# 1 ChibiOS/HAL

## 1.1 Copyright

## 1.2 Introduction

## 1.3 Related Documents

# 2 Deprecated List

# 3 Module Index

## 3.1 Modules

# 4 Hierarchical Index

## 4.1 Class Hierarchy

# 5 Data Structure Index

## 5.1 Data Structures

# 6 File Index

## 6.1 File List

# 7 Module Documentation

## 7.1 HAL Driver

### 7.1.1 Detailed Description

### 7.1.2 Macro Definition Documentation

#### 7.1.2.1 __CHIBIOS_HAL__

#### 7.1.2.2 CH_HAL_STABLE

#### 7.1.2.3 HAL_VERSION

#### 7.1.2.4 CH_HAL_MAJOR

#### 7.1.2.5 CH_HAL_MINOR

#### 7.1.2.6 CH_HAL_PATCH

#### 7.1.2.7 HAL_SUCCESS

#### 7.1.2.8 HAL_FAILED

#### 7.1.2.9 HAL_LLD_USE_CLOCK_MANAGEMENT

### 7.1.3 Typedef Documentation

#### 7.1.3.1 halclkpt_t

#### 7.1.3.2 halfreq_t

### 7.1.4 Function Documentation

#### 7.1.4.1 halInit()

#### 7.1.4.2 halClockSwitchMode()

#### 7.1.4.3 halClockGetPointX()

#### 7.1.4.4 hal_lld_init()

#### 7.1.4.5 hal_lld_clock_switch_mode()

#### 7.1.4.6 hal_lld_get_clock_point()

## 7.2 ADC Driver

### 7.2.1 Detailed Description
7.2.7.8 adcStopConversionI()  ................................................................. 50
7.2.7.9 adcConvert() ........................................................................... 50
7.2.7.10 adcAcquireBus() ................................................................. 51
7.2.7.11 adcReleaseBus() ................................................................. 52
7.2.7.12 adc_lld_init() ................................................................. 53
7.2.7.13 adc_lld_start() ................................................................. 53
7.2.7.14 adc_lld_stop() ................................................................. 53
7.2.7.15 adc_lld_start_conversion() .................................................. 54
7.2.7.16 adc_lld_stop_conversion() ................................................... 54
7.2.8 Variable Documentation ............................................................... 54
7.2.8.1 ADCD1 .................................................................................. 55

7.3 I/O Buffers Queues ........................................................................ 56
7.3.1 Detailed Description ................................................................ 56
7.3.2 Macro Definition Documentation .............................................. 58
7.3.2.1 BUFFERS_CHUNKS_SIZE ...................................................... 58
7.3.2.2 BQ_BUFFER_SIZE ................................................................ 58
7.3.2.3 bqSizeX ............................................................................... 59
7.3.2.4 bqSpaceI ............................................................................. 59
7.3.2.5 bqGetLinkX ........................................................................ 59
7.3.2.6 bqSetLinkX ........................................................................ 60
7.3.2.7 bqIsSuspendedX .................................................................. 60
7.3.2.8 bqSuspendI ......................................................................... 61
7.3.2.9 bqResumeX ......................................................................... 61
7.3.2.10 ibqIsEmptyI ....................................................................... 62
7.3.2.11 ibqIsFullI .......................................................................... 62
7.3.2.12 obqIsEmptyI ....................................................................... 63
7.3.2.13 obqIsFullI .......................................................................... 64

7.3.3 Typedef Documentation ............................................................. 64
7.3.3.1 io_buffers_queue_t ................................................................ 64
7.3.3.2 bqnotify_t ........................................................................... 64
7.3.3.3 input_buffers_queue_t .......................................................... 65
7.3.3.4 output_buffers_queue_t ......................................................... 65

7.3.4 Function Documentation ............................................................ 65
7.3.4.1 ibqObjectInit() ................................................................... 65
7.3.4.2 ibqResetI() ........................................................................ 66
7.3.4.3 ibqGetEmptyBufferI ............................................................ 67
7.3.4.4 ibqPostFullBufferI .............................................................. 67
7.3.4.5 ibqGetFullBufferTimeout() .................................................. 68
7.3.4.6 ibqGetFullBufferTimeoutS() .................................................. 69
7.3.4.7 ibqReleaseEmptyBufferI ....................................................... 70
7.3.4.8 ibqReleaseEmptyBufferS() .................................................... 71
7.3.4.9 ibqGetTimeout() ................................................................. 71
7.3.4.10 ibqReadTimeout() ................................................. 72
7.3.4.11 obqObjectInit() .............................................. 73
7.3.4.12 obqResetI() .................................................. 74
7.3.4.13 obqGetFullBufferI() ....................................... 75
7.3.4.14 obqReleaseEmptyBufferI() ................................. 75
7.3.4.15 obqGetEmptyBufferTimeout() .............................. 76
7.3.4.16 obqGetEmptyBufferTimeoutS() ............................ 77
7.3.4.17 obqPostFullBuffer() ....................................... 78
7.3.4.18 obqPostFullBufferS() ..................................... 79
7.3.4.19 obqPutTimeout() ............................................ 79
7.3.4.20 obqWriteTimeout() ......................................... 80
7.3.4.21 obqTryFlushI() ............................................ 81
7.3.4.22 obqFlush() .................................................. 82
7.4 CAN Driver .......................................................... 83
  7.4.1 Detailed Description ........................................ 83
  7.4.2 Driver State Machine ....................................... 83
  7.4.3 Macro Definition Documentation ......................... 86
    7.4.3.1 CAN_LIMIT_WARNING .................................... 86
    7.4.3.2 CAN_LIMIT_ERROR ...................................... 86
    7.4.3.3 CAN_BUS_OFF_ERROR .................................... 87
    7.4.3.4 CAN_FRAMING_ERROR ................................... 87
    7.4.3.5 CAN_OVERFLOW_ERROR .................................. 87
    7.4.3.6 CAN_ANY_MAILBOX ...................................... 87
    7.4.3.7 CAN_USE_SLEEP_MODE .................................. 87
    7.4.3.8 CAN_ENFORCE_USE_CALLBACKS ........................... 87
    7.4.3.9 CAN_MAILBOX_TO_MASK ................................ 88
    7.4.3.10 canTransmit ........................................... 88
    7.4.3.11 canReceive ............................................ 88
    7.4.3.12 _can_tx_empty_isr .................................. 88
    7.4.3.13 _can_rx_full_isr .................................... 89
    7.4.3.14 _can_wakeup_isr ..................................... 89
    7.4.3.15 _can_error_isr ....................................... 89
    7.4.3.16 CAN_TX_MAILBOXES .................................. 89
    7.4.3.17 CAN_RX_MAILBOXES .................................. 90
    7.4.3.18 PLATFORM_CAN_USE_CAN1 .............................. 90
  7.4.4 Typedef Documentation .................................... 90
    7.4.4.1 CANDriver ................................................. 90
    7.4.4.2 canmbx_t ................................................ 90
    7.4.4.3 can_callback_t ......................................... 90
  7.4.5 Enumeration Type Documentation ......................... 91
    7.4.5.1 canstate_t ............................................. 91
  7.4.6 Function Documentation ................................... 91
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryStop()</td>
<td>117</td>
</tr>
<tr>
<td>cryLoadAESTransientKey()</td>
<td>117</td>
</tr>
<tr>
<td>cryEncryptAES()</td>
<td>119</td>
</tr>
<tr>
<td>cryDecryptAES()</td>
<td>120</td>
</tr>
<tr>
<td>cryEncryptAES_ECB()</td>
<td>121</td>
</tr>
<tr>
<td>cryDecryptAES_ECB()</td>
<td>122</td>
</tr>
<tr>
<td>cryEncryptAES_CBC()</td>
<td>123</td>
</tr>
<tr>
<td>cryDecryptAES_CBC()</td>
<td>124</td>
</tr>
<tr>
<td>cryEncryptAES_CFB()</td>
<td>125</td>
</tr>
<tr>
<td>cryDecryptAES_CFB()</td>
<td>126</td>
</tr>
<tr>
<td>cryEncryptAES_CTR()</td>
<td>127</td>
</tr>
<tr>
<td>cryDecryptAES_CTR()</td>
<td>128</td>
</tr>
<tr>
<td>cryEncryptAES_GCM()</td>
<td>129</td>
</tr>
<tr>
<td>cryDecryptAES_GCM()</td>
<td>130</td>
</tr>
<tr>
<td>cryLoadDESTransientKey()</td>
<td>132</td>
</tr>
<tr>
<td>cryEncryptDES()</td>
<td>133</td>
</tr>
<tr>
<td>cryDecryptDES()</td>
<td>134</td>
</tr>
<tr>
<td>cryEncryptDES_ECB()</td>
<td>135</td>
</tr>
<tr>
<td>cryDecryptDES_ECB()</td>
<td>136</td>
</tr>
<tr>
<td>cryEncryptDES_CBC()</td>
<td>137</td>
</tr>
<tr>
<td>cryDecryptDES_CBC()</td>
<td>138</td>
</tr>
<tr>
<td>crySHA1Init()</td>
<td>139</td>
</tr>
<tr>
<td>crySHA1Update()</td>
<td>140</td>
</tr>
<tr>
<td>crySHA1Final()</td>
<td>141</td>
</tr>
<tr>
<td>crySHA256Init()</td>
<td>142</td>
</tr>
<tr>
<td>crySHA256Update()</td>
<td>143</td>
</tr>
<tr>
<td>crySHA256Final()</td>
<td>144</td>
</tr>
<tr>
<td>crySHA512Init()</td>
<td>145</td>
</tr>
<tr>
<td>crySHA512Update()</td>
<td>146</td>
</tr>
<tr>
<td>crySHA512Final()</td>
<td>147</td>
</tr>
<tr>
<td>cryLoadHMACTransientKey()</td>
<td>148</td>
</tr>
<tr>
<td>cryHMACHSHA256Init()</td>
<td>149</td>
</tr>
<tr>
<td>cryHMACHSHA256Update()</td>
<td>150</td>
</tr>
<tr>
<td>cryHMACHSHA256Final()</td>
<td>151</td>
</tr>
<tr>
<td>cryHMACHSHA512Init()</td>
<td>151</td>
</tr>
<tr>
<td>cryHMACHSHA512Update()</td>
<td>152</td>
</tr>
<tr>
<td>cryHMACHSHA512Final()</td>
<td>153</td>
</tr>
<tr>
<td>cry_lld_init()</td>
<td>154</td>
</tr>
<tr>
<td>cry_lld_start()</td>
<td>154</td>
</tr>
<tr>
<td>cry_lld_stop()</td>
<td>155</td>
</tr>
<tr>
<td>cry_lld_aes_loadkey()</td>
<td>155</td>
</tr>
<tr>
<td>cry_lld_encrypt_AES()</td>
<td>156</td>
</tr>
</tbody>
</table>
7.6.2.4 _dac_wait_s ......................................................... 187
7.6.2.5 _dac_reset_i ....................................................... 188
7.6.2.6 _dac_reset_s ....................................................... 188
7.6.2.7 _dac_wakeup_isr .................................................. 189
7.6.2.8 _dac_timeout_isr .................................................. 189
7.6.2.9 _dac_isr_half_code ............................................... 190
7.6.2.10 _dac_isr_full_code .............................................. 190
7.6.2.11 _dac_isr_error_code ............................................. 191
7.6.2.12 DAC_MAX_CHANNELS .............................................. 192
7.6.2.13 PLATFORM_DAC_USE_DAC1 ................................. 192
7.6.2.14 dac_lld_driver_fields .......................................... 192
7.6.2.15 dac_lld_config_fields .......................................... 192
7.6.2.16 dac_lld_conversion_group_fields ............................ 192

7.6.3 Typedef Documentation ........................................... 193
7.6.3.1 DACDriver .......................................................... 193
7.6.3.2 DACConfig .......................................................... 193
7.6.3.3 DACConversionGroup ............................................ 193
7.6.3.4 daccallback_t ...................................................... 193
7.6.3.5 dacerrorcallback_t ............................................... 193
7.6.3.6 dacchannel_t ....................................................... 194
7.6.3.7 dacsample_t ........................................................ 194

7.6.4 Enumeration Type Documentation ................................. 194
7.6.4.1 dacstate_t .......................................................... 194
7.6.4.2 dacerror_t .......................................................... 195

7.6.5 Function Documentation ........................................... 195
7.6.5.1 dacInit() ............................................................ 195
7.6.5.2 dacObjectInit() .................................................... 195
7.6.5.3 dacStart() .......................................................... 196
7.6.5.4 dacStop() ........................................................... 197
7.6.5.5 dacPutChannelX() ................................................ 198
7.6.5.6 dacStartConversion() ............................................. 199
7.6.5.7 dacStartConversion() ............................................. 199
7.6.5.8 dacStopConversion() .............................................. 200
7.6.5.9 dacStopConversion() .............................................. 201
7.6.5.10 dacConvert() ...................................................... 202
7.6.5.11 dacAcquireBus() ................................................ 203
7.6.5.12 dacReleaseBus() ................................................ 204
7.6.5.13 dac_lld_init() .................................................... 205
7.6.5.14 dac_lld_start() ................................................... 205
7.6.5.15 dac_lld_stop() ................................................... 205
7.6.5.16 dac_lld_put_channel() ......................................... 206
7.6.5.17 dac_lld_start_conversion() ................................. 206
7.8.3.7 _base_flash_methods Alone .................................................. 223
7.8.3.8 _base_flash_methods ............................................................ 224
7.8.3.9 _base_flash_data ................................................................. 224
7.8.3.10 getBaseFlash ................................................................. 224
7.8.3.11 flashGetDescriptor ........................................................... 224
7.8.3.12 flashRead ................................................................. 225
7.8.3.13 flashProgram .............................................................. 225
7.8.3.14 flashStartEraseAll ......................................................... 226
7.8.3.15 flashStartEraseSector ...................................................... 227
7.8.3.16 flashQueryErase ............................................................. 227
7.8.3.17 flashVerifyErase ............................................................ 228
7.8.4 Typedef Documentation ............................................................. 228
7.8.4.1 flash_offset_t ................................................................. 229
7.8.4.2 flash_sector_t ................................................................. 229
7.8.5 Enumeration Type Documentation .............................................. 229
7.8.5.1 flash_state_t ................................................................. 229
7.8.5.2 flash_error_t ................................................................. 229
7.8.6 Function Documentation .......................................................... 229
7.8.6.1 flashWaitErase() ............................................................ 229
7.8.6.2 flashGetSectorOffset() ..................................................... 230
7.8.6.3 flashGetSectorSize() ........................................................ 230
7.9 GPT Driver ........................................................................ 231
7.9.1 Detailed Description ............................................................... 231
7.9.2 Driver State Machine ............................................................. 231
7.9.3 GPT Operations. .................................................................. 232
7.9.4 Macro Definition Documentation ............................................... 234
7.9.4.1 gptChangeInterval() .......................................................... 234
7.9.4.2 gptGetIntervalX ............................................................... 234
7.9.4.3 gptGetCounterX ............................................................... 235
7.9.4.4 _gpt_isr_invoke_cb ........................................................... 236
7.9.4.5 PLATFORM_GPT_USE_GPT1 .................................................. 236
7.9.4.6 gpt_lld_change_interval ....................................................... 236
7.9.5 Typedef Documentation ............................................................. 237
7.9.5.1 GPTDriver ................................................................. 237
7.9.5.2 gptcallback_t ................................................................. 237
7.9.5.3 gptfreq_t ................................................................. 238
7.9.5.4 gptcnt_t ................................................................. 238
7.9.6 Enumeration Type Documentation ............................................... 238
7.9.6.1 gptstate_t ................................................................. 238
7.9.7 Function Documentation .......................................................... 238
7.9.7.1 gptInit() ................................................................. 238
7.9.7.2 gptObjectInit() ............................................................. 239
7.10.5.1 i2cstate_t .................................................. 257
7.10.6 Function Documentation ........................................ 258
  7.10.6.1 i2cInit() ................................................. 258
  7.10.6.2 i2cObjectInit() ......................................... 258
  7.10.6.3 i2cStart() ............................................... 259
  7.10.6.4 i2cStop() ............................................... 260
  7.10.6.5 i2cGetErrors() ........................................... 260
  7.10.6.6 i2cMasterTransmitTimeout() ................................ 261
  7.10.6.7 i2cMasterReceiveTimeout() ................................ 262
  7.10.6.8 i2cAcquireBus() ......................................... 262
  7.10.6.9 i2cReleaseBus() ......................................... 263
  7.10.6.10 i2c_lld_init() .......................................... 264
  7.10.6.11 i2c_lld_start() .......................................... 264
  7.10.6.12 i2c_lld_stop() .......................................... 264
  7.10.6.13 i2c_lld_master_receive_timeout() ......................... 265
  7.10.6.14 i2c_lld_master_transmit_timeout() ....................... 266
7.10.7 Variable Documentation ....................................... 266
  7.10.7.1 I2CD1 .................................................. 266
7.11 I2S Driver .......................................................... 267
  7.11.1 Detailed Description ......................................... 267
  7.11.2 Driver State Machine ....................................... 267
  7.11.3 Macro Definition Documentation ............................. 269
    7.11.3.1 i2sIsBufferComplete .................................. 269
    7.11.3.2 i2sStartExchange ....................................... 269
    7.11.3.3 i2sStopExchange ....................................... 270
    7.11.3.4 _i2s_isr_half_code .................................... 270
    7.11.3.5 _i2s_isr_full_code .................................... 271
    7.11.3.6 PLATFORM_I2S_USE_I2S1 ................................ 271
    7.11.3.7 i2s_lld_driver_fields ................................... 272
    7.11.3.8 i2s_lld_config_fields .................................. 272
  7.11.4 Typedef Documentation ...................................... 272
    7.11.4.1 I2SDriver ............................................. 272
    7.11.4.2 I2SConfig ............................................. 272
    7.11.4.3 i2scallback_t ......................................... 272
  7.11.5 Enumeration Type Documentation ............................. 273
    7.11.5.1 i2sstate_t ............................................. 273
  7.11.6 Function Documentation ...................................... 273
    7.11.6.1 i2sInit() ............................................. 273
    7.11.6.2 i2sObjectInit() ....................................... 274
    7.11.6.3 i2sStart() ............................................. 274
    7.11.6.4 i2sStop() ............................................. 275
    7.11.6.5 i2sStartExchange() .................................... 276
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.11.6.6</td>
<td>i2sStopExchange()</td>
</tr>
<tr>
<td>7.11.6.7</td>
<td>i2s_lld_init()</td>
</tr>
<tr>
<td>7.11.6.8</td>
<td>i2s_lld_start()</td>
</tr>
<tr>
<td>7.11.6.9</td>
<td>i2s_lld_stop()</td>
</tr>
<tr>
<td>7.11.6.10</td>
<td>i2s_lld_start_exchange()</td>
</tr>
<tr>
<td>7.11.6.11</td>
<td>i2s_lld_stop_exchange()</td>
</tr>
<tr>
<td>7.11.7</td>
<td>Variable Documentation</td>
</tr>
<tr>
<td>7.11.7.1</td>
<td>I2SD1</td>
</tr>
<tr>
<td>7.12</td>
<td>ICU Driver</td>
</tr>
<tr>
<td>7.12.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>7.12.2</td>
<td>Driver State Machine</td>
</tr>
<tr>
<td>7.12.3</td>
<td>ICU Operations</td>
</tr>
<tr>
<td>7.12.4</td>
<td>Macro Definition Documentation</td>
</tr>
<tr>
<td>7.12.4.1</td>
<td>icuStartCaptureI</td>
</tr>
<tr>
<td>7.12.4.2</td>
<td>icuStopCaptureI</td>
</tr>
<tr>
<td>7.12.4.3</td>
<td>icuEnableNotificationsI</td>
</tr>
<tr>
<td>7.12.4.4</td>
<td>icuDisableNotificationsI</td>
</tr>
<tr>
<td>7.12.4.5</td>
<td>icuAreNotificationsEnabledX</td>
</tr>
<tr>
<td>7.12.4.6</td>
<td>icuGetWidthX</td>
</tr>
<tr>
<td>7.12.4.7</td>
<td>icuGetPeriodX</td>
</tr>
<tr>
<td>7.12.4.8</td>
<td>_icu_isr_invoke_width_cb</td>
</tr>
<tr>
<td>7.12.4.9</td>
<td>_icu_isr_invoke_period_cb</td>
</tr>
<tr>
<td>7.12.4.10</td>
<td>_icu_isr_invoke_overflow_cb</td>
</tr>
<tr>
<td>7.12.4.11</td>
<td>PLATFORM_ICU_USE_ICU1</td>
</tr>
<tr>
<td>7.12.4.12</td>
<td>icu_lld_get_width</td>
</tr>
<tr>
<td>7.12.4.13</td>
<td>icu_lld_get_period</td>
</tr>
<tr>
<td>7.12.4.14</td>
<td>icu_lld_are_notifications_enabled</td>
</tr>
<tr>
<td>7.12.5</td>
<td>Typedef Documentation</td>
</tr>
<tr>
<td>7.12.5.1</td>
<td>ICUDriver</td>
</tr>
<tr>
<td>7.12.5.2</td>
<td>icucallback_t</td>
</tr>
<tr>
<td>7.12.5.3</td>
<td>icufreq_t</td>
</tr>
<tr>
<td>7.12.5.4</td>
<td>icucnt_t</td>
</tr>
<tr>
<td>7.12.6</td>
<td>Enumeration Type Documentation</td>
</tr>
<tr>
<td>7.12.6.1</td>
<td>icustate_t</td>
</tr>
<tr>
<td>7.12.6.2</td>
<td>icumode_t</td>
</tr>
<tr>
<td>7.12.7</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>7.12.7.1</td>
<td>icuInit()</td>
</tr>
<tr>
<td>7.12.7.2</td>
<td>icuObjectInit()</td>
</tr>
<tr>
<td>7.12.7.3</td>
<td>icuStart()</td>
</tr>
<tr>
<td>7.12.7.4</td>
<td>icuStop()</td>
</tr>
<tr>
<td>7.12.7.5</td>
<td>icuStartCapture()</td>
</tr>
<tr>
<td>7.12.7.6</td>
<td>icuWaitCapture()</td>
</tr>
</tbody>
</table>
7.16.2.11 chnReadTimeout .................................................. 318
7.16.2.12 chnControl ..................................................... 318
7.16.2.13 CHN_NO_ERROR .................................................. 319
7.16.2.14 CHN_CONNECTED ................................................. 319
7.16.2.15 CHN_DISCONNECTED ............................................. 319
7.16.2.16 CHN_INPUT_AVAILABLE ......................................... 319
7.16.2.17 CHN_OUTPUT_EMPTY ............................................. 320
7.16.2.18 CHN_TRANSMISSION_END ....................................... 320
7.16.2.19 _base_asynchronous_channel_methods .......................... 320
7.16.2.20 _base_asynchronous_channel_data ............................. 320
7.16.2.21 chnGetEventSource ............................................ 320
7.16.2.22 chnAddFlagsI ................................................... 321
7.17 Abstract Files Interface ............................................. 322
  7.17.1 Detailed Description ............................................ 322
  7.17.2 Macro Definition Documentation ................................. 323
     7.17.2.1 FILE_OK ..................................................... 323
     7.17.2.2 FILE_ERROR ................................................ 323
     7.17.2.3 FILE_EOF .................................................... 323
     7.17.2.4 _file_stream_methods ....................................... 324
     7.17.2.5 _file_stream_data .......................................... 324
     7.17.2.6 fileStreamWrite ............................................ 324
     7.17.2.7 fileStreamRead ............................................. 325
     7.17.2.8 fileStreamPut .............................................. 325
     7.17.2.9 fileStreamGet .............................................. 326
     7.17.2.10 fileStreamClose .......................................... 327
     7.17.2.11 fileStreamGetError ....................................... 327
     7.17.2.12 fileStreamGetSize ......................................... 328
     7.17.2.13 fileStreamGetPosition .................................... 328
     7.17.2.14 fileStreamSetPosition .................................... 329
  7.17.3 Typedef Documentation ......................................... 329
     7.17.3.1 fileoffset_t ................................................ 329
7.18 Abstract Persistent Storage Interface ............................. 330
  7.18.1 Detailed Description ............................................ 330
  7.18.2 Macro Definition Documentation ................................. 331
     7.18.2.1 _base_pers_storage_methods_alone .......................... 331
     7.18.2.2 _base_pers_storage_methods ................................ 331
     7.18.2.3 _base_persistent_storage_data ............................. 331
     7.18.2.4 getBasePersistentStorage .................................. 331
     7.18.2.5 psGetStorageSize .......................................... 331
     7.18.2.6 psRead ..................................................... 332
     7.18.2.7 psWrite .................................................... 333
  7.18.3 Typedef Documentation ......................................... 333
7.24.2 Driver State Machine .................................................. 353
7.24.3 Macro Definition Documentation ................................. 354
  7.24.3.1 _base_block_device_methods ................................. 354
  7.24.3.2 _base_block_device_data .................................. 355
  7.24.3.3 blkGetDriverState ........................................... 355
  7.24.3.4 blkIsTransferring .......................................... 355
  7.24.3.5 blkIsInserted ............................................... 356
  7.24.3.6 blkIsWriteProtected ....................................... 357
  7.24.3.7 blkConnect ................................................. 357
  7.24.3.8 blkDisconnect .............................................. 358
  7.24.3.9 blkRead ..................................................... 358
  7.24.3.10 blkWrite .................................................. 359
  7.24.3.11 blkSync ................................................... 360
  7.24.3.12 blkGetInfo ............................................... 360
7.24.4 Enumeration Type Documentation .................................. 361
  7.24.4.1 blkstate_t .................................................. 361
7.25 MAC Driver ............................................................. 362
  7.25.1 Detailed Description ........................................... 362
  7.25.2 Macro Definition Documentation ............................... 364
    7.25.2.1 MAC_USE_ZERO_COPY ...................................... 364
    7.25.2.2 MAC_USE_EVENTS ........................................ 364
    7.25.2.3 macGetReceiveEventSource ............................... 364
    7.25.2.4 macWriteTransmitDescriptor ............................ 365
    7.25.2.5 macReadReceiveDescriptor ............................... 365
    7.25.2.6 macGetNextTransmitBuffer ............................... 366
    7.25.2.7 macGetNextReceiveBuffer ............................... 367
    7.25.2.8 MAC_SUPPORTS_ZERO_COPY ................................ 367
    7.25.2.9 PLATFORM_MAC_USE_MAC1 ................................ 367
  7.25.3 Typedef Documentation ........................................... 368
    7.25.3.1 MACDriver ................................................ 368
  7.25.4 Enumeration Type Documentation ................................ 368
    7.25.4.1 macstate_t .............................................. 368
  7.25.5 Function Documentation ........................................ 368
    7.25.5.1 macInit() ................................................ 368
    7.25.5.2 macObjectInit() ......................................... 369
    7.25.5.3 macStart() ............................................... 370
    7.25.5.4 macStop() ............................................... 370
    7.25.5.5 macWaitTransmitDescriptor() ........................... 371
    7.25.5.6 macReleaseTransmitDescriptor() ......................... 372
    7.25.5.7 macWaitReceiveDescriptor() ............................. 373
    7.25.5.8 macReleaseReceiveDescriptor() .......................... 374
    7.25.5.9 macPollLinkStatus() .................................... 374
7.26.5.6 mfs_bank_write_header() ................................. 391
7.26.5.7 mfs_bank_check_header() ............................... 392
7.26.5.8 mfs_bank_scan_records() ................................. 392
7.26.5.9 mfs_bank_get_state() .................................. 393
7.26.5.10 mfs_garbage_collect() ................................. 393
7.26.5.11 mfs_try_mount() ....................................... 394
7.26.5.12 mfs_mount() ............................................. 394
7.26.5.13 mfsObjectInit() ....................................... 395
7.26.5.14 mfsStart() ............................................. 395
7.26.5.15 mfsStop() ................................................ 396
7.26.5.16 mfsErase() .............................................. 396
7.26.5.17 mfsReadRecord() ....................................... 397
7.26.5.18 mfsWriteRecord() ...................................... 398
7.26.5.19 mfsEraseRecord() ....................................... 398
7.26.5.20 mfsPerformGarbageCollection() ....................... 400
7.26.5.21 mfsStartTransaction() ................................. 401
7.26.5.22 mfsCommitTransaction() ............................... 402
7.26.5.23 mfsRollbackTransaction() ............................. 402

