEY_SIZE  

ChibiOS/HAL
Crypto Driver, 116
state, 990
cryEncryptAES
Crypto Driver, 121
cryEncryptAES_CBC
Crypto Driver, 125
cryEncryptAES_CFB
Crypto Driver, 127
cryEncryptAES_CTR
Crypto Driver, 129
cryEncryptAES_ECB
Crypto Driver, 123
cryEncryptAES_GCM
Crypto Driver, 131
cryEncryptDES
Crypto Driver, 135
cryEncryptDES_CBC
Crypto Driver, 139
cryEncryptDES_ECB
Crypto Driver, 137
cryerror_t
Crypto Driver, 116
cryHMACSHA256Final
Crypto Driver, 152
cryHMACSHA256Init
Crypto Driver, 151
cryHMACSHA256Update
Crypto Driver, 152
cryHMACSHA512Final
Crypto Driver, 155
cryHMACSHA512Init
Crypto Driver, 153
cryHMACSHA512Update
Crypto Driver, 154
cryInit
Crypto Driver, 117
crykey_t
Crypto Driver, 115
cryLoadAESTransientKey
Crypto Driver, 120
cryLoadDESTransientKey
Crypto Driver, 134
cryLoadHMACTransientKey
Crypto Driver, 150
cryObjectInit
Crypto Driver, 118
Crypto Driver, 109
bitsize_t, 115
cry_algo_aes, 117
cry_algo_des, 117
cry_algo_hmac, 117
CRY_ERR_AUTH_FAILED, 117
CRY_ERR_INV_ALGO, 116
CRY_ERR_INV_KEY_ID, 117
CRY_ERR_INV_KEY_SIZE, 116
CRY_ERR_INV_KEY_TYPE, 117
CRY_ERR_OP_FAILURE, 117
cry_lld_aes_loadkey, 157
cry_decrypt_AES, 159
cry_decrypt_AES_CBC, 162
cry_decrypt_AES_CFB, 164
cry_decrypt_AES_CTR, 165
cry_decrypt_AES_ECB, 160
cry_decrypt_AES_GCM, 167
cry_decrypt_DES, 170
cry_decrypt_DES_CBC, 173
cry_decrypt_DES_ECB, 172
cry_decrypt_des_loadkey, 168
cry_decrypt_AES, 158
cry_decrypt_AES_CBC, 161
cry_decrypt_AES_CFB, 163
cry_decrypt_AES_CTR, 165
cry_decrypt_AES_ECB, 159
cry_decrypt_AES_GCM, 166
cry_decrypt_DES, 169
cry_decrypt_DES_CBC, 173
cry_decrypt_DES_ECB, 171
cry_hmac_loadkey, 180
cry_HMACSHA256_final, 182
cry_HMACSHA256_init, 181
cry_HMACSHA256_update, 181
cry_HMACSHA512_final, 184
cry_HMACSHA512_init, 183
cry_HMACSHA512_update, 183
cry_init, 156
cry_SHA1_final, 176
cry_SHA1_init, 174
cry_SHA1_update, 175
cry_SHA256_final, 178
cry_SHA256_init, 176
cry_SHA256_update, 177
cry_SHA512_final, 179
cry_SHA512_init, 178
cry_SHA512_update, 179
cry_start, 156
cry_stop, 157
CRY_NOERROR, 116
CRY_READY, 116
CRY_STOP, 116
CRY_UNINIT, 116
cryalgorithm_t, 117
CRYD1, 185
cryDecryptAES, 122
cryDecryptAES_CBC, 126
cryDecryptAES_CFB, 128
cryDecryptAES_CTR, 130
cryDecryptAES_ECB, 124
cryDecryptAES_GCM, 132
cryDecryptDES, 136
cryDecryptDES_CBC, 140
cryDecryptDES_ECB, 138
CRYDriver, 116
cryEncryptAES, 121
cryEncryptAES_CBC, 125
cryEncryptAES_CFB, 127
cryEncryptAES_CTR, 129
ChibiOS/HAL
cryEncryptAES_ECB, 123
cryEncryptAES_GCM, 131
cryEncryptDES, 135
cryEncryptDES_CBC, 139
cryEncryptDES_ECB, 137
cryerror_t, 116
cryHMACSHA256Final, 152
cryHMACSHA256init, 151
cryHMACSHA256Update, 152
cryHMACSHA512Final, 155
cryHMACSHA512init, 153
cryHMACSHA512Update, 154
cryInit, 117
crykey_t, 115
cryLoadAESTransientKey, 120
cryLoadDESTransientKey, 134
cryLoadHMACTransientKey, 150
cryObjectInit, 118
crySHA1Final, 143
crySHA1Init, 141
crySHA1Update, 142
crySHA256Final, 146
crySHA256Init, 144
crySHA256Update, 145
crySHA512Final, 149
crySHA512Init, 147
crySHA512Update, 148
cryStart, 118
crystate_t, 116
cryStop, 119
HAL_CRY_ENFORCE_FALLBACK, 115
HAL_CRY_USE_FALLBACK, 114
PLATFORM_CRY_USE_CRY1, 115
crySHA1Final
  Crypto Driver, 143
crySHA1Init
  Crypto Driver, 141
crySHA1Update
  Crypto Driver, 142
crySHA256Final
  Crypto Driver, 146
crySHA256Init
  Crypto Driver, 144
crySHA256Update
  Crypto Driver, 145
crySHA512Final
  Crypto Driver, 149
crySHA512Init
  Crypto Driver, 147
crySHA512Update
  Crypto Driver, 148
cryStart
  Crypto Driver, 118
crystate_t
  Crypto Driver, 116
cryStop
  Crypto Driver, 119
current_bank

MFSDriver, 1078
current_counter
MFSDriver, 1078

DAC Driver, 186
  _dac_isr_error_code, 193
  _dac_isr_full_code, 192
  _dac_isr_half_code, 191
  _dac_reset_i, 190
  _dac_reset_s, 190
  _dac_timeout_isr, 191
  _dac_wait_s, 189
  _dac_wakeup_isr, 191
DAC_ACTIVE, 196
DAC_COMPLETE, 196
DAC_ERR_DMAFAILURE, 197
DAC_ERR_UNDERFLOW, 197
DAC_ERROR, 196
dac_ild_config_fields, 194
dac_ild_conversion_group_fields, 194
dac_ild_driver_fields, 194
dac_ild_init, 206
dac_ild_put_channel, 208
dac_ild_start, 207
dac_ild_start_conversion, 208
dac_ild_stop, 207
dac_ild_stop_conversion, 209
DAC_MAX_CHANNELS, 193
DAC_READY, 196
DAC_STOP, 196
DAC_UNINIT, 196
DAC_USE_MUTUAL_EXCLUSION, 188
DAC_USE_WAIT, 188
dacAcquireBus, 205
dacCallback_t, 195
dacchannel_t, 196
DACConfig, 195
dacConvertGroup, 195
dacConvert, 204
DACD1, 209
DACDriver, 195
dacerror_t, 196
dacerrorcallback_t, 195
dacInIt, 197
dacIsBufferComplete, 189
dacObjectInit, 197
dacPutChannelX, 200
dacReleaseBus, 206
dacsample_t, 196
dacStart, 198
dacStartConversion, 201
dacStartConversionI, 201
dacstate_t, 196
dacStop, 199
dacStopConversion, 202
dacStopConversionI, 203
PLATFORM_DAC_USE_DAC1, 194
DAC_ACTIVE
  DAC Driver, 196

ChibiOS/HAL
DAC_COMPLETE
  DAC Driver, 196
DAC_ERR_DMAFAILURE
  DAC Driver, 197
DAC_ERR_UNDERFLOW
  DAC Driver, 197
DAC_ERROR
  DAC Driver, 196
dac_lld_config_fields
  DAC Driver, 194
dac_lld_conversion_group_fields
  DAC Driver, 194
dac_lld_driver_fields
  DAC Driver, 194
dac_lld_init
  DAC Driver, 206
dac_lld_put_channel
  DAC Driver, 208
dac_lld_start
  DAC Driver, 207
dac_lld_start_conversion
  DAC Driver, 208
dac_lld_stop
  DAC Driver, 207
dac_lld_stop_conversion
  DAC Driver, 209
DAC_MAX_CHANNELS
  DAC Driver, 193
DAC_READY
  DAC Driver, 196
DAC_STOP
  DAC Driver, 196
DAC_UNINIT
  DAC Driver, 196
DAC_USE_MUTUAL_EXCLUSION
  Configuration, 906
  DAC Driver, 188
DAC_USE_WAIT
  Configuration, 906
  DAC Driver, 188
dacAcquireBus
  DAC Driver, 205
daccallback_t
  DAC Driver, 195
dacchannel_t
  DAC Driver, 196
DACConfig
  DAC Driver, 195
DACConversionGroup
  DAC Driver, 195
dacConvert
  DAC Driver, 204
DACD1
  DAC Driver, 209
DACDriver
  DAC Driver, 195
dacerror_t
  DAC Driver, 196
dacerrorcallback_t
  DAC Driver, 195
dacInit
  DAC Driver, 197
daclsBufferComplete
  DAC Driver, 189
dacObjectInit
  DAC Driver, 197
daPutChannelX
  DAC Driver, 200
daReleaseBus
  DAC Driver, 206
daSample_t
  DAC Driver, 196
dacStart
  DAC Driver, 198
daStartConversion
  DAC Driver, 201
daStartConversionI
  DAC Driver, 201
daState_t
  DAC Driver, 196
daStop
  DAC Driver, 199
daStopConversion
  DAC Driver, 202
daStopConversionI
  DAC Driver, 203
data16
  CANRxFrame, 986
  CANTxFrame, 988
data32
  CANRxFrame, 986
  CANTxFrame, 988
data8
  CANRxFrame, 985
  CANTxFrame, 988
data_cb
  hal_spi_config, 1041
day
  RTCDatetime, 1103
dayofweek
  RTCDatetime, 1103
default_config
  Serial Driver, 656
default_handler
  USB Driver, 835
depth
  hal_adc_driver, 1007
  hal_dac_driver, 1017
descriptors
  MFSDriver, 1079
detect_bus_clk
  SDC Driver, 618
DLC
  CANRxFrame, 985
  CANTxFrame, 987
dstflag

ChibiOS/HAL
RTCDatetime, 1102
dummy
term
wspi_command_t, 1145
early
hal_uart_driver, 1049
EFL Driver, 210
_elf_driver_data, 212
_elf_flash_methods, 212
_elf_flash_methods_alone, 211
elf_lld_config_fields, 213
efl_lld_driver_fields, 212
efl_lld_get_descriptor, 217
efl_lld_init, 216
efl_lld_program, 218
efl_lld_query_erase, 220
efl_lld_read, 217
efl_lld_start, 216
efl_lld_start_erase_all, 219
efl_lld_start_erase_sector, 220
efl_lld_stop, 217
efl_lld_verify_erase, 221
EFL_USE_MUTUAL_EXCLUSION, 211
EFlashConfig, 213
EFlashDriver, 213
EFLD1, 221
eflInit, 213
eflObjectInit, 214
eflStart, 214
eflStop, 215
PLATFORM_EFL_USE_EFL1, 212
efl_lld_config_fields
EFL Driver, 213
efl_lld_driver_fields
EFL Driver, 212
efl_lld_get_descriptor
EFL Driver, 217
efl_lld_init
EFL Driver, 216
efl_lld_program
EFL Driver, 218
efl_lld_query_erase
EFL Driver, 220
efl_lld_read
EFL Driver, 217
efl_lld_start
EFL Driver, 216
efl_lld_start_erase_all
EFL Driver, 219
efl_lld_start_erase_sector
EFL Driver, 220
efl_lld_stop
EFL Driver, 217
efl_lld_verify_erase
EFL Driver, 221
EFL_USE_MUTUAL_EXCLUSION
EFL Driver, 211
EFlashConfig
EFL Driver, 213
EFlashDriver
EFL Driver, 213
EFlashDriverVMT, 991
EFLD1
EFL Driver, 221
eflInit
EFL Driver, 213
eflObjectInit
EFL Driver, 214
eflStart
EFL Driver, 214
eflStop
EFL Driver, 215
EID
CANRxFrame, 985
CANTxFrame, 988
enabled
hal_sio_driver, 1037
PWMDriver, 1100
end_cb
hal_adc_configuration_group, 1005
hal_dac_conversion_group, 1015
hal_i2s_config, 1024
hal_spi_config, 1040
hal_wspi_config, 1051
ep0_state
USB Driver, 857
ep0config
USB Driver, 858
ep0endcb
USBDriver, 1135
ep0n
USBDriver, 1135
ep0next
USBDriver, 1135
ep0state
USBDriver, 1134
ep_mode
USBEndpointConfig, 1138
EP_STATUS_ACTIVE
USB Driver, 833
EP_STATUS_DISABLED
USB Driver, 833
EP_STATUS_STALLED
USB Driver, 833
epc
USBDriver, 1134
erased
MFSConfig, 1075
error_cb
hal_adc_configuration_group, 1005
hal_dac_conversion_group, 1015
hal_spi_config, 1041
hal_wspi_config, 1051
error_event
hal_can_driver, 1012
errors
hal_i2c_driver, 1023
flags
  event_source, 993
  hal_mac_driver, 1028
FLASH_ATTR_ECC_CAPABLE
  Generic Flash Interface, 226
FLASH_ATTR_ECC_ZERO_LINE_CAPABLE
  Generic Flash Interface, 226
FLASH_ATTR_ERASED_IS_ONE
  Generic Flash Interface, 225
FLASH_ATTR_MEMORY_MAPPED
  Generic Flash Interface, 225
FLASH_ATTR_REWRITABLE
  Generic Flash Interface, 226
FLASH_ATTR_SUSPEND_ERASE_CAPABLE
  Generic Flash Interface, 226
flash_descriptor_t, 997
  address, 998
  attributes, 998
  page_size, 998
  sectors, 998
  sectors_count, 998
  sectors_size, 998
  size, 999
flash_error_t
  Generic Flash Interface, 233
flash_offset_t
  Generic Flash Interface, 233
flash_sector_descriptor_t, 999
  offset, 1000
  size, 1000
flash_sector_t
  Generic Flash Interface, 233
flash_state_t
  Generic Flash Interface, 233
flashAcquireExclusive
  Generic Flash Interface, 232
flashGetAddressOffset
  Generic Flash Interface, 236
flashGetDescriptor
  Generic Flash Interface, 228
flashGetOffsetAddress
  Generic Flash Interface, 236
flashGetOffsetSector
  Generic Flash Interface, 235
flashGetSectorOffset
  Generic Flash Interface, 234
flashGetSectorSize
  Generic Flash Interface, 235
flashp
  MFSConfig, 1075
flashProgram
  Generic Flash Interface, 229
flashQueryErase
  Generic Flash Interface, 231
flashRead
  Generic Flash Interface, 228
flashReleaseExclusive
  Generic Flash Interface, 232
GPT Driver, 240

gptcnt_t
GPT Driver, 244

GPTConfig, 1000

callback, 1001
frequency, 1001

GPTD1
GPT Driver, 255

GPTDriver, 1002
config, 1002
GPT Driver, 243
state, 1002

gptfreq_t
GPT Driver, 243

gptGetCounterX
GPT Driver, 241

gptGetIntervalX
GPT Driver, 240

gptInit
GPT Driver, 244

gptObjectInit
GPT Driver, 245

gptPolledDelay
GPT Driver, 252

gptStart
GPT Driver, 245

gptStartContinuous
GPT Driver, 248

gptStartContinuousI
GPT Driver, 248

gptStartOneShot
GPT Driver, 249

gptStartOneShotI
GPT Driver, 250

gptstate_t
GPT Driver, 244

gptStop
GPT Driver, 246

gptStopTimer
GPT Driver, 250

gptStopTimerI
GPT Driver, 251

grpp
hal_adc_driver, 1007
hal_dac_driver, 1017

HAL, 895
HAL Driver, 23

__CHIBIOS_HAL__, 25
CH_HAL_MAJOR, 25
CH_HAL_MINOR, 25
CH_HAL_PATCH, 25
CH_HAL_STABLE, 25
CH_HAL_VERSION, 25
HAL_FAILED, 26
HAL_LLD_SELECT_SPI_V2, 27
HAL_RET_CONFIG_ERROR, 26
HAL_RET_HW_BUSY, 26
HAL_RET_HW_FAILURE, 27
HAL_RET_NO_RESOURCE, 26
HAL_RET_UNKNOWN_CTL, 27
HAL_SUCCESS, 26
halclkpt_t, 27
halClockGetPointX, 30
halClockSwitchMode, 29
halfreq_t, 27
halInit, 28

hal.c, 1150
hal.h, 1151
hal_adc.c, 1152
hal_adc.h, 1153
hal_adc_config, 1003
hal_adc_configuration_group, 1003

circular, 1005
end_cb, 1005
error_cb, 1005
num_channels, 1005

hal_adc_driver, 1006
config, 1007
depth, 1007
grpp, 1007
mutex, 1008
samples, 1007
state, 1007
thread, 1008

hal_adc_lld.c, 1155
hal_adc_lld.h, 1156
hal_buffered_serial, 1008
vmt, 1009

hal_buffered_serial.h, 1157
hal_buffers.c, 1158
hal_buffers.h, 1160
hal_can.c, 1162
hal_can.h, 1163
hal_can_config, 1010
hal_can_driver, 1010

config, 1011
error_event, 1012
rxfull_event, 1012
rxqueue, 1012
sleep_event, 1013
state, 1011
txempty_event, 1012
txqueue, 1011
wakeup_event, 1013

hal_can_lld.c, 1164
hal_can_lld.h, 1165
hal_channels.h, 1167
HAL_CRY_ENFORCE_FALLBACK

Configuration, 906
Crypto Driver, 115
HAL_CRY_USE_FALLBACK

Configuration, 905
INDEX

Crypto Driver, 114
hal_crypto.c, 1168
hal_crypto.h, 1171
hal_crypto_lld.c, 1174
hal_crypto_lld.h, 1176
hal_dac.c, 1179
hal_dac.h, 1180
hal_dac_config, 1013
hal_dac_conversion_group, 1014
   end_cb, 1015
   error_cb, 1015
   num_channels, 1014
hal_dac_driver, 1015
   config, 1017
   depth, 1017
   grpp, 1017
   mutex, 1018
   samples, 1017
   state, 1017
   thread, 1017
hal_dac_lld.c, 1182
hal_dac_lld.h, 1183
hal_efl.c, 1184
hal_efl.h, 1184
hal_efl_config, 1018
hal_efl_driver, 1019
   mutex, 1020
   vmt, 1020
hal_efl_lld.c, 1185
hal_efl_lld.h, 1186
HAL_FAILED
   HAL Driver, 26
hal_files.h, 1187
hal_flash.c, 1188
hal_flash.h, 1189
hal_gpt.c, 1191
hal_gpt.h, 1192
hal_gpt_lld.c, 1193
hal_gpt_lld.h, 1194
hal_i2c.c, 1195
hal_i2c.h, 1195
hal_i2c_config, 1021
hal_i2c_driver, 1021
   config, 1022
   errors, 1023
   state, 1022
hal_i2c_lld.c, 1197
hal_i2c_lld.h, 1198
hal_i2s.c, 1199
hal_i2s.h, 1200
hal_i2s_config, 1023
   end_cb, 1024
   rx_buffer, 1024
   size, 1024
   tx_buffer, 1024
hal_i2s_driver, 1025
   config, 1025
   state, 1025
hal_i2s_lld.c, 1201
hal_i2s_lld.h, 1202
hal_icu.c, 1202
hal_icu.h, 1203
hal_icu_lld.c, 1205
hal_icu_lld.h, 1205
hal_ioblock.h, 1207
hal_lld.c, 1208
hal_lld.h, 1208
hal_lld_clock_switch_mode
   HAL Driver, 31
hal_lld_get_clock_point
   HAL Driver, 32
hal_lld_init
   HAL Driver, 31
HAL_LLD_SELECT_SPI_V2
   HAL Driver, 27
HAL_LLD_USE_CLOCK_MANAGEMENT
   HAL Driver, 27
hal_mac.c, 1209
hal_mac.h, 1210
hal_mac_config, 1026
hal_mac_driver, 1026
   arg, 1029
   cb, 1029
   config, 1028
   es, 1028
   flags, 1028
   rdqueue, 1028
   state, 1028
   tdqueue, 1028
hal_mac_lld.c, 1212
hal_mac_lld.h, 1213
hal_mac_receive_descriptor, 1029
   offset, 1030
   size, 1030
hal_mac_transmit_descriptor, 1030
   offset, 1031
   size, 1031
hal_mfs.c, 1214
hal_mfs.h, 1215
hal_mii.h, 1217
hal_mmc_spi.c, 1220
hal_mmc_spi.h, 1221
hal_mmcsd.c, 1223
hal_mmcsd.h, 1224
hal_objects.h, 1228
hal_pal.c, 1228
hal_pal.h, 1229
hal_pal_lld.c, 1233
hal_pal_lld.h, 1233
hal_persistent.h, 1235
hal_pwm.c, 1236
hal_pwm.h, 1237
hal_pwm_lld.c, 1239
hal_pwm_lld.h, 1240
hal_queues.c, 1241
hal_queues.h, 1242
HAL_RET_CONFIG_ERROR
HAL Driver, 26

HAL_RET_HW_BUSY
HAL Driver, 26

HAL_RET_HW_FAILURE
HAL Driver, 27

HAL_RET_NO_RESOURCE
HAL Driver, 26

HAL_RET_UNKNOWN_CTL
HAL Driver, 27

hal_rtc.c, 1244
hal_rtc.h, 1245
hal_rtc_lld.c, 1247
hal_rtc_lld.h, 1247
hal_sdc.c, 1249
hal_sdc.h, 1250
hal_sdc_lld.c, 1252
hal_sdc_lld.h, 1253
hal_serial.c, 1255
hal_serial.h, 1256
hal_serial_config, 1031
    speed, 1032
hal_serial_driver, 1032
    vmt, 1034
hal_serial_lld.c, 1258
hal_serial_lld.h, 1258
hal_serial_nor.c, 1259
hal_serial_nor.h, 1260
hal_serial_usb.c, 1262
hal_serial_usb.h, 1263
hal_sio.c, 1264
hal_sio.h, 1265
hal_sio_config, 1035
hal_sio_driver, 1035
    arg, 1038
    cb, 1037
    config, 1037
    enabled, 1037
    state, 1037
    sync_rx, 1038
    sync_rxidle, 1038
    sync_tx, 1038
    sync_txend, 1038
    vmt, 1037
hal_sio_lld.c, 1269
hal_sio_lld.h, 1270
hal_spi.c, 1271
hal_spi.h, 1271
hal_spi_config, 1039
    circular, 1040
    data_cb, 1041
    end_cb, 1040
    error_cb, 1041
    slave, 1041
    ssline, 1040
    ssmask, 1041
    sspad, 1041
    ssport, 1040
hal_spi_driver, 1042
    config, 1043
    mutex, 1043
    state, 1043
    sync_transfer, 1043
    thread, 1043
hal_spi_lld.c, 1271
hal_spi_lld.h, 1272
hal_spi_v1.h, 1273
hal_spi_v2.h, 1275
hal_spi_v2_lld.c, 1276
hal_spi_v2_lld.h, 1277
hal_st.c, 1278
hal_st.h, 1279
hal_st_lld.c, 1280
hal_st_lld.h, 1280
hal_streams.h, 1281
HAL_SUCCESS
HAL Driver, 26
hal_trng.c, 1282
hal_trng.h, 1282
hal_trng_config, 1044
hal_trng_driver, 1044
    config, 1045
    state, 1045
hal_trng_lld.c, 1283
hal_trng_lld.h, 1284
hal_uart.c, 1284
hal_uart.h, 1285
hal_uart_config, 1045
    rxchar_cb, 1047
    rxend_cb, 1047
    rxerr_cb, 1047
    txend1_cb, 1047
    txend2_cb, 1047
hal_uart_driver, 1048
    config, 1049
    early, 1049
    mutex, 1050
    rxstate, 1049
    state, 1049
    threadrx, 1050
    threadtx, 1050
    txstate, 1049
hal_uart_lld.c, 1288
hal_uart_lld.h, 1288
hal_usb.c, 1289
hal_usb.h, 1291
hal_usb_cdc.h, 1294
hal_usb_lld.c, 1295
    in, 1297
    out, 1297
hal_usb_lld.h, 1297
HAL_USE_ADC
    Configuration, 900
HAL_USE_CAN
    Configuration, 901
HAL_USE_CRY
input_queue_t, 577
io_queue_t, 576
iq_read, 577
iqGet, 573
iqGetEmptyI, 571
iqGetFullI, 571
iqGet, 571
iqGetTimeout, 581
iqIsEmptyI, 572
iqIsFullI, 572
iqObjectInit, 578
iqPut, 580
iqRead, 582
iqReadTimeout, 583
iqReset, 579
oq_write, 578
oqGetEmpty, 574
oqGetFull, 573
oqGet, 578
oqIsEmpty, 575
oqIsFull, 575
oqObjectInit, 584
oqPut, 576
oqPut, 586
oqPutTimeout, 586
oqReset, 585
oqWrite, 588
oqWriteTimeout, 589
output_queue_t, 577
Q_EMPTY, 568
Q_FULL, 568
Q_OK, 568
Q_RESET, 568
Q_TIMEOUT, 568
qGetLink, 570
qnotify_t, 577
qSetLink, 570
qSizeX, 569
qSpaCexl, 569
12C Driver, 256
_i2c_wakeup_isr, 261
_i2c_wakeup_isr, 260
I2C_ACK_FAILURE, 259
I2C_ACTIVE_RX, 263
I2C_ACTIVE_TX, 263
I2C_ARBITRATION_LOST, 259
I2C_BUS_ERROR, 259
i2c_lld_get_errors, 262
i2c_lld_init, 270
i2c_lld_master_receive_timeout, 271
i2c_lld_master_transmit_timeout, 271
i2c_lld_start, 270
i2c_lld_stop, 270
I2C_LOCKED, 263
I2C_NO_ERROR, 259
I2C_OVERRUN, 259
I2C_PEC_ERROR, 260
I2C_READY, 263
I2C_SMB_ALERT, 260
I2C_STOP, 263
I2C_TIMEOUT, 260
I2C_UNINIT, 263
I2C_USE_MUTUAL_EXCLUSION, 260
i2cAcquireBus, 268
i2caddr_t, 262
i2cConfig, 263
I2CD1, 272
I2CDriver, 263
i2cflags_t, 263
i2cGetErrors, 266
i2cInit, 264
i2cMasterReceive, 261
i2cMasterReceiveTimeout, 267
i2cMasterTransmit, 261
i2cMasterTransmitTimeout, 267
i2cObjectInit, 264
i2cReleaseBus, 269
i2cStart, 265
i2cstate_t, 263
i2cStop, 265
PLATFORM_I2C_USE_I2C1, 262
I2C_ACK_FAILURE
I2C Driver, 259
I2C_ACTIVE_RX
I2C Driver, 263
I2C_ACTIVE_TX
I2C Driver, 263
I2C_ARBITRATION_LOST
I2C Driver, 259
I2C_BUS_ERROR
I2C Driver, 259
i2c_lld_get_errors
I2C Driver, 262
i2c_lld_init
I2C Driver, 270
i2c_lld_master_receive_timeout
I2C Driver, 271
i2c_lld_master_transmit_timeout
I2C Driver, 271
i2c_lld_start
I2C Driver, 270
i2c_lld_stop
I2C Driver, 270
I2C_LOCKED
I2C Driver, 263
I2C_NO_ERROR
I2C Driver, 259
I2C_OVERRUN
I2C Driver, 259
I2C_PEC_ERROR
I2C Driver, 260
I2C_READY
I2C Driver, 263
I2C_SMB_ALERT
I2C Driver, 260
I2C_STOP
INDEX