7.27 MII/RMII Header .............................................. 404
  7.27.1 Detailed Description ..................................... 404
  7.27.2 Macro Definition Documentation ....................... 406
    7.27.2.1 MII_BMCR .......................................... 406
    7.27.2.2 MII_BMSR .......................................... 406
    7.27.2.3 MII_PHYSID1 ....................................... 407
    7.27.2.4 MII_PHYSID2 ....................................... 407
    7.27.2.5 MII_ADVERTISE ..................................... 407
    7.27.2.6 MII_LPA ............................................ 407
    7.27.2.7 MII_EXPANSION ..................................... 407
    7.27.2.8 MII_ANNPTR ........................................ 407
    7.27.2.9 MII_CTRL1000 ...................................... 407
    7.27.2.10 MII_STAT1000 ..................................... 408
    7.27.2.11 MII_ESTATUS ..................................... 408
    7.27.2.12 MII_PHYSTS ...................................... 408
    7.27.2.13 MII_MICR .......................................... 408
    7.27.2.14 MII_DCOUNTER .................................... 408
    7.27.2.15 MII_FSCOUNTER ................................. 408
    7.27.2.16 MII_NWAYTEST .................................... 408
    7.27.2.17 MII_RERRCOUNTER ............................... 409
    7.27.2.18 MII_SREVISION ................................... 409
    7.27.2.19 MII_RESV1 ........................................ 409
    7.27.2.20 MII_LBRERROR .................................... 409
    7.27.2.21 MII_PHYADDR .................................... 409
7.27.2.22 MII_RESV2 .......................................................... 409
7.27.2.23 MII_TPSTATUS .................................................... 409
7.27.2.24 MII_NCONFIG ...................................................... 410
7.27.2.25 BMCR_RESV ....................................................... 410
7.27.2.26 BMCR_CTST ....................................................... 410
7.27.2.27 BMCR_FULLDPLX ............................................... 410
7.27.2.28 BMCR_ANRESTART ............................................. 410
7.27.2.29 BMCR_ISOLATE .................................................. 410
7.27.2.30 BMCR_PDOWN ................................................... 410
7.27.2.31 BMCR_ANENABLE ............................................... 411
7.27.2.32 BMCR_SPEED100 ............................................... 411
7.27.2.33 BMCR_LOOPBACK .............................................. 411
7.27.2.34 BMCR_RESET .................................................... 411
7.27.2.35 BMSR_ERCAP ................................................... 411
7.27.2.36 BMSR_JCD ....................................................... 411
7.27.2.37 BMSR_LSTATUS ................................................ 411
7.27.2.38 BMSR_ANECAPABLE .......................................... 412
7.27.2.39 BMSR_RFAULT ................................................... 412
7.27.2.40 BMSR_ANEGCOMPLETE ...................................... 412
7.27.2.41 BMSR_MFPRESUPPCAP .................................... 412
7.27.2.42 BMSR_RESV ..................................................... 412
7.27.2.43 BMSR_10HALF .................................................. 412
7.27.2.44 BMSR_10FULL ................................................... 412
7.27.2.45 BMSR_100HALF ................................................ 413
7.27.2.46 BMSR_100FULL .................................................. 413
7.27.2.47 BMSR_100BASE4 .............................................. 413
7.27.2.48 ADVERTISE_SLCT ............................................ 413
7.27.2.49 ADVERTISE_CSMA ............................................. 413
7.27.2.50 ADVERTISE_10HALF ......................................... 413
7.27.2.51 ADVERTISE_10FULL ........................................ 413
7.27.2.52 ADVERTISE_100HALF ........................................ 414
7.27.2.53 ADVERTISE_100FULL ....................................... 414
7.27.2.54 ADVERTISE_100BASE4 .................................... 414
7.27.2.55 ADVERTISE_PAUSE_CAP .................................. 414
7.27.2.56 ADVERTISE_PAUSE_ASYM .................................. 414
7.27.2.57 ADVERTISE_RESV ............................................. 414
7.27.2.58 ADVERTISE_RFAULT ......................................... 414
7.27.2.59 ADVERTISE_LPACK .......................................... 415
7.27.2.60 ADVERTISE_NPAGE ......................................... 415
7.27.2.61 LPA_SLCT ....................................................... 415
7.27.2.62 LPA_10HALF ................................................... 415
7.27.2.63 LPA_10FULL .................................................... 415
7.27.2.64 LPA_100HALF ........................................... 415
7.27.2.65 LPA_100FULL ........................................... 415
7.27.2.66 LPA_100BASE4 ......................................... 416
7.27.2.67 LPA_PAUSE_CAP ....................................... 416
7.27.2.68 LPA_PAUSE_ASYM ..................................... 416
7.27.2.69 LPA_RESV .............................................. 416
7.27.2.70 LPA_RFAULT ........................................... 416
7.27.2.71 LPA_LPACK ............................................. 416
7.27.2.72 LPA_NPAGE ............................................. 416
7.27.2.73 EXPANSION_NWAY ................................... 417
7.27.2.74 EXPANSION_LCWPF .................................. 417
7.27.2.75 EXPANSION_ENABLEPAGE ............................. 417
7.27.2.76 EXPANSION_NPCAPABLE ............................... 417
7.27.2.77 EXPANSION_MFAULTS ................................ 417
7.27.2.78 EXPANSION_RESV .................................... 417
7.27.2.79 NWAYTEST_RESV1 .................................... 417
7.27.2.80 NWAYTEST_LOOPBACK ................................. 418
7.27.2.81 NWAYTEST_RESV2 .................................... 418
7.28 MMC over SPI Driver ....................................... 419
  7.28.1 Detailed Description ................................... 419
  7.28.2 Driver State Machine ................................... 419
  7.28.3 Driver Operations ...................................... 419
  7.28.4 Macro Definition Documentation ....................... 421
    7.28.4.1 MMC_NICE_WAITING ................................ 421
    7.28.4.2 _mmc_driver_methods ................................ 421
    7.28.4.3 mmclIsCardInserted ................................ 421
    7.28.4.4 mmclIsWriteProtected ................................ 422
  7.28.5 Function Documentation ................................ 422
    7.28.5.1 crc7() ............................................. 423
    7.28.5.2 wait() ............................................. 423
    7.28.5.3 send_hdr() ......................................... 424
    7.28.5.4 recvr1() ........................................... 424
    7.28.5.5 recvr3() ........................................... 425
    7.28.5.6 send_command_R1() ................................ 426
    7.28.5.7 send_command_R3() ................................ 427
    7.28.5.8 read_CxD() ......................................... 428
    7.28.5.9 sync() ............................................. 429
    7.28.5.10 mmcInit() ......................................... 430
    7.28.5.11 mmcObjectInit() .................................. 430
    7.28.5.12 mmcStart() ........................................ 431
    7.28.5.13 mmcStop() ......................................... 431
    7.28.5.14 mmcConnect() ...................................... 432
7.28.5.15 mmcDisconnect() ................................................................. 433
7.28.5.16 mmcStartSequentialRead() ................................................... 434
7.28.5.17 mmcSequentialRead() ............................................................ 435
7.28.5.18 mmcStopSequentialRead() ...................................................... 436
7.28.5.19 mmcStartSequentialWrite() .................................................... 436
7.28.5.20 mmcSequentialWrite() ........................................................... 437
7.28.5.21 mmcStopSequentialWrite() ..................................................... 438
7.28.5.22 mmcSync() ........................................................................ 439
7.28.5.23 mmcGetInfo() .................................................................... 440
7.28.5.24 mmcErase() ........................................................................ 441
7.28.6 Variable Documentation .............................................................. 441
7.28.6.1 mmc_vmt ........................................................................... 442
7.28.6.2 crc7_lookup_table ................................................................. 442
7.29 MMC/SD Block Device ................................................................. 443
7.29.1 Detailed Description .................................................................. 443
7.29.2 Macro Definition Documentation ............................................... 447
7.29.2.1 MMCSD_BLOCK_SIZE ......................................................... 447
7.29.2.2 MMCSD_R1_ERROR_MASK .................................................... 448
7.29.2.3 MMCSD_CMD8_PATTERN ..................................................... 448
7.29.2.4 _mmcsd_block_device_methods .............................................. 448
7.29.2.5 _mmcsd_block_device_data ................................................... 448
7.29.2.6 MMCSD_R1_ERROR ............................................................... 448
7.29.2.7 MMCSD_R1_STS ................................................................. 449
7.29.2.8 MMCSD_R1_IS_CARD_LOCKED .............................................. 449
7.29.2.9 mmcsdGetCardCapacity ......................................................... 449
7.29.3 Function Documentation ............................................................ 450
7.29.3.1 _mmcsd_get_slice() ............................................................... 450
7.29.3.2 _mmcsd_get_capacity() ......................................................... 450
7.29.3.3 _mmcsd_get_capacity_ext() .................................................... 451
7.29.3.4 _mmcsd_unpack_sdc_cid() ..................................................... 452
7.29.3.5 _mmcsd_unpack_mmc_cid() .................................................... 452
7.29.3.6 _mmcsd_unpack_csd_mmc() .................................................... 453
7.29.3.7 _mmcsd_unpack_csd_v10() ...................................................... 454
7.29.3.8 _mmcsd_unpack_csd_v20() ...................................................... 454
7.30 Serial NOR Flash Driver ............................................................... 456
7.30.1 Detailed Description .................................................................. 456
7.30.2 Driver State Machine ............................................................... 456
7.30.3 Macro Definition Documentation ............................................... 458
7.30.3.1 SNOR_BUS_DRIVER .............................................................. 458
7.30.3.2 SNOR_SHARED_BUS ............................................................ 458
7.30.3.3 _snor_flash_methods Alone ........................................................ 459
7.30.3.4 _snor_flash_methods .............................................................. 459
7.30.4 Function Documentation ................................................................. 459
  7.30.4.1 snor_get_descriptor() .............................................................. 459
  7.30.4.2 bus_acquire() .......................................................................... 459
  7.30.4.3 bus_release() ........................................................................... 460
  7.30.4.4 bus_stop() ................................................................................ 460
  7.30.4.5 bus_cmd() ................................................................................ 461
  7.30.4.6 bus_cmd_send() ......................................................................... 461
  7.30.4.7 bus_cmd_receive() ................................................................. 461
  7.30.4.8 bus_cmd_addr() ........................................................................ 462
  7.30.4.9 bus_cmd_addr_send() ............................................................... 462
  7.30.4.10 bus_cmd_addr_receive() ......................................................... 463
  7.30.4.11 bus_cmd_dummy_receive() ....................................................... 463
  7.30.4.12 bus_cmd_addr_dummy_receive() .............................................. 464
  7.30.4.13 snorObjectInit() ..................................................................... 464
  7.30.4.14 snorStart() ........................................................................... 465
  7.30.4.15 snorStop() ............................................................................. 465
  7.30.4.16 snorMemoryMap() ................................................................. 465
  7.30.4.17 snorMemoryUnmap() ............................................................. 466

7.30.5 Variable Documentation ................................................................. 467
  7.30.5.1 snor_vmt ................................................................................. 467

7.31 Base Object ....................................................................................... 468
  7.31.1 Detailed Description ..................................................................... 468
  7.31.2 Macro Definition Documentation ............................................... 468
    7.31.2.1 _base_object_methods ........................................................... 468
    7.31.2.2 _base_object_data ................................................................. 469
    7.31.2.3 objGetInstance ..................................................................... 469

7.32 PAL Driver ......................................................................................... 470
  7.32.1 Detailed Description ................................................................... 470
  7.32.2 Implementation Rules .................................................................. 470
    7.32.2.1 Writing on input pads ............................................................... 470
    7.32.2.2 Reading from output pads ....................................................... 470
    7.32.2.3 Writing unused or unimplemented port bits ............................. 471
    7.32.2.4 Reading from unused or unimplemented port bits .................. 471
    7.32.2.5 Reading or writing on pins associated to other functionalities .... 471
  7.32.3 Macro Definition Documentation ............................................... 476
    7.32.3.1 PAL_MODE_RESET ................................................................. 476
    7.32.3.2 PAL_MODE_UNCONNECTED .................................................. 477
    7.32.3.3 PAL_MODE_INPUT ................................................................. 477
    7.32.3.4 PAL_MODE_INPUT_PULLUP .................................................... 477
    7.32.3.5 PAL_MODE_INPUT_PULLDOWN .............................................. 477
    7.32.3.6 PAL_MODE_INPUT_ANALOG .................................................. 477
    7.32.3.7 PAL_MODE_OUTPUT_PUSHPULL ............................................. 477
7.32.3.50 palClearLine
7.32.3.51 palToggleLine
7.32.3.52 palSetLineMode
7.32.3.53 palEnablePadEventI
7.32.3.54 palDisablePadEventI
7.32.3.55 palEnablePadEvent
7.32.3.56 palDisablePadEvent
7.32.3.57 palEnableLineEventI
7.32.3.58 palDisableLineEventI
7.32.3.59 palEnableLineEvent
7.32.3.60 palDisableLineEvent
7.32.3.61 palIsPadEventEnabledX
7.32.3.62 palIsLineEventEnabledX
7.32.3.63 palSetPadCallback
7.32.3.64 palSetLineCallback
7.32.3.65 PAL_IOPORTS_WIDTH
7.32.3.66 PAL_WHOLE_PORT
7.32.3.67 PAL_LINE
7.32.3.68 PAL_PORT
7.32.3.69 PAL_PAD
7.32.3.70 PAL_NOLINE
7.32.3.71 IPORT1
7.32.3.72 pal_lld_init
7.32.3.73 pal_lld_readport
7.32.3.74 pal_lld_readlatch
7.32.3.75 pal_lld_writeport
7.32.3.76 pal_lld_setport
7.32.3.77 pal_lld_clearport
7.32.3.78 pal_lld_toggleport
7.32.3.79 pal_lld_readgroup
7.32.3.80 pal_lld_writegroup
7.32.3.81 pal_lld_setgroupmode
7.32.3.82 pal_lld_readpad
7.32.3.83 pal_lld_writepad
7.32.3.84 pal_lld_setpad
7.32.3.85 pal_lld_clearpad
7.32.3.86 pal_lld_togglepad
7.32.3.87 pal_lld_setpadmode
7.32.3.88 pal_lld_get_pad_event
7.32.3.89 pal_lld_get_line_event

7.32.4 Typedef Documentation
7.32.4.1 palcallback_t
7.32.4.2 iomode_t .................................................. 515
7.32.4.3 ioline_t .................................................. 515
7.32.4.4 iopadid_t .................................................. 516
7.32.4.5 iopadid_t .................................................. 516
7.32.4.6 ioportid_t .................................................. 516

7.32.5 Function Documentation ...................................... 516
  7.32.5.1 palReadBus() ............................................ 516
  7.32.5.2 palWriteBus() ............................................ 516
  7.32.5.3 palSetBusMode() ........................................ 517
  7.32.5.4 palSetPadCallbackI() ................................... 518
  7.32.5.5 palSetLineCallbackI() ................................... 518
  7.32.5.6 palWaitPadTimeoutS() ................................... 518
  7.32.5.7 palWaitPadTimeout() .................................... 519
  7.32.5.8 palWaitLineTimeoutS() ................................... 520
  7.32.5.9 palWaitLineTimeout() .................................... 521
  7.32.5.10 __pal_lld_init() ....................................... 522
  7.32.5.11 __pal_lld_setgroupmode() ................................ 522

7.33 Peripheral Interfaces ........................................... 524
  7.33.1 Detailed Description ....................................... 524

7.34 PWM Driver ..................................................... 525
  7.34.1 Detailed Description ....................................... 525
  7.34.2 Driver State Machine ...................................... 525
  7.34.3 PWM Operations ............................................ 525
  7.34.4 Macro Definition Documentation ............................ 528
    7.34.4.1 PWM_OUTPUT_MASK ..................................... 528
    7.34.4.2 PWM_OUTPUT_DISABLED ................................ 529
    7.34.4.3 PWM_OUTPUT_ACTIVE_HIGH ................................ 529
    7.34.4.4 PWM_OUTPUT_ACTIVE_LOW ................................ 529
    7.34.4.5 PWM_FRACTION_TO_WIDTH ................................ 529
    7.34.4.6 PWM_DEGREES_TO_WIDTH ................................ 530
    7.34.4.7 PWM_PERCENTAGE_TO_WIDTH ................................ 530
    7.34.4.8 pwmChangePeriodI ..................................... 531
    7.34.4.9 pwmEnableChannelI ..................................... 532
    7.34.4.10 pwmDisableChannelI ................................... 532
    7.34.4.11 pwmIsChannelEnabledI .................................. 533
    7.34.4.12 pwmEnablePeriodicNotificationI ....................... 534
    7.34.4.13 pwmDisablePeriodicNotificationI ...................... 534
    7.34.4.14 pwmEnableChannelNotificationI ....................... 535
    7.34.4.15 pwmDisableChannelNotificationI ...................... 535
    7.34.4.16 PWM_CHANNELS ......................................... 536
    7.34.4.17 PLATFORM_PWM_USE_PWM1 ................................ 536
    7.34.4.18 pwm_lld_change_period ................................ 537
7.34.5 Typedef Documentation

7.34.5.1 PWMDriver

7.34.5.2 pwmcallback_t

7.34.5.3 pwmmode_t

7.34.5.4 pwmchannel_t

7.34.5.5 pwmcnmsk_t

7.34.5.6 pwmcnt_t

7.34.6 Enumeration Type Documentation

7.34.6.1 pwmstate_t

7.34.7 Function Documentation

7.34.7.1 pwmInit()

7.34.7.2 pwmObjectInit()

7.34.7.3 pwmStart()

7.34.7.4 pwmStop()

7.34.7.5 pwmChangePeriod()

7.34.7.6 pwmEnableChannel()

7.34.7.7 pwmDisableChannel()

7.34.7.8 pwmEnablePeriodicNotification()

7.34.7.9 pwmDisablePeriodicNotification()

7.34.7.10 pwmEnableChannelNotification()

7.34.7.11 pwmDisableChannelNotification()

7.34.7.12 pwm_lld_init()

7.34.7.13 pwm_lld_start()

7.34.7.14 pwm_lld_stop()

7.34.7.15 pwm_lld_enable_channel()

7.34.7.16 pwm_lld_disable_channel()

7.34.7.17 pwm_lld_enable_periodic_notification()

7.34.7.18 pwm_lld_disable_periodic_notification()

7.34.7.19 pwm_lld_enable_channel_notification()

7.34.7.20 pwm_lld_disable_channel_notification()

7.34.8 Variable Documentation

7.34.8.1 PWMD1

7.35 I/O Bytes Queues

7.35.1 Detailed Description

7.35.2 Macro Definition Documentation

7.35.2.1 Q_OK

7.35.2.2 Q_TIMEOUT

7.35.2.3 Q_RESET

7.35.2.4 Q_EMPTY

7.35.2.5 Q_FULL

7.35.2.6 qSizeX

7.35.2.7 qSpaceI

ChibiOS/HAL
7.35.2.8 qGetLink ......................................................... 558
7.35.2.9 qSetLink ......................................................... 558
7.35.2.10 iqGetFullI ....................................................... 559
7.35.2.11 iqGetEmptyI .................................................... 559
7.35.2.12 iqIsEmptyI ....................................................... 560
7.35.2.13 iqIsFullI ......................................................... 560
7.35.2.14 iqGet ............................................................ 561
7.35.2.15 oqGetFullI ....................................................... 562
7.35.2.16 oqGetEmptyI .................................................... 562
7.35.2.17 oqIsEmptyI ....................................................... 563
7.35.2.18 oqIsFullI ......................................................... 563
7.35.2.19 oqPut ............................................................ 564
7.35.3 Typedef Documentation ........................................... 564
7.35.3.1 io_queue_t ......................................................... 565
7.35.3.2 qnotify_t ........................................................ 565
7.35.3.3 input_queue_t ................................................... 565
7.35.3.4 output_queue_t ................................................. 565
7.35.4 Function Documentation .......................................... 565
7.35.4.1 iq_read() ........................................................ 565
7.35.4.2 oq_write() ....................................................... 566
7.35.4.3 iqObjectInit() ................................................... 566
7.35.4.4 iqResetI() ........................................................ 567
7.35.4.5 iqPutI() ........................................................... 568
7.35.4.6 iqGet() ............................................................ 569
7.35.4.7 iqGetTimeout() .................................................. 570
7.35.4.8 iqReadI() ........................................................ 571
7.35.4.9 iqReadTimeout() ................................................. 571
7.35.4.10 oqObjectInit() .................................................. 572
7.35.4.11 oqResetI() ........................................................ 573
7.35.4.12 oqPutI() ........................................................... 574
7.35.4.13 oqPutTimeout() .................................................. 574
7.35.4.14 oqGet() ........................................................... 576
7.35.4.15 oqWriteI() ........................................................ 576
7.35.4.16 oqWriteTimeout() .............................................. 577
7.36 RTC Driver ............................................................ 579
7.36.1 Detailed Description .............................................. 579
7.36.2 Macro Definition Documentation ................................. 581
7.36.2.1 RTC_BASE_YEAR .................................................. 581
7.36.2.2 _rtc_driver_methods .......................................... 581
7.36.2.3 RTC_SUPPORTS_CALLBACKS ................................... 581
7.36.2.4 RTC_ALARMS ...................................................... 581
7.36.2.5 RTC_HAS_STORAGE .............................................. 582
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.37.6</td>
<td>Enumeration Type Documentation</td>
<td>600</td>
</tr>
<tr>
<td>7.37.6.1</td>
<td>mmc_switch_t</td>
<td>600</td>
</tr>
<tr>
<td>7.37.6.2</td>
<td>sd_switch_t</td>
<td>600</td>
</tr>
<tr>
<td>7.37.6.3</td>
<td>sd_switch_function_t</td>
<td>600</td>
</tr>
<tr>
<td>7.37.6.4</td>
<td>sdcbusmode_t</td>
<td>600</td>
</tr>
<tr>
<td>7.37.6.5</td>
<td>sdcbusclk_t</td>
<td>600</td>
</tr>
<tr>
<td>7.37.7</td>
<td>Function Documentation</td>
<td>600</td>
</tr>
<tr>
<td>7.37.7.1</td>
<td>mode_detect()</td>
<td>600</td>
</tr>
<tr>
<td>7.37.7.2</td>
<td>mmc_init()</td>
<td>601</td>
</tr>
<tr>
<td>7.37.7.3</td>
<td>sdc_init()</td>
<td>602</td>
</tr>
<tr>
<td>7.37.7.4</td>
<td>mmc_cmd6_construct()</td>
<td>602</td>
</tr>
<tr>
<td>7.37.7.5</td>
<td>sdc_cmd6_construct()</td>
<td>604</td>
</tr>
<tr>
<td>7.37.7.6</td>
<td>sdc_cmd6_extract_info()</td>
<td>604</td>
</tr>
<tr>
<td>7.37.7.7</td>
<td>sdc_cmd6_check_status()</td>
<td>605</td>
</tr>
<tr>
<td>7.37.7.8</td>
<td>sdc_detect_bus_clk()</td>
<td>605</td>
</tr>
<tr>
<td>7.37.7.9</td>
<td>mmc_detect_bus_clk()</td>
<td>606</td>
</tr>
<tr>
<td>7.37.7.10</td>
<td>detect_bus_clk()</td>
<td>607</td>
</tr>
<tr>
<td>7.37.7.11</td>
<td>sdc_set_bus_width()</td>
<td>607</td>
</tr>
<tr>
<td>7.37.7.12</td>
<td>mmc_set_bus_width()</td>
<td>608</td>
</tr>
<tr>
<td>7.37.7.13</td>
<td>_sdc_wait_for_transfer_state()</td>
<td>608</td>
</tr>
<tr>
<td>7.37.7.14</td>
<td>sdcInit()</td>
<td>609</td>
</tr>
<tr>
<td>7.37.7.15</td>
<td>sdcObjectInit()</td>
<td>610</td>
</tr>
<tr>
<td>7.37.7.16</td>
<td>sdcStart()</td>
<td>610</td>
</tr>
<tr>
<td>7.37.7.17</td>
<td>sdcStop()</td>
<td>611</td>
</tr>
<tr>
<td>7.37.7.18</td>
<td>sdcConnect()</td>
<td>612</td>
</tr>
<tr>
<td>7.37.7.19</td>
<td>sdcDisconnect()</td>
<td>613</td>
</tr>
<tr>
<td>7.37.7.20</td>
<td>sdcRead()</td>
<td>613</td>
</tr>
<tr>
<td>7.37.7.21</td>
<td>sdcWrite()</td>
<td>614</td>
</tr>
<tr>
<td>7.37.7.22</td>
<td>sdcGetAndClearErrors()</td>
<td>615</td>
</tr>
<tr>
<td>7.37.7.23</td>
<td>sdcSync()</td>
<td>615</td>
</tr>
<tr>
<td>7.37.7.24</td>
<td>sdcGetInfo()</td>
<td>616</td>
</tr>
<tr>
<td>7.37.7.25</td>
<td>sdcErase()</td>
<td>617</td>
</tr>
<tr>
<td>7.37.7.26</td>
<td>sdc_lld_init()</td>
<td>617</td>
</tr>
<tr>
<td>7.37.7.27</td>
<td>sdc_lld_start()</td>
<td>618</td>
</tr>
<tr>
<td>7.37.7.28</td>
<td>sdc_lld_stop()</td>
<td>618</td>
</tr>
<tr>
<td>7.37.7.29</td>
<td>sdc_lld_start_clk()</td>
<td>619</td>
</tr>
<tr>
<td>7.37.7.30</td>
<td>sdc_lld_set_data_clk()</td>
<td>619</td>
</tr>
<tr>
<td>7.37.7.31</td>
<td>sdc_lld_stop_clk()</td>
<td>619</td>
</tr>
<tr>
<td>7.37.7.32</td>
<td>sdc_lld_set_bus_mode()</td>
<td>620</td>
</tr>
<tr>
<td>7.37.7.33</td>
<td>sdc_lld_send_cmd_none()</td>
<td>620</td>
</tr>
<tr>
<td>7.37.7.34</td>
<td>sdc_lld_send_cmd_short()</td>
<td>620</td>
</tr>
<tr>
<td>7.37.7.35</td>
<td>sdc_lld_send_cmd_short_crc()</td>
<td>621</td>
</tr>
</tbody>
</table>
7.37.36 sdc_lld_send_cmd_long_crc() .......................................................... 622
7.37.37 sdc_lld_read() .............................................................................. 623
7.37.38 sdc_lld_write() ........................................................................... 623
7.37.39 sdc_lld_sync() .............................................................................. 624
7.37.8 Variable Documentation ................................................................. 624
7.37.8.1 sdc_vmt ...................................................................................... 624
7.37.8.2 SDCD1 ....................................................................................... 625
7.38 Serial Driver .................................................................................... 626
7.38.1 Detailed Description ..................................................................... 626
7.38.2 Driver State Machine .................................................................. 626
7.38.3 Macro Definition Documentation ................................................. 629
7.38.3.1 SD_PARITY_ERROR ..................................................................... 629
7.38.3.2 SD_FRAMING_ERROR ................................................................. 629
7.38.3.3 SD_OVERRUN_ERROR .............................................................. 629
7.38.3.4 SD_NOISE_ERROR ...................................................................... 629
7.38.3.5 SD_BREAK_DETECTED ............................................................... 630
7.38.3.6 SD_QUEUE_FULL_ERROR .......................................................... 630
7.38.3.7 SERIAL_DEFAULT_BITRATE ....................................................... 630
7.38.3.8 SERIAL_BUFFERS_SIZE .......................................................... 630
7.38.3.9 _serial_driver_methods ............................................................. 630
7.38.3.10 sdPutl ..................................................................................... 631
7.38.3.11 sdPut ...................................................................................... 631
7.38.3.12 sdPutTimeout ......................................................................... 631
7.38.3.13 sdGetl ..................................................................................... 632
7.38.3.14 sdGet ...................................................................................... 632
7.38.3.15 sdGetTimeout ........................................................................ 632
7.38.3.16 sdWritel .................................................................................. 633
7.38.3.17 sdWrite ................................................................................... 633
7.38.3.18 sdWriteTimeout ..................................................................... 633
7.38.3.19 sdAsynchronousWrite .............................................................. 634
7.38.3.20 sdReadl ................................................................................... 634
7.38.3.21 sdRead ....................................................................................... 634
7.38.3.22 sdReadTimeout ...................................................................... 635
7.38.3.23 sdAsynchronousRead .............................................................. 635
7.38.3.24 PLATFORM_SERIAL_USE_USART1 ....................................... 635
7.38.3.25 _serial_driver_data ................................................................. 636
7.38.4 Typedef Documentation ................................................................ 636
7.38.4.1 SerialDriver ............................................................................. 636
7.38.5 Enumeration Type Documentation ................................................ 636
7.38.5.1 sdstate_t .................................................................................. 636
7.38.6 Function Documentation .............................................................. 637
7.38.6.1 sdInit() .................................................................................... 637
7.38.6.2 sdObjectInit() .................................................. 637
7.38.6.3 sdStart() ......................................................... 638
7.38.6.4 sdStop() .......................................................... 638
7.38.6.5 sdIncomingDataI() ............................................. 639
7.38.6.6 sdRequestDataI() ............................................... 640
7.38.6.7 sdPutWouldBlock() ........................................... 641
7.38.6.8 sdGetWouldBlock() .......................................... 642
7.38.6.9 sdControl() ..................................................... 643
7.38.6.10 sd_lld_init() .................................................. 644
7.38.6.11 sd_lld_start() ................................................. 644
7.38.6.12 sd_lld_stop() .................................................. 644

7.38.7 Variable Documentation ........................................ 645

7.38.7.1 SD1 ............................................................... 645
7.38.7.2 default_config ............................................... 645

7.39 Serial over USB Driver .............................................. 646

7.39.1 Detailed Description ............................................ 646
7.39.2 Driver State Machine ......................................... 646

7.39.3 Macro Definition Documentation ................................. 648

7.39.3.1 SERIAL_USB_BUFFERS_SIZE .................................. 648
7.39.3.2 SERIAL_USB_BUFFERS_NUMBER ................................ 648
7.39.3.3 _serial_usb_driver_data ..................................... 648
7.39.3.4 _serial_usb_driver_methods .................................. 649

7.39.4 Typedef Documentation ......................................... 649

7.39.4.1 SerialUSBDriver .............................................. 649

7.39.5 Enumeration Type Documentation ................................. 649

7.39.5.1 sdustate_t ...................................................... 649

7.39.6 Function Documentation ......................................... 649

7.39.6.1 ibnotify() ...................................................... 649
7.39.6.2 obnotify() ...................................................... 650
7.39.6.3 sduInit() ....................................................... 650
7.39.6.4 sduObjectInit() ............................................... 651
7.39.6.5 sduStart() ..................................................... 651
7.39.6.6 sduStop() ....................................................... 652
7.39.6.7 sduSuspendHookI() ........................................... 652
7.39.6.8 sduWakeupHookI() ............................................ 653
7.39.6.9 sduConfigureHookI() ........................................ 653
7.39.6.10 sduRequestsHook() ......................................... 654
7.39.6.11 sduSOFHookI() ............................................... 655
7.39.6.12 sduDataTransmitted() ...................................... 655
7.39.6.13 sduDataReceived() .......................................... 656
7.39.6.14 sduInterruptTransmitted() ................................. 657
7.39.6.15 sduControl() .................................................. 657
7.40 SIO Driver .............................................................. 658
7.40.1 Detailed Description ............................................. 658
7.40.2 Driver State Machine ............................................. 658
7.40.3 Macro Definition Documentation ................................. 662
  7.40.3.1 SIO_NO_ERROR ............................................. 662
  7.40.3.2 SIO_PARITY_ERROR ....................................... 662
  7.40.3.3 SIO_FRAMING_ERROR ...................................... 662
  7.40.3.4 SIO_OVERRUN_ERROR ..................................... 663
  7.40.3.5 SIO_NOISE_ERROR ........................................ 663
  7.40.3.6 SIO_BREAK_DETECTED ...................................... 663
  7.40.3.7 SIO_DEFAULT_BITRATE .................................... 663
  7.40.3.8 SIO_USE_SYNCHRONIZATION ................................. 663
  7.40.3.9 _sio_driver_methods ....................................... 663
  7.40.3.10 sioGetFlagsX ........................................... 664
  7.40.3.11 siolsRXEmptyX ........................................... 664
  7.40.3.12 siolsTXFullX ............................................ 664
  7.40.3.13 sioGetAndClearEventsI .................................. 665
  7.40.3.14 sioGetX ................................................ 665
  7.40.3.15 sioPutX ................................................ 666
  7.40.3.16 sioAsyncReadI .......................................... 666
  7.40.3.17 sioAsyncWriteI ......................................... 667
  7.40.3.18 sioControlI ............................................ 668
  7.40.3.19 __sio_callback_rx ...................................... 668
  7.40.3.20 __sio_callback_rx_idle .................................. 669
  7.40.3.21 __sio_callback_tx ...................................... 669
  7.40.3.22 __sio_callback_tx_end .................................. 670
  7.40.3.23 __sio_callback_rx_evt .................................. 670
  7.40.3.24 __sio_wakeup_rx ........................................ 670
  7.40.3.25 __sio_wakeup_tx ........................................ 671
  7.40.3.26 __sio_wakeup_txend ..................................... 671
  7.40.3.27 PLATFORM_SIO_USE_SIO1 ................................ 672
  7.40.3.28 sio_lld_driver_fields ................................... 672
  7.40.3.29 sio_lld_config_fields ................................... 672
  7.40.3.30 sio_lld_is_rx_empty .................................... 672
  7.40.3.31 sio_lld_is_tx_full ...................................... 673
  7.40.3.32 sio_lld_is_tx_ongoing ................................... 673
7.40.4 Typedef Documentation .......................................... 674
  7.40.4.1 sioflags_t ............................................... 674
  7.40.4.2 SIODriver ................................................ 674
  7.40.4.3 SIOConfig ................................................ 674
  7.40.4.4 SIOOperation ............................................. 675
  7.40.4.5 siocb_t .................................................. 675

ChibiOS/HAL
7.41.3.10 spiStartSendI ............................................. 697
7.41.3.11 spiStartReceiveI ........................................ 698
7.41.3.12 spiPolledExchange ....................................... 699
7.41.3.13 _spi_wakeup_isr ........................................ 699
7.41.3.14 _spi_isr_code ........................................... 700
7.41.3.15 _spi_isr_half_code ..................................... 701
7.41.3.16 _spi_isr_full_code ..................................... 701
7.41.3.17 SPI_SUPPORTS_CIRCULAR .................................. 702
7.41.3.18 PLATFORM_SPI_USE_SPI1 ............................... 702
7.41.3.19 spi_lld_driver_fields .................................... 702
7.41.3.20 spi_lld_config_fields .................................... 702
7.41.4 Typedef Documentation ..................................... 703
  7.41.4.1 SPIDriver .................................................. 703
  7.41.4.2 SPIConfig .................................................. 703
  7.41.4.3 spicallback_t ............................................. 703
7.41.5 Enumeration Type Documentation ............................ 703
  7.41.5.1 spistate_t .................................................. 703
7.41.6 Function Documentation .................................... 704
  7.41.6.1 spiInit() ................................................... 704
  7.41.6.2 spiObjectInit() .......................................... 704
  7.41.6.3 spiStart() ................................................ 705
  7.41.6.4 spiStop() .................................................. 706
  7.41.6.5 spiSelect() ............................................... 707
  7.41.6.6 spiUnselect() ............................................. 708
  7.41.6.7 spiStartIgnore() ........................................ 708
  7.41.6.8 spiStartExchange() ...................................... 709
  7.41.6.9 spiStartSend() ........................................... 710
  7.41.6.10 spiStartReceive() ...................................... 711
  7.41.6.11 spiAbortI() .............................................. 712
  7.41.6.12 spiAbort() ................................................ 713
  7.41.6.13 spiIgnore() .............................................. 714
  7.41.6.14 spiExchange() .......................................... 715
  7.41.6.15 spiSend() ................................................ 716
  7.41.6.16 spiReceive() ............................................ 717
  7.41.6.17 spiAcquireBus() ....................................... 718
  7.41.6.18 spiReleaseBus() ....................................... 719
  7.41.6.19 spi_lld_init() .......................................... 720
  7.41.6.20 spi_lld_start() ......................................... 720
  7.41.6.21 spi_lld_stop() .......................................... 720
  7.41.6.22 spi_lld_select() ....................................... 721
  7.41.6.23 spi_lld_unselect() ..................................... 721
  7.41.6.24 spi_lld_ignore() ....................................... 722
7.41.6.25 spi_lld_exchange() .................................................. 722
7.41.6.26 spi_lld_send() ....................................................... 723
7.41.6.27 spi_lld_receive() ..................................................... 723
7.41.6.28 spi_lld_abort() ....................................................... 724
7.41.6.29 spi_lld_polled_exchange() .................................... 724

7.41.7 Variable Documentation .......................................................... 725
7.41.7.1 SPID1 .................................................................... 725