I2C Driver, 263
I2C_TIMEOUT
  I2C Driver, 260
I2C_UNINIT
  I2C Driver, 263
I2C_USE_MUTUAL_EXCLUSION
    Configuration, 906
  I2C Driver, 260
i2cAcquireBus
  I2C Driver, 268
i2caddr_t
  I2C Driver, 262
I2CConfig
  I2C Driver, 263
I2CD1
  I2C Driver, 272
I2CDriver
  I2C Driver, 263
i2cflags_t
  I2C Driver, 263
i2cGetErrors
  I2C Driver, 266
i2cInit
  I2C Driver, 264
i2cMasterReceive
  I2C Driver, 261
i2cMasterReceiveTimeout
  I2C Driver, 267
i2cMasterTransmit
  I2C Driver, 261
i2cMasterTransmitTimeout
  I2C Driver, 267
i2cObjectInit
  I2C Driver, 264
i2cReleaseBus
  I2C Driver, 269
i2cStart
  I2C Driver, 265
i2cstate_t
  I2C Driver, 263
i2cStop
  I2C Driver, 265
I2S Driver, 273
  _i2s_isr_full_code, 277
  _i2s_isr_half_code, 276
I2S_ACTIVE, 279
I2S_COMPLETE, 279
i2s_lld_config_fields, 278
i2s_lld_driver_fields, 278
i2s_lld_init, 283
i2s_lld_start, 283
i2s_lld_start_exchange, 284
i2s_lld_stop, 284
i2s_lld_stop_exchange, 285
I2S_READY, 279
I2S_STOP, 279
I2S_UNINIT, 279
i2scallback_t, 278
I2SConfig, 278
I2SD1, 285
I2SDriver, 278
i2sInit, 279
i2sIsBufferComplete, 275
i2sObjectInit, 280
i2sStart, 280
i2sStartExchange, 282
i2sStartExchange, 275
i2sstate_t, 279
i2sStop, 281
i2sStopExchange, 282
i2sStopExchange, 276
PLATFORM_I2S_USE_I2S1, 277
I2S_ACTIVE
  I2S Driver, 279
I2S_COMPLETE
  I2S Driver, 279
i2s_lld_config_fields
  I2S Driver, 278
i2s_lld_driver_fields
  I2S Driver, 278
i2s_lld_init
  I2S Driver, 283
i2s_lld_start
  I2S Driver, 283
i2s_lld_start_exchange
  I2S Driver, 284
i2s_lld_stop
  I2S Driver, 284
i2s_lld_stop_exchange
  I2S Driver, 285
I2S_READY
  I2S Driver, 279
I2S_STOP
  I2S Driver, 279
I2S_UNINIT
  I2S Driver, 279
i2sObjectInit
  I2S Driver, 278
i2scallback_t
  I2S Driver, 278
I2SConfig
  I2S Driver, 278
I2SD1
  I2S Driver, 285
I2SDriver
  I2S Driver, 278
i2sInit
  I2S Driver, 279
i2sIsBufferComplete
  I2S Driver, 275
i2sObjectInit
  I2S Driver, 280
i2sStart
  I2S Driver, 280
i2sStartExchange
  I2S Driver, 282
i2sStartExchange
  I2S Driver, 275
i2sstate_t
  I2S Driver, 279
i2sStop
  I2S Driver, 281
i2sStopExchange
  I2S Driver, 282
i2sStopExchangeI
  I2S Driver, 276
ibnotify
  Serial over USB Driver, 660
ibqGetEmptyBuffer
  I/O Buffers Queues, 68
ibqGetFullBufferTimeout
  I/O Buffers Queues, 70
ibqGetFullBufferTimeoutS
  I/O Buffers Queues, 71
ibqGetTimeout
  I/O Buffers Queues, 73
ibqIsEmptyI
  I/O Buffers Queues, 64
ibqIsFullI
  I/O Buffers Queues, 64
ibqObjInit
  I/O Buffers Queues, 67
ibqPostFullBuffer
  I/O Buffers Queues, 69
ibqReadTimeout
  I/O Buffers Queues, 74
ibqReleaseEmptyBuffer
  I/O Buffers Queues, 72
ibqReleaseEmptyBufferS
  I/O Buffers Queues, 73
ibqResetI
  I/O Buffers Queues, 68
ICU Driver, 286
  _icu_isr_invoke_overflow_cb, 294
  _icu_isr_invoke_period_cb, 293
  _icu_isr_invoke_width_cb, 293
  ICU_ACTIVE, 297
  ICU_INPUT_ACTIVE_HIGH, 297
  ICU_INPUT_ACTIVE_LOW, 297
  icu_lld_are_notifications_enabled, 295
  icu_lld_disable_notifications, 308
  icu_lld_enable_notifications, 307
  icu_lld_get_period, 295
  icu_lld_get_width, 294
  icu_lld_init, 305
  icu_lld_start, 305
  icu_lld_start_capture, 306
  icu_lld_stop, 306
  icu_lld_stop_capture, 307
  icu_lld_wait_capture, 306
  ICU_ACTIVE, 297
  ICU_STOP, 297
  ICU_UNINIT, 297
  icuAreNotificationsEnabledX, 291
  icucallback_t, 296
  icucnt_t, 296
  ICUD1, 308
  icuDisableNotifications, 304
  icuDisableNotificationsI, 291
  ICUDriver, 296
  icuEnableNotifications, 303
  icuEnableNotificationsI, 290
  icufreq_t, 296
  icuGetPeriodX, 292
  icuGetWidthX, 292
  icuInit, 297
  icumode_t, 297
  icuObjectInit, 298
  icuStart, 298
  icuStartCapture, 301
  icuStartCaptureI, 289
  icustate_t, 297
  icuStop, 300
  icuStopCapture, 303
  icuStopCaptureI, 290
  icuWaitCapture, 301
  PLATFORM_ICU_USE_ICU1, 294
  ICU_ACTIVE
  ICU Driver, 297
  ICU_INPUT_ACTIVE_HIGH
  ICU Driver, 297
  ICU_INPUT_ACTIVE_LOW
  ICU Driver, 297
  icu_lld_are_notifications_enabled
  ICU Driver, 295
  icu_lld_disable_notifications
  ICU Driver, 308
  icu_lld_enable_notifications
  ICU Driver, 307
  icu_lld_get_period
  ICU Driver, 295
  icu_lld_get_width
  ICU Driver, 294
  icu_lld_init
  ICU Driver, 305
  icu_lld_start
  ICU Driver, 305
  icu_lld_start_capture
  ICU Driver, 306
  icu_lld_stop
  ICU Driver, 306
  icu_lld_stop_capture
  ICU Driver, 307
  icu_lld_wait_capture
  ICU Driver, 306
  ICU_READY
  ICU Driver, 297
  ICU_STOP
  ICU Driver, 297
  ICU_UNINIT
  ICU Driver, 297
  ICU_WAITING
  ICU Driver, 297
icuAreNotificationsEnabledX
   ICU Driver, 291
icucallback_t
   ICU Driver, 296
icucnt_t
   ICU Driver, 296
ICUConfig, 1055
   frequency, 1056
   mode, 1056
   overflow_cb, 1056
   period_cb, 1056
   width_cb, 1056
ICUD1
   ICU Driver, 308
icuDisableNotifications
   ICU Driver, 304
icuDisableNotificationsI
   ICU Driver, 291
ICUDriver, 1057
   config, 1057
   ICU Driver, 296
   state, 1057
icuEnableNotifications
   ICU Driver, 303
icuEnableNotificationsI
   ICU Driver, 290
icufreq_t
   ICU Driver, 296
icuGetPeriodX
   ICU Driver, 292
icuGetWidthX
   ICU Driver, 292
icuInit
   ICU Driver, 297
icumode_t
   ICU Driver, 297
icuObjectInit
   ICU Driver, 298
icuStart
   ICU Driver, 298
icuStartCapture
   ICU Driver, 301
icuStartCaptureI
   ICU Driver, 290
icustate_t
   ICU Driver, 297
icuStop
   ICU Driver, 300
icuStopCapture
   ICU Driver, 303
icuStopCaptureI
   ICU Driver, 290
icuWaitCapture
   ICU Driver, 301
id
   mfs_data_header_t, 1071
   mfs_transaction_op_t, 1073
IDE
CANRxFrame, 985
CANTxFrame, 988
in
   hal_usb_lld.c, 1297
   USB Driver, 857
in_cb
   USBEndpointConfig, 1138
in_maxsize
   USBEndpointConfig, 1139
in_params
   USBDriver, 1134
in_state
   USBEndpointConfig, 1139
Inner Code, 917
Input Formatter Utility, 359
   chscanf, 360
   CHSCNF_USE_FLOAT, 359
   chsnscanf, 361
   chvsscanf, 359
   chvsnscanf, 363
input_buffers_queue_t
   I/O Buffers Queues, 67
input_queue_t
   I/O Bytes Queues, 577
int_in
   SerialUSBConfig, 1114
Interfaces and Classes, 916
io_buffers_queue, 1058
   bcounter, 1059
   bn, 1060
   brdptr, 1060
   bsize, 1060
   btop, 1060
   buffers, 1060
   bwrptr, 1059
   link, 1061
   notify, 1061
   ptr, 1061
   suspended, 1059
   top, 1061
   waiting, 1059
io_buffers_queue_t
   I/O Buffers Queues, 66
io_queue, 1062
   q_buffer, 1063
   q_counter, 1063
   q_link, 1064
   q_notify, 1064
   q_rdptr, 1064
   q_top, 1063
   q_waiting, 1063
   q_wrptr, 1063
io_queue_t
   I/O Bytes Queues, 576
IOBus, 1064
   mask, 1065
   offset, 1065
   portid, 1065
ChibiOS HAL
<table>
<thead>
<tr>
<th>Macro/Constant</th>
<th>Type/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOBUS_DECL</td>
<td>PAL Driver</td>
<td>493</td>
</tr>
<tr>
<td>ioline_t</td>
<td>PAL Driver</td>
<td>527</td>
</tr>
<tr>
<td>iomode_t</td>
<td>PAL Driver</td>
<td>527</td>
</tr>
<tr>
<td>iopadid_t</td>
<td>PAL Driver</td>
<td>527</td>
</tr>
<tr>
<td>IOPORT1</td>
<td>PAL Driver</td>
<td>517</td>
</tr>
<tr>
<td>ioportid_t</td>
<td>PAL Driver</td>
<td>527</td>
</tr>
<tr>
<td>ioportmask_t</td>
<td>PAL Driver</td>
<td>527</td>
</tr>
<tr>
<td>IOPORT1</td>
<td>PAL Driver</td>
<td>517</td>
</tr>
<tr>
<td>ioportid_t</td>
<td>PAL Driver</td>
<td>527</td>
</tr>
<tr>
<td>ioportmask_t</td>
<td>PAL Driver</td>
<td>527</td>
</tr>
<tr>
<td>iq_read</td>
<td>I/O Bytes Queues</td>
<td>577</td>
</tr>
<tr>
<td>iqGet</td>
<td>I/O Bytes Queues</td>
<td>573</td>
</tr>
<tr>
<td>iqGetEmpty</td>
<td>I/O Bytes Queues</td>
<td>571</td>
</tr>
<tr>
<td>iqGetFull</td>
<td>I/O Bytes Queues</td>
<td>571</td>
</tr>
<tr>
<td>iqGet</td>
<td>I/O Bytes Queues</td>
<td>581</td>
</tr>
<tr>
<td>iqGetTimeout</td>
<td>I/O Bytes Queues</td>
<td>581</td>
</tr>
<tr>
<td>iqIsEmpty</td>
<td>I/O Bytes Queues</td>
<td>572</td>
</tr>
<tr>
<td>iqIsFull</td>
<td>I/O Bytes Queues</td>
<td>572</td>
</tr>
<tr>
<td>iqObjectInit</td>
<td>I/O Bytes Queues</td>
<td>578</td>
</tr>
<tr>
<td>iqPut</td>
<td>I/O Bytes Queues</td>
<td>580</td>
</tr>
<tr>
<td>iqReadl</td>
<td>I/O Bytes Queues</td>
<td>582</td>
</tr>
<tr>
<td>iqReadTimeout</td>
<td>I/O Bytes Queues</td>
<td>583</td>
</tr>
<tr>
<td>iqResetl</td>
<td>I/O Bytes Queues</td>
<td>579</td>
</tr>
<tr>
<td>link</td>
<td>io_buffers_queue</td>
<td>1061</td>
</tr>
<tr>
<td>LPA_100BASE4</td>
<td>MII/RMII Header</td>
<td>431</td>
</tr>
<tr>
<td>LPA_100FULL</td>
<td>MII/RMII Header</td>
<td>431</td>
</tr>
<tr>
<td>LPA_100HALF</td>
<td>MII/RMII Header</td>
<td>431</td>
</tr>
<tr>
<td>LPA_10FULL</td>
<td>MII/RMII Header</td>
<td>431</td>
</tr>
<tr>
<td>LPA_10HALF</td>
<td>MII/RMII Header</td>
<td>431</td>
</tr>
<tr>
<td>LPA_LPACK</td>
<td>MII/RMII Header</td>
<td>432</td>
</tr>
<tr>
<td>LPA_NPAGE</td>
<td>MII/RMII Header</td>
<td>431</td>
</tr>
<tr>
<td>LPA_PAUSE_ASYM</td>
<td>MII/RMII Header</td>
<td>432</td>
</tr>
<tr>
<td>LPA_PAUSE_CAP</td>
<td>MII/RMII Header</td>
<td>432</td>
</tr>
<tr>
<td>LPA_RESV</td>
<td>MII/RMII Header</td>
<td>432</td>
</tr>
<tr>
<td>LPA_RFault</td>
<td>MII/RMII Header</td>
<td>432</td>
</tr>
<tr>
<td>LPA_SLCT</td>
<td>MII/RMII Header</td>
<td>431</td>
</tr>
<tr>
<td>lscfg</td>
<td>mmc_spi_config_t</td>
<td>1082</td>
</tr>
<tr>
<td>MAC Driver</td>
<td>374</td>
<td></td>
</tr>
<tr>
<td>__mac_callback</td>
<td>377</td>
<td></td>
</tr>
<tr>
<td>__mac_rx_wakeup</td>
<td>378</td>
<td></td>
</tr>
<tr>
<td>__mac_tx_wakeup</td>
<td>377</td>
<td></td>
</tr>
<tr>
<td>ETHD1</td>
<td>398</td>
<td></td>
</tr>
<tr>
<td>MAC_ACTIVE</td>
<td>386</td>
<td></td>
</tr>
<tr>
<td>mac_lld_config_fields</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>mac_lld_driver_fields</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>mac_lld_get_next_receive_buffer</td>
<td>397</td>
<td></td>
</tr>
<tr>
<td>mac_lld_get_next_transmit_buffer</td>
<td>396</td>
<td></td>
</tr>
<tr>
<td>mac_lld_get_descriptor</td>
<td>393</td>
<td></td>
</tr>
<tr>
<td>mac_lld_init</td>
<td>392</td>
<td></td>
</tr>
<tr>
<td>mac_lld_poll_link_status</td>
<td>395</td>
<td></td>
</tr>
<tr>
<td>mac_lld_read_receive_descriptor</td>
<td>396</td>
<td></td>
</tr>
<tr>
<td>mac_lld_receive_descriptor_fields</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>mac_lld_release_receive_descriptor</td>
<td>395</td>
<td></td>
</tr>
<tr>
<td>mac_lld_release_transmit_descriptor</td>
<td>394</td>
<td></td>
</tr>
<tr>
<td>mac_lld_start</td>
<td>392</td>
<td></td>
</tr>
<tr>
<td>mac_lld_stop</td>
<td>393</td>
<td></td>
</tr>
<tr>
<td>mac_lld_transmit_descriptor_fields</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>mac_lld_write_transmit_descriptor</td>
<td>395</td>
<td></td>
</tr>
<tr>
<td>MAC_STOP</td>
<td>386</td>
<td></td>
</tr>
<tr>
<td>MAC_SUPPORTS_ZERO_COPY</td>
<td>383</td>
<td></td>
</tr>
<tr>
<td>MAC_UNINIT</td>
<td>386</td>
<td></td>
</tr>
<tr>
<td>MAC_USE_EVENTS</td>
<td>377</td>
<td></td>
</tr>
<tr>
<td>MAC_USE_ZERO_COPY</td>
<td>377</td>
<td></td>
</tr>
<tr>
<td>macccb_t</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>MACConfig</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>MACDriver</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>macGetAndClearEventsI</td>
<td>388</td>
<td></td>
</tr>
<tr>
<td>macGetEventSource</td>
<td>379</td>
<td></td>
</tr>
<tr>
<td>macGetNextReceiveBuffer</td>
<td>383</td>
<td></td>
</tr>
<tr>
<td>macGetNextTransmitBuffer</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>macGetReceiveDescriptorX</td>
<td>380</td>
<td></td>
</tr>
<tr>
<td>macGetTransmitDescriptorX</td>
<td>379</td>
<td></td>
</tr>
<tr>
<td>macInit</td>
<td>386</td>
<td></td>
</tr>
<tr>
<td>macObjectInit</td>
<td>386</td>
<td></td>
</tr>
<tr>
<td>macPollLinkStatus</td>
<td>391</td>
<td></td>
</tr>
<tr>
<td>macReadReceiveDescriptor</td>
<td>381</td>
<td></td>
</tr>
<tr>
<td>MACReceiveDescriptor</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>macReleaseReceiveDescriptorX</td>
<td>381</td>
<td></td>
</tr>
<tr>
<td>macReleaseTransmitDescriptorX</td>
<td>380</td>
<td></td>
</tr>
<tr>
<td>macSetCallbackX</td>
<td>378</td>
<td></td>
</tr>
<tr>
<td>macStart</td>
<td>387</td>
<td></td>
</tr>
<tr>
<td>macstate_t</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>macStop</td>
<td>388</td>
<td></td>
</tr>
<tr>
<td>MACTransmitDescriptor</td>
<td>385</td>
<td></td>
</tr>
</tbody>
</table>
MFS_CFG_BUFFER_SIZE, 403
MFS_CFG_MAX_RECORDS, 402
MFS_CFG_MAX_REPAIR_ATTEMPTS, 403
MFS_CFG_MEMORY_ALIGNMENT, 403
MFS_CFG_STRONG_CHECKING, 403
MFS_CFG_TRANSACTION_MAX, 404
MFS_CFG_WRITE_VERIFY, 403
mfs__error_t, 405
mfs_flash_copy, 406
mfs_flash_read, 405
mfs_flash_write, 406
mfs__garbage_collect, 411
mfs_id_t, 404
mfs__mount, 412
mfs__nocache_buffer_t, 404
mfs__state_t, 404
mfs__try__mount, 411
mfsCommitTransaction, 418
mfsErase, 414
mfsEraseRecord, 416
mfsObjectInit, 412
mfsPerformGarbageCollection, 417
mfsReadRecord, 414
mfsRollbackTransaction, 419
mfsStart, 413
mfsStartTransaction, 417
mfsStop, 413
mfsWriteRecord, 415
PAIR, 402
RET_ON_ERROR, 402
mask
IOBus, 1065
Memory Streams Class, 342
  _memory_stream_data, 342
  mfsObjectInit, 342
MemoryStream, 1066
vmt, 1067
memstreams.c, 1309
memstreams.h, 1310
MemStreamVMT, 1068
mfs__bank_check_header
  Managed Flash Storage Driver, 409
mfs__bank__erase
  Managed Flash Storage Driver, 407
mfs__bank__get__state
  Managed Flash Storage Driver, 410
mfs__bank__header_t, 1068
counter, 1069
crc, 1069
magic1, 1069
magic2, 1069
reserved1, 1069
mfs__bank__scan__records
  Managed Flash Storage Driver, 409
mfs__bank__state_t
  Managed Flash Storage Driver, 405
mfs__bank_t
  Managed Flash Storage Driver, 404
mfs_bank__verify__erase
  Managed Flash Storage Driver, 408
mfs_bank__write__header
  Managed Flash Storage Driver, 408
MFS_CFG_BUFFER_SIZE
  Managed Flash Storage Driver, 403
MFS_CFG_MAX_RECORDS
  Managed Flash Storage Driver, 402
MFS_CFG_MAX_REPAIR_ATTEMPTS
  Managed Flash Storage Driver, 403
MFS_CFG_MEMORY_ALIGNMENT
  Managed Flash Storage Driver, 403
MFS_CFG_STRONG_CHECKING
  Managed Flash Storage Driver, 403
MFS_CFG_TRANSACTION_MAX
  Managed Flash Storage Driver, 404
MFS_CFG_WRITE_VERIFY
  Managed Flash Storage Driver, 403
mfs_data__header_t, 1070
crc, 1071
id, 1071
magic1, 1070
magic2, 1070
size, 1071
mfs__error_t
  Managed Flash Storage Driver, 405
mfs_flash_copy
  Managed Flash Storage Driver, 406
mfs_flash_read
  Managed Flash Storage Driver, 405
mfs_flash_write
  Managed Flash Storage Driver, 406
mfs__garbage__collect
  Managed Flash Storage Driver, 411
mfs_id_t
  Managed Flash Storage Driver, 411
mfs__mount
  Managed Flash Storage Driver, 404
mfs__nocache_buffer
  Managed Flash Storage Driver, 412
mfs__nocache_buffer_t
  Managed Flash Storage Driver, 404
mfs__state_t
  Managed Flash Storage Driver, 404
mfs__transaction__op_t, 1072
id, 1073
offset, 1073
size, 1073
mfs__try__mount
  Managed Flash Storage Driver, 411
mfsCommitTransaction
  Managed Flash Storage Driver, 418
MFSConfig, 1074
  bank0__sectors, 1076
  bank0__start, 1075
  bank1__sectors, 1076
  bank1__start, 1076
  bank__size, 1075
  erased, 1075
<table>
<thead>
<tr>
<th>Flash Memory Driver Functions</th>
<th>Managed Flash Storage Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mfsErase</code></td>
<td>Managed Flash Storage Driver, 414</td>
</tr>
<tr>
<td><code>mfsEraseRecord</code></td>
<td>Managed Flash Storage Driver, 416</td>
</tr>
<tr>
<td><code>mfsObjectInit</code></td>
<td>Managed Flash Storage Driver, 412</td>
</tr>
<tr>
<td><code>mfsPerformGarbageCollection</code></td>
<td>Managed Flash Storage Driver, 417</td>
</tr>
<tr>
<td><code>mfsReadRecord</code></td>
<td>Managed Flash Storage Driver, 414</td>
</tr>
<tr>
<td><code>mfsRollbackTransaction</code></td>
<td>Managed Flash Storage Driver, 419</td>
</tr>
<tr>
<td><code>mfsStart</code></td>
<td>Managed Flash Storage Driver, 413</td>
</tr>
<tr>
<td><code>mfsStartTransaction</code></td>
<td>Managed Flash Storage Driver, 417</td>
</tr>
<tr>
<td><code>mfsStop</code></td>
<td>Managed Flash Storage Driver, 413</td>
</tr>
<tr>
<td><code>mfsWriteRecord</code></td>
<td>Managed Flash Storage Driver, 415</td>
</tr>
</tbody>
</table>

### MII/RMII Header

<table>
<thead>
<tr>
<th>MII/RMII Header Functions</th>
<th>MII/RMII Header</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADVERTISE_100BASE4</code></td>
<td>430</td>
</tr>
<tr>
<td><code>ADVERTISE_100FULL</code></td>
<td>430</td>
</tr>
<tr>
<td><code>ADVERTISE_100HALF</code></td>
<td>429</td>
</tr>
<tr>
<td><code>ADVERTISE_10FULL</code></td>
<td>429</td>
</tr>
<tr>
<td><code>ADVERTISE_1HALF</code></td>
<td>429</td>
</tr>
<tr>
<td><code>ADVERTISE_CSMA</code></td>
<td>429</td>
</tr>
<tr>
<td><code>ADVERTISE_LPACK</code></td>
<td>430</td>
</tr>
<tr>
<td><code>ADVERTISE_NPAGE</code></td>
<td>431</td>
</tr>
<tr>
<td><code>ADVERTISE_PAUSEASYM</code></td>
<td>430</td>
</tr>
<tr>
<td><code>ADVERTISE_PAUSECAP</code></td>
<td>430</td>
</tr>
<tr>
<td><code>ADVERTISE_RESV</code></td>
<td>430</td>
</tr>
<tr>
<td><code>ADVERTISE_RFAULT</code></td>
<td>430</td>
</tr>
<tr>
<td><code>ADVERTISE_SLCT</code></td>
<td>429</td>
</tr>
<tr>
<td><code>BMCR_ANENABLE</code></td>
<td>426</td>
</tr>
<tr>
<td><code>BMCR_ANRESTART</code></td>
<td>426</td>
</tr>
<tr>
<td><code>BMCR_CTST</code></td>
<td>426</td>
</tr>
<tr>
<td><code>BMCR_FULDLPLX</code></td>
<td>426</td>
</tr>
<tr>
<td><code>BMCR_ISOLATE</code></td>
<td>426</td>
</tr>
<tr>
<td><code>BMCR_LOOPBACK</code></td>
<td>427</td>
</tr>
<tr>
<td><code>BMCR_PDOWN</code></td>
<td>426</td>
</tr>
<tr>
<td><code>BMCR_RESET</code></td>
<td>427</td>
</tr>
<tr>
<td><code>BMCR_RESV</code></td>
<td>426</td>
</tr>
<tr>
<td><code>BMCR_SPEED100</code></td>
<td>427</td>
</tr>
<tr>
<td><code>BMSR_100BASE4</code></td>
<td>429</td>
</tr>
<tr>
<td><code>BMSR_100FULL</code></td>
<td>429</td>
</tr>
<tr>
<td><code>BMSR_100HALF</code></td>
<td>428</td>
</tr>
<tr>
<td><code>BMSR_10FULL</code></td>
<td>428</td>
</tr>
<tr>
<td><code>BMSR_10HALF</code></td>
<td>428</td>
</tr>
<tr>
<td><code>BMSR_ANEGCAPABLE</code></td>
<td>427</td>
</tr>
<tr>
<td><code>BMSR_ANEGCOMPLETE</code></td>
<td>428</td>
</tr>
<tr>
<td><code>BMSR_ERCCAP</code></td>
<td>427</td>
</tr>
<tr>
<td><code>BMSR_JCD</code></td>
<td>427</td>
</tr>
<tr>
<td><code>BMSR_LSTATUS</code></td>
<td>427</td>
</tr>
<tr>
<td><code>BMSR_MFPRESCAPACITY</code></td>
<td>428</td>
</tr>
<tr>
<td><code>BMSR_RESV</code></td>
<td>428</td>
</tr>
<tr>
<td><code>BMSR_RFAULT</code></td>
<td>428</td>
</tr>
<tr>
<td><code>EXPANSION_ENABLENPAGE</code></td>
<td>433</td>
</tr>
<tr>
<td><code>EXPANSION_LCPWP</code></td>
<td>433</td>
</tr>
<tr>
<td><code>EXPANSION_MFAULTS</code></td>
<td>433</td>
</tr>
<tr>
<td><code>EXPANSION_NCPABILITY</code></td>
<td>433</td>
</tr>
<tr>
<td><code>EXPANSION_NWAY</code></td>
<td>432</td>
</tr>
<tr>
<td><code>EXPANSION_RESV</code></td>
<td>433</td>
</tr>
<tr>
<td><code>LPA_100BASE4</code></td>
<td>431</td>
</tr>
<tr>
<td><code>LPA_100FULL</code></td>
<td>431</td>
</tr>
<tr>
<td><code>LPA_100HALF</code></td>
<td>431</td>
</tr>
<tr>
<td><code>LPA_10FULL</code></td>
<td>431</td>
</tr>
<tr>
<td><code>LPA_10HALF</code></td>
<td>431</td>
</tr>
<tr>
<td><code>LPA_NPAGE</code></td>
<td>432</td>
</tr>
<tr>
<td><code>LPA_PAUSEASYM</code></td>
<td>432</td>
</tr>
<tr>
<td><code>LPA_PAUSECAP</code></td>
<td>432</td>
</tr>
<tr>
<td><code>LPA_RESV</code></td>
<td>432</td>
</tr>
<tr>
<td><code>LPA_RFAULT</code></td>
<td>432</td>
</tr>
<tr>
<td><code>LPA_SLCT</code></td>
<td>431</td>
</tr>
<tr>
<td><code>MII_ADVERTISE</code></td>
<td>423</td>
</tr>
<tr>
<td><code>MII_ANNPTR</code></td>
<td>423</td>
</tr>
<tr>
<td><code>MII_BMCR</code></td>
<td>422</td>
</tr>
<tr>
<td><code>MII_BMSR</code></td>
<td>422</td>
</tr>
<tr>
<td><code>MII_CTRL1000</code></td>
<td>423</td>
</tr>
<tr>
<td><code>MII_DCOUNTER</code></td>
<td>424</td>
</tr>
<tr>
<td><code>MII_ESTATUS</code></td>
<td>424</td>
</tr>
<tr>
<td><code>MII_EXPANSION</code></td>
<td>423</td>
</tr>
<tr>
<td><code>MII_FCSCOUNTER</code></td>
<td>424</td>
</tr>
<tr>
<td><code>MII_LBERROR</code></td>
<td>425</td>
</tr>
<tr>
<td><code>MII_LPA</code></td>
<td>423</td>
</tr>
<tr>
<td><code>MII_MICR</code></td>
<td>424</td>
</tr>
<tr>
<td><code>MII_NCONFIG</code></td>
<td>425</td>
</tr>
<tr>
<td><code>MII_NWAYTEST</code></td>
<td>424</td>
</tr>
<tr>
<td><code>MII_PHYADDR</code></td>
<td>425</td>
</tr>
<tr>
<td><code>MII_PHYADD1</code></td>
<td>422</td>
</tr>
<tr>
<td><code>MII_PHYADD2</code></td>
<td>423</td>
</tr>
<tr>
<td><code>MII_PHYSTS</code></td>
<td>424</td>
</tr>
<tr>
<td><code>MII_PHYST</code></td>
<td>424</td>
</tr>
<tr>
<td><code>MII_RESV1</code></td>
<td>425</td>
</tr>
<tr>
<td><code>MII_RESV2</code></td>
<td>425</td>
</tr>
<tr>
<td><code>MII_SREVISION</code></td>
<td>425</td>
</tr>
<tr>
<td><code>MII_STAT1000</code></td>
<td>423</td>
</tr>
<tr>
<td><code>MII_TPSTATUS</code></td>
<td>425</td>
</tr>
<tr>
<td><code>NWAYTEST_LOOPBACK</code></td>
<td>433</td>
</tr>
<tr>
<td><code>NWAYTEST_RESV1</code></td>
<td>433</td>
</tr>
<tr>
<td><code>NWAYTEST_RESV2</code></td>
<td>434</td>
</tr>
</tbody>
</table>
MII ADVERTISE
MII/RMII Header, 423