7.42 ST Driver .............................................................................. 726
7.42.1 Detailed Description ............................................................... 726
7.42.2 Function Documentation ......................................................... 727
7.42.2.1 stInit() ................................................................... 727
7.42.2.2 stBind() .................................................................. 727
7.42.2.3 stGetCounter() .......................................................... 728
7.42.2.4 stStartAlarm() ............................................................ 728
7.42.2.5 stStopAlarm() ................................................................ 729
7.42.2.6 stSetAlarm() ................................................................ 729
7.42.2.7 stGetAlarm() ................................................................ 730
7.42.2.8 stIsAlarmActive() ....................................................... 731
7.42.2.9 stSetCallback() ............................................................ 731
7.42.2.10 stBindAlarmN() .......................................................... 732
7.42.2.11 stStartAlarmN() ........................................................... 732
7.42.2.12 stStopAlarmN() ............................................................ 733
7.42.2.13 stSetAlarmN() ............................................................. 733
7.42.2.14 stGetAlarmN() ............................................................. 734
7.42.2.15 stIsAlarmActiveN() .................................................... 734
7.42.2.16 st_lld_init() ................................................................. 735
7.42.2.17 st_lld_get_counter() ................................................... 735
7.42.2.18 st_lld_start_alarm() .................................................... 735
7.42.2.19 st_lld_stop_alarm() ..................................................... 736
7.42.2.20 st_lld_set_alarm() ....................................................... 736
7.42.2.21 st_lld_get_alarm() ....................................................... 736
7.42.2.22 st_lld_is_alarm_active() ........................................... 737

7.42.3 Variable Documentation .......................................................... 737
7.42.3.1 st_callbacks ................................................................. 737

7.43 TRNG Driver ........................................................................ 738
7.43.1 Detailed Description ............................................................... 738
7.43.2 Macro Definition Documentation ........................................... 739
7.43.2.1 PLATFORM_TRNG_USE_TRNG1 ................................ 739
7.43.2.2 trng_lld_driver_fields ................................................ 740
7.43.2.3 trng_lld_config_fields ................................................ 740

7.43.3 Typedef Documentation .......................................................... 740
7.43.3.1 TRNGDriver ................................................................. 740
### 7.43 TRNG Config

#### 7.43.4 Enumeration Type Documentation

- **trngstate_t**

#### 7.43.5 Function Documentation

- **trngInit()**
- **trngObjectInit()**
- **trngStart()**
- **trngStop()**
- **trngGenerate()**
- **trng_lld_init()**
- **trng_lld_start()**
- **trng_lld_stop()**
- **trng_lld_generate()**

#### 7.43.6 Variable Documentation

- **TRNGD1**

### 7.44 UART Driver

#### 7.44.1 Detailed Description

#### 7.44.2 Driver State Machine

- **Transmitter sub State Machine**
- **Receiver sub State Machine**

#### 7.44.3 Macro Definition Documentation

- **UART_NO_ERROR**
- **UART_PARITY_ERROR**
- **UART_FRAMING_ERROR**
- **UART_OVERRUN_ERROR**
- **UART_NOISE_ERROR**
- **UART_BREAK_DETECTED**
- **UART_USE_WAIT**
- **UART_USE_MUTUAL_EXCLUSION**
- **_uart_wakeup_tx1_isr**
- **_uart_wakeup_tx2_isr**
- **_uart_wakeup_rx_complete_isr**
- **_uart_wakeup_rx_error_isr**
- **_uart_wakeup_rx_cm_isr**
- **_uart_wakeup_rx_timeout_isr**
- **_uart_tx1_isr_code**
- **_uart_tx2_isr_code**
- **_uart_rx_complete_isr_code**
- **_uart_rx_error_isr_code**
- **_uart_rx_idle_code**
- **_uart_timeout_isr_code**
- **_uart_rx_char_match_isr_code**
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.45.7.25 usb_lld_reset()</td>
<td>823</td>
</tr>
<tr>
<td>7.45.7.26 usb_lld_set_address()</td>
<td>824</td>
</tr>
<tr>
<td>7.45.7.27 usb_lld_init_endpoint()</td>
<td>824</td>
</tr>
<tr>
<td>7.45.7.28 usb_lld_disable_endpoints()</td>
<td>825</td>
</tr>
<tr>
<td>7.45.7.29 usb_lld_get_status_out()</td>
<td>825</td>
</tr>
<tr>
<td>7.45.7.30 usb_lld_get_status_in()</td>
<td>826</td>
</tr>
<tr>
<td>7.45.7.31 usb_lld_read_setup()</td>
<td>826</td>
</tr>
<tr>
<td>7.45.7.32 usb_lld_prepare_receive()</td>
<td>827</td>
</tr>
<tr>
<td>7.45.7.33 usb_lld_prepare_transmit()</td>
<td>827</td>
</tr>
<tr>
<td>7.45.7.34 usb_lld_start_out()</td>
<td>828</td>
</tr>
<tr>
<td>7.45.7.35 usb_lld_start_in()</td>
<td>828</td>
</tr>
<tr>
<td>7.45.7.36 usb_lld_stall_out()</td>
<td>828</td>
</tr>
<tr>
<td>7.45.7.37 usb_lld_stall_in()</td>
<td>829</td>
</tr>
<tr>
<td>7.45.7.38 usb_lld_clear_out()</td>
<td>829</td>
</tr>
<tr>
<td>7.45.7.39 usb_lld_clear_in()</td>
<td>829</td>
</tr>
<tr>
<td>7.45.8 Variable Documentation</td>
<td>830</td>
</tr>
<tr>
<td>7.45.8.1 usbgetdescriptor_t</td>
<td>830</td>
</tr>
<tr>
<td>7.45.8.2 USBD1</td>
<td>830</td>
</tr>
<tr>
<td>7.45.8.3 ep0_state</td>
<td>830</td>
</tr>
<tr>
<td>7.45.8.4 in</td>
<td>830</td>
</tr>
<tr>
<td>7.45.8.5 out</td>
<td>831</td>
</tr>
<tr>
<td>7.45.8.6 ep0config</td>
<td>831</td>
</tr>
<tr>
<td>7.46 USB CDC Header</td>
<td>832</td>
</tr>
<tr>
<td>7.46.1 Detailed Description</td>
<td>832</td>
</tr>
<tr>
<td>7.47 WDG Driver</td>
<td>834</td>
</tr>
<tr>
<td>7.47.1 Detailed Description</td>
<td>834</td>
</tr>
<tr>
<td>7.47.2 Macro Definition Documentation</td>
<td>835</td>
</tr>
<tr>
<td>7.47.2.1 wdgResetl</td>
<td>835</td>
</tr>
<tr>
<td>7.47.2.2 PLATFORM_WDG_USE_WDG1</td>
<td>835</td>
</tr>
<tr>
<td>7.47.3 Typedef Documentation</td>
<td>835</td>
</tr>
<tr>
<td>7.47.3.1 WDGDriver</td>
<td>835</td>
</tr>
<tr>
<td>7.47.4 Enumeration Type Documentation</td>
<td>836</td>
</tr>
<tr>
<td>7.47.4.1 wdgstate_t</td>
<td>836</td>
</tr>
<tr>
<td>7.47.5 Function Documentation</td>
<td>836</td>
</tr>
<tr>
<td>7.47.5.1 wdgInit()</td>
<td>836</td>
</tr>
<tr>
<td>7.47.5.2 wdgStart()</td>
<td>837</td>
</tr>
<tr>
<td>7.47.5.3 wdgStop()</td>
<td>837</td>
</tr>
<tr>
<td>7.47.5.4 wdgReset()</td>
<td>838</td>
</tr>
<tr>
<td>7.47.5.5 wdg_lld_init()</td>
<td>839</td>
</tr>
<tr>
<td>7.47.5.6 wdg_lld_start()</td>
<td>839</td>
</tr>
<tr>
<td>7.47.5.7 wdg_lld_stop()</td>
<td>839</td>
</tr>
<tr>
<td>7.47.5.8 wdg_lld_reset()</td>
<td>840</td>
</tr>
</tbody>
</table>
7.48  WSPI Driver ................................................................. 841
  7.48.1 Detailed Description .............................................. 841
  7.48.2 Macro Definition Documentation ................................. 844
    7.48.2.1 WSPI_USE_WAIT .............................................. 845
    7.48.2.2 WSPI_USE_MUTUAL_EXCLUSION ............................ 845
    7.48.2.3 wspiStartCommandI ........................................ 845
    7.48.2.4 wspiStartSendI ............................................ 846
    7.48.2.5 wspiStartReceiveI ......................................... 846
    7.48.2.6 wspiMapFlashI ............................................. 847
    7.48.2.7 wspiUnmapFlashI .......................................... 848
    7.48.2.8 _wspi_wakeup_isr ......................................... 848
    7.48.2.9 _wspi_isr_code ........................................... 849
    7.48.2.10 _wspi_error_code ....................................... 849
    7.48.2.11 PLATFORM_WSPI_USE_WSPI1 ............................... 850
    7.48.2.12 wspi_lld_driver_fields ................................ 850
    7.48.2.13 wspi_lld_config_fields ................................ 851
  7.48.3 Typedef Documentation .......................................... 851
    7.48.3.1 WSPIDriver ................................................ 851
    7.48.3.2 WSPIConfig ................................................ 851
    7.48.3.3 wspicallback_t ........................................... 851
  7.48.4 Enumeration Type Documentation ............................... 851
    7.48.4.1 wspistate_t ............................................... 851
  7.48.5 Function Documentation ....................................... 852
    7.48.5.1 wspiInit() ................................................ 852
    7.48.5.2 wspiObjectInit() ........................................ 853
    7.48.5.3 wspiStart() ............................................... 854
    7.48.5.4 wspiStop() ................................................ 855
    7.48.5.5 wspiStartCommand() ..................................... 856
    7.48.5.6 wspiStartSend() ......................................... 857
    7.48.5.7 wspiStartReceive() ..................................... 858
    7.48.5.8 wspiCommand() ........................................... 859
    7.48.5.9 wspiSend() ............................................... 860
    7.48.5.10 wspiReceive() ........................................... 860
    7.48.5.11 wspiMapFlash() ......................................... 861
    7.48.5.12 wspiUnmapFlash() ....................................... 862
    7.48.5.13 wspiAcquireBus() ...................................... 862
    7.48.5.14 wspiReleaseBus() ...................................... 863
    7.48.5.15 wspi_lld_init() ........................................ 864
    7.48.5.16 wspi_lld_start() ....................................... 864
    7.48.5.17 wspi_lld_stop() ....................................... 864
    7.48.5.18 wspi_lld_command() .................................... 865
    7.48.5.19 wspi_lld_send() ....................................... 865
7.56.2.1 OSAL_ST_RESOLUTION
7.56.2.2 OSAL_ST_FREQUENCY
7.56.2.3 OSAL_ST_MODE
7.56.2.4 OSAL_IRQ_PRIORITY_LEVELS
7.56.2.5 OSAL_IRQ_MAXIMUM_PRIORITY
7.56.2.6 OSAL_DBG_ENABLE_ASSERTS
7.56.2.7 OSAL_DBG_ENABLE_CHECKS
7.56.2.8 osalDbgAssert
7.56.2.9 osalDbgCheck
7.56.2.10 osalDbgCheckClassI
7.56.2.11 osalDbgCheckClassS
7.56.2.12 OSAL_IRQ_IS_VALID_PRIORITY
7.56.2.13 OSAL_IRQ_PROLOGUE
7.56.2.14 OSAL_IRQ_EPILOGUE
7.56.2.15 OSAL_IRQ_HANDLER
7.56.2.16 OSAL_S2I
7.56.2.17 OSAL_MS2I
7.56.2.18 OSAL_US2I
7.56.2.19 OSAL_I2S
7.56.2.20 OSAL_I2MS
7.56.2.21 OSAL_I2US
7.56.2.22 OSAL_S2RTC
7.56.2.23 OSAL_MS2RTC
7.56.2.24 OSAL_US2RTC
7.56.2.25 osalThreadSleepSeconds
7.56.2.26 osalThreadSleepMilliseconds
7.56.2.27 osalThreadSleepMicroseconds
7.56.3 Typedef Documentation
7.56.3.1 syssts_t
7.56.3.2 msg_t
7.56.3.3 systime_t
7.56.3.4 sysinterval_t
7.56.3.5 time_conv_t
7.56.3.6 rtcnt_t
7.56.3.7 thread_reference_t
7.56.3.8 eventflags_t
7.56.3.9 event_source_t
7.56.3.10 eventcallback_t
7.56.3.11 mutex_t
7.56.4 Function Documentation ......................................................... 910
  7.56.4.1 osalInit() ................................................................. 910
  7.56.4.2 osalSysHalt() .......................................................... 910
  7.56.4.3 osalSysPolledDelayX() ............................................... 911
  7.56.4.4 osalOsTimerHandler() ............................................... 911
  7.56.4.5 osalOsRescheduleS() ................................................ 912
  7.56.4.6 osalOsGetSystemTimeX() ........................................... 912
  7.56.4.7 osalThreadSleepS() ................................................ 912
  7.56.4.8 osalThreadSleep() .................................................. 913
  7.56.4.9 osalThreadSuspendS() ............................................... 913
  7.56.4.10 osalThreadSuspendTimeoutS() .................................... 914
  7.56.4.11 osalThreadResumeI() .............................................. 914
  7.56.4.12 osalThreadResumeS() .............................................. 915
  7.56.4.13 osalThreadEnqueueTimeoutS() .................................... 915
  7.56.4.14 osalThreadDequeueNextI() ....................................... 916
  7.56.4.15 osalThreadDequeueAllI() ....................................... 916
  7.56.4.16 osalEventBroadcastFlagsI() ..................................... 918
  7.56.4.17 osalEventBroadcastFlags() .................................... 918
  7.56.4.18 osalEventSetCallback() .......................................... 919
  7.56.4.19 osalMutexLock() .................................................. 920
  7.56.4.20 osalMutexUnlock() ................................................ 920
  7.56.4.21 osalSysDisable() .................................................. 920
  7.56.4.22 osalSysEnable() ................................................... 921
  7.56.4.23 osalSysLock() ..................................................... 921
  7.56.4.24 osalSysUnlock() ................................................... 921
  7.56.4.25 osalSysLockFromISR() ........................................... 922
  7.56.4.26 osalSysUnlockFromISR() ......................................... 922
  7.56.4.27 osalSysGetStatusAndLockX() ................................... 922
  7.56.4.28 osalSysRestoreStatusX() ....................................... 923
  7.56.4.29 osalTimeAddX() ................................................... 923
  7.56.4.30 osalTimeDiffX() .................................................. 923
  7.56.4.31 osalTimeIsInRangeX() ............................................ 924
  7.56.4.32 osalThreadQueueObjectInit() .................................. 925
  7.56.4.33 osalEventObjectInit() .......................................... 925
  7.56.4.34 osalMutexObjectInit() ........................................... 925

7.56.5 Variable Documentation .................................................. 926
  7.56.5.1 osal_halt_msg [1/2] ............................................... 926
  7.56.5.2 osal_halt_msg [2/2] ............................................... 926

8 Data Structure Documentation .................................................. 927
  8.1 BaseAsynchronousChannel Struct Reference ................................ 927
    8.1.1 Detailed Description ................................................... 928
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1.2</td>
<td>Field Documentation</td>
<td>928</td>
</tr>
<tr>
<td>8.1.2.1</td>
<td>vmt</td>
<td>928</td>
</tr>
<tr>
<td>8.2</td>
<td>BaseAsynchronousChannelVMT Struct Reference</td>
<td>929</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Detailed Description</td>
<td>930</td>
</tr>
<tr>
<td>8.3</td>
<td>BaseBlockDevice Struct Reference</td>
<td>930</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Detailed Description</td>
<td>932</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Field Documentation</td>
<td>932</td>
</tr>
<tr>
<td>8.3.2.1</td>
<td>vmt</td>
<td>932</td>
</tr>
<tr>
<td>8.4</td>
<td>BaseBlockDeviceVMT Struct Reference</td>
<td>933</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Detailed Description</td>
<td>933</td>
</tr>
<tr>
<td>8.5</td>
<td>BaseBufferedStream Struct Reference</td>
<td>934</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Detailed Description</td>
<td>935</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Field Documentation</td>
<td>935</td>
</tr>
<tr>
<td>8.5.2.1</td>
<td>vmt</td>
<td>935</td>
</tr>
<tr>
<td>8.6</td>
<td>BaseBufferedStreamVMT Struct Reference</td>
<td>936</td>
</tr>
<tr>
<td>8.6.1</td>
<td>Detailed Description</td>
<td>936</td>
</tr>
<tr>
<td>8.7</td>
<td>BaseChannel Struct Reference</td>
<td>937</td>
</tr>
<tr>
<td>8.7.1</td>
<td>Detailed Description</td>
<td>938</td>
</tr>
<tr>
<td>8.7.2</td>
<td>Field Documentation</td>
<td>938</td>
</tr>
<tr>
<td>8.7.2.1</td>
<td>vmt</td>
<td>938</td>
</tr>
<tr>
<td>8.8</td>
<td>BaseChannelVMT Struct Reference</td>
<td>939</td>
</tr>
<tr>
<td>8.8.1</td>
<td>Detailed Description</td>
<td>940</td>
</tr>
<tr>
<td>8.9</td>
<td>BaseFlash Struct Reference</td>
<td>940</td>
</tr>
<tr>
<td>8.9.1</td>
<td>Detailed Description</td>
<td>942</td>
</tr>
<tr>
<td>8.9.2</td>
<td>Field Documentation</td>
<td>942</td>
</tr>
<tr>
<td>8.9.2.1</td>
<td>vmt</td>
<td>942</td>
</tr>
<tr>
<td>8.10</td>
<td>BaseFlashVMT Struct Reference</td>
<td>943</td>
</tr>
<tr>
<td>8.10.1</td>
<td>Detailed Description</td>
<td>943</td>
</tr>
<tr>
<td>8.11</td>
<td>BaseObject Struct Reference</td>
<td>943</td>
</tr>
<tr>
<td>8.11.1</td>
<td>Detailed Description</td>
<td>944</td>
</tr>
<tr>
<td>8.11.2</td>
<td>Field Documentation</td>
<td>944</td>
</tr>
<tr>
<td>8.11.2.1</td>
<td>vmt</td>
<td>945</td>
</tr>
<tr>
<td>8.12</td>
<td>BaseObjectVMT Struct Reference</td>
<td>945</td>
</tr>
<tr>
<td>8.12.1</td>
<td>Detailed Description</td>
<td>945</td>
</tr>
<tr>
<td>8.13</td>
<td>BasePersistentStorage Struct Reference</td>
<td>945</td>
</tr>
<tr>
<td>8.13.1</td>
<td>Detailed Description</td>
<td>946</td>
</tr>
<tr>
<td>8.13.2</td>
<td>Field Documentation</td>
<td>946</td>
</tr>
<tr>
<td>8.13.2.1</td>
<td>vmt</td>
<td>946</td>
</tr>
<tr>
<td>8.14</td>
<td>BasePersistentStorageVMT Struct Reference</td>
<td>947</td>
</tr>
<tr>
<td>8.14.1</td>
<td>Detailed Description</td>
<td>947</td>
</tr>
<tr>
<td>8.15</td>
<td>BaseSequentialStream Struct Reference</td>
<td>947</td>
</tr>
<tr>
<td>8.15.1</td>
<td>Detailed Description</td>
<td>949</td>
</tr>
</tbody>
</table>
8.23 CANTxFrame Struct Reference .......................................................... 960
8.23.1 Detailed Description ..................................................................... 961
8.23.2 Field Documentation ................................................................... 961
  8.23.2.1 DLC ................................................................. 961
  8.23.2.2 RTR ............................................................... 962
  8.23.2.3 IDE .................................................................. 962
  8.23.2.4 SID .................................................................. 962
  8.23.2.5 EID .................................................................. 962
  8.23.2.6 data8 .............................................................. 962
  8.23.2.7 data16 ............................................................ 962
  8.23.2.8 data32 ............................................................ 963
8.24 cdc_linecoding_t Struct Reference .................................................. 963
  8.24.1 Detailed Description ................................................................. 963
8.25 CRYConfig Struct Reference ........................................................... 963
  8.25.1 Detailed Description ................................................................. 964
8.26 CRYDriver Struct Reference ............................................................ 964
  8.26.1 Detailed Description ................................................................. 964
  8.26.2 Field Documentation ................................................................ 964
    8.26.2.1 state ............................................................. 965
    8.26.2.2 config ........................................................... 965
8.27 EFlashDriverVMT Struct Reference .................................................. 965
  8.27.1 Detailed Description ................................................................. 966
8.28 event_source Struct Reference ......................................................... 966
  8.28.1 Detailed Description ................................................................. 967
  8.28.2 Field Documentation ................................................................ 967
    8.28.2.1 flags ............................................................... 967
    8.28.2.2 cb ................................................................. 967
    8.28.2.3 param ........................................................... 967
8.29 FileStream Struct Reference ............................................................. 968
  8.29.1 Detailed Description ................................................................. 969
  8.29.2 Field Documentation ................................................................ 969
    8.29.2.1 vmt ............................................................... 969
8.30 FileStreamVMT Struct Reference ...................................................... 970
  8.30.1 Detailed Description ................................................................. 970
8.31 flash_descriptor_t Struct Reference .................................................. 971
  8.31.1 Detailed Description ................................................................. 971
  8.31.2 Field Documentation ................................................................ 972
    8.31.2.1 attributes ....................................................... 972
    8.31.2.2 page_size ....................................................... 972
    8.31.2.3 sectors_count ............................................... 972
    8.31.2.4 sectors ......................................................... 972
    8.31.2.5 sectors_size .................................................. 972
8.31.2.6 address ......................................................... 973
8.31.2.7 size .......................................................... 973
8.32 flash_sector_descriptor_t Struct Reference .................... 973
  8.32.1 Detailed Description ........................................ 973
  8.32.2 Field Documentation ....................................... 974
    8.32.2.1 offset .................................................... 974
    8.32.2.2 size ...................................................... 974
8.33 GPTConfig Struct Reference ..................................... 974
  8.33.1 Detailed Description ....................................... 975
  8.33.2 Field Documentation ....................................... 975
    8.33.2.1 frequency ............................................... 975
    8.33.2.2 callback ................................................. 975
8.34 GPTDriver Struct Reference .................................... 976
  8.34.1 Detailed Description ....................................... 976
  8.34.2 Field Documentation ....................................... 976
    8.34.2.1 state ..................................................... 976
    8.34.2.2 config ................................................... 977
8.35 hal_adc_config Struct Reference ................................ 977
  8.35.1 Detailed Description ....................................... 977
8.36 hal_adc_configuration_group Struct Reference .................. 977
  8.36.1 Detailed Description ....................................... 978
  8.36.2 Field Documentation ....................................... 979
    8.36.2.1 circular ................................................ 979
    8.36.2.2 num_channels ......................................... 979
    8.36.2.3 end_cb .................................................. 979
    8.36.2.4 error_cb ............................................... 979
8.37 hal_adc_driver Struct Reference ................................ 980
  8.37.1 Detailed Description ....................................... 981
  8.37.2 Field Documentation ....................................... 981
    8.37.2.1 state ..................................................... 981
    8.37.2.2 config ................................................... 981
    8.37.2.3 samples ................................................ 981
    8.37.2.4 depth ................................................... 981
    8.37.2.5 grpp .................................................... 982
    8.37.2.6 thread .................................................. 982
    8.37.2.7 mutex ................................................... 982
8.38 hal_dac_config Struct Reference ................................ 982
  8.38.1 Detailed Description ....................................... 982
8.39 hal_dac_conversion_group Struct Reference .................... 983
  8.39.1 Detailed Description ....................................... 983
  8.39.2 Field Documentation ....................................... 983
    8.39.2.1 num_channels ......................................... 984
8.74.2.3 current_bank ............................................ 1038
8.74.2.4 current_counter ................................. 1038
8.74.2.5 next_offset ........................................... 1038
8.74.2.6 used_space .......................................... 1038
8.74.2.7 descriptors ........................................ 1038
8.74.2.8 tr_next_offset .................................... 1039
8.74.2.9 tr_limit_offset .................................. 1039
8.74.2.10 tr_nops .............................................. 1039
8.74.2.11 tr_ops ................................................. 1039
8.74.2.12 buffer ................................................ 1039

8.75 MMCConfig Struct Reference .......................... 1040
8.75.1 Detailed Description ................................. 1040
8.75.2 Field Documentation .................................. 1041
8.75.2.1 spip .................................................. 1041
8.75.2.2 lscfg ................................................ 1041
8.75.2.3 hscfg ................................................. 1041

8.76 MMCDriver Struct Reference .......................... 1041
8.76.1 Detailed Description ................................. 1043
8.76.2 Field Documentation .................................. 1043
8.76.2.1 vmt .................................................. 1044
8.76.2.2 config .............................................. 1044
8.76.2.3 block_addresses ................................. 1044

8.77 MMCDriverVMT Struct Reference ....................... 1044
8.77.1 Detailed Description ................................. 1045

8.78 MMCSDBlockDevice Struct Reference .................. 1045
8.78.1 Detailed Description ................................. 1047
8.78.2 Field Documentation .................................. 1047
8.78.2.1 vmt .................................................. 1047

8.79 MMCSDBlockDeviceVMT Struct Reference ............... 1048
8.79.1 Detailed Description ................................. 1049

8.80 NullStream Struct Reference ............................ 1049
8.80.1 Detailed Description ................................. 1050
8.80.2 Field Documentation .................................. 1050
8.80.2.1 vmt .................................................. 1050

8.81 NullStreamVMT Struct Reference ....................... 1051
8.81.1 Detailed Description ................................. 1051

8.82 PALConfig Struct Reference ............................ 1051
8.82.1 Detailed Description ................................. 1051

8.83 palevent_t Struct Reference ........................... 1052
8.83.1 Detailed Description ................................. 1052
8.83.2 Field Documentation .................................. 1052
8.83.2.1 threads ............................................. 1052
8.83.2.2 cb ................................................................. 1053
8.83.2.3 arg ............................................................... 1053
8.84 PWMChannelConfig Struct Reference ................................................................. 1053
8.84.1 Detailed Description .......................................................................................... 1054
8.84.2 Field Documentation ......................................................................................... 1054
8.84.2.1 mode ......................................................................................... 1054
8.84.2.2 callback .................................................................................. 1054
8.85 PWMConfig Struct Reference ............................................................................. 1055
8.85.1 Detailed Description ......................................................................................... 1055
8.85.2 Field Documentation ......................................................................................... 1056
8.85.2.1 frequency .............................................................................. 1056
8.85.2.2 period ................................................................................. 1056
8.85.2.3 callback .............................................................................. 1056
8.85.2.4 channels ........................................................................... 1056
8.86 PWMDriver Struct Reference .............................................................................. 1057
8.86.1 Detailed Description ......................................................................................... 1058
8.86.2 Field Documentation ......................................................................................... 1058
8.86.2.1 state .................................................................................. 1058
8.86.2.2 config ................................................................................. 1058
8.86.2.3 period ................................................................................ 1058
8.86.2.4 enabled ............................................................................ 1058
8.86.2.5 channels ........................................................................... 1058
8.87 RTCAlarm Struct Reference .............................................................................. 1059
8.87.1 Detailed Description ......................................................................................... 1059
8.88 RTCDateTime Struct Reference .......................................................................... 1059
8.88.1 Detailed Description ......................................................................................... 1060
8.88.2 Field Documentation ......................................................................................... 1060
8.88.2.1 year .................................................................................. 1060
8.88.2.2 month .............................................................................. 1060
8.88.2.3 dstflag ............................................................................ 1061
8.88.2.4 dayofweek ...................................................................... 1061
8.88.2.5 day ................................................................................ 1061
8.88.2.6 millisecond ....................................................................... 1061
8.89 RTCDriver Struct Reference .............................................................................. 1061
8.89.1 Detailed Description ......................................................................................... 1062
8.89.2 Field Documentation ......................................................................................... 1062
8.89.2.1 vmt .................................................................................. 1062
8.90 RTCDriverVMT Struct Reference ........................................................................ 1063
8.90.1 Detailed Description ......................................................................................... 1064
8.91 SDCConfig Struct Reference .............................................................................. 1064
8.91.1 Detailed Description ......................................................................................... 1065
8.91.2 Field Documentation ......................................................................................... 1065
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.91.2.1</td>
<td>bus_width</td>
<td>1065</td>
</tr>
<tr>
<td>8.92</td>
<td>SDCDriver Struct Reference</td>
<td>1065</td>
</tr>
<tr>
<td>8.92.1</td>
<td>Detailed Description</td>
<td>1066</td>
</tr>
<tr>
<td>8.92.2</td>
<td>Field Documentation</td>
<td>1067</td>
</tr>
<tr>
<td>8.92.2.1</td>
<td>vmt</td>
<td>1067</td>
</tr>
<tr>
<td>8.92.2.2</td>
<td>config</td>
<td>1067</td>
</tr>
<tr>
<td>8.92.2.3</td>
<td>cardmode</td>
<td>1067</td>
</tr>
<tr>
<td>8.92.2.4</td>
<td>errors</td>
<td>1067</td>
</tr>
<tr>
<td>8.92.2.5</td>
<td>rca</td>
<td>1067</td>
</tr>
<tr>
<td>8.92.2.6</td>
<td>buf</td>
<td>1067</td>
</tr>
<tr>
<td>8.93</td>
<td>SDCDriverVMT Struct Reference</td>
<td>1068</td>
</tr>
<tr>
<td>8.93.1</td>
<td>Detailed Description</td>
<td>1069</td>
</tr>
<tr>
<td>8.94</td>
<td>SerialConfig Struct Reference</td>
<td>1069</td>
</tr>
<tr>
<td>8.94.1</td>
<td>Detailed Description</td>
<td>1070</td>
</tr>
<tr>
<td>8.94.2</td>
<td>Field Documentation</td>
<td>1070</td>
</tr>
<tr>
<td>8.94.2.1</td>
<td>speed</td>
<td>1070</td>
</tr>
<tr>
<td>8.95</td>
<td>SerialDriver Struct Reference</td>
<td>1070</td>
</tr>
<tr>
<td>8.95.1</td>
<td>Detailed Description</td>
<td>1072</td>
</tr>
<tr>
<td>8.95.2</td>
<td>Field Documentation</td>
<td>1072</td>
</tr>
<tr>
<td>8.95.2.1</td>
<td>vmt</td>
<td>1072</td>
</tr>
<tr>
<td>8.96</td>
<td>SerialDriverVMT Struct Reference</td>
<td>1073</td>
</tr>
<tr>
<td>8.96.1</td>
<td>Detailed Description</td>
<td>1074</td>
</tr>
<tr>
<td>8.97</td>
<td>SerialUSBConfig Struct Reference</td>
<td>1074</td>
</tr>
<tr>
<td>8.97.1</td>
<td>Detailed Description</td>
<td>1075</td>
</tr>
<tr>
<td>8.97.2</td>
<td>Field Documentation</td>
<td>1075</td>
</tr>
<tr>
<td>8.97.2.1</td>
<td>usbp</td>
<td>1076</td>
</tr>
<tr>
<td>8.97.2.2</td>
<td>bulk_in</td>
<td>1076</td>
</tr>
<tr>
<td>8.97.2.3</td>
<td>bulk_out</td>
<td>1076</td>
</tr>
<tr>
<td>8.97.2.4</td>
<td>int_in</td>
<td>1076</td>
</tr>
<tr>
<td>8.98</td>
<td>SerialUSBDriver Struct Reference</td>
<td>1077</td>
</tr>
<tr>
<td>8.98.1</td>
<td>Detailed Description</td>
<td>1078</td>
</tr>
<tr>
<td>8.98.2</td>
<td>Field Documentation</td>
<td>1078</td>
</tr>
<tr>
<td>8.98.2.1</td>
<td>vmt</td>
<td>1078</td>
</tr>
<tr>
<td>8.99</td>
<td>SerialUSBDriverVMT Struct Reference</td>
<td>1079</td>
</tr>
<tr>
<td>8.99.1</td>
<td>Detailed Description</td>
<td>1080</td>
</tr>
<tr>
<td>8.100</td>
<td>SHA1Context Struct Reference</td>
<td>1080</td>
</tr>
<tr>
<td>8.100.1</td>
<td>Detailed Description</td>
<td>1081</td>
</tr>
<tr>
<td>8.101</td>
<td>SHA256Context Struct Reference</td>
<td>1081</td>
</tr>
<tr>
<td>8.101.1</td>
<td>Detailed Description</td>
<td>1081</td>
</tr>
<tr>
<td>8.102</td>
<td>SHA512Context Struct Reference</td>
<td>1081</td>
</tr>
<tr>
<td>8.102.1</td>
<td>Detailed Description</td>
<td>1082</td>
</tr>
<tr>
<td>8.103</td>
<td>sio_driver_vmt Struct Reference</td>
<td>1082</td>
</tr>
</tbody>
</table>
8.119 USBinEndpointState Struct Reference ................................................. 1107
  8.119.1 Detailed Description ................................................................. 1107
  8.119.2 Field Documentation .................................................................... 1107
    8.119.2.1 txsize ..................................................................................... 1107
    8.119.2.2 txcnt ...................................................................................... 1108
    8.119.2.3 txbuf ...................................................................................... 1108
    8.119.2.4 thread ...................................................................................... 1108
8.120 USBOutEndpointState Struct Reference ............................................... 1108
  8.120.1 Detailed Description ................................................................. 1109
  8.120.2 Field Documentation .................................................................... 1109
    8.120.2.1 rxsize ..................................................................................... 1109
    8.120.2.2 rxcnt ...................................................................................... 1109
    8.120.2.3 rxbuf ...................................................................................... 1109
    8.120.2.4 thread ...................................................................................... 1109
8.121 WDGConfig Struct Reference ............................................................. 1110
  8.121.1 Detailed Description ................................................................. 1110
8.122 WDGDriver Struct Reference .............................................................. 1110
  8.122.1 Detailed Description ................................................................. 1111
  8.122.2 Field Documentation .................................................................... 1111
    8.122.2.1 state ......................................................................................... 1111
    8.122.2.2 config ...................................................................................... 1111
8.123 wspi_command_t Struct Reference ...................................................... 1111
  8.123.1 Detailed Description ................................................................. 1112
  8.123.2 Field Documentation .................................................................... 1112
    8.123.2.1 cfg ......................................................................................... 1112
    8.123.2.2 cmd ......................................................................................... 1112
    8.123.2.3 addr ......................................................................................... 1112
    8.123.2.4 alt ............................................................................................ 1112
    8.123.2.5 dummy ................................................................................... 1112