MII_ANNPTR
MII/RMII Header, 423

MII_BMCR
MII/RMII Header, 422

MII_BMSR
MII/RMII Header, 422

MII_CTRL
MII/RMII Header, 423

MII_DCOUNTER
MII/RMII Header, 424

MII_ESTATUS
MII/RMII Header, 424

MII_EXPANSION
MII/RMII Header, 424

MII_FCSCOUNTER
MII/RMII Header, 424

MII_LBRERROR
MII/RMII Header, 425

MII_LPA
MII/RMII Header, 423

MII_MICR
MII/RMII Header, 424

MII_NCONFIG
MII/RMII Header, 425

MII_NWAYTEST
MII/RMII Header, 424

MII_PHYADDR
MII/RMII Header, 425

MII_PHYID1
MII/RMII Header, 422

MII_PHYID2
MII/RMII Header, 423

MII_PHYSTS
MII/RMII Header, 424

MII_RERRCOUNTER
MII/RMII Header, 424

MII_RESV1
MII/RMII Header, 425

MII_RESV2
MII/RMII Header, 425

MII_SREVISION
MII/RMII Header, 425

MII_STAT1000
MII/RMII Header, 423

MII_TPISTATUS
MII/RMII Header, 425

millisecond

RTCDateTime, 1103

MMC over SPI Driver, 435

_mmcsd_block_device_data, 460
_mmcsd_block_device_methods, 460
_mmcsd_get_capacity, 462
_mmcsd_get_capacity_ext, 463
_mmcsd_get_slice, 462
_mmcsd_unpack_csd_mmc, 465
_mmcsd_unpack_csd_v10, 465
_mmcsd_unpack_csd_v20, 466
_mmcsd_unpack_mmc_cid, 464
_mmcsd_unpack_sdc_cid, 463
MMCSD_BLOCK_SIZE, 459
MMCSD_CMD8_PATTERN, 460
MMCSD_R1_ERROR, 460
MMCSD_R1_ERROR_MASK, 459
MMCSD_R1_IS_CARD_LOCKED, 461
MMCSD_R1_STS, 461
_mmcsdGetCardCapacity, 461

MMC_BUFFER_SIZE
MMC over SPI Driver, 437

mmc_cmd6_construct
SDC Driver, 614

mmc_crc7
MMC over SPI Driver, 440

mmc_crc7_lookup_table
MMC over SPI Driver, 454

mmc_detect_bus_clk
SDC Driver, 618

MMC_IDLE_TIMEOUT_MS
Configuration, 907
MMC over SPI Driver, 437

mmc_init
### ChibiOS/HAL

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDC Driver</td>
<td>MMC/SD Block Device</td>
</tr>
<tr>
<td>mmc_read_CxD</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_recvr1</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_recvr3</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_send_command_R1</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_send_command_R3</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_send_hdr</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_set_bus_width</td>
<td>SDC Driver</td>
</tr>
<tr>
<td>mmc_spi_config_t, 1080</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>hscfg, 1082</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>lscfg, 1082</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>spip, 1082</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_spi_driver_t, 1082</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>block_addresses, 1085</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>buffer, 1085</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>config, 1085</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>vmt, 1085</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_spi_driver_vmt, 1086</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_wait_idle</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_C.resources</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>MMC_CONFIG</td>
<td>Configuration, 907</td>
</tr>
<tr>
<td>MMC over SPI Driver</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_vmt</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_wait_idle</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmc_C.resources</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcConnect</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcDisconnect</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcDriver</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcCSD_R1</td>
<td>MMC/SD Block Device</td>
</tr>
<tr>
<td>MMCSD_CSD_R1_ERROR</td>
<td>MMC/SD Block Device</td>
</tr>
<tr>
<td>MMCSD_CSD_R1_ERROR_MASK</td>
<td>MMC/SD Block Device</td>
</tr>
<tr>
<td>MMCSD_CSD_R1_IS_CARD_LOCKED</td>
<td>MMC/SD Block Device</td>
</tr>
<tr>
<td>MMCSD_CSD_R1_STS</td>
<td>MMC/SD Block Device</td>
</tr>
<tr>
<td>MMCSDBlockDevice, 1087</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>vmt, 1089</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>MMCSDBlockDeviceVMT, 1090</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcsdGetCardCapacity</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcSequentialRead</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcSequentialWrite</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcStart</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcStartSequentialRead</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcStartSequentialWrite</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcStop</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcStopSequentialRead</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcStopSequentialWrite</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mmcSync</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>MMC over SPI Driver</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mode</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>ICUConfig</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>PWMChannelConfig, 1096</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>month</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>RTCDriver</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>msg_t</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>msObjectInit</td>
<td>MMC over SPI Driver</td>
</tr>
<tr>
<td>mutex</td>
<td>Memory Streams Class, 342</td>
</tr>
<tr>
<td>mutex_t</td>
<td>Memory Streams Class, 342</td>
</tr>
<tr>
<td>ncbuf</td>
<td>MFSDriver, 1080</td>
</tr>
<tr>
<td>next_offset</td>
<td>MFSDriver, 1079</td>
</tr>
<tr>
<td>nocache</td>
<td>SNORDriver, 1124</td>
</tr>
</tbody>
</table>
Normal Drivers, 913
notify
io_buffers_queue, 1061
Null Streams Class, 344
_null_stream_data, 344
nullObjectInit, 344
num_channels
hal_adc_configuration_group, 1005
hal_dac_conversion_group, 1014
NWAYTEST_LOOPBACK
MII/RMII Header, 433
NWAYTEST_RESV1
MII/RMII Header, 433
NWAYTEST_RESV2
MII/RMII Header, 434
objcGetInstance
Base Object, 481
obnotify
Serial over USB Driver, 661
obqFlush
I/O Buffers Queues, 84
obqGetEmptyBufferTimeout
I/O Buffers Queues, 78
obqGetEmptyBufferTimeoutS
I/O Buffers Queues, 79
obqGetFullBuffer
I/O Buffers Queues, 76
obqIsEmpty
I/O Buffers Queues, 65
obqIsFull
I/O Buffers Queues, 66
obqObjectInit
I/O Buffers Queues, 75
obqPostFullBuffer
I/O Buffers Queues, 80
obqPostFullBufferS
I/O Buffers Queues, 81
obqPutTimeout
I/O Buffers Queues, 81
obqReleaseEmptyBuffer
I/O Buffers Queues, 77
obqReset
I/O Buffers Queues, 76
obqTryFlush
I/O Buffers Queues, 83
obqWriteTimeout
I/O Buffers Queues, 82
offset
flash_sector_descriptor_t, 1000
hal_mac_receive_descriptor, 1030
hal_mac_transmit_descriptor, 1031
IOBus, 1065
mfs_transaction_op_t, 1073
oq_write
I/O Bytes Queues, 578
oqGetEmpty
I/O Bytes Queues, 574
oqGetFull
I/O Bytes Queues, 573
oqGet
I/O Bytes Queues, 587
oqIsEmpty
I/O Bytes Queues, 575
oqIsFull
I/O Bytes Queues, 575
oqObjectInit
I/O Bytes Queues, 584
oqPut
I/O Bytes Queues, 576
oqPutI
I/O Bytes Queues, 586
oqPutTimeout
I/O Bytes Queues, 586
oqReset
I/O Bytes Queues, 585
oqWrite
I/O Bytes Queues, 588
oqWriteTimeout
I/O Bytes Queues, 589
OSAL, 919
event_source_t, 936
eventcallback_t, 936
eventflags_t, 936
msg_t, 935
mutex_t, 936
OSAL_DBG_ENABLE_ASSERTS, 925
OSAL_DBG_ENABLE_CHECKS, 925
osal_halt_msg, 953
OSAL_I2MS, 930
OSAL_I2S, 929
OSAL_I2US, 931
OSAL_IRQ_EPILOGUE, 927
OSAL_IRQ_HANDLER, 927
OSAL_IRQ_IS_VALID_PRIORITY, 927
OSAL_IRQ_MAXIMUM_PRIORITY, 925
OSAL_IRQ_PRIORITY_LEVELS, 925
OSAL_IRQ_PROLOGUE, 927
OSAL_MS2I, 928
OSAL_MS2RTC, 932
OSAL_S2I, 928
OSAL_S2RTC, 931
osalDbgAssert, 925
osalDbgCheck, 926
osalDbgCheckClassI, 926
osalDbgCheckClassS, 926
osalEventBroadcastFlags, 945
osalEventBroadcastFlagsI, 945
osalEventSetCallback, 946
osalInit, 937
osalMutexLock, 946
osalMutexObjectInit, 952
osalMutexUnlock, 947
osalOsGetSystemTimeX, 939
osalOsRescheduleS, 938
osalOsTimerHandlerI, 938
osalSysDisable, 947
osalSysEnable, 948
osalSysGetStatusAndLockX, 949
osalSysHalt, 937
osalSysLock, 948
osalSysLockFromISR, 948
osalSysPolledDelayX, 938
osalSysRestoreStatusX, 949
osalSysUnlock, 948
osalSysUnlockFromISR, 949
osalThreadDequeAllI, 943
osalThreadDequeNextI, 943
osalThreadEnqueueTimeoutS, 942
osalThreadResumeI, 951
osalThreadResumeS, 942
osalThreadSleep, 940
osalThreadSleepMicroseconds, 934
osalThreadSleepMilliseconds, 934
osalThreadSleepS, 939
osalThreadSleepSeconds, 933
osalThreadSuspendI, 940
osalThreadSuspendTimeoutS, 941
osalTimeAddX, 950
osalTimeDiffX, 950
osalTimeIsInRangeX, 951
rtcnt_t, 935
sysinterval_t, 935
syssts_t, 935
systime_t, 935
thread_reference_t, 936
time_conv_t, 935
osal.c, 1311
osal.h, 1313
OSAL_DBG_ENABLE_ASSERTS
osal, 925
OSAL_DBG_ENABLE_CHECKS
osal, 925
osal_halt_msg
osal, 953
OSAL_I2MS
osal, 930
OSAL_I2S
osal, 929
OSAL_I2US
osal, 931
OSAL_IRQ_EPILOGUE
osal, 927
OSAL_IRQ_HANDLER
osal, 927
OSAL_IRQ_IS_VALID_PRIORITY
osal, 927
OSAL_IRQ_MAXIMUM_PRIORITY
osal, 925
OSAL_IRQ_PRIORITY_LEVELS
osal, 925
OSAL_IRQ_PROLOGUE
osal, 927
OSAL_MS2I
osal, 928
OSAL_MS2RTC
osal, 932
OSAL_S2I
osal, 928
OSAL_S2RTC
osal, 931
OSAL_ST_FREQUENCY
osal, 924
OSAL_ST_MODE
osal, 924
OSAL_ST_RESOLUTION
osal, 924
OSAL_US2I
osal, 929
OSAL_US2RTC
osal, 933
osalDbgAssert
osal, 925
osalDbgCheck
osal, 926
osalDbgCheckClassS
osal, 926
osalDbgCheckClassS
osal, 926
osalEventBroadcastFlags
osal, 945
osalEventBroadcastFlagsI
osal, 945
osalEventObjectInit
osal, 952
osalInit
osal, 937
osalMutexLock
osal, 946
osalMutexObjectInit
osal, 952
osalMutexUnlock
osal, 947
osalOsGetSystemTimeX
osal, 939
osalOsRescheduleS
osal, 938
osalOsTimerHandlerI
OSAL, 938
osalSysDisable
OSAL, 947
osalSysEnable
OSAL, 948
osalSysGetStatusAndLockX
OSAL, 949
osalSysHalt
OSAL, 937
osalSysLock
OSAL, 948
osalSysLockFromISR
OSAL, 948
osalSysPolledDelayX
OSAL, 938
osalSysRestoreStatusX
OSAL, 949
osalSysUnlock
OSAL, 948
osalSysUnlockFromISR
OSAL, 949
osalThreadDequeueAllI
OSAL, 943
osalThreadDequeueNextI
OSAL, 943
osalThreadEnqueueTimeoutS
OSAL, 942
osalThreadQueueObjectInit
OSAL, 951
osalThreadResumeI
OSAL, 941
osalThreadResumeS
OSAL, 941
osalThreadSleep
OSAL, 940
osalThreadSleepMicroseconds
OSAL, 934
osalThreadSleepMilliseconds
OSAL, 934
osalThreadSleepS
OSAL, 939
osalThreadSleepSeconds
OSAL, 933
osalThreadSuspendS
OSAL, 940
osalThreadSuspendTimeoutS
OSAL, 941
osalTimeAddX
OSAL, 950
osalTimeDiffX
OSAL, 950
osalTimelsInRangeX
OSAL, 951
out
hal_usb_lld.c, 1297
USB Driver, 857
out_cb
USBEndpointConfig, 1139
out_maxsize
USBEndpointConfig, 1139
out_params
USBDriver, 1134
out_state
USBEndpointConfig, 1139
Output Formatter Utility, 353
chprintf, 354
CHPRINTF_USE_FLOAT, 353
chsnprintf, 355
chvprintf, 353
chvsnprintf, 357
output_buffers_queue_t
I/O Buffers Queues, 67
output_queue_t
I/O Bytes Queues, 577
overflow_cb
ICUConfig, 1056
page_size
flash_descriptor_t, 998
PAIR
Managed Flash Storage Driver, 402
PAL Driver, 482
_IOBUS_DATA, 492
_pal_clear_event, 494, 495
_pal_init_event, 493, 494
_pal_isr_code, 496, 497
_pal_lld_init, 534
_pal_lld_setgroupmode, 534
IUSB_DIGEST, 493
ioline_t, 527
iomode_t, 527
iopadid_t, 527
IOPORT1, 527
ioportid_t, 527
ioportmask_t, 527
PAL_EVENT_MODE_BOTH_EDGES, 491
PAL_EVENT_MODE_DISABLED, 490
PAL_EVENT_MODE_EDGES_MASK, 490
PAL_EVENT_MODE_FALLING_EDGE, 490
PAL_EVENT_MODE_RISING_EDGE, 490
PAL_GROUP_MASK, 492
PAL_HIGH, 490
PAL_IOPORTS_WIDTH, 516
PAL_LINE, 516
pal_lld_clearpad, 524
pal_lld_clearport, 519
pal_lld_get_line_event, 526
pal_lld_get_pad_event, 526
pal_lld_init, 517
pal_lld_readgroup, 520
pal_lld_readlatch, 518
pal_lld_readpad, 522
pal_lld_readport, 517
pal_lld_setgroupmode, 521
pal_lld_setpad, 523
pal_lld_setpadmode, 525
palWaitLineTimeout
   PAL Driver, 533
palWaitLineTimeoutS
   PAL Driver, 532
palWaitPadTimeout
   PAL Driver, 531
palWaitPadTimeoutS
   PAL Driver, 530
palWriteBus
   PAL Driver, 528
palWriteGroup
   PAL Driver, 502
palWriteLine
   PAL Driver, 507
palWritePad
   PAL Driver, 504
palWritePort
   PAL Driver, 498
param
   event_source, 993
   period
      PWMConfig, 1098
      PWMDriver, 1100
period_cb
   ICUConfig, 1056
Peripheral Interfaces, 536
PLATFORM_ADC_USE_ADC1
   ADC Driver, 42
PLATFORM_CAN_USE_CAN1
   CAN Driver, 92
PLATFORM_CRY_USE_CRY1
   Crypto Driver, 115
PLATFORM_DAC_USE_DAC1
   DAC Driver, 194
PLATFORM_EFL_USE_EFL1
   EFL Driver, 212
PLATFORM_GPT_USE_GPT1
   GPT Driver, 242
PLATFORM_I2C_USE_I2C1
   I2C Driver, 262
PLATFORM_I2S_USE_I2S1
   I2S Driver, 277
PLATFORM_ICU_USE_ICU1
   ICU Driver, 294
PLATFORM_MAC_USE_MAC1
   MAC Driver, 383
PLATFORM_PWM_USE_PWM1
   PWM Driver, 548
PLATFORM_RTC_USE_RTC1
   RTC Driver, 594
PLATFORM_SDC_USE_SDC1
   SDC Driver, 610
PLATFORM_SERIAL_USE_USART1
   Serial Driver, 646
PLATFORM_SIO_USE_SIO1
   SIO Driver, 687
PLATFORM_SPI_USE_SPI1
   SPIv1 Driver, 723
   SPIv2 Driver, 742
PLATFORM_TRNG_USE_TRNG1
   TRNG Driver, 765
PLATFORM_UART_USE_UART1
   UART Driver, 787
PLATFORM_USB_USE_USB1
   USB Driver, 829
PLATFORM_WGD_USE_WGD1
   WGD Driver, 862
PLATFORM_WSPI_USE_WSPI1
   WSPI Driver, 877
portid
   IOBus, 1065
ps_error_t
   Abstract Persistent Storage Interface, 340
ps_offset_t
   Abstract Persistent Storage Interface, 340
psGetStorageSize
   Abstract Persistent Storage Interface, 338
psRead
   Abstract Persistent Storage Interface, 339
psWrite
   Abstract Persistent Storage Interface, 339
ptr
   io_buffers_queue, 1061
PWM Driver, 537
   PLATFORM_PWM_USE_PWM1, 548
   PWM_CHANNELS, 548
   PWM_DEGREES_TO_WIDTH, 542
   PWM_FRACTION_TO_WIDTH, 541
   pwm_lld_change_period, 548
   pwm_lld_disabled_channels, 562
   pwm_lld_disable_channel_notification, 564
   pwm_lld_disable_periodic_notification, 563
   pwm_lld_enable_channel, 561
   pwm_lld_enable_channel_notification, 563
   pwm_lld_enable_periodic_notification, 562
   pwm_lld_init, 560
   pwm_lld_start, 560
   pwm_lld_stop, 561
   PWM_OUTPUT_ACTIVE_HIGH, 541
   PWM_OUTPUT_ACTIVE_LOW, 541
   PWM_OUTPUT_DISABLED, 540
   PWM_OUTPUT_MASK, 540
   PWM_PERCENTAGE_TO_WIDTH, 542
   PWM_READY, 551
   PWM_STOP, 551
   PWM_UNINIT, 551
   pwmcallback_t, 549
   pwmChangePeriod, 553
   pwmChangePeriod_t, 543
   pwmchannel_t, 550
   pwmchnmsk_t, 550
   pwmcnt_t, 550
   PWMD1, 564
   pwmDisableChannel, 555
   pwmDisableChannel_t, 544
   pwmDisableChannelNotification, 559
pwmDisableChannelNotificationI, 547
pwmDisablePeriodicNotification, 557
pwmDisablePeriodicNotificationI, 546
PWMDriver, 549
pwmEnableChannel, 554
pwmEnableChannelI, 544
pwmEnableChannelNotification, 558
pwmEnableChannelNotificationI, 547
pwmEnablePeriodicNotification, 556
pwmEnablePeriodicNotificationI, 546
pwmInit, 551
pwmIsChannelEnabledI, 545
pwmmode_t, 550
pwmObjectInit, 551
pwmStart, 552
pwmstate_t, 550
pwmStop, 552
PWM_CHANNELS
PWMDriver, 548
PWM_DEGREES_TO_WIDTH
PWMDriver, 542
PWM_FRACTION_TO_WIDTH
PWMDriver, 541
pwm_lld_change_period
PWMDriver, 548
pwm_lld_disable_channel
PWMDriver, 562
pwm_lld_disable_channel_notification
PWMDriver, 564
pwm_lld_disable_periodic_notification
PWMDriver, 563
pwm_lld_enable_channel
PWMDriver, 561
pwm_lld_enable_channel_notification
PWMDriver, 563
pwm_lld_enable_periodic_notification
PWMDriver, 562
pwm_lld_init
PWMDriver, 560
pwm_lld_start
PWMDriver, 560
pwm_lld_stop
PWMDriver, 561
PWM_OUTPUT_ACTIVE_HIGH
PWMDriver, 541
PWM_OUTPUT_ACTIVE_LOW
PWMDriver, 541
PWM_OUTPUT_DISABLED
PWMDriver, 540
PWM_OUTPUT_MASK
PWMDriver, 540
PWM_PERCENTAGE_TO_WIDTH
PWMDriver, 542
PWM_READY
PWMDriver, 551
PWM_STOP
PWMDriver, 551
PWM_UNINIT
PWMDriver, 551
pwmcallback_t
PWMDriver, 549
pwmChangePeriod
PWMDriver, 553
pwmChangePeriodI
PWMDriver, 543
pwmcallback_t
PWMDriver, 550
PWMChannelConfig, 1095
callback, 1096
mode, 1096
pwmcallback_t
PWMDriver, 550
PWMConfig, 1097
callback, 1098
channels, 1098
frequency, 1098
period, 1098
PWMD1
PWMDriver, 564
pwmDisableChannel
PWMDriver, 555
pwmDisableChannelI
PWMDriver, 544
pwmDisableChannelNotification
PWMDriver, 559
pwmDisableChannelNotificationI
PWMDriver, 547
pwmDisablePeriodicNotification
PWMDriver, 557
pwmDisablePeriodicNotificationI
PWMDriver, 546
PWMDriver, 1099
channels, 1100
config, 1100
enabled, 1100
period, 1100
PWMDriver, 549
state, 1100
pwmEnableChannel
PWMDriver, 554
pwmEnableChannelI
PWMDriver, 544
pwmEnableChannelNotification
PWMDriver, 558
pwmEnableChannelNotificationI
PWMDriver, 547
pwmEnablePeriodicNotification
PWMDriver, 556
pwmEnablePeriodicNotificationI
PWMDriver, 546
pwmInit
PWMDriver, 551
pwmIsChannelEnabledI
PWMDriver, 545
INDEX