9 File Documentation ..................................................................................... 1113
  9.1 bufstreams.c File Reference ................................................................. 1113
    9.1.1 Detailed Description ..................................................................... 1113
  9.2 bufstreams.h File Reference ................................................................. 1113
    9.2.1 Detailed Description ..................................................................... 1114
  9.3 chprintf.c File Reference ..................................................................... 1114
    9.3.1 Detailed Description ..................................................................... 1114
  9.4 chprintf.h File Reference ................................................................... 1114
    9.4.1 Detailed Description ................................................................... 1115
  9.5 chscanf.c File Reference ..................................................................... 1115
    9.5.1 Detailed Description ................................................................... 1115
  9.6 chscanf.h File Reference ..................................................................... 1116
9.6.1 Detailed Description .................................................. 1116
9.7 hal.c File Reference ......................................................... 1116
  9.7.1 Detailed Description .............................................. 1116
9.8 hal.h File Reference ......................................................... 1117
  9.8.1 Detailed Description .............................................. 1118
9.9 hal_adc.c File Reference .................................................. 1118
  9.9.1 Detailed Description .............................................. 1119
9.10 hal_adc.h File Reference .................................................. 1119
  9.10.1 Detailed Description ............................................. 1121
9.11 hal_adc_lld.c File Reference .............................................. 1121
  9.11.1 Detailed Description ............................................. 1121
9.12 hal_adc_lld.h File Reference .............................................. 1122
  9.12.1 Detailed Description ............................................. 1122
9.13 hal_buffers.c File Reference .............................................. 1123
  9.13.1 Detailed Description ............................................. 1124
9.14 hal_buffers.h File Reference .............................................. 1124
  9.14.1 Detailed Description ............................................. 1126
9.15 hal_can.c File Reference .................................................. 1126
  9.15.1 Detailed Description ............................................. 1127
9.16 hal_can.h File Reference .................................................. 1127
  9.16.1 Detailed Description ............................................. 1128
9.17 hal_can_lld.c File Reference .............................................. 1128
  9.17.1 Detailed Description ............................................. 1129
9.18 hal_can_lld.h File Reference .............................................. 1129
  9.18.1 Detailed Description ............................................. 1130
9.19 hal_channels.h File Reference .............................................. 1131
  9.19.1 Detailed Description ............................................. 1132
9.20 hal_crypto.c File Reference .............................................. 1132
  9.20.1 Detailed Description ............................................. 1134
9.21 hal_crypto.h File Reference .............................................. 1134
  9.21.1 Detailed Description ............................................. 1137
9.22 hal_crypto_lld.c File Reference .............................................. 1138
  9.22.1 Detailed Description ............................................. 1140
9.23 hal_crypto_lld.h File Reference .............................................. 1140
  9.23.1 Detailed Description ............................................. 1143
9.24 hal_dac.c File Reference .................................................. 1143
  9.24.1 Detailed Description ............................................. 1144
9.25 hal_dac.h File Reference .................................................. 1144
  9.25.1 Detailed Description ............................................. 1146
9.26 hal_dac_lld.c File Reference .............................................. 1146
  9.26.1 Detailed Description ............................................. 1147
9.27 hal_dac_lld.h File Reference .............................................. 1147
<table>
<thead>
<tr>
<th>Section</th>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.27.1</td>
<td>Detailed Description</td>
<td>1148</td>
</tr>
<tr>
<td>9.28</td>
<td>hal_efl.c File Reference</td>
<td>1148</td>
</tr>
<tr>
<td>9.28.1</td>
<td>Detailed Description</td>
<td>1148</td>
</tr>
<tr>
<td>9.29</td>
<td>hal_efl.h File Reference</td>
<td>1148</td>
</tr>
<tr>
<td>9.29.1</td>
<td>Detailed Description</td>
<td>1149</td>
</tr>
<tr>
<td>9.30</td>
<td>hal_efl_lld.c File Reference</td>
<td>1149</td>
</tr>
<tr>
<td>9.30.1</td>
<td>Detailed Description</td>
<td>1150</td>
</tr>
<tr>
<td>9.31</td>
<td>hal_efl_lld.h File Reference</td>
<td>1150</td>
</tr>
<tr>
<td>9.31.1</td>
<td>Detailed Description</td>
<td>1151</td>
</tr>
<tr>
<td>9.32</td>
<td>hal_files.h File Reference</td>
<td>1151</td>
</tr>
<tr>
<td>9.32.1</td>
<td>Detailed Description</td>
<td>1152</td>
</tr>
<tr>
<td>9.33</td>
<td>hal_flash.c File Reference</td>
<td>1152</td>
</tr>
<tr>
<td>9.33.1</td>
<td>Detailed Description</td>
<td>1153</td>
</tr>
<tr>
<td>9.34</td>
<td>hal_flash.h File Reference</td>
<td>1153</td>
</tr>
<tr>
<td>9.34.1</td>
<td>Detailed Description</td>
<td>1154</td>
</tr>
<tr>
<td>9.35</td>
<td>hal_gpt.c File Reference</td>
<td>1155</td>
</tr>
<tr>
<td>9.35.1</td>
<td>Detailed Description</td>
<td>1155</td>
</tr>
<tr>
<td>9.36</td>
<td>hal_gpt.h File Reference</td>
<td>1155</td>
</tr>
<tr>
<td>9.36.1</td>
<td>Detailed Description</td>
<td>1157</td>
</tr>
<tr>
<td>9.37</td>
<td>hal_gpt_lld.c File Reference</td>
<td>1157</td>
</tr>
<tr>
<td>9.37.1</td>
<td>Detailed Description</td>
<td>1157</td>
</tr>
<tr>
<td>9.38</td>
<td>hal_gpt_lld.h File Reference</td>
<td>1157</td>
</tr>
<tr>
<td>9.38.1</td>
<td>Detailed Description</td>
<td>1158</td>
</tr>
<tr>
<td>9.39</td>
<td>hal_i2c.c File Reference</td>
<td>1158</td>
</tr>
<tr>
<td>9.39.1</td>
<td>Detailed Description</td>
<td>1159</td>
</tr>
<tr>
<td>9.40</td>
<td>hal_i2c.h File Reference</td>
<td>1159</td>
</tr>
<tr>
<td>9.40.1</td>
<td>Detailed Description</td>
<td>1161</td>
</tr>
<tr>
<td>9.41</td>
<td>hal_i2c_lld.c File Reference</td>
<td>1161</td>
</tr>
<tr>
<td>9.41.1</td>
<td>Detailed Description</td>
<td>1161</td>
</tr>
<tr>
<td>9.42</td>
<td>hal_i2c_lld.h File Reference</td>
<td>1161</td>
</tr>
<tr>
<td>9.42.1</td>
<td>Detailed Description</td>
<td>1162</td>
</tr>
<tr>
<td>9.43</td>
<td>hal_i2s.c File Reference</td>
<td>1162</td>
</tr>
<tr>
<td>9.43.1</td>
<td>Detailed Description</td>
<td>1163</td>
</tr>
<tr>
<td>9.44</td>
<td>hal_i2s.h File Reference</td>
<td>1163</td>
</tr>
<tr>
<td>9.44.1</td>
<td>Detailed Description</td>
<td>1164</td>
</tr>
<tr>
<td>9.45</td>
<td>hal_i2s_lld.c File Reference</td>
<td>1164</td>
</tr>
<tr>
<td>9.45.1</td>
<td>Detailed Description</td>
<td>1165</td>
</tr>
<tr>
<td>9.46</td>
<td>hal_i2s_lld.h File Reference</td>
<td>1165</td>
</tr>
<tr>
<td>9.46.1</td>
<td>Detailed Description</td>
<td>1166</td>
</tr>
<tr>
<td>9.47</td>
<td>hal_icu.c File Reference</td>
<td>1166</td>
</tr>
<tr>
<td>9.47.1</td>
<td>Detailed Description</td>
<td>1166</td>
</tr>
<tr>
<td>9.48</td>
<td>hal_icu.h File Reference</td>
<td>1166</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>9.48.1</td>
<td>Detailed Description</td>
<td>1168</td>
</tr>
<tr>
<td>9.49</td>
<td>hal_icu_ld.c File Reference</td>
<td>1168</td>
</tr>
<tr>
<td>9.49.1</td>
<td>Detailed Description</td>
<td>1169</td>
</tr>
<tr>
<td>9.50</td>
<td>hal_icu_ld.h File Reference</td>
<td>1169</td>
</tr>
<tr>
<td>9.50.1</td>
<td>Detailed Description</td>
<td>1170</td>
</tr>
<tr>
<td>9.51</td>
<td>hal_ioblock.h File Reference</td>
<td>1170</td>
</tr>
<tr>
<td>9.51.1</td>
<td>Detailed Description</td>
<td>1171</td>
</tr>
<tr>
<td>9.52</td>
<td>hal_ld.c File Reference</td>
<td>1171</td>
</tr>
<tr>
<td>9.52.1</td>
<td>Detailed Description</td>
<td>1172</td>
</tr>
<tr>
<td>9.53</td>
<td>hal_ld.h File Reference</td>
<td>1172</td>
</tr>
<tr>
<td>9.53.1</td>
<td>Detailed Description</td>
<td>1172</td>
</tr>
<tr>
<td>9.54</td>
<td>hal_mac.c File Reference</td>
<td>1173</td>
</tr>
<tr>
<td>9.54.1</td>
<td>Detailed Description</td>
<td>1173</td>
</tr>
<tr>
<td>9.55</td>
<td>hal_mac.h File Reference</td>
<td>1173</td>
</tr>
<tr>
<td>9.55.1</td>
<td>Detailed Description</td>
<td>1175</td>
</tr>
<tr>
<td>9.56</td>
<td>hal_mac_ld.c File Reference</td>
<td>1175</td>
</tr>
<tr>
<td>9.56.1</td>
<td>Detailed Description</td>
<td>1175</td>
</tr>
<tr>
<td>9.57</td>
<td>hal_mac_ld.h File Reference</td>
<td>1176</td>
</tr>
<tr>
<td>9.57.1</td>
<td>Detailed Description</td>
<td>1177</td>
</tr>
<tr>
<td>9.58</td>
<td>hal_mfs.c File Reference</td>
<td>1177</td>
</tr>
<tr>
<td>9.58.1</td>
<td>Detailed Description</td>
<td>1178</td>
</tr>
<tr>
<td>9.59</td>
<td>hal_mfs.h File Reference</td>
<td>1178</td>
</tr>
<tr>
<td>9.59.1</td>
<td>Detailed Description</td>
<td>1180</td>
</tr>
<tr>
<td>9.60</td>
<td>hal_mii.h File Reference</td>
<td>1180</td>
</tr>
<tr>
<td>9.60.1</td>
<td>Detailed Description</td>
<td>1183</td>
</tr>
<tr>
<td>9.61</td>
<td>hal_mmc_spi.c File Reference</td>
<td>1183</td>
</tr>
<tr>
<td>9.61.1</td>
<td>Detailed Description</td>
<td>1184</td>
</tr>
<tr>
<td>9.62</td>
<td>hal_mmc_spi.h File Reference</td>
<td>1184</td>
</tr>
<tr>
<td>9.62.1</td>
<td>Detailed Description</td>
<td>1185</td>
</tr>
<tr>
<td>9.63</td>
<td>hal_mmcsd.c File Reference</td>
<td>1185</td>
</tr>
<tr>
<td>9.63.1</td>
<td>Detailed Description</td>
<td>1186</td>
</tr>
<tr>
<td>9.64</td>
<td>hal_mmcsd.h File Reference</td>
<td>1186</td>
</tr>
<tr>
<td>9.64.1</td>
<td>Detailed Description</td>
<td>1190</td>
</tr>
<tr>
<td>9.65</td>
<td>hal_objects.h File Reference</td>
<td>1190</td>
</tr>
<tr>
<td>9.65.1</td>
<td>Detailed Description</td>
<td>1191</td>
</tr>
<tr>
<td>9.66</td>
<td>hal_pal.c File Reference</td>
<td>1191</td>
</tr>
<tr>
<td>9.66.1</td>
<td>Detailed Description</td>
<td>1192</td>
</tr>
<tr>
<td>9.67</td>
<td>hal_pal.h File Reference</td>
<td>1192</td>
</tr>
<tr>
<td>9.67.1</td>
<td>Detailed Description</td>
<td>1195</td>
</tr>
<tr>
<td>9.68</td>
<td>hal_pal_ld.c File Reference</td>
<td>1195</td>
</tr>
<tr>
<td>9.68.1</td>
<td>Detailed Description</td>
<td>1196</td>
</tr>
<tr>
<td>9.69</td>
<td>hal_pal_ld.h File Reference</td>
<td>1196</td>
</tr>
</tbody>
</table>
9.90.1 Detailed Description ........................................... 1224
9.91 hal_serial_usb.c File Reference .............................. 1224
  9.91.1 Detailed Description ....................................... 1224
9.92 hal_serial_usb.h File Reference .............................. 1225
  9.92.1 Detailed Description ....................................... 1226
9.93 hal_sio.c File Reference ....................................... 1226
  9.93.1 Detailed Description ....................................... 1227
9.94 hal_sio.h File Reference ....................................... 1227
  9.94.1 Detailed Description ....................................... 1230
9.95 hal_sio_lld.c File Reference .................................. 1230
  9.95.1 Detailed Description ....................................... 1231
9.96 hal_sio_lld.h File Reference .................................. 1231
  9.96.1 Detailed Description ....................................... 1232
9.97 hal_spi.c File Reference ....................................... 1232
  9.97.1 Detailed Description ....................................... 1233
9.98 hal_spi.h File Reference ....................................... 1233
  9.98.1 Detailed Description ....................................... 1235
9.99 hal_spi_lld.c File Reference .................................. 1236
  9.99.1 Detailed Description ....................................... 1236
9.100 hal_spi_lld.h File Reference ................................ 1236
  9.100.1 Detailed Description ..................................... 1237
9.101 hal_st.c File Reference ...................................... 1237
  9.101.1 Detailed Description ..................................... 1238
9.102 hal_st.h File Reference ...................................... 1238
  9.102.1 Detailed Description ..................................... 1239
9.103 hal_st_lld.c File Reference ................................ 1239
  9.103.1 Detailed Description ..................................... 1239
9.104 hal_st_lld.h File Reference ................................ 1239
  9.104.1 Detailed Description ..................................... 1240
9.105 hal_streams.h File Reference ................................ 1240
  9.105.1 Detailed Description ..................................... 1241
9.106 hal_trng.c File Reference .................................... 1241
  9.106.1 Detailed Description ..................................... 1242
9.107 hal_trng.h File Reference .................................... 1242
  9.107.1 Detailed Description ..................................... 1243
9.108 hal_trng_lld.c File Reference ............................... 1243
  9.108.1 Detailed Description ..................................... 1243
9.109 hal_trng_lld.h File Reference ............................... 1244
  9.109.1 Detailed Description ..................................... 1244
9.110 hal_uart.c File Reference .................................... 1244
  9.110.1 Detailed Description ..................................... 1245
9.111 hal_uart.h File Reference .................................... 1245
9.131 nullstreams.h File Reference .......................................................... 1271
9.131.1 Detailed Description ................................................................. 1271
9.132 osal.c File Reference ................................................................. 1271
9.132.1 Detailed Description ................................................................. 1273
9.133 osal.h File Reference ................................................................. 1273
9.133.1 Detailed Description ................................................................. 1277

Index ......................................................................................... 1279
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 `sdGetWouldBlock` (SerialDriver *sdp)

 Global `sdPutWouldBlock` (SerialDriver *sdp)
### 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>868</td>
</tr>
<tr>
<td>Peripheral Interfaces</td>
<td>524</td>
</tr>
<tr>
<td>Generic Flash Interface</td>
<td>220</td>
</tr>
<tr>
<td>Configuration</td>
<td>871</td>
</tr>
<tr>
<td>Normal Drivers</td>
<td>886</td>
</tr>
<tr>
<td>HAL Driver</td>
<td>23</td>
</tr>
<tr>
<td>ADC Driver</td>
<td>32</td>
</tr>
<tr>
<td>CAN Driver</td>
<td>83</td>
</tr>
<tr>
<td>Crypto Driver</td>
<td>107</td>
</tr>
<tr>
<td>DAC Driver</td>
<td>184</td>
</tr>
<tr>
<td>EFL Driver</td>
<td>208</td>
</tr>
<tr>
<td>GPT Driver</td>
<td>231</td>
</tr>
<tr>
<td>I2C Driver</td>
<td>250</td>
</tr>
<tr>
<td>I2S Driver</td>
<td>267</td>
</tr>
<tr>
<td>ICU Driver</td>
<td>280</td>
</tr>
<tr>
<td>MAC Driver</td>
<td>362</td>
</tr>
<tr>
<td>PAL Driver</td>
<td>470</td>
</tr>
<tr>
<td>PWM Driver</td>
<td>525</td>
</tr>
<tr>
<td>RTC Driver</td>
<td>579</td>
</tr>
<tr>
<td>SDC Driver</td>
<td>592</td>
</tr>
<tr>
<td>Serial Driver</td>
<td>626</td>
</tr>
<tr>
<td>SIO Driver</td>
<td>658</td>
</tr>
<tr>
<td>SPI Driver</td>
<td>690</td>
</tr>
<tr>
<td>ST Driver</td>
<td>726</td>
</tr>
<tr>
<td>TRNG Driver</td>
<td>738</td>
</tr>
<tr>
<td>UART Driver</td>
<td>747</td>
</tr>
<tr>
<td>USB Driver</td>
<td>783</td>
</tr>
<tr>
<td>WDG Driver</td>
<td>834</td>
</tr>
<tr>
<td>WSPI Driver</td>
<td>841</td>
</tr>
<tr>
<td>Complex Drivers</td>
<td>888</td>
</tr>
<tr>
<td>Managed Flash Storage Driver</td>
<td>382</td>
</tr>
<tr>
<td>MMC over SPI Driver</td>
<td>419</td>
</tr>
<tr>
<td>Serial NOR Flash Driver</td>
<td>456</td>
</tr>
<tr>
<td>Serial over USB Driver</td>
<td>646</td>
</tr>
<tr>
<td>Interfaces and Classes</td>
<td>889</td>
</tr>
<tr>
<td>Module</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Blocks</td>
<td>304</td>
</tr>
<tr>
<td>Abstract I/O Block Device</td>
<td>353</td>
</tr>
<tr>
<td>Streams</td>
<td>305</td>
</tr>
<tr>
<td>Abstract Streams Interface</td>
<td>306</td>
</tr>
<tr>
<td>Abstract I/O Channel Interface</td>
<td>312</td>
</tr>
<tr>
<td>Abstract Files Interface</td>
<td>322</td>
</tr>
<tr>
<td>Abstract Persistent Storage Interface</td>
<td>330</td>
</tr>
<tr>
<td>Memory Streams Class</td>
<td>335</td>
</tr>
<tr>
<td>Null Streams Class</td>
<td>337</td>
</tr>
<tr>
<td>Buffered Streams Class</td>
<td>339</td>
</tr>
<tr>
<td>Output Formatter Utility</td>
<td>341</td>
</tr>
<tr>
<td>Input Formatter Utility</td>
<td>347</td>
</tr>
<tr>
<td>Base Object</td>
<td>468</td>
</tr>
<tr>
<td>Inner Code</td>
<td>890</td>
</tr>
<tr>
<td>I/O Buffers Queues</td>
<td>56</td>
</tr>
<tr>
<td>MMC/SD Block Device</td>
<td>443</td>
</tr>
<tr>
<td>I/O Bytes Queues</td>
<td>554</td>
</tr>
<tr>
<td>Support Code</td>
<td>891</td>
</tr>
<tr>
<td>MII/RMII Header</td>
<td>404</td>
</tr>
<tr>
<td>USB CDC Header</td>
<td>832</td>
</tr>
<tr>
<td>OSAL</td>
<td>892</td>
</tr>
</tbody>
</table>
Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

BaseBlockDeviceVMT .................................................. 933
  MMCSDBlockDeviceVMT ........................................... 1048
  MMCDriverVMT .................................................... 1044
  SDCDriverVMT .................................................... 1068
BaseFlashVMT .......................................................... 943
  EFlashDriverVMT .................................................. 965
  SNORDriverVMT ................................................... 1086
BaseObject ............................................................ 943
  BaseBlockDevice .................................................. 930
    MMCSDBlockDevice ............................................ 1045
    MMCDriver ...................................................... 1041
  BaseFlash .......................................................... 940
    hal_eff_driver .................................................. 988
    SNORDriver ..................................................... 1084
  BasePersistentStorage .......................................... 945
  BaseSequentialStream ............................................ 947
  BaseBufferedStream .............................................. 934
    NullStream ...................................................... 1049
  BaseChannel ........................................................ 937
    BaseAsynchronousChannel .................................... 927
      SerialDriver ................................................ 1070
      SerialUSBDriver .......................................... 1077
    FileStream ..................................................... 968
    RTCDriverVMT .................................................. 1063
    MemoryStream .................................................. 1026
BaseObjectVMT ........................................................ 945
BasePersistentStorageVMT .......................................... 947
BaseSequentialStreamVMT .......................................... 950
  BaseBufferedStreamVMT ......................................... 936
  BaseChannelVMT .................................................. 939
    BaseAsynchronousChannelVMT ................................ 929
      SerialDriverVMT ............................................ 1073
      SerialUSBDriverVMT ....................................... 1079
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sio_driver_vmt</td>
<td></td>
<td>1082</td>
</tr>
<tr>
<td>FileStreamVMT</td>
<td></td>
<td>970</td>
</tr>
<tr>
<td>BlockDeviceInfo</td>
<td></td>
<td>951</td>
</tr>
<tr>
<td>BufferedStreamAdapter</td>
<td></td>
<td>952</td>
</tr>
<tr>
<td>BufferedStreamAdapterVMT</td>
<td></td>
<td>953</td>
</tr>
<tr>
<td>CANConfig</td>
<td></td>
<td>953</td>
</tr>
<tr>
<td>CANDriver</td>
<td></td>
<td>954</td>
</tr>
<tr>
<td>CANRxFrame</td>
<td></td>
<td>957</td>
</tr>
<tr>
<td>CANTxFrame</td>
<td></td>
<td>960</td>
</tr>
<tr>
<td>cdc_linecoding_t</td>
<td></td>
<td>963</td>
</tr>
<tr>
<td>CRYConfig</td>
<td></td>
<td>963</td>
</tr>
<tr>
<td>CRYDriver</td>
<td></td>
<td>964</td>
</tr>
<tr>
<td>event_source</td>
<td></td>
<td>966</td>
</tr>
<tr>
<td>flash_descriptor_t</td>
<td></td>
<td>971</td>
</tr>
<tr>
<td>flash_sector_descriptor_t</td>
<td></td>
<td>973</td>
</tr>
<tr>
<td>GPTConfig</td>
<td></td>
<td>974</td>
</tr>
<tr>
<td>GPTDriver</td>
<td></td>
<td>976</td>
</tr>
<tr>
<td>hal_adc_config</td>
<td></td>
<td>977</td>
</tr>
<tr>
<td>hal_adc_configuration_group</td>
<td></td>
<td>977</td>
</tr>
<tr>
<td>hal_adc_driver</td>
<td></td>
<td>980</td>
</tr>
<tr>
<td>hal_dac_config</td>
<td></td>
<td>982</td>
</tr>
<tr>
<td>hal_dac_conversion_group</td>
<td></td>
<td>983</td>
</tr>
<tr>
<td>hal_dac_driver</td>
<td></td>
<td>984</td>
</tr>
<tr>
<td>hal_efi_config</td>
<td></td>
<td>987</td>
</tr>
<tr>
<td>hal_i2s_config</td>
<td></td>
<td>990</td>
</tr>
<tr>
<td>hal_i2s_driver</td>
<td></td>
<td>992</td>
</tr>
<tr>
<td>hal_sio_config</td>
<td></td>
<td>993</td>
</tr>
<tr>
<td>hal_sio_driver</td>
<td></td>
<td>993</td>
</tr>
<tr>
<td>hal_sio_operation</td>
<td></td>
<td>996</td>
</tr>
<tr>
<td>hal_spi_config</td>
<td></td>
<td>998</td>
</tr>
<tr>
<td>hal_spi_driver</td>
<td></td>
<td>1001</td>
</tr>
<tr>
<td>hal_trng_config</td>
<td></td>
<td>1002</td>
</tr>
<tr>
<td>hal_trng_driver</td>
<td></td>
<td>1003</td>
</tr>
<tr>
<td>hal_wspi_config</td>
<td></td>
<td>1004</td>
</tr>
<tr>
<td>hal_wspi_driver</td>
<td></td>
<td>1005</td>
</tr>
<tr>
<td>halclkcfg_t</td>
<td></td>
<td>1007</td>
</tr>
<tr>
<td>HMACSHA256Context</td>
<td></td>
<td>1007</td>
</tr>
<tr>
<td>HMACSHA512Context</td>
<td></td>
<td>1008</td>
</tr>
<tr>
<td>l2CConfig</td>
<td></td>
<td>1008</td>
</tr>
<tr>
<td>l2CDriver</td>
<td></td>
<td>1009</td>
</tr>
<tr>
<td>ICUConfig</td>
<td></td>
<td>1010</td>
</tr>
<tr>
<td>ICUDriver</td>
<td></td>
<td>1012</td>
</tr>
<tr>
<td>io_buffers_queue</td>
<td></td>
<td>1013</td>
</tr>
<tr>
<td>io_queue</td>
<td></td>
<td>1018</td>
</tr>
<tr>
<td>IOBus</td>
<td></td>
<td>1020</td>
</tr>
<tr>
<td>MACConfig</td>
<td></td>
<td>1022</td>
</tr>
<tr>
<td>MACDriver</td>
<td></td>
<td>1023</td>
</tr>
<tr>
<td>MACReceiveDescriptor</td>
<td></td>
<td>1024</td>
</tr>
<tr>
<td>MACTransmitDescriptor</td>
<td></td>
<td>1025</td>
</tr>
<tr>
<td>MemStreamVMT</td>
<td></td>
<td>1029</td>
</tr>
<tr>
<td>mls_bank_header_t</td>
<td></td>
<td>1029</td>
</tr>
<tr>
<td>mls_data_header_t</td>
<td></td>
<td>1031</td>
</tr>
<tr>
<td>mls_transaction_op_t</td>
<td></td>
<td>1032</td>
</tr>
<tr>
<td>MFSConfig</td>
<td></td>
<td>1034</td>
</tr>
<tr>
<td>MFSDriver</td>
<td></td>
<td>1036</td>
</tr>
<tr>
<td>MMCCConfig</td>
<td></td>
<td>1040</td>
</tr>
<tr>
<td>NullStreamVMT</td>
<td></td>
<td>1051</td>
</tr>
<tr>
<td>PALConfig</td>
<td></td>
<td>1051</td>
</tr>
</tbody>
</table>
4.1 Class Hierarchy

- palevent_t .......................................................... 1052
- PWMChannelConfig .................................................. 1053
- PWMConfig .......................................................... 1055
- PWMDriver ........................................................... 1057
- RTCAlarm ............................................................. 1059
- RTCDateTime .......................................................... 1059
- RTCDriver .............................................................. 1061
- SDCConfig ............................................................ 1064
- SDCDriver ............................................................. 1065
- SerialConfig .......................................................... 1069
- SerialUSBConfig ....................................................... 1074
- SHA1Context ........................................................... 1080
- SHA256Context ........................................................ 1081
- SHA512Context ........................................................ 1081
- SNORConfig ............................................................ 1083
- threads_queue_t ......................................................... 1087
- UARTConfig ............................................................. 1088
- UARTDriver ............................................................ 1090
- unpacked_mmc_cid_t .................................................. 1092
- unpacked_mmc_csd_1 .................................................. 1093
- unpacked_sdc_cid_t ................................................... 1093
- unpacked_sdc_csd_10_t ............................................... 1094
- unpacked_sdc_csd_20_t ............................................... 1095
- USBConfig ............................................................. 1095
- USBDriver .............................................................. 1098
- USBDescriptor .......................................................... 1099
- USBEndpointConfig .................................................... 1103
- USBIPEndpointState .................................................... 1107
- USBOutEndpointState ................................................... 1108
- WDGConfig ............................................................. 1110
- WDGDriver ............................................................. 1110
- wspi_command_t ......................................................... 1111
Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

- **BaseAsynchronousChannel**
  - Base asynchronous channel class ............................................. 927
- **BaseAsynchronousChannelVMT**
  - BaseAsynchronousChannel virtual methods table ............................ 929
- **BaseBlockDevice**
  - Base block device class ............................................................ 930
- **BaseBlockDeviceVMT**
  - BaseBlockDevice virtual methods table ......................................... 933
- **BaseBufferedStream**
  - Buffered stream class ............................................................... 934
- **BaseBufferedStreamVMT**
  - BaseBufferedStream virtual methods table .................................... 936
- **BaseChannel**
  - Base channel class ................................................................. 937
- **BaseChannelVMT**
  - BaseChannel virtual methods table ............................................ 939
- **BaseFlash**
  - Base flash class ..................................................................... 940
- **BaseFlashVMT**
  - BaseFlash virtual methods table ................................................ 943
- **BaseObject**
  - Base stream class ................................................................. 943
- **BaseObjectVMT**
  - BaseObject virtual methods table .............................................. 945
- **BasePersistentStorage**
  - Base persistent storage class ..................................................... 945
- **BasePersistentStorageVMT**
  - BasePersistentStorage virtual methods table ................................. 947
- **BaseSequentialStream**
  - Base stream class ................................................................. 947
- **BaseSequentialStreamVMT**
  - BaseSequentialStream virtual methods table .................................. 950
- **BlockDeviceInfo**
  - Block device info .................................................................. 951
- **BufferedStreamAdapter**
  - Buffered stream adapter object ................................................ 952
<table>
<thead>
<tr>
<th>Structure/Type</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferedStreamAdapterVMT</td>
<td>BufferedStreamAdapter virtual methods table</td>
<td>953</td>
</tr>
<tr>
<td>CANConfig</td>
<td>Driver configuration structure</td>
<td>953</td>
</tr>
<tr>
<td>CANDriver</td>
<td>Structure representing an CAN driver</td>
<td>954</td>
</tr>
<tr>
<td>CANRxFrame</td>
<td>CAN received frame</td>
<td>957</td>
</tr>
<tr>
<td>CANTxFrame</td>
<td>CAN transmission frame</td>
<td>957</td>
</tr>
<tr>
<td>cdc_linecoding_t</td>
<td>Type of Line Coding structure</td>
<td>960</td>
</tr>
<tr>
<td>CRYConfig</td>
<td>Driver configuration structure</td>
<td>963</td>
</tr>
<tr>
<td>CRYDriver</td>
<td>Structure representing a CRY driver</td>
<td>964</td>
</tr>
<tr>
<td>EFFlashDriverVMT</td>
<td>EFFlash virtual methods table</td>
<td>965</td>
</tr>
<tr>
<td>event_source</td>
<td>Events source object</td>
<td>966</td>
</tr>
<tr>
<td>FileStream</td>
<td>Base file stream class</td>
<td>968</td>
</tr>
<tr>
<td>FileStreamVMT</td>
<td>FileStream virtual methods table</td>
<td>970</td>
</tr>
<tr>
<td>flash_descriptor_t</td>
<td>Type of a flash device descriptor</td>
<td>971</td>
</tr>
<tr>
<td>flash_sector_descriptor_t</td>
<td>Flash sector descriptor</td>
<td>973</td>
</tr>
<tr>
<td>GPTConfig</td>
<td>Driver configuration structure</td>
<td>974</td>
</tr>
<tr>
<td>GPTDriver</td>
<td>Structure representing a GPT driver</td>
<td>976</td>
</tr>
<tr>
<td>hal_adc_config</td>
<td>Driver configuration structure</td>
<td>977</td>
</tr>
<tr>
<td>hal_adc_configuration_group</td>
<td>Conversion group configuration structure</td>
<td>977</td>
</tr>
<tr>
<td>hal_adc_driver</td>
<td>Structure representing an ADC driver</td>
<td>980</td>
</tr>
<tr>
<td>hal_dac_config</td>
<td>Driver configuration structure</td>
<td>982</td>
</tr>
<tr>
<td>hal_dac_conversion_group</td>
<td>DAC Conversion group structure</td>
<td>983</td>
</tr>
<tr>
<td>hal_dac_driver</td>
<td>Structure representing a DAC driver</td>
<td>984</td>
</tr>
<tr>
<td>hal_efi_config</td>
<td>Type of a structure representing a flash driver config</td>
<td>987</td>
</tr>
<tr>
<td>hal_efi_driver</td>
<td>Structure representing an embedded flash driver</td>
<td>988</td>
</tr>
<tr>
<td>hal_i2s_config</td>
<td>Driver configuration structure</td>
<td>990</td>
</tr>
<tr>
<td>hal_i2s_driver</td>
<td>Structure representing an I2S driver</td>
<td>992</td>
</tr>
<tr>
<td>hal_sio_config</td>
<td>Driver configuration structure</td>
<td>993</td>
</tr>
<tr>
<td>hal_sio_driver</td>
<td>Structure representing a SIO driver</td>
<td>993</td>
</tr>
<tr>
<td>hal_sio_operation</td>
<td>Structure representing a SIO operation</td>
<td>996</td>
</tr>
</tbody>
</table>
### 5.1 Data Structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>hal_spi_config</td>
<td>Driver configuration structure</td>
<td>998</td>
</tr>
<tr>
<td>hal_spi_driver</td>
<td>Structure representing an SPI driver</td>
<td>1001</td>
</tr>
<tr>
<td>hal_trng_config</td>
<td>Driver configuration structure</td>
<td>1002</td>
</tr>
<tr>
<td>hal_trng_driver</td>
<td>Structure representing a TRNG driver</td>
<td>1003</td>
</tr>
<tr>
<td>hal_wspi_config</td>
<td>Driver configuration structure</td>
<td>1004</td>
</tr>
<tr>
<td>hal_wspi_driver</td>
<td>Structure representing a WSPI driver</td>
<td>1005</td>
</tr>
<tr>
<td>halclkcfg_t</td>
<td>Type of a clock configuration structure</td>
<td>1007</td>
</tr>
<tr>
<td>HMACSHA256Context</td>
<td>Type of a HMAC_SHA256 context</td>
<td>1007</td>
</tr>
<tr>
<td>HMACSHA512Context</td>
<td>Type of a HMAC_SHA512 context</td>
<td>1008</td>
</tr>
<tr>
<td>I2CConfig</td>
<td>Type of I2C driver configuration structure</td>
<td>1008</td>
</tr>
<tr>
<td>I2CDriver</td>
<td>Structure representing an I2C driver</td>
<td>1009</td>
</tr>
<tr>
<td>ICUConfig</td>
<td>Driver configuration structure</td>
<td>1010</td>
</tr>
<tr>
<td>ICUDriver</td>
<td>Structure representing an ICU driver</td>
<td>1012</td>
</tr>
<tr>
<td>io_buffers_queue</td>
<td>Structure of a generic buffers queue</td>
<td>1013</td>
</tr>
<tr>
<td>io_queue</td>
<td>Generic I/O queue structure</td>
<td>1018</td>
</tr>
<tr>
<td>IOBus</td>
<td>I/O bus descriptor</td>
<td>1020</td>
</tr>
<tr>
<td>MACConfig</td>
<td>Driver configuration structure</td>
<td>1022</td>
</tr>
<tr>
<td>MACDriver</td>
<td>Structure representing a MAC driver</td>
<td>1023</td>
</tr>
<tr>
<td>MACReceiveDescriptor</td>
<td>Structure representing a receive descriptor</td>
<td>1024</td>
</tr>
<tr>
<td>MACTransmitDescriptor</td>
<td>Structure representing a transmit descriptor</td>
<td>1025</td>
</tr>
<tr>
<td>MemoryStream</td>
<td>Memory stream object</td>
<td>1026</td>
</tr>
<tr>
<td>MemStreamVMT</td>
<td>MemStream virtual methods table</td>
<td>1029</td>
</tr>
<tr>
<td>mfs_bank_header_t</td>
<td>Type of a bank header</td>
<td>1029</td>
</tr>
<tr>
<td>mfs_data_header_t</td>
<td>Type of a data block header</td>
<td>1031</td>
</tr>
<tr>
<td>mfs_transaction_op_t</td>
<td>Type of a buffered write/erase operation within a transaction</td>
<td>1032</td>
</tr>
<tr>
<td>MFSConfig</td>
<td>Type of a MFS configuration structure</td>
<td>1034</td>
</tr>
<tr>
<td>MFSDriver</td>
<td>Type of an MFS instance</td>
<td>1036</td>
</tr>
<tr>
<td>MMCConfig</td>
<td>MMC/SD over SPI driver configuration structure</td>
<td>1040</td>
</tr>
<tr>
<td>MMCDriver</td>
<td>Structure representing a MMC/SD over SPI driver</td>
<td>1041</td>
</tr>
</tbody>
</table>
MMCDriverVMT  MMCDriver virtual methods table  1044
MMCSDBlockDevice  MCC/SD block device class  1045
MMCSDBlockDeviceVMT  MMCSDBlockDevice virtual methods table  1048
NullStream  Null stream object  1049
NullStreamVMT  NullStream virtual methods table  1051
PALConfig  Generic I/O ports static initializer  1051
palevent_t  Type of a PAL event record  1052
PWMChannelConfig  Type of a PWM driver channel configuration structure  1053
PWMConfig  Type of a PWM driver configuration structure  1055
PWMDriver  Structure representing a PWM driver  1057
RTCAlarm  Type of a structure representing an RTC alarm time stamp  1059
RTCDateTime  Type of a structure representing an RTC date/time stamp  1059
RTCDriver  Structure representing an RTC driver  1061
RTCDriverVMT  RTCDriver virtual methods table  1063
SDCConfig  Driver configuration structure  1064
SDCDriver  Structure representing an SDC driver  1065
SDCDriverVMT  SCDriver virtual methods table  1068
SerialConfig  PLATFORM Serial Driver configuration structure  1069
SerialDriver  Full duplex serial driver class  1070
SerialDriverVMT  SerialDriver virtual methods table  1073
SerialUSBConfig  Serial over USB Driver configuration structure  1074
SerialUSBDriver  Full duplex serial driver class  1077
SerialUSBDriverVMT  SerialDriver virtual methods table  1079
SHA1Context  Type of a SHA1 context  1080
SHA256Context  Type of a SHA256 context  1081
SHA512Context  Type of a SHA512 context  1081
sio_driver_vmt  SIODriver virtual methods table  1082
SNORConfig  Type of a SNOR configuration structure  1083
SNORDriver  Type of SNOR flash class  1084
<table>
<thead>
<tr>
<th>Class/Structure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNORDriverVMT</td>
<td>SNOR virtual methods table</td>
<td>1086</td>
</tr>
<tr>
<td>threads_queue_t</td>
<td>Type of a thread queue</td>
<td>1087</td>
</tr>
<tr>
<td>UARTConfig</td>
<td>Driver configuration structure</td>
<td>1088</td>
</tr>
<tr>
<td>UARTDriver</td>
<td>Structure representing an UART driver</td>
<td>1090</td>
</tr>
<tr>
<td>unpacked_mmc_cid_t</td>
<td>Unpacked CID register from MMC</td>
<td>1092</td>
</tr>
<tr>
<td>unpacked_mmc_csd_t</td>
<td>Unpacked CSD register from MMC</td>
<td>1093</td>
</tr>
<tr>
<td>unpacked_sdc_cid_t</td>
<td>Unpacked CID register from SDC</td>
<td>1093</td>
</tr>
<tr>
<td>unpacked_sdc_csd_10_t</td>
<td>Unpacked CSD v1.0 register from SDC</td>
<td>1094</td>
</tr>
<tr>
<td>unpacked_sdc_csd_20_t</td>
<td>Unpacked CSD v2.0 register from SDC</td>
<td>1095</td>
</tr>
<tr>
<td>USBConfig</td>
<td>Type of an USB driver configuration structure</td>
<td>1095</td>
</tr>
<tr>
<td>USBDescriptor</td>
<td>Type of an USB descriptor</td>
<td>1098</td>
</tr>
<tr>
<td>USBDriver</td>
<td>Structure representing an USB driver</td>
<td>1099</td>
</tr>
<tr>
<td>USBEndpointConfig</td>
<td>Type of an USB endpoint configuration structure</td>
<td>1103</td>
</tr>
<tr>
<td>USBInEndpointState</td>
<td>Type of an IN endpoint state structure</td>
<td>1107</td>
</tr>
<tr>
<td>USBOutEndpointState</td>
<td>Type of an OUT endpoint state structure</td>
<td>1108</td>
</tr>
<tr>
<td>WDConfig</td>
<td>Driver configuration structure</td>
<td>1110</td>
</tr>
<tr>
<td>WDGDriver</td>
<td>Structure representing an WDG driver</td>
<td>1110</td>
</tr>
<tr>
<td>wspi_command_t</td>
<td>Type of a WSPI command descriptor</td>
<td>1111</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
- **bufstreams.h**: Buffered streams structures and macros
- **chprintf.c**: Mini printf-like functionality
- **chprintf.h**: Mini printf-like functionality
- **chscanf.c**: Mini scanf-like functionality
- **chscanf.h**: Mini scanf-like functionality
- **hal.c**: HAL subsystem code
- **hal.h**: HAL subsystem header
- **hal_adc.c**: ADC Driver code
- **hal_adc.h**: ADC Driver macros and structures
- **hal_adc_lld.c**: PLATFORM ADC subsystem low level driver source
- **hal_adc_lld.h**: PLATFORM ADC subsystem low level driver header
- **hal_buffers.c**: I/O Buffers code
- **hal_buffers.h**: I/O Buffers macros and structures
- **hal_can.c**: CAN Driver code
- **hal_can.h**: CAN Driver macros and structures
- **hal_can_lld.c**: PLATFORM CAN subsystem low level driver source
- **hal_can_lld.h**: PLATFORM CAN subsystem low level driver header
<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>hal_channels.h</td>
<td>I/O channels access</td>
<td>1131</td>
</tr>
<tr>
<td>hal_crypto.c</td>
<td>Cryptographic Driver code</td>
<td>1132</td>
</tr>
<tr>
<td>hal_crypto.h</td>
<td>Cryptographic Driver macros and structures</td>
<td>1134</td>
</tr>
<tr>
<td>hal_crypto_lld.c</td>
<td>PLATFORM cryptographic subsystem low level driver source</td>
<td>1138</td>
</tr>
<tr>
<td>hal_crypto_lld.h</td>
<td>PLATFORM cryptographic subsystem low level driver header</td>
<td>1140</td>
</tr>
<tr>
<td>hal_dac.c</td>
<td>DAC Driver code</td>
<td>1143</td>
</tr>
<tr>
<td>hal_dac.h</td>
<td>DAC Driver macros and structures</td>
<td>1144</td>
</tr>
<tr>
<td>hal_dac_lld.c</td>
<td>PLATFORM DAC subsystem low level driver source</td>
<td>1146</td>
</tr>
<tr>
<td>hal_dac_lld.h</td>
<td>PLATFORM DAC subsystem low level driver header</td>
<td>1147</td>
</tr>
<tr>
<td>hal_efl.c</td>
<td>Embedded Flash Driver code</td>
<td>1148</td>
</tr>
<tr>
<td>hal_efl.h</td>
<td>Embedded Flash Driver macros and structures</td>
<td>1148</td>
</tr>
<tr>
<td>hal_efl_lld.c</td>
<td>PLATFORM Embedded Flash subsystem low level driver source</td>
<td>1149</td>
</tr>
<tr>
<td>hal_efl_lld.h</td>
<td>PLATFORM Embedded Flash subsystem low level driver header</td>
<td>1150</td>
</tr>
<tr>
<td>hal_files.h</td>
<td>Data files</td>
<td>1151</td>
</tr>
<tr>
<td>hal_flash.c</td>
<td>Generic flash driver class code</td>
<td>1152</td>
</tr>
<tr>
<td>hal_flash.h</td>
<td>Generic flash driver class header</td>
<td>1153</td>
</tr>
<tr>
<td>hal_gpt.c</td>
<td>GPT Driver code</td>
<td>1155</td>
</tr>
<tr>
<td>hal_gpt.h</td>
<td>GPT Driver macros and structures</td>
<td>1155</td>
</tr>
<tr>
<td>hal_gpt_lld.c</td>
<td>PLATFORM GPT subsystem low level driver source</td>
<td>1157</td>
</tr>
<tr>
<td>hal_gpt_lld.h</td>
<td>PLATFORM GPT subsystem low level driver header</td>
<td>1157</td>
</tr>
<tr>
<td>hal_i2c.c</td>
<td>I2C Driver code</td>
<td>1158</td>
</tr>
<tr>
<td>hal_i2c.h</td>
<td>I2C Driver macros and structures</td>
<td>1159</td>
</tr>
<tr>
<td>hal_i2c_lld.c</td>
<td>PLATFORM I2C subsystem low level driver source</td>
<td>1161</td>
</tr>
<tr>
<td>hal_i2c_lld.h</td>
<td>PLATFORM I2C subsystem low level driver header</td>
<td>1161</td>
</tr>
<tr>
<td>hal_i2s.c</td>
<td>I2S Driver code</td>
<td>1162</td>
</tr>
<tr>
<td>hal_i2s.h</td>
<td>I2S Driver macros and structures</td>
<td>1163</td>
</tr>
<tr>
<td>hal_i2s_lld.c</td>
<td>PLATFORM I2S subsystem low level driver source</td>
<td>1164</td>
</tr>
<tr>
<td>hal_i2s_lld.h</td>
<td>PLATFORM I2S subsystem low level driver header</td>
<td>1165</td>
</tr>
<tr>
<td>hal_icu.c</td>
<td>ICU Driver code</td>
<td>1166</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.h</td>
<td>ICU Driver macros and structures</td>
<td>1166</td>
</tr>
<tr>
<td>hal_icu_lld.c</td>
<td>PLATFORM ADC subsystem low level driver source</td>
<td>1168</td>
</tr>
<tr>
<td>hal_icu_lld.h</td>
<td>PLATFORM ICU subsystem low level driver header</td>
<td>1169</td>
</tr>
<tr>
<td>hal_ioblock.h</td>
<td>I/O block devices access</td>
<td>1170</td>
</tr>
<tr>
<td>hal_lld.c</td>
<td>PLATFORM HAL subsystem low level driver source</td>
<td>1171</td>
</tr>
<tr>
<td>hal_lld.h</td>
<td>PLATFORM HAL subsystem low level driver header</td>
<td>1172</td>
</tr>
<tr>
<td>hal_mac.c</td>
<td>MAC Driver code</td>
<td>1173</td>
</tr>
<tr>
<td>hal_mac.h</td>
<td>MAC Driver macros and structures</td>
<td>1173</td>
</tr>
<tr>
<td>hal_mac_lld.c</td>
<td>PLATFORM MAC subsystem low level driver source</td>
<td>1175</td>
</tr>
<tr>
<td>hal_mac_lld.h</td>
<td>PLATFORM MAC subsystem low level driver header</td>
<td>1176</td>
</tr>
<tr>
<td>hal_mfs.c</td>
<td>Managed Flash Storage module code</td>
<td>1177</td>
</tr>
<tr>
<td>hal_mfs.h</td>
<td>Managed Flash Storage module header</td>
<td>1178</td>
</tr>
<tr>
<td>hal_mii.h</td>
<td>MII macros and structures</td>
<td>1180</td>
</tr>
<tr>
<td>hal_mmc_spi.c</td>
<td>MMC over SPI driver code</td>
<td>1183</td>
</tr>
<tr>
<td>hal_mmc_spi.h</td>
<td>MMC over SPI driver header</td>
<td>1184</td>
</tr>
<tr>
<td>hal_mmcsd.c</td>
<td>MMC/SD cards common code</td>
<td>1185</td>
</tr>
<tr>
<td>hal_mmcsd.h</td>
<td>MMC/SD cards common header</td>
<td>1186</td>
</tr>
<tr>
<td>hal_objects.h</td>
<td>Base object</td>
<td>1190</td>
</tr>
<tr>
<td>hal_pal.c</td>
<td>I/O Ports Abstraction Layer code</td>
<td>1191</td>
</tr>
<tr>
<td>hal_pal.h</td>
<td>I/O Ports Abstraction Layer macros, types and structures</td>
<td>1192</td>
</tr>
<tr>
<td>hal_pal_lld.c</td>
<td>PLATFORM PAL subsystem low level driver source</td>
<td>1195</td>
</tr>
<tr>
<td>hal_pal_lld.h</td>
<td>PLATFORM PAL subsystem low level driver header</td>
<td>1196</td>
</tr>
<tr>
<td>hal_persistent.h</td>
<td>Generic persistent storage class header</td>
<td>1198</td>
</tr>
<tr>
<td>hal_pwm.c</td>
<td>PWM Driver code</td>
<td>1199</td>
</tr>
<tr>
<td>hal_pwm.h</td>
<td>PWM Driver macros and structures</td>
<td>1199</td>
</tr>
<tr>
<td>hal_pwm_lld.c</td>
<td>PLATFORM PWM subsystem low level driver source</td>
<td>1201</td>
</tr>
<tr>
<td>hal_pwm_lld.h</td>
<td>PLATFORM PWM subsystem low level driver header</td>
<td>1202</td>
</tr>
<tr>
<td>hal_queues.c</td>
<td>I/O Queues code</td>
<td>1204</td>
</tr>
<tr>
<td>hal_queues.h</td>
<td>I/O Queues macros and structures</td>
<td>1205</td>
</tr>
</tbody>
</table>
hal_rtc.c
RTC Driver code .......................................................... 1207
hal_rtc.h
RTC Driver macros and structures ........................................ 1207
hal_rtc_lld.c
PLATFORM RTC subsystem low level driver source ....................... 1209
hal_rtc_lld.h
PLATFORM RTC subsystem low level driver header ...................... 1209
hal_sdc.c
SDC Driver code ............................................................ 1211
hal_sdc.h
SDC Driver macros and structures ........................................ 1212
hal_sdc_lld.c
PLATFORM SDC subsystem low level driver source ....................... 1214
hal_sdc_lld.h
PLATFORM SDC subsystem low level driver header ...................... 1215
hal_serial.c
Serial Driver code .......................................................... 1217
hal_serial.h
Serial Driver macros and structures ..................................... 1218
hal_serial_lld.c
PLATFORM serial subsystem low level driver source ................... 1220
hal_serial_lld.h
PLATFORM serial subsystem low level driver header .................. 1220
hal_serial_nor.c
Serial NOR serial flash driver code ..................................... 1221
hal_serial_nor.h
Serial NOR driver header ................................................ 1222
hal_serial_usb.c
Serial over USB Driver code .............................................. 1224
hal_serial_usb.h
Serial over USB Driver macros and structures ......................... 1225
hal_sio.c
SIO Driver code ............................................................ 1226
hal_sio.h
SIO Driver macros and structures ........................................ 1227
hal_sio_lld.c
PLATFORM SIO subsystem low level driver source ....................... 1230
hal_sio_lld.h
PLATFORM SIO subsystem low level driver header ...................... 1231
hal_spi.c
SPI Driver code ............................................................ 1232
hal_spi.h
SPI Driver macros and structures ........................................ 1233
hal_spi_lld.c
PLATFORM SPI subsystem low level driver source ....................... 1236
hal_spi_lld.h
PLATFORM SPI subsystem low level driver header ...................... 1236
hal_st.c
ST Driver code ............................................................. 1237
hal_st.h
ST Driver macros and structures ........................................ 1238
hal_st_lld.c
PLATFORM ST subsystem low level driver source ....................... 1239
hal_st_lld.h
PLATFORM ST subsystem low level driver header ...................... 1239
hal_streams.h
Data streams ............................................................... 1240
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>hal_trng.c</td>
<td>TRNG Driver code</td>
<td>1241</td>
</tr>
<tr>
<td>hal_trng.h</td>
<td>TRNG Driver macros and structures</td>
<td>1242</td>
</tr>
<tr>
<td>hal_trng_lld.c</td>
<td>PLATFORM TRNG subsystem low level driver source</td>
<td>1243</td>
</tr>
<tr>
<td>hal_trng_lld.h</td>
<td>PLATFORM TRNG subsystem low level driver header</td>
<td>1244</td>
</tr>
<tr>
<td>hal_uart.c</td>
<td>UART Driver code</td>
<td>1244</td>
</tr>
<tr>
<td>hal_uart.h</td>
<td>UART Driver macros and structures</td>
<td>1245</td>
</tr>
<tr>
<td>hal_uart_lld.c</td>
<td>PLATFORM UART subsystem low level driver source</td>
<td>1248</td>
</tr>
<tr>
<td>hal_uart_lld.h</td>
<td>PLATFORM UART subsystem low level driver header</td>
<td>1248</td>
</tr>
<tr>
<td>hal_usb.c</td>
<td>USB Driver code</td>
<td>1249</td>
</tr>
<tr>
<td>hal_usb.h</td>
<td>USB Driver macros and structures</td>
<td>1251</td>
</tr>
<tr>
<td>hal_usb_cdc.h</td>
<td>USB CDC macros and structures</td>
<td>1254</td>
</tr>
<tr>
<td>hal_usb_lld.c</td>
<td>PLATFORM USB subsystem low level driver source</td>
<td>1255</td>
</tr>
<tr>
<td>hal_usb_lld.h</td>
<td>PLATFORM USB subsystem low level driver header</td>
<td>1257</td>
</tr>
<tr>
<td>hal_wdg.c</td>
<td>WDG Driver code</td>
<td>1259</td>
</tr>
<tr>
<td>hal_wdg.h</td>
<td>WDG Driver macros and structures</td>
<td>1259</td>
</tr>
<tr>
<td>hal_wdg_lld.c</td>
<td>WDG Driver subsystem low level driver source</td>
<td>1260</td>
</tr>
<tr>
<td>hal_wdg_lld.h</td>
<td>WDG Driver subsystem low level driver header</td>
<td>1261</td>
</tr>
<tr>
<td>hal_wspi.c</td>
<td>WSPI Driver code</td>
<td>1261</td>
</tr>
<tr>
<td>hal_wspi.h</td>
<td>WSPI Driver macros and structures</td>
<td>1262</td>
</tr>
<tr>
<td>hal_wspi_lld.c</td>
<td>PLATFORM WSPI subsystem low level driver source</td>
<td>1265</td>
</tr>
<tr>
<td>hal_wspi_lld.h</td>
<td>PLATFORM WSPI subsystem low level driver header</td>
<td>1266</td>
</tr>
<tr>
<td>halconf.h</td>
<td>HAL configuration header</td>
<td>1267</td>
</tr>
<tr>
<td>memstreams.c</td>
<td>Memory streams code</td>
<td>1269</td>
</tr>
<tr>
<td>memstreams.h</td>
<td>Memory streams structures and macros</td>
<td>1270</td>
</tr>
<tr>
<td>nullstreams.c</td>
<td>Null streams code</td>
<td>1270</td>
</tr>
<tr>
<td>nullstreams.h</td>
<td>Null streams structures and macros</td>
<td>1271</td>
</tr>
<tr>
<td>osal.c</td>
<td>OSAL module code</td>
<td>1271</td>
</tr>
<tr>
<td>osal.h</td>
<td>OSAL module header</td>
<td>1273</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 not 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 0`
  Stable release flag.
- `#define HAL_LLD_USE_CLOCK_MANAGEMENT`
  Specifies implementation of dynamic clock management.

ChibiOS/HAL version identification

- `#define HAL_VERSION "7.2.0"`
  HAL version string.
- `#define CH_HAL_MAJOR 7`
  HAL version major number.
- `#define CH_HAL_MINOR 2`
  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.

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.

7.1.2 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 0

Stable release flag.

7.1.2.3 HAL_VERSION

#define HAL_VERSION "7.2.0"

HAL version string.

7.1.2.4 CH_HAL_MAJOR

#define CH_HAL_MAJOR 7

HAL version major number.

7.1.2.5 CH_HAL_MINOR

#define CH_HAL_MINOR 2

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.
7.1.2.8 **HAL_FAILED**

```c
#define HAL_FAILED true
```

HAL operation failed.

7.1.2.9 **HAL_LLD_USE_CLOCK_MANAGEMENT**

```c
#define HAL_LLD_USE_CLOCK_MANAGEMENT
```

Specifies implementation of dynamic clock management.

7.1.3 **Typedef Documentation**

7.1.3.1 **halclkpt_t**

```c
typedef unsigned halclkpt_t
```

Type of a clock point identifier.

7.1.3.2 **halfreq_t**

```c
typedef uint32_t halfreq_t
```

Type of a clock point frequency in Hz.

7.1.4 **Function Documentation**
7.1 HAL Driver

7.1.4.1 halInit()

```c
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()

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:

```
halClockSwitchMode    hal_lld_clock_switch_mode
```

7.1.4.3 halClockGetPointX()

```
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 |

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:

![Call Graph](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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>ccp</code> pointer to clock a <code>halclkcfg_t</code> structure</td>
</tr>
</tbody>
</table>

### Returns

The clock switch result.

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td>if the clock switch succeeded</td>
</tr>
<tr>
<td><code>true</code></td>
<td>if the clock switch failed</td>
</tr>
</tbody>
</table>
7.1.4.6  hal_lld_get_clock_point()

```c
halfreq_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 ADC Driver

7.2.3.1 ADC Conversion Groups

The ADCConversionGroup is the object 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 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.

- `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.
7.2 ADC Driver

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 a ADCDriver structure.
- void 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(adc) ((bool)((adc)->state == ADC_COMPLETE))
```

Buffer state.

**Note**

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

**Parameters**

- **in adc** pointer to the ADCDriver object
7.2 ADC Driver

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.

7.2.4.4  \_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`

```c
#define _adc_wakeup_isr(
    adcp
)

Value:
{
    \  
    osalSysLockFromISR();
    osalThreadResumeI(&(adcp)->thread, MSG_OK);
    osalSysUnlockFromISR();
}
```

Wakes up the waiting thread.

**Parameters**

| in | adcp | pointer to the ADCDriver object |

**Function Class:**

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

### 7.2.4.7 `_adc_timeout_isr`

```c
#define _adc_timeout_isr(
    adcp
)

Value:
{
    \  
    osalSysLockFromISR();
    osalThreadResumeI(&(adcp)->thread, MSG_TIMEOUT);
    osalSysUnlockFromISR();
}
```

Wakes up the waiting thread with a timeout message.

**Parameters**

| in | adcp | 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`

```c
#define _adc_isr_half_code(
    adcp
)
```

---

*ChibiOS/HAL*
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 adcp 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

#define _adc_isr_full_code(

ado

Value:

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;
            (adcp)->grpp = NULL;
        }
    }
    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:
7.2.4.10  _adc_isr_error_code

#define _adc_isr_error_code(
    adcp,
    err 
)

Value:

    adc_lld_stop_conversion(adcp);
    \   
    if ((adcp)->grpp->error_cb != NULL) {
        (adcp)->state = ADC_ERROR;
        (adcp)->grpp->error_cb(adcp, err);
    } 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

in  adcp  pointer to the ADCDriver object

Function Class:

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

Note

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

<table>
<thead>
<tr>
<th>in</th>
<th>adcp</th>
<th>pointer to the ADCDriver object</th>
</tr>
</thead>
<tbody>
<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.

7.2.4.11 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.
7.2.4.15 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

| in | adcp | pointer to the ADCDriver object triggering the callback |

7.2.5.5 adcerrorcallback_t

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

Type of an ADC error callback.

Parameters

| in | adcp | pointer to the ADCDriver object triggering the callback |
|    | err  | ADC error code |

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.

7.2.5.8 adceerror_t

typedef uint32_t adceerror_t

Type of an ADC error mask.

7.2.6 Enumeration Type Documentation

7.2.6.1 adcstate_t

enum adcstate_t

Driver state machine possible states.

Enumerator

<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.
7.2 ADC Driver

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.

Here is the call graph for this function:

![Call Graph]

7.2.7.2 adcObjectInit()

\begin{verbatim}
void adcObjectInit ( \\
    ADCDriver * adcp )
\end{verbatim}

Initializes the standard part of a \texttt{ADCDriver} structure.

Parameters

\begin{tabular}{ll}
  \textbf{out adcp} & pointer to the \texttt{ADCDriver} object \\
\end{tabular}

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]
7.2.7.3 adcStart()

void adcStart (  
    ADCDriver * adcp,  
    const ADCConfig * config  
)

Configures and activates the ADC peripheral.

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 config</td>
<td>pointer to the ADCConfig object. Depending on the implementation the value can be 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:

```
adcStart
    - adc_lld_start
    - osalSysLock
    - osalSysUnlock
```

7.2.7.4 adcStop()

void adcStop (  
    ADCDriver * adcp  
)

Deactivates the ADC peripheral.

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>
</tbody>
</table>

Function Class:

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

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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>adcp</td>
</tr>
<tr>
<td>in</td>
<td>grpp</td>
</tr>
<tr>
<td>out</td>
<td>samples</td>
</tr>
<tr>
<td>in</td>
<td>depth</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]

7.2.7.6 adcStartConversionI()

```c
void adcStartConversionI ( 
    ADCDriver * adcp,
    const 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>Type</th>
<th>Description</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>
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:

```
adcStopConversionI adc_lld_stop_conversion
```

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.
7.2.7.9 adcConvert()

```c
void adcConvert (
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.
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:

```
adcAcquireBus 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 ADC_USE_MUTUAL_EXCLUSION must be enabled.

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:

```
adcReleaseBus 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.
Parameters

| in | adcp | 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()

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

Starts an ADC conversion.

Parameters

| in | adcp | 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()

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

Stops an ongoing conversion.

Parameters

| in | adcp | pointer to the ADCDriver object |

Function Class:

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

### 7.2.8 Variable Documentation

ChibiOS/HAL
7.2.8.1 ADCD1

ADCDriver ADCD1

ADC1 driver identifier.
7.3 I/O Buffers Queues

7.3.1 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

- `#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 bqlsSuspendedX(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)`
  
  Evaluates to true if the specified output buffers queue is full.
7.3 I/O Buffers Queues

**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.

**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, 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 only, 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.
Module Documentation

7.3.2 Macro Definition Documentation

7.3.2.1 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.

7.3.2.2 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>
7.3.2.3 bqSizeX

```c
#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.

7.3.2.4 bqSpaceI

```c
#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 ioBuffers_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.

7.3.2.5 bqGetLinkX

```c
#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.

### 7.3.2.6 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.

### 7.3.2.7 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

<table>
<thead>
<tr>
<th>false</th>
<th>if blocking access to the queue is enabled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if blocking access to the queue is suspended.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.3.2.8 bqSuspendI

```c
#define bqSuspendI(bqp)  
{  
    (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 in 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.

### 7.3.2.9 bqResumeX

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

### 7.3.2.10 ibqIsEmptyI

```c
#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

- `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.

### 7.3.2.11 ibqIsFullI

```c
#define ibqIsFullI( ibqp )
```

Value:

```c
/*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.
7.3 I/O Buffers Queues

Parameters

\textbf{in } \textit{ibqp} \hspace{1cm} \textbf{pointer to an input\_buffers\_queue\_t structure}

Returns

The queue status.

Return values

| \textbf{false} | if the queue is not full. |
| \textbf{true} | if the queue is full. |

Function Class:

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

7.3.2.12 \texttt{obqIsEmptyI}

\begin{verbatim}
#define obqIsEmptyI( obqp )

Value:

\texttt{
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/

\{(bool)\{(obqp)->bwrptr == (obqp)->brdptr && (obqp)->bcounter != 0U\}\}}

/*lint -restore*/
\end{verbatim}

Evaluates to \texttt{true} if the specified output buffers queue is empty.

Parameters

\textbf{in } \textit{obqp} \hspace{1cm} \textbf{pointer to an output\_buffers\_queue\_t structure}

Returns

The queue status.