SDC_NICE_WAITING, 608
sdc_set_bus_width, 619
sdc_vmt, 636
sdcbusclk_t, 612
sdcbusmode_t, 612
sdcConnect, 624
SDCD1, 637
sdcDisconnect, 624
SDCDriver, 611
sdcErase, 629
sdcflags_t, 611
sdcGetAndClearErrors, 627
sdcGetInfo, 628
sdcInit, 621
sdclsCardInserted, 609
sdclsWriteProtected, 610
sdcmode_t, 611
sdcObjectInit, 622
sdcRead, 625
sdcStart, 622
sdcStop, 623
sdcSync, 627
sdcWrite, 626
sdc_cmd6_check_status
SDC Driver, 617
sdc_cmd6_construct
SDC Driver, 616
sdc_cmd6_extract_info
SDC Driver, 616
sdc_detect_bus_clk
SDC Driver, 617
sdc_init
SDC Driver, 614
SDC_INIT_OCR
Configuration, 908
SDC Driver, 608
SDC_INIT_OCR_V20
Configuration, 908
SDC Driver, 608
SDC_INIT_RETRY
Configuration, 907
SDC Driver, 608
sdc_lid_init
SDC Driver, 629
sdc_lid_read
SDC Driver, 634
sdc_lid_send_cmd_long_crc
SDC Driver, 634
sdc_lid_send_cmd_none
SDC Driver, 632
sdc_lid_send_cmd_short
SDC Driver, 632
sdc_lid_send_cmd_short_crc
SDC Driver, 633
sdc_lid_set_bus_mode
SDC Driver, 632
sdc_lid_set_data_clk
SDC Driver, 631
sdc_lid_start
SDC Driver, 630
sdc_lid_start_clk
SDC Driver, 630
sdc_lid_stop
SDC Driver, 630
sdc_lid_stop_clk
SDC Driver, 631
sdc_lid_sync
SDC Driver, 636
sdc_lid_write
SDC Driver, 635
SDC_MMC_SUPPORT
Configuration, 908
SDC Driver, 608
SDC_NICE_WAITING
Configuration, 908
SDC Driver, 608
sdc_set_bus_width
SDC Driver, 619
sdc_vmt
SDC Driver, 636
sdcbusclk_t
SDC Driver, 612
sdcbusmode_t
SDC Driver, 612
SDCDriver, 1107
buf, 1109
cardmode, 1109
config, 1109
errors, 1109
rca, 1109
SDC Driver, 611
vmt, 1109
SDCDriverVMT, 1110
sdcErase
SDC Driver, 629
sdcflags_t
SDC Driver, 611
sdclsCardInserted
SDC Driver, 609
sdclsWriteProtected
SDC Driver, 610
sdcmode_t
SDC Driver, 621
ChibiOS/HAL
SDC Driver, 611
sdcObjectInit
SDC Driver, 622
sdControl
Serial Driver, 654
sdcRead
SDC Driver, 625
sdcStart
SDC Driver, 622
sdStop
SDC Driver, 623
sdcSync
SDC Driver, 627
sdWrite
SDC Driver, 626
sdGet
Serial Driver, 643
sdGetI
Serial Driver, 642
sdGetTimeout
Serial Driver, 643
sdGetWouldBlock
Serial Driver, 653
sdIncomingDataI
Serial Driver, 650
sdInit
Serial Driver, 648
sdObjectInit
Serial Driver, 648
sdPut
Serial Driver, 642
sdPutI
Serial Driver, 641
sdPutTimeout
Serial Driver, 642
sdPutWouldBlock
Serial Driver, 652
sdRead
Serial Driver, 645
sdReadI
Serial Driver, 645
sdReadTimeout
Serial Driver, 645
sdRequestDataI
Serial Driver, 651
sdStart
Serial Driver, 649
sdstate_t
Serial Driver, 647
sdStop
Serial Driver, 649
SDU_READY
Serial over USB Driver, 660
SDU_STOP
Serial over USB Driver, 660
SDU_UNINIT
Serial over USB Driver, 660
sduConfigureHookI
Serial over USB Driver, 664
sduControl
Serial over USB Driver, 668
sduDataReceived
Serial over USB Driver, 667
sduDataTransmitted
Serial over USB Driver, 666
sduInit
Serial over USB Driver, 661
sduInterruptTransmitted
Serial over USB Driver, 667
sduObjectInit
Serial over USB Driver, 661
sduRequestsHook
Serial over USB Driver, 665
sduSOFHookI
Serial over USB Driver, 665
sduStart
Serial over USB Driver, 662
sdustate_t
Serial over USB Driver, 660
sduStop
Serial over USB Driver, 663
sduSuspendHookI
Serial over USB Driver, 663
sduWakeupHookI
Serial over USB Driver, 664
sdWrite
Serial Driver, 644
sdWriteI
Serial Driver, 643
sdWriteTimeout
Serial Driver, 644
sectors
flash_descriptor_t, 998
sectors_count
flash_descriptor_t, 998
sectors_size
flash_descriptor_t, 998
Serial Driver, 638
_serial_driver_data, 646
_serial_driver_methods, 641
default_config, 656
PLATFORM_SERIAL_USE_USART1, 646
SD1, 656
sd_lld_init, 654
sd_lld_start, 655
ds_lld_stop, 655
SD_READY, 648
SD_STOP, 648
SD_UNINIT, 648
sdAsynchronousRead, 646
sdAsynchronousWrite, 644
dsControl, 654
sdGet, 643
sdGetI, 642
sdGetTimeout, 643
sdGetWouldBlock, 653
sdIncomingDataI, 650
sdInit, 648
sdObjectInit, 648
sdPut, 642
sdPutI, 641
sdPutTimeout, 642
sdPutWouldBlock, 652
sdRead, 645
sdReadI, 645
sdReadTimeout, 645
sdRequestDataI, 651
sdStart, 649
sdstate_t, 649
sdStop, 649
sdWrite, 644
sdWriteI, 643
sdWriteTimeout, 644
SERIAL_BUFFERS_SIZE, 641
SERIAL_DEFAULT_BITRATE, 641
SerialConfig, 647
SerialDriver, 647
Serial NOR Flash Driver, 468
_snor_flash_methods, 471
_snor_flash_methods_alone, 471
bus_acquire, 472
bus_cmd, 473
bus_cmd_addr, 474
bus_cmd_addr_dummy_receive, 476
bus_cmd_addr_receive, 475
bus_cmd_addr_send, 474
bus_cmd_dummy_receive, 475
bus_cmd_receive, 474
bus_cmd_send, 473
bus_release, 472
bus_stop, 472
SNOR_BUFFER_SIZE, 470
SNOR_BUS_DRIVER, 470
snor_get_descriptor, 471
SNOR_SHARED_BUS, 470
SNOR_SPI_4BYTES_ADDRESS, 471
SNOR_USE_MUTUAL_ExCLUSION, 470
snor_vmt, 479
snorMemoryMap, 478
snorMemoryUnmap, 478
snorObjectInit, 476
snorStart, 477
snorStop, 477
Serial over USB Driver, 657
_serial_usb_driver_data, 659
_serial_usb_driver_methods, 659
ibnotify, 660
obnotify, 661
SDU_READY, 660
SDU_STOP, 660
SDU_UNINIT, 660
sduConfigureHookI, 664
sduControl, 668
sduDataReceived, 667
sduDataTransmitted, 666
sduInit, 661
sduInterruptTransmitted, 667
sduObjectInit, 661
sduRequestsHook, 665
sduSOFHookI, 665
sduStart, 662
sdustate_t, 660
sduStop, 663
sduSuspendHookI, 663
sduWakeupHookI, 664
SERIAL_USB_BUFFERS_NUMBER, 659
SERIAL_USB_BUFFERS_SIZE, 659
SerialUSBDriver, 660
SERIAL_BUFFERS_SIZE
Configuration, 909
Serial Driver, 641
SERIAL_DEFAULT_BITRATE
Configuration, 908
Serial Driver, 641
SERIAL_USB_BUFFERS_NUMBER
Configuration, 909
Serial over USB Driver, 659
SERIAL_USB_BUFFERS_SIZE
Configuration, 909
Serial over USB Driver, 659
SerialConfig
Serial Driver, 647
SerialDriver
Serial Driver, 647
SerialDriverVMT, 1111
SerialUSBConfig, 1113
bulk_in, 1114
bulk_out, 1114
int_in, 1114
usp,b, 1114
SerialUSBDriver, 1115
Serial over USB Driver, 660
vmt, 1116
SerialUSBDriverVMT, 1117
set_address
USB Driver, 835
setup
USBDriver, 1135
setup_cb
USBEndpointConfig, 1138
SHA1Context, 1118
SHA256Context, 1119
SHA512Context, 1119
SID
CANRxFrame, 985
CANTxFrame, 988
SIO Driver, 669
__sio_callback, 683
__sio_reloc_field, 686
__sio_wakeup_errors, 684
__sio_wakeup_rx, 684
__sio_wakeup_rxidle, 685
INDEX