Return values

| \textbf{false} | if the queue is not empty. |
| \textbf{true} | if the queue is empty. |

Function Class:

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

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

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

Parameters

| in | obqp | pointer to an outputBuffers_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.

7.3.3 Typedef Documentation

7.3.3.1 io_buffers_queue_t

typedef struct io_buffers_queue io_buffers_queue_t

Type of a generic queue of buffers.

7.3.3.2 bqnotify_t

typedef void(* bqnotify_t) (ioBuffers_queue_t *bqp)

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

Parameters

| in  | iodbp | the buffers queue pointer |

7.3.3.3 input_buffers_queue_t

typedef io_buffers_queue_t input_buffers_queue_t

Type of an input buffers queue.

7.3.4 output_buffers_queue_t

typedef io_buffers_queue_t output_buffers_queue_t

Type of an output buffers queue.

7.3.4 Function Documentation

7.3.4.1 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
```

### 7.3.4.2 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
```
7.3.4.3 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**

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

**Returns**

A pointer to the next buffer to be filled.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</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.

---

7.3.4.4 ibqPostFullBufferI()

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

Posts a new filled buffer to the queue.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ibqp</td>
<td>pointer to the <code>input_buffers_queue_t</code> object</td>
</tr>
<tr>
<td>in size</td>
<td>used size of the buffer, cannot be zero</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:

```
ibqPostFullBufferI

osalThreadDequeueNextI
```

### 7.3.4.5 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:

![Call graph]

7.3.4.6 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>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:

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

### 7.3.4.7 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:

```
ibqReleaseEmptyBufferS

ibqReleaseEmptyBuffer

osalSysLock

osalSysUnlock
```
### 7.3.4.8 ibqReleaseEmptyBufferS()

```c
void ibqReleaseEmptyBufferS ( 
    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:**

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

### 7.3.4.9 ibqGetTimeout()

```c
msg_t ibqGetTimeout ( 
    input_buffers_queue_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 `input_buffers_queue_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

<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 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:

```
ibqGetTimeout
ibqGetFullBufferTimeoutS
osalSysLock
```

### 7.3.4.10 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>
7.3 I/O Buffers Queues

Returns

The number of bytes effectively transferred.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if a timeout 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.

Here is the call graph for this function:

![Call Graph Diagram]

7.3.4.11 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>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>suspended</code></td>
<td>initial state of the queue</td>
</tr>
<tr>
<td><code>bp</code></td>
<td>pointer to a memory area allocated for buffers</td>
</tr>
<tr>
<td><code>size</code></td>
<td>buffers size</td>
</tr>
<tr>
<td><code>n</code></td>
<td>number of buffers</td>
</tr>
<tr>
<td><code>onfy</code></td>
<td>callback called when a buffer is posted in the queue</td>
</tr>
<tr>
<td><code>link</code></td>
<td>application defined pointer</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Initialize, 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
```

### 7.3.4.12 obqResetI()

```c
void obqResetI (outputBuffers_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**

- **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:

```
    obqResetI
        → osalThreadDequeueAll
```
7.3.4.13  `obqGetFullBufferI()`

```c
uint8_t * obqGetFullBufferI ( 
          output_buffers_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 <code>outputBuffersQueue_t</code> 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.

7.3.4.14  `obqReleaseEmptyBufferI()`

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

Releases the next filled buffer back in the queue.

Parameters

| in  | obqp | pointer to the `outputBuffersQueue_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:

```
+-----------------+----------------------+
| obqReleaseEmptyBufferI | osalThreadDequeueNextI |
+-----------------+----------------------+
```

### 7.3.4.15 obqGetEmptyBufferTimeout()

```c
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><code>in obqp</code></td>
<td>pointer to the <code>output_buffers_queue_t</code> object</td>
</tr>
</tbody>
</table>
| `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**

| 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:

```
    obqGetEmptyBufferTimeout
    obqGetEmptyBufferTimeoutS
    osalSysLock
    osalSysUnlock
```

### 7.3.4.16 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>in</th>
<th>obqp</th>
<th>pointer to the output_buffers_queue_t 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

<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.

### 7.3.4.17 obqPostFullBuffer()

```c
void obqPostFullBuffer (  
    outputBuffersQueue_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 outputBuffersQueue_t 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:
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)

### 7.3.4.18 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 output_buffers_queue_t 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.

### 7.3.4.19 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 output_buffers_queue_t 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>• 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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:

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)

7.3.4.20 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.
7.3 I/O Buffers Queues

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>obqp</th>
<th>pointer to the output_buffers_queue_t object</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>
<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.

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:

---

7.3.4.21 obqTryFlushI()

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.

### 7.3.4.22 obqFlush()

```c
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:

```
/ \                  /
|   |                  |
|   |                  |
|   | obqFlush          |
| / |                 |
|   |                  |
|   | obqPostFullBufferS|
|   |                  |
|   |                  |
|   | osalSysLock       |
|   |                  |
|   |                  |
|   | osalSysUnlock     |
```
7.4 CAN Driver

Generic CAN Driver.

7.4.1 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`.

7.4.2 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 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.
Typedefs

- typedef struct CANDriver 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.

Data Structures

- struct CANTxFrame
  CAN transmission frame.
- struct CANRxFrame
  CAN received frame.
- struct CANConfig
  Driver configuration structure.
- struct CANDriver
  Structure representing an CAN driver.

Functions

- void canInit (void)
  CAN Driver initialization.
- void canObjectInit (CANDriver *canp)
  Initializes the standard part of a CANDriver structure.
- void 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)
  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.

7.4.3 Macro Definition Documentation

7.4.3.1 CAN_LIMIT_WARNING

#define CAN_LIMIT.WARNING 1U

Errors rate warning.

7.4.3.2 CAN_LIMIT_ERROR

#define CAN_LIMIT.ERROR 2U

Errors rate error.
7.4.3.3 CAN_BUS_OFF_ERROR

#define CAN_BUS_OFF_ERROR 4U

Bus off condition reached.

7.4.3.4 CAN_FRAMING_ERROR

#define CAN_FRAMING_ERROR 8U

Framing error of some kind on the CAN bus.

7.4.3.5 CAN_OVERFLOW_ERROR

#define CAN_OVERFLOW_ERROR 16U

Overflow in receive queue.

7.4.3.6 CAN_ANY_MAILBOX

#define CAN_ANY_MAILBOX 0U

Special mailbox identifier.

7.4.3.7 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.

7.4.3.8 CAN_ENFORCE_USE_CALLBACKS

#define CAN_ENFORCE_USE_CALLBACKS FALSE

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

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

Converts a mailbox index to a bit mask.

7.4.3.10 canTransmit

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

Legacy name for canTransmitTimeout().

Deprecated

7.4.3.11 canReceive

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

Legacy name for canReceiveTimeout().

Deprecated

7.4.3.12 _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.
7.4 CAN Driver

7.4.3.13 _can_rx_full_isr

#define _can_rx_full_isr(
    canp,
    flags
)

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

RX mailbox empty full event.

7.4.3.14 _can_wakeup_isr

#define _can_wakeup_isr(
    canp
)

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

Wakeup event.

7.4.3.15 _can_error_isr

#define _can_error_isr(
    canp,
    flags
)

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

Error event.

7.4.3.16 CAN_TX_MAILBOXES

#define CAN_TX_MAILBOXES 1

Number of transmit mailboxes.
7.4.3.17 CAN_RX_MAILBOXES

#define CAN_RX_MAILBOXES 1

Number of receive mailboxes.

7.4.3.18 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.

7.4.4 Typedef Documentation

7.4.4.1 CANDriver

typedef struct CANDriver CANDriver

Type of a structure representing an CAN driver.

7.4.4.2 canmbx_t

typedef uint32_t canmbx_t

Type of a transmission mailbox index.

7.4.4.3 can_callback_t

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

Type of a CAN notification callback.
7.4 CAN Driver

Parameters

| in  | canp   | pointer to the CANDriver object triggering the callback |
| in  | flags  | flags associated to the mailbox callback               |

7.4.5 Enumeration Type Documentation

7.4.5.1 canstate_t

def canstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</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>CANREADY</td>
<td>Ready.</td>
</tr>
<tr>
<td>CAN_SLEEP</td>
<td>Sleep state.</td>
</tr>
</tbody>
</table>

7.4.6 Function Documentation

7.4.6.1 canInit()

```c
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:

![Call Graph]

7.4.6.2 canObjectInit()

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

Initializes the standard part of a CANDriver structure.

Parameters

| out | 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:

![Call Graph]
### 7.4.6.3 `canStart()`

```c
void 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>canp</code></td>
<td>pointer to the <code>CANDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>config</code></td>
<td>pointer to the <code>CANConfig</code> object. Depending on the implementation the value can be <code>NULL</code>.</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)

### 7.4.6.4 `canStop()`

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

Deactivates the CAN peripheral.
7.4.6.5 canTryTransmitI()

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.
7.4 CAN Driver

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:

![Call Graph](image)

7.4.6.6 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:

```
    canTryReceiveI
      can_lld_is_rx_nonempty
        can_lld_receive
```

### 7.4.6.7 canTryAbortX()

void canTryAbortX (
    CANDriver * canp,
    canmbx_t mailbox)

Tries to abort an ongoing transmission.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>canp</code></td>
<td>pointer to the CANDriver object</td>
</tr>
<tr>
<td><code>mailbox</code></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:

```
    canTryAbortX
      can_lld_abort
```
7.4.6.8 canTransmitTimeout()

```c
msg_t canTransmitTimeout (  
    CANDriver * canp,  
    canmbx_t mailbox,  
    const CANTxFrame * ctfp,  
    sysinterval_t timeout  
)
```

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>
<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 result.

**Return values**

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>the frame has been queued for transmission.</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>
7.4.6.9  canReceiveTimeout()

```
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_INFINITE no timeout.</td>
</tr>
</tbody>
</table>
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
```

7.4.6.10 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_SUPPORT_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:

```
canSleep
    
    can_lld_sleep
    osalEventBroadcastFlagsI
    osalOsRescheduleS
    osalSysLock
    osalSysUnlock

7.4.6.11 canWakeup()

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

| in  | canp | pointer to the CANDriver object |
Here is the call graph for this function:

```
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:
7.4.6.13  can_lld_start()

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

Configures and activates the CAN peripheral.
7.4 CAN Driver

Parameters

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

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

7.4.6.14 can_lld_stop()

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

Deactivates the CAN peripheral.

Parameters

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

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

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

Returns
The queue space availability.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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
7.4.6.16 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>canp</code></td>
<td>pointer to the CANDriver object</td>
</tr>
<tr>
<td><code>ctfp</code></td>
<td>pointer to the CAN frame to be transmitted</td>
</tr>
<tr>
<td><code>mailbox</code></td>
<td>mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
</tbody>
</table>

7.4.6.17 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>canp</code></td>
<td>pointer to the CANDriver object</td>
</tr>
<tr>
<td><code>mailbox</code></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><code>false</code></td>
<td>no space in the transmit queue.</td>
</tr>
<tr>
<td><code>true</code></td>
<td>transmit slot available.</td>
</tr>
</tbody>
</table>
Function Class:
    Not an API, this function is for internal use only.

### 7.4.6.18 can_lld_receive()  

```c
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.

### 7.4.6.19 can_lld_abort()  

```c
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.

### 7.4.6.20 can_lld_sleep()  

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

ChibiOS/HAL
Enters the sleep mode.

Parameters

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

Function Class:

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

7.4.6.21 can_lld_wakeup()

void can_lld_wakeup (CANDriver * canp)

Enforces leaving the sleep mode.

Parameters

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

Function Class:

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

7.4.7 Variable Documentation

7.4.7.1 CAND1

CANDriver CAND1

CAND1 driver identifier.
7.5 Crypto Driver

Generic Crypto Driver.

7.5.1 Detailed Description

Generic 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_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 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.
- void 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)
  Decryption operation using AES-ECB.
7.5 Crypto Driver

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 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 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, const uint8_t *tag_in)

Encryption operation using (T)DES.

- 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 *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 *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 *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 *iv)
  Decryption operation using AES-CFB.
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_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_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.

7.5.2 Macro Definition Documentation

7.5.2.1 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 underlying hardware.

Note

Fall-back implementations may not be present for all algorithms.
7.5.2.2 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.

7.5.2.3 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.

7.5.2.4 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.

7.5.3 Typedef Documentation

7.5.3.1 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.
7.5.3.2 crykey_t

typedef uint32_t crykey_t

CRY key identifier type.

7.5.3.3 CRYDriver

typedef struct CRYDriver CRYDriver

Type of a structure representing an CRY driver.

7.5.4 Enumeration Type Documentation

7.5.4.1 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>

7.5.4.2 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>
7.5 Crypto Driver

7.5.4.3 cryalgorithm_t

def 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_ERR_INV_KEY_TYPE</td>
<td>Invalid key type.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_ID</td>
<td>Invalid key identifier.</td>
</tr>
<tr>
<td>CRY_ERR_AUTH_FAILED</td>
<td>Failed authentication.</td>
</tr>
<tr>
<td>CRY_ERR_OP_FAILURE</td>
<td>Failed operation.</td>
</tr>
</tbody>
</table>

7.5.5 Function Documentation

7.5.5.1 cryInit()

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
```

7.5.5.2 cryObjectInit()

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.

7.5.5.3 cryStart()

void 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. |
7.5 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:

```
7.5.5.4 cryStop()

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:

![Call Graph](image)

7.5.5.5 **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 → cry_lld_aes_loadkey
```

7.5.5.6 cryEncryptAES()

cryptor_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>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:

![Call graph](image)

7.5.5.7 cryDecryptAES()

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

<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>cryerror_t</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:

```
    cryDecryptAES  cry_lld_decrypt_AES
```

### 7.5.5.8 cryEncryptAES_ECB()

```c
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>Error 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
```

7.5.5.9 cryDecryptAES_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.

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>
</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:

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:

```
cretecryptAES_CBC  
cryDecryptAES_ECB
```

7.5.5.10 cryEncryptAES_CBC()

```c
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.

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>ChiedoMA</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 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_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:

7.5.5.11 cryDecryptAES_CBC()

```c
cherror_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>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 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>
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:

7.5.5.12 cryEncryptAES_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-CFB.

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

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>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 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](cryEncryptAES_CFB -> cry_lld_encrypt_AES_CFB)

7.5.5.13 cryDecryptAES_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.

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 input 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>
<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 diagram]

7.5.5.14 cryEncryptAES_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-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 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>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_CTR  ->  cry_lld_encrypt_AES_CTR
```

7.5.5.15 cryDecryptAES_CTR()

crypterror_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>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<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 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 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 Diagram]

7.5.5.16 cryEncryptAES_GCM()

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

<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

<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:

```
cryEncryptAES_GCM  cry_lld_encrypt_AES_GCM
```

7.5.5.17 cryDecryptAES_GCM()

cryerror_t cryDecryptAES_GCM (crykey_t key_id, CRYDriver cryp) {
Decryption operation using AES-GCM.

Note

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

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>cryp</strong></td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>key_id</strong></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><strong>in</strong></td>
<td><strong>auth_size</strong></td>
<td>size of the data buffer to be authenticated</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>auth_in</strong></td>
<td>buffer containing the data to be authenticated</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>text_size</strong></td>
<td>size of the text buffer</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>text_in</strong></td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>text_out</strong></td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>iv</strong></td>
<td>128 bits input vector</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>tag_size</strong></td>
<td>size of the authentication tag, this number must be between 1 and 16</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>tag_in</strong></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_AUTH_FAILED** | authentication failed |
| **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:

![Call Graph]

7.5.5.18 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:

7.5.5.19 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>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:

```
cryEncryptDES cry_lld_encrypt_DES
```

7.5.5.20 cryDecryptDES()

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:

![Call graph for cryDecryptDES and cry_lld_decrypt_DES]

7.5.5.21 cryEncryptDES_ECB()

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 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 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>Module</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:

![Call Graph](image)

7.5.5.22 cryDecryptDES_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.

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 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 8</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>
7.5 Crypto Driver

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:

7.5.5.23 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>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>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 plaintext</td>
</tr>
<tr>
<td>out out</td>
<td>buffer for the output ciphertext</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:

```
 cryEncryptDES_CBC  cry_lld_encrypt_DES_CBC
```

7.5.5.24 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>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
<td></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>
<td></td>
</tr>
<tr>
<td>size</td>
<td>size of both buffers, this number must be a multiple of 8</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>buffer containing the input ciphertext</td>
<td></td>
</tr>
<tr>
<td>out</td>
<td>buffer for the output plaintext</td>
<td></td>
</tr>
<tr>
<td>iv</td>
<td>64 bits input vector</td>
<td>ChibiOS/HAL</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:

```
| cryDecryptDES_CBC               | cry_lld_decrypt_DES_CBC |
```

### 7.5.5.25 crySHA1Init()

```c
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>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>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return 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_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:

```
    crySHA1Init -> cry_lld_SHA1_init
```

### 7.5.5.26 crySHA1Update()

```c
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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cryp</code></td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td><code>sha1ctxp</code></td>
<td>pointer to a SHA1 context</td>
</tr>
<tr>
<td><code>size</code></td>
<td>size of input buffer</td>
</tr>
<tr>
<td><code>in</code></td>
<td>buffer containing the input text</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:

```
crySHA1Update cry_lld_SHA1_update
```

### 7.5.5.27 crySHA1Final()

```c
cryerror_t crySHA1Final (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>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>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:

```plaintext
crySHA1Final    cry_lld_SHA1_final
```

7.5.5.28  crySHA256Init()

cryerror_t crySHA256Init (  
    CRYDriver * cryp,  
    SHA256Context * sha256ctxp )

Hash initialization using SHA256.

Parameters

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<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>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:

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:

```
crySHA256Update cry_lld_SHA256_update
```

7.5.5.30 crySHA256Final()

ccryerror_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

<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:

```
crySHA256Final    cry_lld_SHA256_final
```

### 7.5.5.31 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
```

### 7.5.5.32 crySHA512Update()

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

Hash update 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>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>
7.5 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:

```
crySHA512Update cry_lld_SHA512_update
```

7.5.5.33 crySHA512Final()

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

Hash finalization using SHA512.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td><strong>in</strong> sha512ctxp</td>
<td>pointer to a SHA512 context</td>
</tr>
<tr>
<td><strong>out</strong> out</td>
<td>512 bits output buffer</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><strong>CRY_NOERROR</strong></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><strong>CRY_ERR_INV_ALGO</strong></td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td><strong>CRY_ERR_OP_FAILURE</strong></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
```

### 7.5.5.34 cryLoadHMACTransientKey()

```c
cryerror_t cryLoadHMACTransientKey (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**

| CRY_NOERROR | if the operation succeeded. |
| CRY_ERR_INV_ALGO | if the algorithm is unsupported. |
| CRY_ERR_INV_KEY_SIZE | if the specified key size is invalid for the specified algorithm. |

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
```

### 7.5.5.35 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**

| 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:

```
cryHMACSHA256Init -> cry_lld_HMACSHA256_init
```
7.5.5.36  cryHMACSHA256Update()

cryerror_t cryHMACSHA256Update (  
    CRYDriver * cryp,  
    HMACSHA256Context * hmacsha256ctxp,  
    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>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:

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:
7.5.5.37 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 diagram]

7.5.5.38 cryHMACSHA512Init()

cryerror_t cryHMACSHA512Init (  
  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></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:

```
cryHMACSHA512Init -- cry_lld_HMACSHA512_init
```

7.5.5.39 cryHMACSHA512Update()

```c
error_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
```

### 7.5.5.40 cryHMACSHA512Final()

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

Hash finalization using HMAC.

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 hmacsha512ctxp</td>
<td>pointer to a HMAC_SHA512 context</td>
</tr>
<tr>
<td>out out</td>
<td>512 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:

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)

### 7.5.5.41 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.

### 7.5.5.42 cry_lld_start()

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

Configures and activates the crypto peripheral.

**Parameters**

| in  | cryp | pointer to the CRYDriver object |

ChibiOS/HAL
7.5 Crypto Driver

Function Class:

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

7.5.5.43 cry_lld_stop()

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

Deactivates the crypto peripheral.

Parameters

| in  | cryp | pointer to the CRYDriver object |

Function Class:

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

7.5.5.44 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

| in | cryp | pointer to the CRYDriver object |
| in | size | key size in bytes               |
| in | keyp | pointer to the key data         |

Returns

The operation status.

Return values

| CRY_NOERROR | if the operation succeeded. |

ChibiOS/HAL
Return values

<table>
<thead>
<tr>
<th>Return Value</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.

7.5.5.45 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>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>in buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out 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:

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

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>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.

7.5.5.47 cry_lld_encrypt_AES_ECB()

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

| 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 both buffers, this number must be a multiple of 16 |
| in in | buffer containing the input plaintext |
| out out | buffer for the output ciphertext |

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.

7.5.5.48 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.
7.5 Crypto Driver

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

| 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.

7.5.5.49 cry_lld_encrypt_AES_CBC()

cryerror_t cry_lld_encrypt_AES_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>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>
<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>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.

7.5.5.50 cry_lld_decrypt_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.

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 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>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.

7.5.5.51 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>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</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>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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.

7.5.5.52 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

<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 initial vector</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.

### 7.5.5.53 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>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</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>
<tr>
<td><code>iv</code></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>Return 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.
7.5.5.54 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.

7.5.5.55 cry_lld_encrypt_AES_GCM()

```c
cryerror_t cry_lld_encrypt_AES_GCM ( 
    CRYDriver * cryp,
    crykey_t key_id,
```
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

<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.

7.5.5.56 cry_lld_decrypt_AES_GCM()

cryerror_t cry_lld_decrypt_AES_GCM (cryp, 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>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>in</td>
<td>tag_in</td>
<td>buffer for the generated authentication tag</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_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.
7.5.5.57 cry_lld_des_loadkey()

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>key size in bytes</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>keyp</td>
<td>pointer to the key data</td>
<td></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 algorithm is unsupported.</td>
<td></td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>if the specified key size is invalid for the specified algorithm.</td>
<td></td>
</tr>
</tbody>
</table>

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

7.5.5.58 cry_lld_encrypt_DES()

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.
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

| 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.

7.5.5.59 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 plaintext</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.

7.5.5.60 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></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></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>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.

7.5.5.61 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>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 the plaintext buffer, this number must be a multiple of 8</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>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>
7.5 Crypto Driver

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

7.5.5.62 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

<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 the plaintext buffer, 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>
<tr>
<td>in</td>
<td>iv</td>
<td>64 bits input vector</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.
7.5.5.63 cry_lld_decrypt_DES_CBC()

```c
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

<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 the plaintext buffer, this number must be a multiple of 8</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>
<tr>
<td><code>iv</code></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:

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

7.5.5.64 cry_lld_SHA1_init()

```c
cryerror_t cry_lld_SHA1_init ( 
    CRYDriver ∗ cryp,
    SHA1Context ∗ sha1ctxp )
```

Hash initialization 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>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.

7.5.5.65 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></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.

7.5.5.66 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></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></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.
7.5.5.67 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.

7.5.5.68 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>
</table>
Return values

<table>
<thead>
<tr>
<th>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.

7.5.5.69  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>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>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

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

7.5.5.70  cry_lld_SHA512_init()

cryerror_t cry_lld_SHA512_init (  
    CRYDriver * cryp,
    SHA512Context * sha512ctxp)

Hash initialization using SHA512.
7.5 Crypto Driver

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>

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.

7.5.5.71 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

<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>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.
7.5.5.72 cry_lld_SHA512_final()

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.

7.5.5.73 cry_lld_hmac_loadkey()

cryerror_t cry_lld_hmac_loadkey (  
    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>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYP_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRYP_ERR_INV_ALGO</td>
<td>if the algorithm is unsupported.</td>
</tr>
<tr>
<td>CRYP_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.

7.5.5.74 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 cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>out 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>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYP_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRYP_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRYP_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.

7.5.5.75 cry_lld_HMACSHA256_update()

cryerror_t cry_lld_HMACSHA256_update {
    CRYDriver * cryp,

**HMACSHA256Context**  

```c
HMACSHA256Context * hmacsha256ctxp,
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 <strong>CRYDriver</strong> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>hmacsha256ctxp</td>
<td>pointer to a <strong>HMAC_SHA256</strong> 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.

---

### 7.5.5.76 cry_lld_HMACSHA256_final()

```c
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 <strong>CRYDriver</strong> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>hmacsha256ctxp</td>
<td>pointer to a <strong>HMAC_SHA256</strong> 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></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CRY_NOERROR</strong></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><strong>CRY_ERR_INV_ALGO</strong></td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td><strong>CRY_ERR_OP_FAILURE</strong></td>
<td>if the operation failed, implementation dependent.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.5.5.77 cry_lld_HMACSHA512_init()

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

Function Class:

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

### 7.5.5.78 cry_lld_HMACSHA512_update()

```c
cryerror_t cry_lld_HMACSHA512_update (  
    CRYDriver * cryp,  
    HMACSHA512Context * hmacsha512ctxp,  
    size_t size,  
    const uint8_t * in  
)
```

Hash update using HMAC.
### 7.5.5.79 cry_lld_HMACSHA512_final()

```c
cryerror_t cry_lld_HMACSHA512_final (  
    CRYDriver * cryp,
    HMACSHA512Context * hmacsha512ctxp,
    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>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>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.
Function Class:
   Not an API, this function is for internal use only.

7.5.6 Variable Documentation

7.5.6.1 CRYD1

**CRYDriver CRYD1**

CRY1 driver identifier.
7.6 DAC Driver

Generic DAC Driver.

7.6.1 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.
7.6 DAC Driver

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.

- `void 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.

7.6.2 Macro Definition Documentation

7.6.2.1 DAC_USE_WAIT

#define DAC_USE_WAIT TRUE  
Enables synchronous APIs.

Note  
Disabling this option saves both code and data space.
7.6.2.2 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.

7.6.2.3 dacIsBufferComplete

#define dacIsBufferComplete(
    dACP)
    ((bool)((dACP)->state == DAC_COMPLETE))

Buffer state.

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

Parameters
- **in** `dACP` 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.

7.6.2.4 _dac_wait_s

#define _dac_wait_s(
    dACP)
    osalThreadSuspendS(&(dACP)->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

<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>
</tbody>
</table>

Function Class:

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

7.6.2.5 _dac_reset_i

```c
#define _dac_reset_i(
    dacp ) osalThreadResumeI(&(dacp)->thread, MSG_RESET)
```

Resumes a thread waiting for a conversion completion.

Parameters

<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>
</tbody>
</table>

Function Class:

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

7.6.2.6 _dac_reset_s

```c
#define _dac_reset_s(
    dacp ) osalThreadResumeS(&(dacp)->thread, MSG_RESET)
```

Resumes a thread waiting for a conversion completion.

Parameters

<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>
</tbody>
</table>
Function Class:

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

### 7.6.2.7 _dac_wakeup_isr

```c
#define _dac_wakeup_isr(
    dACP
)
```

**Value:**

```
  \
  ( osalSysLockFromISR();
  osalThreadResumeI(&(dACP)->thread, MSG_OK);
  osalSysUnlockFromISR();
 )
```

Wakes up the waiting thread.

**Parameters**

| in   | dACP | pointer to the DACDriver object |

Function Class:

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

### 7.6.2.8 _dac_timeout_isr

```c
#define _dac_timeout_isr(
    dACP
)
```

**Value:**

```
  \
  ( osalSysLockFromISR();
  osalThreadResumeI(&(dACP)->thread, MSG_TIMEOUT);
  osalSysUnlockFromISR();
 )
```

Wakes up the waiting thread with a timeout message.

**Parameters**

| in   | dACP | pointer to the DACDriver object |

Function Class:

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

#define _dac_isr_half_code(
dACP)

Value:
{}

if ((ACP)->grpp->end_cb != NULL) {
    (ACP)->grpp->end_cb(ACP);
}

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.

7.6.2.10 _dac_isr_full_code

#define _dac_isr_full_code(
dACP)

Value:
{}

if ((ACP)->grpp->end_cb) {
    (ACP)->state = DAC_COMPLETE;
    (ACP)->grpp->end_cb(ACP);
    if ((ACP)->state == DAC_COMPLETE)
        (ACP)->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.
7.6 DAC Driver

Parameters

| in  | dacp | pointer to the DACDriver object |

Function Class:

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

7.6.2.11 _dac_isr_error_code

#define _dac_isr_error_code(
  dacp,
  err
)

Value:

| dacp_lld_stop_conversion(dacp); \  |
| if ((dacp)->grpp->error_cb != NULL) { \  |
|   (dacp)->state = DAC_ERROR; \  |
|   (dacp)->grpp->error_cb(dacp, err); \  |
|   if ((dacp)->state == DAC_ERROR) \  |
|     (dacp)->state = DAC_READY; \  |
| } \  |
| (dacp)->grpp = NULL; \  |
| _dac_timeout_isr(dacp); \  |

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

| in  | dacp | pointer to the DACDriver object |
| in  | err  | platform dependent error code |

Function Class:

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

#define DAC_MAX_CHANNELS 2

Maximum number of DAC channels per unit.

7.6.2.13 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.

7.6.2.14 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.

7.6.2.15 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.

7.6.2.16 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.
7.6 DAC Driver

7.6.3 Typedef Documentation

7.6.3.1 DACDriver

typedef struct hal_dac_driver DACDriver

Type of a structure representing an DAC driver.

7.6.3.2 DACConfig

typedef struct hal_dac_config DACConfig

Type of a structure representing an DAC driver configuration.

7.6.3.3 DACConversionGroup

typedef struct hal_dac_conversion_group DACConversionGroup

Type of a DAC conversion group.

7.6.3.4 daccallback_t

typedef void(* daccallback_t) (DACDriver *dacp)

DAC notification callback type.
Parameters

| in | dacp | pointer to the DACDriver object triggering the |

7.6.3.5 dacerrorcallback_t

typedef void(* dacerrorcallback_t) (DACDriver *dacp, dacerror_t err)

DAC error callback type.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dacp</th>
<th>pointer to the DACDriver object triggering the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>err</td>
<td>DAC error code</td>
</tr>
</tbody>
</table>

7.6.3.6 dacchannel_t

typedef uint32_t dacchannel_t

Type of a DAC channel index.

7.6.3.7 dacsample_t

typedef uint16_t dacsample_t

Type representing a DAC sample.

7.6.4 Enumeration Type Documentation

7.6.4.1 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>
7.6 DAC Driver

7.6.4.2 dacerror_t

enum dacerror_t

Possible DAC failure causes.

Note

Error codes are architecture dependent and should not relied upon.

Enumerator

<table>
<thead>
<tr>
<th>DAC_ERR_DMAFAILURE</th>
<th>DMA operations failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_ERR_UNDERFLOW</td>
<td>DAC overflow condition.</td>
</tr>
</tbody>
</table>

7.6.5 Function Documentation

7.6.5.1 dacInit()

void dacInit (
    void
)

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:

```
  dacInit -> dac_lld_init -> dacObjectInit
```

7.6.5.2 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]

7.6.5.3 dacStart()

```c
void dacStart (  
    DACDriver * dacp,  
    const DACConfig * config)
```

Configures and activates the DAC peripheral.

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>config</td>
<td>pointer to the DACConfig object, it can be NULL if the low level driver implementation supports a default 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.

Here is the call graph for this function:

```
dacStart
  ↓
dac_lld_start
  ↓
osal SysLock
  ↓
osal SysUnlock
```

### 7.6.5.4 dacStop()

```c
void dacStop (DACDriver * dacp )
```

Deactivates the DAC peripheral.

**Note**

Deactivating the peripheral also enforces a release of the slave select line.

**Parameters**

| in  | dacp | pointer to the DACDriver 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 for dacStop]

7.6.5.5 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>in</th>
<th>dacp</th>
<th>pointer to the DACDriver object</th>
</tr>
</thead>
<tbody>
<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:

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

Here is the call graph for this function:

![Call Graph for dacPutChannelX]
7.6.5.6 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*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 |
| in  | samples | pointer to the samples buffer |
| in  | depth  | buffer depth (matrix rows number). The buffer depth must be one or an even number. |

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:

7.6.5.7 dacStartConversionI()
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

| in | dcp | pointer to the DACDriver object |
| in | grpp | pointer to a DACConversionGroup object |
| in | samples | pointer to the samples buffer |
| in | depth | buffer depth (matrix rows number). The buffer depth must be one or an even number. |

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:

7.6.5.8 dacStopConversion()

void dacStopConversion (  
   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.
7.6 DAC Driver

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>
</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:

```
    dacStopConversion
    ▼
     |  
    ▼  ▼
  osalSysLock
  |   |
  v  v
dac_lld_stop_conversion
```

7.6.5.9 dacStopConversionI()

```c
void dacStopConversionI (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

<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>
</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:

\[
\text{dacStopConversionI} \rightarrow \text{dac_lld_stop_conversion}
\]

7.6.5.10 dacConvert()

\[
\text{msg_t dacConvert (}
\begin{align*}
&\text{DACDriver } \ast \text{ dacp}, \\
&\text{const DACConversionGroup } \ast \text{ grpp}, \\
&\text{dacsample_t } \ast \text{ samples}, \\
&\text{size_t } \text{ depth }
\end{align*}
\text{)}
\]

Performs a DAC conversion.

Performs a synchronous 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>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>

Returns

The operation result.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>Conversion finished.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>The conversion has been stopped using \text{acdStopConversion()} or \text{acdStopConversionI()}. The result buffer may contain incorrect data.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>The conversion has been stopped because an hardware 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:

![Call Graph]

7.6.5.11 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:

```
| dacAcquireBus | osalMutexLock |
```

### 7.6.5.12 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**
- `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:

```
| dacReleaseBus | osalMutexUnlock |
```
7.6.5.13 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:

![Call Graph](image)

7.6.5.14 dac_lld_start()

```c
void dac_lld_start (  
    DACDriver * dacp )
```

Configures and activates the DAC peripheral.

**Parameters**

* `dACP` pointer to the DACDriver object

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

7.6.5.15 dac_lld_stop()

```c
void dac_lld_stop (  
    DACDriver * dacp )
```

Deactivates the DAC peripheral.
Parameters

<table>
<thead>
<tr>
<th>in</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.

7.6.5.16 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.

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>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.

7.6.5.17 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.
7.6 DAC Driver

Parameters

| in | dacp | pointer to the DACDriver object |

Function Class:

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

7.6.5.18 dac_lld_stop_conversion()

```
void dac_lld_stop_conversion(
    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

| in | dacp | 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.

7.6.6 Variable Documentation

7.6.6.1 DACD1

```
DACDriver DACD1
```

DAC1 driver identifier.
7.7 EFL Driver

Generic Embedded Flash Driver.

7.7.1 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_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.
- void 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 a 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.

7.7.2 Macro Definition Documentation

7.7.2.1 _efl_flash_methods_alone

#define _efl_flash_methods_alone

EFlashDriver specific methods.
7.7.2.2 _efl_flash_methods

#define _efl_flash_methods

Value:

_base_flash_methods
_efl_flash_methods_alone

EFlashDriver specific methods with inherited ones.

7.7.2.3 _efl_driver_data

#define _efl_driver_data

Value:

_base_flash_data
/* Current configuration data. */
const EFlashConfig *config;

EFlashDriver specific data.

7.7.2.4 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.

7.7.2.5 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.
7.7 EFL Driver

7.7.2.6 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.

7.7.3 Typedef Documentation

7.7.3.1 EFlashDriver

typedef struct hal_efl_driver EFlashDriver

Type of external flash driver class.

7.7.3.2 EFlashConfig

typedef struct hal_efl_config EFlashConfig

Type of a structure representing a flash driver configuration.

7.7.4 Function Documentation

7.7.4.1 eflInit()

void eflInit()

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:

[Call graph diagram]
7.7.4.2 eflObjectInit()

```c
void eflObjectInit ( 
    EFlashDriver * eflp )
```

Initializes a generic EFlashDriver object.

**Parameters**

| out | eflp | pointer to a EFlashDriver structure |

**Function Class:**

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

7.7.4.3 eflStart()

```c
void 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>

**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:
7.7.4.4 eflStop()

```c
void eflStop (EFlashDriver *eflp)
```

Stops the driver.

Parameters

```c
in  eflp  pointer to a EFlashDriver structure
```

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:

```
  eflStop    osalSysLock
```

7.7.4.5 efl_lld_init()

```c
void efl_lld_init (void)
```

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:

```
  efl_lld_init    eflObjectInit
```
7.7.4.6  efl_lld_start()

```c
void efl_lld_start (  
    EFlashDriver * efip  )
```

Configures and activates the Embedded Flash peripheral.

Parameters
- **in** efip  pointer to a EFlashDriver structure

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

7.7.4.7  efl_lld_stop()

```c
void efl_lld_stop (  
    EFlashDriver * efip  )
```

Deactivates the Embedded Flash peripheral.

Parameters
- **in** efip  pointer to a EFlashDriver structure

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

7.7.4.8  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.

7.7.4.9  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>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_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.

7.7.4.10  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.
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 programmed</td>
</tr>
<tr>
<td>in</td>
<td>pp</td>
<td>pointer to the data buffer</td>
</tr>
</tbody>
</table>

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Flash 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:

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

7.7.4.11  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

<table>
<thead>
<tr>
<th>Flash 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.

7.7.4.12 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>Flash 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.

7.7.4.13 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>

ChibiOS/HAL
Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>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_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.

### 7.7.4.14  efl_lld_verify_erase()

```c
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><code>instance</code></td>
<td>pointer to a EFlashDriver instance</td>
</tr>
<tr>
<td><code>sector</code></td>
<td>sector to be verified</td>
</tr>
</tbody>
</table>

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>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:

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

### 7.7.5  Variable Documentation
7.7.5.1 EFLD1

EFlashDriver EFLD1

EFL1 driver identifier.
7.8 Generic Flash Interface

HAL Generic Flash Driver Interface.

7.8.1 Detailed Description

HAL Generic Flash Driver Interface.

7.8.2 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 _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.

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.

Enumerations

- `enum flash_state_t`
  Driver state machine possible states.
- `enum flash_error_t`
  Type of a flash error code.

7.8.3 Macro Definition Documentation

7.8.3.1 FLASH_ATTR_ERASED_IS_ONE

#define FLASH_ATTR_ERASED_IS_ONE 0x00000001U

Defines one as the erased bit state.

7.8.3.2 FLASH_ATTR_MEMORY_MAPPED

#define FLASH_ATTR_MEMORY_MAPPED 0x00000002U

The memory is accessible in a memory mapped mode.

7.8.3.3 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.
7.8 Generic Flash Interface

7.8.3.4 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.

7.8.3.5 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.

7.8.3.6 FLASH_ATTR_SUSPEND_ERASE_CAPABLE

#define FLASH_ATTR_SUSPEND_ERASE_CAPABLE 0x00000020U

The device is able to suspend erase operations.

7.8.3.7 _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 *rp);
/* 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);
flash_error_t (*query_erase)(void *instance, uint32_t *wait_time);
/* Verify erase single sector */
flash_error_t (*verify_erase)(void *instance, flash_sector_t sector);

BaseFlash specific methods.
7.8.3.8 _base_flash_methods

#define _base_flash_methods

Value:
_base_object_methods
_base_flash_methods_alone

_BaseFlash specific methods with inherited ones.

7.8.3.9 _base_flash_data

#define _base_flash_data

Value:
_base_object_data
/* Driver state. */
flash_state_t state;

_BaseFlash specific data.

7.8.3.10 getBaseFlash

#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.

7.8.3.11 flashGetDescriptor

#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.

7.8.3.12 flashRead

#define flashRead(
    ip,
    offset,
    n,
    rp ) (ip)->vmt->read(ip, offset, n, rp)

Read operation.

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>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>Enum</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.

7.8.3.13 flashProgram

#define flashProgram(
    ip,
    offset,
    n,
    pp ) (ip)->vmt->program(ip, offset, n, pp)

Program operation.
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>offset</td>
<td>flash offset</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to be programmed</td>
</tr>
<tr>
<td>in</td>
<td>pp</td>
<td>pointer to the data buffer</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_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.

7.8.3.14 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>
7.8 Generic Flash Interface

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

7.8.3.15 flashStartEraseSector

#define flashStartEraseSector(ip, sector) (ip)->vmt->start_erase_sector(ip, sector)

Starts an sector erase operation.

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 erased</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_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.

7.8.3.16 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>

ChibiOS/HAL
Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Enum</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.

7.8.3.17 flashVerifyErase

#define flashVerifyErase(ip, sector) (ip)->vmt->verify_erase(ip, sector)

Returns the erase state of a sector.

Parameters

<table>
<thead>
<tr>
<th>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 sector</td>
<td>sector to be verified</td>
</tr>
</tbody>
</table>

Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Enum</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.

7.8.4 Typedef Documentation
7.8.4.1 flash_offset_t

typedef uint32_t flash_offset_t

Type of a flash offset.

7.8.4.2 flash_sector_t

typedef uint32_t flash_sector_t

Type of a flash sector number.

7.8.5 Enumeration Type Documentation

7.8.5.1 flash_state_t

enum flash_state_t

Driver state machine possible states.

7.8.5.2 flash_error_t

enum flash_error_t

Type of a flash error code.

7.8.6 Function Documentation

7.8.6.1 flashWaitErase()

flash_error_t flashWaitErase ( BaseFlash * devp )

Waits until the current erase operation is finished.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>devp</td>
<td>pointer to a BaseFlash object</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Name</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_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>

7.8.6.2 flashGetSectorOffset()

```c
flash_offset_t flashGetSectorOffset (
    BaseFlash * devp,
    flash_sector_t sector )
```

Returns the offset of a sector.

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>sector</td>
<td>flash sector number</td>
</tr>
</tbody>
</table>

Returns

the offset of the sector

7.8.6.3 flashGetSectorSize()

```c
uint32_t flashGetSectorSize ( 
    BaseFlash * devp,
    flash_sector_t sector )
```

Returns the size of a sector.

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>sector</td>
<td>flash sector number</td>
</tr>
</tbody>
</table>

Returns

the size of the sector
7.9 GPT Driver

Generic GPT Driver.

7.9.1 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.

7.9.2 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).
7.9.3 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 *gpt)`  
  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.

- void 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.

7.9.4 Macro Definition Documentation

7.9.4.1 gptChangeIntervalI

```c
#define gptChangeIntervalI( 
    gptp, 
    interval ) \
    gpt_lld_change_interval(gptp, interval); \
```

**Value:**

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>in</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:**

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

7.9.4.2 gptGetIntervalX

```c
#define gptGetIntervalX( 
    gptp ) gpt_lld_get_interval(gptp) 
```

Returns the interval of GPT peripheral.
Precondition

The GPT unit must be running in continuous mode.

Parameters

| in  | gptp | 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.

7.9.4.3  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

| in  | gptp | 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.
7.9.4.4 _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.

7.9.4.5 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.

7.9.4.6 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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>gptp</strong></td>
<td>pointer to a <code>GPTDriver</code> object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>interval</strong></td>
<td>new cycle time in timer ticks</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.9.5 Typedef Documentation

#### 7.9.5.1 GPTDriver

typedef struct GPTDriver GPTDriver

Type of a structure representing a GPT driver.

#### 7.9.5.2 gptcallback_t

typedef void(* gptcallback_t) (GPTDriver *gptp)

GPT notification callback type.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>gptp</strong></td>
<td>pointer to a <code>GPTDriver</code> object</td>
</tr>
</tbody>
</table>
7.9.5.3  gptfreq_t

typedef uint32_t gptfreq_t

GPT frequency type.

7.9.5.4  gptcnt_t

typedef uint16_t gptcnt_t

GPT counter type.

7.9.6  Enumeration Type Documentation

7.9.6.1  gptstate_t

enum gptstate_t

Driver state machine possible states.

Enumerator

<table>
<thead>
<tr>
<th>gptstate_t</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>

7.9.7  Function Documentation

7.9.7.1  gptInit()

void gptInit {
    void
}

GPT Driver initialization.
7.9 GPT Driver

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.

Here is the call graph for this function:

```
    gptInit  →  gpt_lld_init  →  gptObjectInit
```

### 7.9.7.2 gptObjectInit()

```c
void gptObjectInit ( 
    GPTDriver * gptp )
```

Initializes the standard part of a \texttt{GPTDriver} structure.

**Parameters**

- \texttt{out gptp} pointer to the \texttt{GPTDriver} object

**Function Class:**

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

### 7.9.7.3 gptStart()

```c
void gptStart ( 
    GPTDriver * gptp, 
    const GPTConfig * config )
```

Configures and activates the GPT peripheral.

**Parameters**

- \texttt{in gptp} pointer to the \texttt{GPTDriver} object
- \texttt{in config} pointer to the \texttt{GPTConfig} 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:

7.9.7.4  gptStop()

void gptStop (
    GPTDriver * gptp
)

Deactivates the GPT peripheral.

Parameters

| in | gptp | 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]

7.9.7.5 gptChangeInterval()

```c
void gptChangeInterval ( 
    GPTDriver * gtp, 
    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>in</th>
<th>gtp</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:

```
void gptStartContinuous (GPTDriver * gptp, gptcnt_t interval )
```

Starts the timer in continuous 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>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:
### 7.9.7.7 gptStartContinuousI()

```c
void gptStartContinuousI ( 
    GPTDriver * gptp,
    gptcnt_t interval )
```

Starts the timer in continuous 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>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
```

### 7.9.7.8 gptStartOneShot()

```c
void gptStartOneShot ( 
    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:

```
void gptStartOneShotI (  
   GPTDriver * gtp,  
   gptcnt_t interval  
)
```

Starts the timer in one shot mode.

Parameters

| in  | gtp  | pointer to the GPTDriver object |
| in  | interval | time interval in ticks |

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:
7.9.7.10 gptStopTimer()

```c
void gptStopTimer ( 
    GPTDriver  ∗ gptp )
```

Stops the timer.

**Parameters**

```
in  gptp  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]

7.9.7.11 gptStopTimerI()

```c
void gptStopTimerI ( 
    GPTDriver  ∗ gptp )
```

Stops the timer.

**Parameters**

```
in  gptp  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 → gpt_lld_stop_timer
```

### 7.9.7.12 gptPolledDelay()

```c
void gptPolledDelay (  
    GPTDriver ∗ gtp,  
    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>gtp</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:

```
gptPolledDelay → gpt_lld_polled_delay
```
7.9.7.13  gpt_lld_init()

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:

7.9.7.14  gpt_lld_start()

void gpt_lld_start (  
    GPTDriver  * gptp  )

Configures and activates the GPT peripheral.

Parameters

| in | gptp | pointer to the GPTDriver object |

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

7.9.7.15  gpt_lld_stop()

void gpt_lld_stop (  
    GPTDriver  * gptp  )

Deactivates the GPT peripheral.
Parameters

| in  | gptp | pointer to the GPTDriver object |

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

7.9.7.16 gpt_lld_start_timer()

```c
void gpt_lld_start_timer (  
    GPTDriver * gptp,  
    gptcnt_t interval )
```

Starts the timer in continuous mode.

Parameters

| in  | gptp | pointer to the GPTDriver object |

| in  | interval | period in ticks |

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

7.9.7.17 gpt_lld_stop_timer()

```c
void gpt_lld_stop_timer (  
    GPTDriver * gptp )
```

Stops the timer.

Parameters

| in  | gptp | pointer to the GPTDriver object |

Function Class:
Not an API, this function is for internal use only.
7.9.7.18  gpt_lld_polled_delay()

```c
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>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:

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

### 7.9.8  Variable Documentation

#### 7.9.8.1  GPTD1

```c
GPTDriver GPTD1
```

GPTD1 driver identifier.
7.10 I2C Driver

Generic I2C Driver.

7.10.1 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`.

7.10.2 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).

![State Machine Diagram]

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.
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)`
  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 I2CDriver I2CDriver
  Type of a structure representing an I2C driver.

## Data Structures

- struct I2CConfig
  Type of I2C driver configuration structure.
- struct I2CDriver
  Structure representing an I2C driver.

## Functions

- void i2cInit (void)
  I2C Driver initialization.
- void i2cObjectInit (I2CDriver *i2cp)
  Initializes the standard part of a I2CDriver structure.
- void 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.
- 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.

7.10.3 Macro Definition Documentation

7.10.3.1 I2C_NO_ERROR

#define I2C_NO_ERROR 0x00

No error.

7.10.3.2 I2C_BUS_ERROR

#define I2C_BUS_ERROR 0x01

Bus Error.

7.10.3.3 I2C_ARBITRATION_LOST

#define I2C_ARBITRATION_LOST 0x02

Arbitration Lost.

7.10.3.4 I2C_ACK_FAILURE

#define I2C_ACK_FAILURE 0x04

Acknowledge Failure.
7.10.3.5 I2C_OVERRUN

#define I2C_OVERRUN 0x08
Overrun/Underrun.

7.10.3.6 I2C_PEC_ERROR

#define I2C_PEC_ERROR 0x10
PEC Error in reception.

7.10.3.7 I2C_TIMEOUT

#define I2C_TIMEOUT 0x20
Hardware timeout.

7.10.3.8 I2C_SMB_ALERT

#define I2C_SMB_ALERT 0x40
SMBus Alert.

7.10.3.9 I2C_USE_MUTUAL_EXCLUSION

#define I2C_USE_MUTUAL_EXCLUSION TRUE
Enables the mutual exclusion APIs on the I2C bus.

7.10.3.10 _i2c_wakeup_isr

#define _i2c_wakeup_isr(
    i2cp )

Value:

    osalSysLockFromISR();
    osalThreadResume1(&(i2cp)->thread, MSG_OK);
    osalSysUnlockFromISR();
} while (0)

Wakes up the waiting thread notifying no errors.
7.10 I2C Driver

Parameters

| in     | i2cp | pointer to the I2CDriver object |

Function Class:

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

7.10.3.11 _i2c_wakeup_error_isr

#define _i2c_wakeup_error_isr(
    i2cp )

Value:

do {
    osalSysLockFromISR();
    osalThreadResumeI(&(i2cp)->thread, MSG_RESET);
    osalSysUnlockFromISR();
} while (0)

Wakes up the waiting thread notifying errors.

Parameters

| in     | i2cp | pointer to the I2CDriver object |

Function Class:

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

7.10.3.12 i2cMasterTransmit

#define i2cMasterTransmit(
    i2cp,
    addr,
    txbuf,
    txbytes,
    rxbuf,
    rxbytes )

Value:

{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.
7.10.3.13 i2cMasterReceive

#define i2cMasterReceive(
    i2cp,
    addr,
    rxbuf,
    rxbytes ) {i2cMasterReceiveTimeout(i2cp, addr, rxbuf, 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.

7.10.3.14 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.

7.10.3.15 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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
</tr>
</tbody>
</table>

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

7.10.4 Typedef Documentation
7.10.4.1  i2caddr_t

typedef uint16_t i2caddr_t

Type representing an I2C address.

7.10.4.2  i2cflags_t

typedef uint32_t i2cflags_t

Type of I2C Driver condition flags.

7.10.4.3  I2CDriver

typedef struct I2CDriver I2CDriver

Type of a structure representing an I2C driver.

7.10.5  Enumeration Type Documentation

7.10.5.1  i2cstate_t

enum i2cstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerations</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>
7.10.6 Function Documentation

7.10.6.1 i2cInit()

```c
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:

![Call graph](image)

7.10.6.2 i2cObjectInit()

```c
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:

```
ipcObjectInit  
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>osalMutexObjectInit</td>
</tr>
</tbody>
</table>
```

### 7.10.6.3 i2cStart()

```c
def 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>

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:

```
i2cStart  
i2c_lld_start  
osalSysLock  
osalSysUnlock  
```
7.10.6.4  i2cStop()

```c
void i2cStop ( 
    I2CDriver * i2cp )
```

Deactivates the I2C peripheral.

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:

```
i2cStop
    └-- i2c_lld_stop
        ├── osalSysLock
        └-- osalSysUnlock
```

7.10.6.5  i2cGetErrors()

```c
i2cflags_t i2cGetErrors ( 
    I2CDriver * i2cp )
```

Returns the errors mask associated to the previous operation.

Parameters

| in | i2cp | pointer to the I2CDriver object |

Returns

The errors mask.
7.10 I2C Driver

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

7.10.6.6 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>in</th>
<th><code>i2cp</code></th>
<th>pointer to the I2CDriver object</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

• `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. |

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.10.6.7  i2cMasterReceiveTimeout()

```c
msg_t i2cMasterReceiveTimeout ( 
    I2CDriver * i2cp,
    i2caddr_t addr,
    uint8_t * rxbuf,
    size_t rbytes,
    sysinterval_t timeout )
```

Receives data from the I2C bus.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>i2cp</code></td>
<td>pointer to the <code>I2CDriver</code> object</td>
</tr>
<tr>
<td>in <code>addr</code></td>
<td>slave device address (7 bits) without R/W bit</td>
</tr>
<tr>
<td>out <code>rxbuf</code></td>
<td>pointer to receive buffer</td>
</tr>
<tr>
<td>in <code>rbytes</code></td>
<td>number of bytes to be received</td>
</tr>
<tr>
<td>in <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

<table>
<thead>
<tr>
<th>Return 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.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.10.6.8  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.
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:

7.10.6.9  i2cReleaseBus()

void 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:
7.10.6.10  i2c_lld_init()

```c
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
```

7.10.6.11  i2c_lld_start()

```c
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.

7.10.6.12  i2c_lld_stop()

```c
void i2c_lld_stop ( I2CDriver * i2cp )
```

Deactivates the I2C peripheral.
7.10 I2C Driver

Parameters

| in  | i2cp  | pointer to the I2CDriver object |

Function Class:

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

7.10.6.13  i2c_lld_master_receive_timeout()

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.

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.
7.10.6.14  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><code>i2cp</code> pointer to the <code>I2CDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>addr</code> slave device address</td>
</tr>
<tr>
<td>in</td>
<td><code>txbuf</code> pointer to the transmit buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>txbytes</code> number of bytes to be transmitted</td>
</tr>
<tr>
<td>out</td>
<td><code>rxbuf</code> pointer to the receive buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>rxbytes</code> number of bytes to be received</td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code> 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**

<table>
<thead>
<tr>
<th>Value</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. **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.

7.10.7  Variable Documentation

7.10.7.1  I2CD1

```c
I2CDriver  I2CD1
```

I2C1 driver identifier.
7.11 I2S Driver

Generic I2S Driver.

7.11.1 Detailed Description

Generic I2S Driver.

This module implements a generic I2S driver.

Precondition

In order to use the I2S driver the \texttt{HAL\_USE\_I2S} option must be enabled in \texttt{halconf.h}.

7.11.2 Driver State Machine

Macros

- \#define i2s\_lld\_driver\_fields
  
  \textit{Low level fields of the I2S driver structure.}

- \#define i2s\_lld\_config\_fields
  
  \textit{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.
- void 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.
7.11 I2S Driver

Variables

- **I2SDriver I2SD1**
  
  I2S2 driver identifier.

7.11.3 Macro Definition Documentation

7.11.3.1 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

- **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.

7.11.3.2 i2sStartExchangeI

```c
#define i2sStartExchangeI( i2sp )

Value:

i2s_lld_start_exchange(i2sp);
(i2sp)->state = I2S_ACTIVE;
```

Starts a I2S data exchange.
Module Documentation

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.

7.11.3.3 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.

7.11.3.4 _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.

### 7.11.3.5 _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.

### 7.11.3.6 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.
Note

The default is FALSE.

7.11.3.7  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.

7.11.3.8  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.

7.11.4  Typedef Documentation

7.11.4.1  I2SDriver

typedef struct hal_i2s_driver I2SDriver

Type of a structure representing an I2S driver.

7.11.4.2  I2SConfig

typedef struct hal_i2s_config I2SConfig

Type of a structure representing an I2S driver configuration.

7.11.4.3  i2scallback_t

typedef void(* i2scallback_t) (I2SDriver *i2sp)

I2S notification callback type.
Parameters

in i2sp pointer to the I2SDriver object

7.11.5 Enumeration Type Documentation

7.11.5.1 i2sstate_t

denum 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>

7.11.6 Function Documentation

7.11.6.1 i2sInit()

void i2sInit ( )

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:

```
    i2sInit —> i2s_lld_init —> i2sObjectInit
```

### 7.11.6.2 i2sObjectInit()

```c
void i2sObjectInit ( I2SDriver * i2sp )
```

Initializes the standard part of a `I2SDriver` structure.

**Parameters**

- `out i2sp` pointer to the `I2SDriver` object

Function Class:

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

### 7.11.6.3 i2sStart()

```c
void i2sStart ( I2SDriver * i2sp,
                const I2SConfig * config )
```

Configures and activates the I2S peripheral.

**Parameters**

- `in i2sp` pointer to the `I2SDriver` object
- `in config` pointer to the `I2SConfig` object
7.11 I2S 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:

![Call Graph]

7.11.6.4 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
    ↓
i2s_lld_stop
    ↓
osalSysLock
    ↓
osalSysUnlock
```

### 7.11.6.5 i2sStartExchange()

```c
void i2sStartExchange (  
    I2SDriver ∗ i2sp )
```

Starts a I2S data exchange.

**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:

```
i2sStartExchange
    ↓
osalSysLock
    ↓
osalSysUnlock
```
### 7.11.6.6 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 I2S Driver 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:

```
 i2sStopExchange
     |     |
     v     v
osalSysLock
     |
osalSysUnlock
```

---

### 7.11.6.7 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:

```
 i2s_lld_init
     |     |
     v     v
i2sObjectInit
```

---

ChibiOS/HAL
7.11.6.8  i2s_lld_start()

void i2s_lld_start (  
   I2SDriver * i2sp )

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>
</tbody>
</table>

Function Class:

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

7.11.6.9  i2s_lld_stop()

void i2s_lld_stop (  
   I2SDriver * i2sp )

Deactivates 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>
</tbody>
</table>

Function Class:

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

7.11.6.10  i2s_lld_start_exchange()

void i2s_lld_start_exchange (  
   I2SDriver * i2sp )

Starts a I2S data exchange.

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>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.11.6.11 i2s_lld_stop_exchange()

```c
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**

| in | i2sp | pointer to the I2SDriver object |

**Function Class:**

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

---

### 7.11.7 Variable Documentation

#### 7.11.7.1 I2SD1

```c
I2SDriver I2SD1
```

I2S2 driver identifier.
7.12 ICU Driver

Generic ICU Driver.

7.12.1 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**.

7.12.2 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]

7.12.3 ICU Operations.

This driver abstracts a generic Input Capture Unit composed of:

- A clock prescaler.
- A main up counter.

ChibiOS/HAL
• 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 icuStartCaptureI(icup)`
  Starts the input capture.
• `#define icuStopCaptureI(icup)`
  Stops the input capture.
• `#define icuEnableNotificationsI(icup) icu_lld_enable_notifications(icup)`
  Enables notifications.
• `#define icuDisableNotificationsI(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.
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.
- void 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.
7.12 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.

7.12.4 Macro Definition Documentation

**7.12.4.1 icuStartCaptureI**

```c
#define icuStartCaptureI(icup) 
  do { 
    icu_lld_start_capture(icup);
    (icup)->state = ICU_WAITING;
  } while (false)
```

**Value:**

Starts the input capture.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>icup</strong></td>
<td>pointer to the <strong>ICUDriver</strong> 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.

7.12.4.2  icuStopCaptureI

#define icuStopCaptureI( 
     icup )

Value:

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.

7.12.4.3  icuEnableNotificationsI

#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.

### 7.12.4.4 icuDisableNotificationsI

```
#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

- `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.

### 7.12.4.5 icuAreNotificationsEnabledX

```
#define icuAreNotificationsEnabledX(
    icup ) icu_lld_are_notifications_enabled(icup)
```

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.

7.12.4.6 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.

7.12.4.7 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.
### 7.12 ICU Driver

**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.

#### 7.12.4.8 _icu_isr_invoke_width_cb

```c
#define _icu_isr_invoke_width_cb(icup)
```

**Value:**

```c
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 **ICUDriver** object |

**Function Class:**

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

#### 7.12.4.9 _icu_isr_invoke_period_cb

```c
#define _icu_isr_invoke_period_cb(icup)
```

**Value:**

```c
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

in icup pointer to the ICU driver object

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

7.12.4.10 _icu_isr_invoke_overflow_cb

#define _icu_isr_invoke_overflow_cb(
    icup
)

Value:

do { 
    (icup)->config->overflow_cb(icup);
    (icup)->state = ICU_WAITING;
} while (0)

Common ISR code, ICU timer overflow event.

Note
An overflow always brings the driver back to the ICU_WAITING state.

Parameters

in icup pointer to the ICU driver object

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

7.12.4.11 PLATFORM_ICU_USE_ICU1

#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.
7.12.4.12  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.

7.12.4.13  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.

7.12.4.14  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.

### 7.12.5 Typedef Documentation

#### 7.12.5.1 ICUDriver

typedef struct ICUDriver ICUDriver

Type of a structure representing an ICU driver.

#### 7.12.5.2 icucallback_t

typedef void(* icucallback_t)(ICUDriver *icup)

ICU notification callback type.

Parameters

| in  | icup   | pointer to a ICUDriver object |

#### 7.12.5.3 icufreq_t

typedef uint32_t icufreq_t

ICU frequency type.
7.12.5.4 icucnt_t

typedef uint32_t icucnt_t

ICU counter type.

7.12.6 Enumeration Type Documentation

7.12.6.1 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>

7.12.6.2 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>

7.12.7 Function Documentation
7.12.7.1 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]

```
  icuInit  icu_lld_init  icuObjectInit
```

7.12.7.2 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.

7.12.7.3 icuStart()

```c
void icuStart (  
        ICUDriver ∗ icup,  
        const ICUConfig ∗ config )
```
Configures and activates 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>

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:

```
    icuStart
     |            |
     v            v
icu_lld_start osalSysLock
                |
                v
osalSysUnlock
```

7.12.7.4 icuStop()

```c
void icuStop ( 
        ICUDriver * icup )
```

Deactivates the ICU peripheral.

Parameters

| in | icup | pointer to the ICUDriver object |
7.12 ICU 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:

7.12.7.5 icuStartCapture()

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:
7.12.7.6 icuWaitCapture()

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:

```
7.12.7.7 icuStopCapture()

void icuStopCapture (  
    ICUDriver * icup  )
```

Stops the input capture.

Parameters
- `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:
7.12.7.8  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**

- `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:

```
    icuEnableNotifications
        ↘
        ↘
    osalSysLock
    osalSysUnlock
```

7.12.7.9  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.
7.12 ICU Driver

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icup</td>
<td>ICUDriver object pointer</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:

icuDisableNotifications → icu_lld_init → icuObjectInit
osalSysLock
osalSysUnlock

7.12.7.10 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

7.12.7.11 icu_lld_start()

```c
void icu_lld_start (ICUDriver * icup)
```

Configures and activates the ICU peripheral.
Parameters

\[
\begin{array}{lll}
\text{in} & \text{icup} & \text{pointer to the ICUDriver object}
\end{array}
\]

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

### 7.12.7.12 icu_lld_stop()

```c
void icu_lld_stop (  
    ICUDriver * icup  
)
```

Deactivates the ICU peripheral.

Parameters

\[
\begin{array}{lll}
\text{in} & \text{icup} & \text{pointer to the ICUDriver object}
\end{array}
\]

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

### 7.12.7.13 icu_lld_start_capture()

```c
void icu_lld_start_capture (  
    ICUDriver * icup  
)
```

Starts the input capture.

Parameters

\[
\begin{array}{lll}
\text{in} & \text{icup} & \text{pointer to the ICUDriver object}
\end{array}
\]

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

### 7.12.7.14 icu_lld_wait_capture()

```c
bool icu_lld_wait_capture (  
    ICUDriver * icup  
)
```

Waits for a completed capture.
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.

### 7.12.7.15 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.

### 7.12.7.16 icu_lld_enable_notifications()

```c
void icu_lld_enable_notifications ( ICUDriver * icup )
```

Enables notifications.
Precondition

The ICU unit must have been activated using `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.

7.12.7.17  icu_lld_disable_notifications()

```c
void icu_lld_disable_notifications ( 
    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.

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.

7.12.8  Variable Documentation
7.12.8.1 ICUD1

ICUDriver ICUD1

ICUD1 driver identifier.

Note

The driver ICUD1 allocates the complex timer TIM1 when enabled.
7.13 Blocks

7.13.1 Detailed Description

Modules

- Abstract I/O Block Device
7.14 Streams

7.14.1 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 Streams Class
- Output Formatter Utility
- Input Formatter Utility
7.15 Abstract Streams Interface

7.15.1 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.

7.15.2 Macro Definition Documentation

7.15.2.1 _base_sequential_stream_methods

#define _base_sequential_stream_methods

Value:

_value = _base_object_methods

// Stream write buffer method.
size_t (*write)(void *instance, const uint8_t *bp, size_t n);

// Stream read buffer method.
size_t (*read)(void *instance, uint8_t *bp, size_t n);

// Channel put method, blocking.
msg_t (*put)(void *instance, uint8_t b);

// Channel get method, blocking.
msg_t (*get)(void *instance);

BaseSequentialStream specific methods.

7.15.2.2 _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.

7.15.2.3 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>pointer to a <code>BaseSequentialStream</code> or derived class</td>
</tr>
<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.

#### 7.15.2.4 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>pointer to a <code>BaseSequentialStream</code> 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. 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.

#### 7.15.2.5 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.

7.15.2.6 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. |
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.15.2.7 _base_buffered_stream_methods

```c
#define _base_buffered_stream_methods

BaseBufferedStream specific methods.
```

### 7.15.2.8 _base_buffered_stream_data

```c
#define _base_buffered_stream_data _base_sequential_stream_data

BaseBufferedStream specific data.
```

**Note**

It is empty because `BaseBufferedStream` is only an interface without implementation.

### 7.15.2.9 streamUnget

```c
#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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pointer to a <code>BaseBufferedStream</code> or derived class</td>
</tr>
<tr>
<td>in</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.
7.16 Abstract I/O Channel Interface

7.16.1 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.