SIO_STOP
  SIO Driver, 691
SIO_UNINIT
  SIO Driver, 691
SIO_USE_STREAMS_INTERFACE
  SIO Driver, 674
SIO_USE_SYNCHRONIZATION
  Configuration, 909
  SIO Driver, 674
sioAsyncRead
  SIO Driver, 698
sioAsyncReadX
  SIO Driver, 682
sioAsyncWrite
  SIO Driver, 699
sioAsyncWriteX
  SIO Driver, 682
siocb_t
  SIO Driver, 690
sioClearEnableFlags
  SIO Driver, 695
sioClearEnableFlagsX
  SIO Driver, 679
SIOConfig
  SIO Driver, 690
sioControlX
  SIO Driver, 683
SIOD1
  SIO Driver, 710
SIODriver
  SIO Driver, 690
sioevents_t
  SIO Driver, 690
sioGetAndClearErrors
  SIO Driver, 696
sioGetAndClearErrorsX
  SIO Driver, 679
sioGetAndClearEvents
  SIO Driver, 697
sioGetAndClearEventsX
  SIO Driver, 680
sioGetEnableFlagsX
  SIO Driver, 679
sioGetEvents
  SIO Driver, 697
sioGetEventsX
  SIO Driver, 680
sioGetX
  SIO Driver, 681
sioHasRXErrorsX
  SIO Driver, 676
sioinit
  SIO Driver, 691
siotRXEmptyX
  SIO Driver, 675
siotRXIdleX
  SIO Driver, 675
siotTXFullX
  SIO Driver, 675
sioStop
  SIO Driver, 693
sioStopX
  SIO Driver, 681
sioSynchronizeRX
  SIO Driver, 700
sioSynchronizeRXIdle
  SIO Driver, 701
sioSynchronizeTX
  SIO Driver, 702
sioSynchronizeTXEnd
  SIO Driver, 703
sioWriteEnableFlags
  SIO Driver, 694
sioWriteEnableFlagsX
  SIO Driver, 678
size
  flash_descriptor_t, 999
  flash_sector_descriptor_t, 1000
  hal_i2s_config, 1024
  hal_mac_receive_descriptor, 1030
  hal_mac_transmit_descriptor, 1031
  mfs_data_header_t, 1071
  mfs_transaction_op_t, 1073
slave
  hal_spi_config, 1041
sleep_event
  hal_can_driver, 1013
SNOR_BUFFER_SIZE
  Serial NOR Flash Driver, 470
SNOR_BUS_DRIVER
  Serial NOR Flash Driver, 470
snor_get_descriptor
  Serial NOR Flash Driver, 471
SNOR_SHARED_BUS
  Serial NOR Flash Driver, 470
SNOR_SPI_4BYTES_ADDRESS
  Serial NOR Flash Driver, 471
SNOR_USE_MUTUAL_EXCLUSION
  Serial NOR Flash Driver, 470
snor_vmt
  Serial NOR Flash Driver, 479
SNORConfig, 1121
SNORDriver, 1122
config, 1124
mutex, 1124
cache, 1124
vmt, 1123
SNORDriverVMT, 1124
snorMemoryMap
Serial NOR Flash Driver, 478
snorMemoryUnmap
Serial NOR Flash Driver, 478
snorObjectInit
Serial NOR Flash Driver, 476
snorStart
Serial NOR Flash Driver, 477
snorStop
Serial NOR Flash Driver, 477
sof_cb
USBConfig, 1130
speed
hal_serial_config, 1032
SPI Driver Selector, 711
SPI_ACTIVE
SPIv1 Driver, 725
SPIv2 Driver, 744
SPI_COMPLETE
SPIv1 Driver, 725
SPIv2 Driver, 744
spi_lld_abort
SPIv1 Driver, 730
SPIv2 Driver, 742
spi_lld_config_fields
SPIv1 Driver, 724
SPIv2 Driver, 742
spi_lld_driver_fields
SPIv1 Driver, 724
SPIv2 Driver, 742
spi_lld_exchange
SPIv1 Driver, 728
SPIv2 Driver, 746
spi_lld_ignore
SPIv1 Driver, 727
SPIv2 Driver, 746
spi_lld_init
SPIv1 Driver, 725
SPIv2 Driver, 744
spi_lld_polled_exchange
SPIv1 Driver, 730
SPIv2 Driver, 750
spi_lld_receive
SPIv1 Driver, 729
SPIv2 Driver, 749
spi_lld_select
SPIv1 Driver, 726
SPIv2 Driver, 745
spi_lld_send
SPIv1 Driver, 728
SPIv2 Driver, 747
spi_lld_start
SPIv1 Driver, 726
SPIv2 Driver, 744
SPI_lld_stop
SPIv1 Driver, 726
SPIv2 Driver, 745
SPI_lld_stop_transfer
SPIv2 Driver, 749
SPI_lld_unselect
SPIv1 Driver, 727
SPIv2 Driver, 745
SPI_READY
SPIv1 Driver, 725
SPIv2 Driver, 744
SPI_SELECT_MODE
Configuration, 910
SPIv1 Driver, 715
SPIv2 Driver, 735
SPI_STOP
SPIv1 Driver, 725
SPIv2 Driver, 744
SPI_SUPPORTEDS_CIRCULAR
SPIv1 Driver, 723
SPIv2 Driver, 742
SPI_SUPPORTEDS_SLAVE_MODE
SPIv2 Driver, 742
SPI_UNINIT
SPIv1 Driver, 725
SPIv2 Driver, 744
SPI_USE_ASSERT_ON_ERROR
Configuration, 910
SPIv2 Driver, 735
SPI_USE_MUTUAL_EXCLUSION
Configuration, 910
SPIv1 Driver, 715
SPIv2 Driver, 735
SPI_USE_SYNCHRONIZATION
SPIv2 Driver, 735
SPI_USE_WAIT
Configuration, 910
SPIv1 Driver, 715
spiAbort
SPIv2 Driver, 738
spiAbortI
SPIv2 Driver, 738
spicallback_t
SPIv1 Driver, 724
spicb_t
SPIv2 Driver, 743
SPIConfig
SPIv1 Driver, 724
SPIv2 Driver, 743
SPIID1
SPIv1 Driver, 730
SPIv2 Driver, 750
SPIDriver
SPIv1 Driver, 724
SPIv2 Driver, 743
spilIsBufferComplete
SPIv1 Driver, 715
st_lld_get_alarm, 762
st_lld_get_counter, 761
st_lld_init, 761
st_lld_is_alarm_active, 762
st_lld_set_alarm, 762
st_lld_start_alarm, 761
st_lld_stop_alarm, 762
stBind, 753
stBindAlarmN, 758
stGetAlarm, 756
stGetAlarmN, 759
stGetCounter, 753
stinit, 753
stIsAlarmActive, 756
stIsAlarmActiveN, 760
stSetAlarm, 755
stSetAlarmN, 759
stSetCallback, 757
stStartAlarm, 754
stStartAlarmN, 758
stStopAlarm, 755
stStopAlarmN, 759
st_callbacks
  ST Driver, 763
st_lld_get_alarm
  ST Driver, 762
st_lld_get_counter
  ST Driver, 761
st_lld_init
  ST Driver, 761
st_lld_is_alarm_active
  ST Driver, 762
st_lld_set_alarm
  ST Driver, 762
st_lld_start_alarm
  ST Driver, 761
st_lld_stop_alarm
  ST Driver, 762
state
  CRYDriver, 990
  GPTDDriver, 1002
  hal_adc_driver, 1007
  hal_can_driver, 1011
  hal_dac_driver, 1017
  hal_i2c_driver, 1022
  hal_i2s_driver, 1025
  hal_mac_driver, 1028
  hal_sio_driver, 1037
  hal_spi_driver, 1043
  hal_tng_driver, 1045
  hal_uart_driver, 1049
  hal_wspi_driver, 1052
  ICUDriver, 1057
  MFSDDriver, 1078
  PWMDDriver, 1100
  USBDDriver, 1133
  WDGDDriver, 1144
status
  USBDriver, 1135
  stBind
    ST Driver, 753
  stBindAlarmN
    ST Driver, 758
  stGetAlarm
    ST Driver, 756
  stGetAlarmN
    ST Driver, 759
  stGetCounter
    ST Driver, 753
  stinit
    ST Driver, 753
  stIsAlarmActive
    ST Driver, 756
  stIsAlarmActiveN
    ST Driver, 759
  stSetAlarm
    ST Driver, 760
  stSetAlarmN
    ST Driver, 759
  stSetCallback
    ST Driver, 757
  stStartAlarm
    ST Driver, 754
  stStartAlarmN
    ST Driver, 758
  stStopAlarm
    ST Driver, 755
  stStopAlarmN
    ST Driver, 759
  support_code
    Support Code, 918
  suspended
    ioBuffers_queue, 1059
  sync_rx
    hal_sio_driver, 1038
  sync_rxdle
    hal_sio_driver, 1038
  sync_transfer
    hal_spi_driver, 1043
  sync_tx
    hal_sio_driver, 1038
  sync_txend
    hal_sio_driver, 1038
  sysinterval
    sysinterval_t
  osal
    OSAL, 935
  syssts_t

ChibiOS/HAL
OSAL, 935
systime_t
   OSAL, 935

tdqueue
   hal_mac_driver, 1028
thread
   hal_adc_driver, 1008
   hal_dac_driver, 1017
   hal_spi_driver, 1043
   hal_wspi_driver, 1053
   USBInEndpointState, 1141
   USBOutEndpointState, 1142
thread_reference_t
   OSAL, 936
threadrx
   hal_uart_driver, 1050
threads
   palevent_t, 1094
   threads_queue_t, 1125
threaddx
   hal_uart_driver, 1050
TIME
   CANRxFrame, 984
time_conv_t
   OSAL, 935
top
   io_buffers_queue, 1061
tr_limit_offset
   MFSDriver, 1079
tr_next_offset
   MFSDriver, 1079
tr_nops
   MFSDriver, 1079
tr_ops
   MFSDriver, 1080
transmitting
   USBDriver, 1133
TRNG Driver, 764
   PLATFORM_TRNG_USE_TRNG1, 765
   trng_lld_config_fields
      TRNG Driver, 766
   trng_lld_driver_fields
      TRNG Driver, 765
   trng_lld_generate
      TRNG Driver, 772
   trng_lld_init
      TRNG Driver, 771
   trng_lld_start
      TRNG Driver, 771
   trng_lld_stop
      TRNG Driver, 772
   TRNG_READY
      TRNG Driver, 767
   TRNG_RUNNING
      TRNG Driver, 767
   TRNG_STOP
      TRNG Driver, 767
   TRNG_UNINIT
      TRNG Driver, 767
   TRNGConfig
      TRNG Driver, 766
   TRNGD1
      TRNG Driver, 773
   TRNGDriver
      TRNG Driver, 766
   trngGenerate
      TRNG Driver, 770
   trngInit
      TRNG Driver, 767
   trngObjectInit
      TRNG Driver, 767
   trngStart
      TRNG Driver, 768
   trngstate_t
      TRNG Driver, 766
   trngStop
      TRNG Driver, 768
   tx_buffer
      hal_i2s_config, 1024
   txbuf
      USBInEndpointState, 1141
   txcnt
      USBInEndpointState, 1140
   txempty_event
      hal_can_driver, 1012
   txend1_cb
      hal_uart_config, 1047
   txend2_cb
      hal_uart_config, 1047
   txqueue
      hal_can_driver, 1011
   txsize
      USBInEndpointState, 1140
   txstate
      hal_uart_driver, 1049
   UART Driver, 774
      _uart_rx_char_match_isr_code, 787

ChibiOS/HAL
USB_EVENT_SUSPEND, 834
USB_EVENT_UNCONFIGURED, 834
usb_lld_clear_in, 856
usb_lld_clear_out, 856
usb_lld_connect_bus, 830
usb_lld_disable_endpoints, 852
usb_lld_disconnect_bus, 830
usb_lld_get_frame_number, 829
usb_lld_get_status_in, 853
usb_lld_get_status_out, 852
usb_lld_get_transaction_size, 830
usb_lld_init, 849
usb_lld_init_endpoint, 851
usb_lld_prepare_receive, 854
usb_lld_prepare_transmit, 854
usb_lld_read_setup, 853
usb_lld_reset, 850
usb_lld_set_address, 851
usb_lld_stall_in, 856
usb_lld_stall_out, 855
usb_lld_start, 849
usb_lld_start_in, 855
usb_lld_start_out, 854
usb_lld_stop, 850
usb_lld_wakeup_host, 831
USB_MAX_ENDPOINTS, 828
USB_READY, 833
USB_SELECTED, 833
USB_SET_ADDRESS_ACK_HANDLING, 829
USB_SET_ADDRESS_MODE, 829
USB_STOP, 833
USB_SUSPENDED, 833
USB_UNINIT, 833
USB_USE_WAIT, 821
usbcallback_t, 831
usbConnectBus, 822
USB_D1, 857
usbDisableEndpointsI, 839
usbDisconnectBus, 822
USBDriver, 831
usbp0state_t, 834
usbp_t, 831
usbp0callback_t, 832
usbp0status_t, 833
usbevent_t, 834
usbeventcb_t, 832
usbgetdescriptor_t, 857
usbGetDriverStateI, 821
usbGetFrameNumberX, 822
usbGetReceiveStatusI, 823
usbGetReceiveTransactionSizeX, 824
usbGetTransmitStatusI, 823
usbInit, 836
usbInitEndpointI, 838
usbObjectInit, 836
usbReadSetup, 825
usbReceive, 841
usbreghandler_t, 832
usbSetupTransfer, 825
usbStallReceiveI, 843
usbStallTransmitI, 844
usbStart, 837
usbStartReceiveI, 840
usbStartTransmitI, 840
usbstate_t, 833
usbStop, 837
usbTransmit, 842
usbWakeupHost, 845
USB_ACTIVE
USB Driver, 833
USB_DESC_BCD
USB Driver, 817
USB_DESC_BYTE
USB Driver, 817
USB_DESC_CONFIGURATION
USB Driver, 818
USB_DESC_CONFIGURATION_SIZE
USB Driver, 818
USB_DESC_DEVICE
USB Driver, 818
USB_DESC_ENDPOINT
USB Driver, 820
USB_DESC_ENDPOINT_SIZE
USB Driver, 820
USB_DESC_INDEX
USB Driver, 817
USB_DESC_INTERFACE
USB Driver, 819
USB_DESC_INTERFACE_ASSOCIATION
USB Driver, 820
USB_DESC_INTERFACE_ASSOCIATION_SIZE
USB Driver, 819
USB_DESC_INTERFACE_SIZE
USB Driver, 819
USB_DESC_WORD
USB Driver, 817
USB_EP0_ERROR
USB Driver, 834
USB_EP0_IN_SENDING_STS
USB Driver, 834
USB_EP0_IN_TX
USB Driver, 834
USB_EP0_IN_WAITING_TX0
USB Driver, 834
USB_EP0_OUT_RX
USB Driver, 834
USB_EP0_OUT_WAITING_STS
USB Driver, 834
USB_EP0_STATUS_STAGE
USB Driver, 828
USB_EP0_STP_WAITING
USB Driver, 834
USB_EP_MODE_TYPE
USB Driver, 820
USB_EP_MODE_TYPE_BULK
INDEX

USB Driver, 821  USB Driver, 821
USB_EP_MODE_TYPE_CTRL  USB Driver, 821
USB_EP_MODE_TYPE_INTR  USB Driver, 821
USB_EP_MODE_TYPE_ISOC  USB Driver, 821
USB_EVENT_ADDRESS  USB Driver, 834
USB_EVENT_CONFIGURED  USB Driver, 834
USB_EVENT_RESET  USB Driver, 834
USB_EVENT_STALLED  USB Driver, 834
USB_EVENT_SUSPEND  USB Driver, 834
USB_EVENT_UNCONFIGURED  USB Driver, 834
USB_EVENT_WAKEUP  USB Driver, 834
usb_lld_clear_in  USB Driver, 856
usb_lld_clear_out  USB Driver, 856
usb_lld_connect_bus  USB Driver, 830
usb_lld_disable_endpoints  USB Driver, 852
usb_lld_disconnect_bus  USB Driver, 830
usb_lld_get_frame_number  USB Driver, 829
usb_lld_get_status_in  USB Driver, 853
usb_lld_get_status_out  USB Driver, 852
usb_lld_get_transaction_size  USB Driver, 830
usb_lld_init  USB Driver, 849
usb_lld_init_endpoint  USB Driver, 851
usb_lld_prepare_receive  USB Driver, 854
usb_lld_prepare_transmit  USB Driver, 854
usb_lld_read_setup  USB Driver, 853
usb_lld_reset  USB Driver, 850
usb_lld_set_address  USB Driver, 851
usb_lld_stall_in  USB Driver, 856
usb_lld_stall_out  USB Driver, 855
usb_lld_start  USB Driver, 849
usbcallback_t  USB Driver, 831
USBConfig, 1129
  event_cb, 1130
get_descriptor_cb, 1130
requests_hook_cb, 1130
sof_cb, 1130
usbConnectBus  USB Driver, 822
USBConfig, 1129
  event_cb, 1130
get_descriptor_cb, 1130
requests_hook_cb, 1130
sof_cb, 1130
usbConnectBus  USB Driver, 822
USBConfig, 1129
  event_cb, 1130
get_descriptor_cb, 1130
requests_hook_cb, 1130
sof_cb, 1130
usbConnectBus  USB Driver, 822
USBConfig, 1129
  event_cb, 1130
get_descriptor_cb, 1130
requests_hook_cb, 1130
sof_cb, 1130
usbConnectBus  USB Driver, 822
USB_D1  USB Driver, 857
USBDescriptor, 1131
ud_size, 1131
ud_string, 1131
usbDisableEndpointsI  USB Driver, 839
usbDisconnectBus  USB Driver, 822
USBDriver, 1132
  address, 1135
  config, 1133
  configuration, 1136
  ep0endcb, 1135
  ep0n, 1135
  ep0next, 1135
  ep0state, 1134
  epc, 1134
  in_params, 1134
  out_params, 1134
  receiving, 1134

ChibiOS/HAL
saved_state, 1136
setup, 1135
state, 1133
status, 1135
transmitting, 1133
USB Driver, 831
USBEndpointConfig, 1136
ep_mode, 1138
in_cb, 1138
in_maxsize, 1139
in_state, 1139
out_cb, 1139
out_maxsize, 1139
out_state, 1139
setup_cb, 1138
usbep0state_t
USB Driver, 834
usbep_t
USB Driver, 831
usbepcallback_t
USB Driver, 832
usbepstatus_t
USB Driver, 833
usbevent_t
USB Driver, 834
usbeventcb_t
USB Driver, 832
usbgetdescriptor_t
USB Driver, 857
usbGetDriverStateI
USB Driver, 821
usbGetFrameNumberX
USB Driver, 822
usbGetReceiveStatusI
USB Driver, 823
usbGetReceiveTransactionSizeX
USB Driver, 824
usbGetTransmitStatusI
USB Driver, 823
USBInEndpointState, 1140
thread, 1141
txbuf, 1141
txcnt, 1140
txsize, 1140
usbInit
USB Driver, 836
usbInitEndpointI
USB Driver, 838
usbObjectInit
USB Driver, 836
USBOutEndpointState, 1141
rxbuf, 1142
rxcnt, 1142
rxsize, 1142
thread, 1142
usbp
SerialUSBConfig, 1114
usbReadSetup
USB Driver, 825
usbReceive
USB Driver, 841
usbreqhandler_t
USB Driver, 832
usbSetupTransfer
USB Driver, 825
usbStallReceiveI
USB Driver, 843
usbStallTransmitI
USB Driver, 844
usbStart
USB Driver, 837
usbStartReceiveI
USB Driver, 840
usbStartTransmitI
USB Driver, 840
usbstate_t
USB Driver, 833
usbStop
USB Driver, 837
usbTransmit
USB Driver, 842
usbWakeupHost
USB Driver, 845
used_space
MFSDriver, 1079
vmt
BaseAsynchronousChannel, 956
BaseBlockDevice, 960
BaseBufferedStream, 963
BaseChannel, 966
BaseFlash, 970
BaseObject, 973
BasePersistentStorage, 975
BaseSequentialStream, 977
BufferedStreamAdapter, 982
FileStream, 995
hal_buffered_serial, 1009
hal_efl_driver, 1020
hal_serial_driver, 1034
hal_sio_driver, 1037
MemoryStream, 1067
mmc_spi_driver_t, 1085
MMCSDBlockDevice, 1089
NullStream, 1092
RTCDriver, 1104
SDCDriver, 1109
SerialUSBDriver, 1116
SNORDriver, 1123
waiting
io_buffers_queue, 1059
wakeup_event
hal_can_driver, 1013
WDG Driver, 861
PLATFORM_WDG_USE_WDG1, 862
wdg_lld_init, 865
wdg_lld_reset, 866
wdg_lld_start, 866
wdg_lld_stop, 866
WGD_READY, 863
WGD_STOP, 863
WGD_UNINIT, 863
WDGDriver, 862
wdgInit, 863
wdgReset, 865
wdgResetI, 862
wdgStart, 863
wdgstate_t, 863
wdgStop, 864
wdg_lld_init
  WDG Driver, 865
wdg_lld_reset
  WDG Driver, 866
wdg_lld_start
  WDG Driver, 866
wdg_lld_stop
  WDG Driver, 866
WGD_READY
  WDG Driver, 863
WGD_STOP
  WDG Driver, 863
WGD_UNINIT
  WDG Driver, 863
WDGConfig, 1143
WDGDriver, 1143
config, 1144
state, 1144
WDG Driver, 862
wdgInit
  WDG Driver, 863
wdgReset
  WDG Driver, 865
wdgResetI
  WDG Driver, 862
wdgStart
  WDG Driver, 863
wdgstate_t
  WDG Driver, 863
wdgStop
  WDG Driver, 864
width cb
ICUConfig, 1056
WSPI Driver, 868
  _wspi_error_code, 876
  _wspi_isr_code, 876
  _wspi_wakeup_isr, 875
PLATFORM_WSPI_USE_WSPI1, 877
WSPI_COMPLETE, 879
wspi_lld_command, 892
wspi_lld_config_fields, 877
wspi_lld_driver_fields, 877
wspi_lld_init, 891
wspi_lld_map_flash, 893
wspi_lld_receive, 893
wspi_lld_send, 892
wspi_lld_start, 891
wspi_lld_stop, 891
wspi_lld_unmap_flash, 894
WSPI_MEMMAP, 879
WSPI_READY, 879
WSPI_RECEIVE, 879
WSPI_SEND, 879
WSPI_STOP, 879
WSPI_UNINIT, 879
WSPI_USE_MUTUAL_EXCLUSION, 872
WSPI_USE_WAIT, 871
wspiAcquireBus, 889
wspiCallback_t, 878
wspiCommand, 886
WSPIConfig, 878
WSPID1, 894
WSPIDriver, 878
wspilnit, 879
wspiMapFlash, 888
wspiMapFlashI, 874
wspiObjectInit, 879
wspiReceive, 887
wspiReleaseBus, 890
wspiSend, 887
wspiStart, 881
wspiStartCommand, 883
wspiStartCommandI, 872
wspiStartReceive, 885
wspiStartReceiveI, 873
wspiStartSend, 894
wspiStartSendI, 873
wspistate_t, 878
wspiStop, 882
wspiUnmapFlash, 889
wspiUnmapFlashI, 875
wspi_command_t, 1144
  addr, 1145
  alt, 1145
  cfg, 1145
  cmd, 1145
  dummy, 1145
WSPI_COMPLETE
  WSPI Driver, 879
wspi_lld_command
  WSPI Driver, 892
wspi_lld_config_fields
  WSPI Driver, 877
wspi_lld_driver_fields
  WSPI Driver, 877
wspi_lld_init
  WSPI Driver, 891
wspi_lld_map_flash
  WSPI Driver, 893
wspi_lld_receive
  WSPI Driver, 893
wspi_lld_send
  WSPI Driver, 892
wspi_lld_start
  WSPI Driver, 891
wspi_lld_stop
  WSPI Driver, 891
wspi_lld_unmap_flash
  WSPI Driver, 894
WSPI_MEMMAP
  WSPI Driver, 879
WSPI_READY
  WSPI Driver, 879
WSPI_RECEIVE
  WSPI Driver, 879
WSPI_SEND
  WSPI Driver, 879
WSPI_STOP
  WSPI Driver, 879
WSPI_UNINIT
  WSPI Driver, 879
WSPI_USE_MUTUAL_EXCLUSION
  Configuration, 912
  WSPI Driver, 872
WSPI_USE_WAIT
  Configuration, 911
  WSPI Driver, 871
wspiAcquireBus
  WSPI Driver, 889
wspicallback_t
  WSPI Driver, 878
wspiCommand
  WSPI Driver, 886
WSPIConfig
  WSPI Driver, 878
WSPID1
  WSPI Driver, 894
WSPIDriver
  WSPI Driver, 878
wspiInit
  WSPI Driver, 879
wspiMapFlash
  WSPI Driver, 888
wspiMapFlashI
  WSPI Driver, 874
wspiObjectInit
  WSPI Driver, 879
wspiReceive
  WSPI Driver, 887
wspiReleaseBus
  WSPI Driver, 890
wspiSend
  WSPI Driver, 887
wspiStart
  WSPI Driver, 881
wspiStartCommand
  WSPI Driver, 883
wspiStartCommandI
  WSPI Driver, 872
wspiStartReceive
  WSPI Driver, 885
wspiStartReceiveI
  WSPI Driver, 873
wspiStartSend
  WSPI Driver, 884
wspiStartSendI
  WSPI Driver, 873
wspistate_t
  WSPI Driver, 878
wspiStop
  WSPI Driver, 882
wspiUnmapFlash
  WSPI Driver, 889
wspiUnmapFlashI
  WSPI Driver, 875
year
  RTCDateTime, 1102