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.

7.16.2 Macro Definition Documentation

7.16.2.1 CHN_CTL_INVALID

`#define CHN_CTL_INVALID 0`

Invalid operation code.
7.16.2.2 CHN_CTL_NOP

#define CHN_CTL_NOP 1

Does nothing.

7.16.2.3 CHN_CTL_TX_WAIT

#define CHN_CTL_TX_WAIT 2

Wait for TX completion.

7.16.2.4 _base_channel_methods

#define _base_channel_methods

Value:

 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.

7.16.2.5 _base_channel_data

#define _base_channel_data _base_sequential_stream_data

 BaseChannel specific data.

 Note

 It is empty because BaseChannel is only an interface without implementation.

7.16.2.6 chnPutTimeout

#define chnPutTimeout(
    ip,
    b,
    time ) ((ip)->vmt->putt(ip, b, time))

Channel blocking byte write with timeout.

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
<td><strong>ip</strong> pointer to a <strong>BaseChannel</strong> or derived class</td>
</tr>
<tr>
<td>in</td>
<td></td>
<td><strong>b</strong> 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>• <strong>TIME_IMMEDIATE</strong> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITE</strong> no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STM_OK</strong></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><strong>STM_TIMEOUT</strong></td>
<td>if the specified time expired.</td>
</tr>
<tr>
<td><strong>STM_RESET</strong></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.

7.16.2.7 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></td>
<td><strong>ip</strong> pointer to a <strong>BaseChannel</strong> 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>• <strong>TIME_IMMEDIATE</strong> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITE</strong> no timeout.</td>
</tr>
</tbody>
</table>

Returns

A byte value from the queue.
## Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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) 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.

### 7.16.2.8 chnWrite

```c
#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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>pointer to a <code>BaseChannel</code> or derived class</td>
</tr>
<tr>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<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.

### 7.16.2.9 chnWriteTimeout

```c
#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>Type</th>
<th>Name</th>
<th>Description</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>
<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.

7.16.2.10 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>Type</th>
<th>Name</th>
<th>Description</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>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.
### 7.16.2.11 chnReadTimeout

```c
#define chnReadTimeout(
    ip,
    bp,
    n,
    time ) ((ip)->vmt->readt(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 <code>BaseChannel</code> 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>- <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 transferred.

**Function Class:**

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

### 7.16.2.12 chnControl

```c
#define chnControl(
    ip,
    operation,
    arg ) ((ip)->vmt->ctl(ip, operation, arg))
```

Control operation on a channel.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a <code>BaseChannel</code> or derived class</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:

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

7.16.2.13 CHN_NO_ERROR

#define CHN_NO_ERROR (eventflags_t)0

No pending conditions.

7.16.2.14 CHN_CONNECTED

#define CHN_CONNECTED (eventflags_t)1

Connection happened.

7.16.2.15 CHN_DISCONNECTED

#define CHN_DISCONNECTED (eventflags_t)2

Disconnection happened.

7.16.2.16 CHN_INPUT_AVAILABLE

#define CHN_INPUT_AVAILABLE (eventflags_t)4

Data available in the input queue.
7.16.2.17 CHN_OUTPUT_EMPTY

#define CHN_OUTPUT_EMPTY (eventflags_t)8

Output queue empty.

7.16.2.18 CHN_TRANSMISSION_END

#define CHN_TRANSMISSION_END (eventflags_t)16

Transmission end.

7.16.2.19 _base_asynchronous_channel_methods

#define _base_asynchronous_channel_methods _base_channel_methods \ 
BaseAsynchronousChannel specific methods.

7.16.2.20 _base_asynchronous_channel_data

#define _base_asynchronous_channel_data

Value:

_base_channel_data
/* I/O condition event source. */
event_source_t event;

BaseAsynchronousChannel specific data.

7.16.2.21 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.

7.16.2.22 chnAddFlagsI

#define chnAddFlagsI(ip, flags)

Value:

osalEventBroadcastFlagsI(&(ip)->event, flags);

Adds status flags to the listener'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.


7.17 Abstract Files Interface

7.17.1 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 fileStreamGetPosition(ip, offset) ((ip)->vmt->GetPosition(ip, offset))
Moves the file current pointer to an absolute position.
- `#define fileStreamGetOffset(ip, offset) ((ip)->vmt->GetOffset(ip, offset))
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))
Returns the current file pointer position.
- `#define fileStreamRead(ip, bp, n) streamRead(ip, bp, n)
File stream read.
7.17 Abstract Files Interface

**Typedefs**

- typedef uint32_t fileoffset_t
  
  *File offset type.*

**Data Structures**

- struct FileStreamVMT
  
  *FileSteam virtual methods table.*

- struct FileStream
  
  *Base file stream class.*

### 7.17.2 Macro Definition Documentation

#### 7.17.2.1 FILE_OK

#define FILE_OK STM_OK

No error return code.

#### 7.17.2.2 FILE_ERROR

#define FILE_ERROR STM_TIMEOUT

Error code from the file stream methods.

#### 7.17.2.3 FILE_EOF

#define FILE_EOF STM_RESET

End-of-file condition for file get/put methods.
7.17.2.4  _file_stream_methods

#define _file_stream_methods

Value:

_base_sequential_stream_methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg_t (*close)(void *instance);</td>
<td>File close method.</td>
</tr>
<tr>
<td>msg_t (*geterror)(void *instance);</td>
<td>Get last error code method.</td>
</tr>
<tr>
<td>msg_t (*getsize)(void *instance, fileoffset_t *offset);</td>
<td>File get size method.</td>
</tr>
<tr>
<td>msg_t (*getposition)(void *instance, fileoffset_t *offset);</td>
<td>File get current position method.</td>
</tr>
<tr>
<td>msg_t (*setposition)(void *instance, fileoffset_t *offset);</td>
<td>File set current position method.</td>
</tr>
</tbody>
</table>

FileStream specific methods.

7.17.2.5  _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.

7.17.2.6  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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ip</td>
<td>pointer to a FileStream or derived class</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</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.
7.17 Abstract Files Interface

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.

7.17.2.7 fileStreamRead

#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.

7.17.2.8 fileStreamPut

#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.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a <code>FileStream</code> 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></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FILE_OK</code></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><code>FILE_ERROR</code></td>
<td>operation failed.</td>
</tr>
<tr>
<td><code>FILE_EOF</code></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.

7.17.2.9 `fileStreamGet`

```c
#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

| in | ip | pointer to a `FileStream` or derived class |

Returns

A byte value from the queue.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FILE_ERROR</code></td>
<td>operation failed.</td>
</tr>
<tr>
<td><code>FILE_EOF</code></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.
#fileStreamClose

```c
#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.

---

##fileStreamGetError

```c
#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.

### 7.17.2.12 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.

### 7.17.2.13 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>
7.17 Abstract Files Interface

Returns
The current position inside the file.

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.

7.17.2.14 fileStreamSetPosition

```c
#define fileStreamSetPosition(
    ip,
    offset)
    ((ip)->vmt->setPosition(ip, offset))
```

Moves the file current pointer to an absolute position.

Parameters
- **in ip**: pointer to a `FileStream` or derived class
- **in offset**: new absolute position

Returns
The operation status.

Return values
- **FILE_OK**: no error.
- **FILE_ERROR**: operation failed.

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

7.17.3 Typedef Documentation

7.17.3.1 fileoffset_t

```c
typedef uint32_t fileoffset_t
```

File offset type.
7.18 Abstract Persistent Storage Interface

7.18.1 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.
7.18.2 Macro Definition Documentation

7.18.2.1 _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);

BasePersistentStorage specific methods.

7.18.2.2 _base_pers_storage_methods

#define _base_pers_storage_methods

Value:
_base_object_methods
_base_pers_storage_methods_alone

BasePersistentStorage specific methods with inherited ones.

7.18.2.3 _base_persistent_storage_data

#define _base_persistent_storage_data _base_object_data

BasePersistentStorage specific data.

7.18.2.4 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.

7.18.2.5 psGetStorageSize

#define psGetStorageSize(
    ip ) (ip)->vmt->getsize(ip)

Get storage size.
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.

7.18.2.6 psRead

```c
#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 |
| in  | 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>PS_ERROR_READ</td>
<td>if the read operation failed.</td>
</tr>
<tr>
<td>PSERROR_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.
7.18.2.7 psWrite

#define psWrite(
    ip,
    offset,
    n,
    wp ) (ip)->vmt->write(ip, offset, n, wp)

Write operation.

Parameters

| in  | ip            | pointer to a BasePersistentStorage or derived class |
|     | offset        | persistent storage offset                           |
| in  | n             | number of bytes to be written                       |
| in  | wp            | pointer to the data buffer                         |

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.

7.18.3 Typedef Documentation

7.18.3.1 ps_offset_t

typedef uint32_t ps_offset_t

Type of a persistent storage offset.

7.18.4 Enumeration Type Documentation
7.18.4.1 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.
7.19 Memory Streams Class

7.19.1 Detailed Description

Memory buffers handled as streams.

Macros

- #define _memory_stream_data
  MemStream specific data.

Data Structures

- struct MemStreamVMT
  MemStream virtual methods table.
- struct MemStream
  Memory stream object.

Functions

- void msObjectInit (MemoryStream *msp, uint8_t *buffer, size_t size, size_t eos)
  Memory stream object initialization.

7.19.2 Macro Definition Documentation

7.19.2.1 _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.

7.19.3 Function Documentation

7.19.3.1 msObjectInit()

void msObjectInit ( MemoryStream * msp,
                    uint8_t * buffer,
                    size_t size,
                    size_t eos )

Memory stream object initialization.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>msp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>buffer</code></td>
</tr>
<tr>
<td>in</td>
<td><code>size</code></td>
</tr>
<tr>
<td>in</td>
<td><code>eos</code></td>
</tr>
</tbody>
</table>
7.20 Null Streams Class

7.20.1 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.

7.20.2 Macro Definition Documentation

7.20.2.1 _null_stream_data

\#define _null_stream_data _base_buffered_stream_data

*NullStream* specific data.

7.20.3 Function Documentation

7.20.3.1 nullObjectInit()

void nullObjectInit ( 
  
  *NullStream* *nsp *)

Null stream object initialization.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td>nsp</td>
</tr>
</tbody>
</table>
7.21 Buffered Streams Class

7.21.1 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.

7.21.2 Macro Definition Documentation

7.21.2.1 BUFSTREAM_BUFFER_SIZE

`#define BUFSTREAM_BUFFER_SIZE 1`

Buffer size for unget.

7.21.2.2 _buffered_stream_adapter_data

`#define _buffered_stream_adapter_data`

Value:

```c
_base_buffered_stream_data
/* Pointer to a wrapped BaseSequentialStream object */
BaseSequentialStream* bssp;
/* Stream buffer */
uint8_t buffer[BUFSTREAM_BUFFER_SIZE];
/* Index in the stream buffer */
size_t ndx;
```

BufferedStreamAdapter specific data.
7.21.3 Function Documentation

7.21.3.1 bsaObjectInit()

```c
void bsaObjectInit (  
    BufferedStreamAdapter * bsap,
    BaseSequentialStream * bssp )
```

Buffered stream adapter object initialization.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td><code>bsap</code></td>
<td>pointer to the <code>BufferedStream</code> object to be initialized</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>bssp</code></td>
<td>pointer to a <code>BaseSequentialStream</code> fulfilling object to be wrapped</td>
</tr>
</tbody>
</table>
7.22 Output Formatter Utility

7.22.1 Detailed Description

Mini printf-like functionality.

Macros

- #define CHPRINTF_USE_FLOAT FALSE
  Float type support.

Functions

- int chvpprintf (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.

7.22.2 Macro Definition Documentation

7.22.2.1 CHPRINTF_USE_FLOAT

#define CHPRINTF_USE_FLOAT FALSE

Float type support.

7.22.3 Function Documentation
7.22.3.1 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][+][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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>chp</code> pointer to a <code>BaseSequentialStream</code> implementing object</td>
</tr>
<tr>
<td>in</td>
<td><code>fmt</code> formatting string</td>
</tr>
<tr>
<td>in</td>
<td><code>ap</code> list of parameters</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.

7.22.3.2 chprintf()

```c
int chprintf (  
    BaseSequentialStream * chp,  
    ChibiOS/HAL
```
const char * fmt,
... )

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]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

| in | chp | pointer to a BaseSequentialStream implementing object |
| in | fmt | formatting string |

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.

Here is the call graph for this function:
7.22.3.3 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][llL]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.
7.22 Output Formatter Utility

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)

7.22.3.4 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][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.

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>
<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:
7.23 Input Formatter Utility

7.23.1 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.

7.23.2 Macro Definition Documentation

7.23.2.1 CHSCANF_USE_FLOAT

#define CHSCANF_USE_FLOAT FALSE

Float type support.

7.23.3 Function Documentation
### 7.23.3.1 chvscanf()

```c
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: `%(type[<width>[<flags]>]<length>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><code>chp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>fmt</code></td>
</tr>
<tr>
<td>in</td>
<td><code>ap</code></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.

### 7.23.3.2 chscanf()

```c
int chscanf (    
    BaseBufferedStream * chp,
```
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][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>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>
</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
```
### chsnscanf()

```c
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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>str</code> pointer to a buffer</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>size</code> size of the buffer</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>fmt</code> 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:

```
chsnscanf → chvsnscanf → chvscanf
```

7.23.3.4 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:

\%\([+][-][\text{width}][\text{lL}][\text{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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in str</td>
<td>pointer to a buffer</td>
</tr>
<tr>
<td>in size</td>
<td>size of the buffer</td>
</tr>
<tr>
<td>in fmt</td>
<td>formatting string</td>
</tr>
<tr>
<td>in 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.

Here is the call graph for this function:
7.24 Abstract I/O Block Device

7.24.1 Detailed Description

7.24.2 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.

7.24.3 Macro Definition Documentation

7.24.3.1 _base_block_device_methods

#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.
7.24 Abstract I/O Block Device

7.24.3.2 _base_block_device_data

#define _base_block_device_data

Value:
_base_object_data

/* Driver state. */
blkstate_t state;

BaseBlockDevice specific data.

7.24.3.3 blkGetDriverState

#define blkGetDriverState(

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.

7.24.3.4 blkIsTransferring

#define blkIsTransferring(

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.

ChibiOS/HAL
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.

7.24.3.5 blkIsInserted

#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.
7.24 Abstract I/O Block Device

7.24.3.6 blkIsWriteProtected

```c
#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.

7.24.3.7 blkConnect

```c
#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.

7.24.3.8 blkDisconnect

#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

<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.

7.24.3.9 blkRead

#define blkRead(
    ip,
    startblk,
    buf,
    n ) ((ip)->vmt->read(ip, startblk, buf, n))

Reads one or more blocks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseBlockDevice or derived class</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

| 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.

7.24.3.10  blkWrite

```c
#define blkWrite(
    ip,
    startblk,
    buf,
    n ) ((ip)->vmt->write(ip, startblk, buf, n))
```

Writes one or more blocks.

Parameters

| in  | ip          | pointer to a BaseBlockDevice or derived class |
| in  | startblk    | first block to write                         |
| out | buf         | pointer to the write buffer                  |
| in  | n           | number of blocks to write                    |

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.
7.24.3.11 blkSync

#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.

7.24.3.12 blkGetInfo

#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.

### 7.24.4 Enumeration Type Documentation

#### 7.24.4.1 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
7.25 MAC Driver

Generic MAC Driver.

7.25.1 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.

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.

Macro Functions

• `#define macGetReceiveEventSource(macp) (&(macp)->rdevent)`
  Enables the zero-copy API.

• `#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, sizep) mac_lld_get_next_transmit_buffer(tdp, 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.

PLATFORM configuration options

• `#define PLATFORM_MAC_USE_MAC1 FALSE`
  MAC driver enable switch.
Typedefs

- typedef struct MACDriver MACDriver
  
  Type of a structure representing a MAC driver.

Data Structures

- struct MACConfig
  
  Driver configuration structure.
- struct MACDriver
  
  Structure representing a MAC driver.
- struct MACTransmitDescriptor
  
  Structure representing a transmit descriptor.
- struct MACReceiveDescriptor
  
  Structure representing a receive descriptor.

Functions

- void macInit (void)
  
  MAC Driver initialization.
- void macObjectInit (MACDriver *macp)
  
  Initialize the standard part of a MACDriver structure.
- void macStart (MACDriver *macp, const MACConfig *config)
  
  Configures and activates the MAC peripheral.
- void macStop (MACDriver *macp)
  
  Deactivates the MAC peripheral.
- msg_t macWaitTransmitDescriptor (MACDriver *macp, MACTransmitDescriptor *tdp, sysinterval_t timeout)
  
  Allocates a transmission descriptor.
- void macReleaseTransmitDescriptor (MACTransmitDescriptor *tdp)
  
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t macWaitReceiveDescriptor (MACDriver *macp, MACReceiveDescriptor *rdp, sysinterval_t timeout)
  
  Waits for a received frame.
- void macReleaseReceiveDescriptor (MACReceiveDescriptor *rdp)
  
  Releases a receive descriptor.
- 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, 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.

7.25.2 Macro Definition Documentation

7.25.2.1 MAC_USE_ZERO_COPY

```c
#define MAC_USE_ZERO_COPY FALSE
```

Enables an event sources for incoming packets.

7.25.2.2 MAC_USE_EVENTS

```c
#define MAC_USE_EVENTS TRUE
```

Enables an event sources for incoming packets.

7.25.2.3 macGetReceiveEventSource

```c
#define macGetReceiveEventSource(
    macp ) (&(macp)->rdevent)
```

Enables the zero-copy API.
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.

7.25.2.4 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.

7.25.2.5 macReadReceiveDescriptor

#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>in</th>
<th>rdp</th>
<th>pointer to a MACReceiveDescriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>pointer to the buffer that will receive the read data</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>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.

7.25.2.6 *macGetNextTransmitBuffer*

```
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>in</th>
<th>tdp</th>
<th>pointer to a MACTransmitDescriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</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>out</td>
<td>sizep</td>
<td>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>

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.
7.25.2.7  **macGetNextReceiveBuffer**

```c
#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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>rdp</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>sizep</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.

---

7.25.2.8  **MAC_SUPPORTS_ZERO_COPY**

```c
#define MAC_SUPPORTS_ZERO_COPY TRUE
```

This implementation supports the zero-copy mode API.

---

7.25.2.9  **PLATFORM_MAC_USE_MAC1**

```c
#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**.
7.25.3 Typedef Documentation

7.25.3.1 MACDriver

typedef struct MACDriver MACDriver

Type of a structure representing a MAC driver.

7.25.4 Enumeration Type Documentation

7.25.4.1 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>

7.25.5 Function Documentation

7.25.5.1 macInit()

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_url)

### 7.25.5.2 macObjectInit()

```c
void macObjectInit (MACDriver * macp)
```

Initialize the standard part of a MACDriver structure.

**Parameters**

- `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](call_graph_url)
7.25.5.3 macStart()

```c
void macStart (  
    MACDriver * macp,  
    const MACConfig * config )
```

Configures and activates the MAC peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in macp</td>
<td>pointer to the MACDriver object</td>
</tr>
<tr>
<td>in config</td>
<td>pointer to the MACConfig 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)

7.25.5.4 macStop()

```c
void macStop (  
    MACDriver * macp )
```

Deactivates the MAC peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in macp</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
\|-- mac_lld_stop
\   \|-- osalSysLock
       \|-- osalSysUnlock
```

### 7.25.5.5 `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**

<table>
<thead>
<tr>
<th>in</th>
<th><code>macp</code></th>
<th>pointer to the <code>MACDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>tdp</code></td>
<td>pointer to a <code>MACTransmitDescriptor</code> structure</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_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><code>MSG_OK</code></th>
<th>the descriptor was obtained.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_TIMEOUT</code></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:

7.25.5.6 macReleaseTransmitDescriptor()

```c
void macReleaseTransmitDescriptor ( 
  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:

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:
7.25.5.7  macWaitReceiveDescriptor()

```c
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>• 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**

| MSG_OK | the descriptor was obtained. |
| MSG_TIMEOUT | the operation timed out, descriptor not initialized. |

**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:
7.25.5.8 macReleaseReceiveDescriptor()

`void macReleaseReceiveDescriptor (` `MACReceiveDescriptor * rdp )`

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rdp</td>
<td>the pointer to the <code>MACReceiveDescriptor</code> 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.

Here is the call graph for this function:

![Call Graph](image)

7.25.5.9 macPollLinkStatus()

`bool macPollLinkStatus (` `MACDriver * macp )`

Updates and returns the link status.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>macp</td>
<td>pointer to the <code>MACDriver</code> object</td>
</tr>
</tbody>
</table>

**Returns**

The link status.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if the link is active.</td>
</tr>
<tr>
<td>false</td>
<td>if the link is down.</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:

```
macPollLinkStatus -> mac_lld_poll_link_status
```

### 7.25.5.10 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
```

### 7.25.5.11 mac_lld_start()

```c
void mac_lld_start (MACDriver * macp)
```

Configures and activates the MAC peripheral.
Module Documentation

Parameters

| in  | macp | pointer to the MACDriver object |

Function Class:

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

### 7.25.5.12 mac_lld_stop()

```c
void mac_lld_stop (MACDriver * macp)
```

Deactivates the MAC peripheral.

Parameters

| in  | macp | pointer to the MACDriver object |

Returns

The operation status.

Function Class:

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

### 7.25.5.13 mac_lld_get_transmit_descriptor()

```c
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.

### 7.25.5.14 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.

### 7.25.5.15 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>
7.25.5.16  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.

7.25.5.17  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

<table>
<thead>
<tr>
<th>true</th>
<th>if the link is active.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the link is down.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.25.5.18  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

| 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:

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

7.25.5.19  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

| in | rdp | pointer to a MACReceiveDescriptor structure |
| in | buf | pointer to the buffer that will receive the read data |
| in | size | number of bytes to be read |

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.
7.25.5.20  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.

**Note**

The API guarantees that enough buffers can be requested to fill a whole frame.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>tdp</strong></td>
<td>pointer to a <code>MACTransmitDescriptor</code> structure</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>size</strong></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><strong>out</strong></td>
<td><strong>sizep</strong></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.

7.25.5.21  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.

7.25.6 Variable Documentation

7.25.6.1 ETHD1

MACDriver ETHD1

MAC1 driver identifier.
7.26 Managed Flash Storage Driver

Managed Flash Storage Driver.

7.26.1 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.

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.
- `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_error_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.
• 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.
• void mfsObjectInit (MFSDriver *mfsp)
  Initializes an instance.
• mfs_error_t mfsStart (MFSDriver *mfsp, const MFSConfig *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.

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.

7.26.2 Macro Definition Documentation
### 7.26.2.1 ALIGNED_REC_SIZE

```c
#define ALIGNED_REC_SIZE(n) (flash_offset_t)MFS_ALIGN_NEXT(sizeof (mfs_data_header_t) + (size_t)(n))
```

Data record size aligned.

### 7.26.2.2 ALIGNED_DHDR_SIZE

```c
#define ALIGNED_DHDR_SIZE ALIGNED_REC_SIZE(0)
```

Data record header size aligned.

### 7.26.2.3 ALIGNED_SIZEOF

```c
#define ALIGNED_SIZEOF(t) (((sizeof (t) - 1U) | MFS_ALIGN_MASK) + 1U)
```

Aligned size of a type.

### 7.26.2.4 PAIR

```c
#define PAIR(a, b) (((unsigned)(a) << 2U) | (unsigned)(b))
```

Combines two values (0..3) in one (0..15).

### 7.26.2.5 RET_ON_ERROR

```c
#define RET_ON_ERROR(err)  
 do {  
   mfs_error_t e = (err);  
   if (e != MFS_NO_ERROR) {  
      return e;  
   }  
} while (false)
```

Error check helper.
7.26.2.6 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.

7.26.2.7 MFS_CFG_MAX_REPAIR_ATTEMPTS

#define MFS_CFG_MAX_REPAIR_ATTEMPTS 3

Maximum number of repair attempts on partition mount.

7.26.2.8 MFS_CFG_WRITE_VERIFY

#define MFS_CFG_WRITE_VERIFY TRUE

Verify written data.

7.26.2.9 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.

7.26.2.10 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.
7.26 Managed Flash Storage Driver

7.26.2.11 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.

7.26.2.12 MFS_CFG_TRANSACTION_MAX

#define MFS_CFG_TRANSACTION_MAX 16

Maximum number of objects writable in a single transaction.

7.26.3 Typedef Documentation

7.26.3.1 mfs_id_t

typedef uint32_t mfs_id_t

Type of a record identifier.

7.26.4 Enumeration Type Documentation

7.26.4.1 mfs_bank_t

enum mfs_bank_t

Type of a flash bank.

7.26.4.2 mfs_state_t

enum mfs_state_t

Type of driver state machine states.
7.26.3 mfs_error_t

enum mfs_error_t

Type of an MFS error code.

Note
Errors are negative integers, informative warnings are positive integers.

7.26.4 mfs_bank_state_t

enum mfs_bank_state_t

Type of a bank state assessment.

7.26.5 Function Documentation

7.26.5.1 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>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</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
The operation status.

Function Class:
Not an API, this function is for internal use only.
7.26.5.2  mfs_flash_write()

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.

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>offset</td>
<td>flash offset</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to be written</td>
</tr>
<tr>
<td>in</td>
<td>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.

7.26.5.3  mfs_flash_copy()

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

<table>
<thead>
<tr>
<th>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>doffset</td>
<td>destination flash offset</td>
</tr>
<tr>
<td>in</td>
<td>soffset</td>
<td>source flash offset</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to be copied</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_flash_copy
  \--> mfs_flash_read
  \--> mfs_bank_erase
  \--> mfs_flash_write
```

### 7.26.5.4 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mfsp</code></td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td><code>bank</code></td>
<td>bank to be erased</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:
Not an API, this function is for internal use only.
7.26 Managed Flash Storage Driver

7.26.5.5 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mfsp</td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td>in 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.

7.26.5.6 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mfsp</td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td>in bank</td>
<td>bank to be validated</td>
</tr>
<tr>
<td>in 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.
7.26.5.7  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.

7.26.5.8  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.
7.26.5.9 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:

![Call Graph](mfs_bank_get_state.mfs_flash_read)

7.26.5.10 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.

7.26.5.11 mfs_try_mount()

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.

7.26.5.12 mfs_mount()

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

<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_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.

7.26.5.13  mfsObjectInit()

void mfsObjectInit (  
    MFSDriver * mfsp )

Initializes an instance.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out mfsp</td>
<td>pointer to the MFSDriver object</td>
</tr>
</tbody>
</table>

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

7.26.5.14  mfsStart()

mfs_error_t mfsStart (  
    MFSDriver * mfsp,  
    const MFSConfig * config )

Configures and activates a MFS driver.

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 config</td>
<td>pointer to the configuration</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MFS_NO_ERROR</code></td>
<td>if the operation has been completed.</td>
</tr>
<tr>
<td><code>MFS_WARN_GC</code></td>
<td>if the operation triggered a garbage collection.</td>
</tr>
<tr>
<td><code>MFS_ERR_FLASH_FAILURE</code></td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the <code>MFS_ERROR</code> state.</td>
</tr>
<tr>
<td><code>MFS_ERR_INTERNAL</code></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.

### 7.26.5.15 mfsStop()

```c
void mfsStop (MFSDriver * mfsp )
```

Deactivates a MFS driver.

Parameters

- `in` `msp` pointer to the `MFSDriver` object

Function Class:

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

### 7.26.5.16 mfsErase()

```c
mfs_error_t mfsErase (MFSDriver * mfsp )
```

Destroys the state of the managed storage by erasing the flash.

Parameters

- `in` `msp` pointer to the `MFSDriver` object
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MFS_NO_ERROR</code></td>
<td>if the operation has been successfully completed.</td>
</tr>
<tr>
<td><code>MFS_ERR_INV_STATE</code></td>
<td>if the driver is in not in <code>MFS_READY</code> state.</td>
</tr>
<tr>
<td><code>MFS_ERR_INV_SIZE</code></td>
<td>if the passed buffer is not large enough to contain the record data.</td>
</tr>
<tr>
<td><code>MFS_ERR_NOT_FOUND</code></td>
<td>if the specified id does not exists.</td>
</tr>
<tr>
<td><code>MFS_ERR_FLASH_FAILURE</code></td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the <code>MFS_ERROR</code> state.</td>
</tr>
<tr>
<td><code>MFS_ERR_INTERNAL</code></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.

7.26.5.17 mfsReadRecord()

```c
mfs_error_t mfsReadRecord ( 
    MFSDriver * mfsp,
    mfs_id_t id,
    size_t * np,
    uint8_t * buffer )
```

Retrieves and reads a data record.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mfsp</code></td>
<td>pointer to the <code>MFSDriver</code> object</td>
</tr>
<tr>
<td><code>id</code></td>
<td>record numeric identifier, the valid range is between 1 and <code>MFS_CFG_MAX_RECORDS</code></td>
</tr>
<tr>
<td><code>np</code></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><code>buffer</code></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><code>MFS_NO_ERROR</code></td>
<td>if the operation has been successfully completed.</td>
</tr>
<tr>
<td><code>MFS_ERR_INV_STATE</code></td>
<td>if the driver is in not in <code>MFS_READY</code> state.</td>
</tr>
<tr>
<td><code>MFS_ERR_INV_SIZE</code></td>
<td>if the passed buffer is not large enough to contain the record data.</td>
</tr>
<tr>
<td><code>MFS_ERR_NOT_FOUND</code></td>
<td>if the specified id does not exists.</td>
</tr>
<tr>
<td><code>MFS_ERR_FLASH_FAILURE</code></td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the <code>MFS_ERROR</code> state.</td>
</tr>
<tr>
<td><code>MFS_ERR_INTERNAL</code></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.

### 7.26.5.18  mfsWriteRecord()

*Function Definition*

```c
mfs_error_t mfsWriteRecord(
    MFSDriver ∗ mfsp,
    mfs_id_t id,
    size_t n,
    const uint8_t ∗ buffer)
```

*Description*

Creates or updates a data record.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mfsp</code></td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td><code>id</code></td>
<td>record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS</td>
</tr>
<tr>
<td><code>n</code></td>
<td>size of data to be written, it cannot be zero</td>
</tr>
<tr>
<td><code>buffer</code></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_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 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.

### 7.26.5.19  mfsEraseRecord()

*Function Definition*

```c
mfs_error_t mfsEraseRecord(
    MFSDriver ∗ mfsp,
    mfs_id_t id)
```

*Description*

ChibiOS/HAL
Erases a data record.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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>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_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.

7.26.5.20 mfsPerformGarbageCollection()

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>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_NO_ERROR</td>
<td>if the operation has been successfully completed.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</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.

7.26.5.21 mfsStartTransaction()

```c
mfs_error_t mfsStartTransaction (
    MFSDriver * mfsp,
    size_t size )
```

Puts the driver in transaction mode.

Note

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>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_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.

7.26.5.22  mfsCommitTransaction()

```c
mfs_error_t mfsCommitTransaction (  
    MFSDriver * mfsp )
```

A transaction is committed and finalized atomically.

Parameters
- `mfsp` pointer to the MFSDriver object

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th>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_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.

7.26.5.23  mfsRollbackTransaction()

```c
mfs_error_t mfsRollbackTransaction (  
    MFSDriver * 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>Return 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_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.
7.27 MII/RMII Header

MII/RMII Support Header.

7.27.1 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_ANNPTR 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_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 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

7.27.2 Macro Definition Documentation

7.27.2.1 MII_BMCR

#define MII_BMCR 0x00

Basic mode control register.

7.27.2.2 MII_BMSR

#define MII_BMSR 0x01

Basic mode status register.
7.27.2.3 MII_PHYSID1

#define MII_PHYSID1 0x02
PHYS ID 1.

7.27.2.4 MII_PHYSID2

#define MII_PHYSID2 0x03
PHYS ID 2.

7.27.2.5 MII_ADVERTISE

#define MII_ADVERTISE 0x04
Advertisement control reg.

7.27.2.6 MII_LPA

#define MII_LPA 0x05
Link partner ability reg.

7.27.2.7 MII_EXPANSION

#define MII_EXPANSION 0x06
Expansion register.

7.27.2.8 MII_ANNPTR

#define MII_ANNPTR 0x07
1000BASE-T control.

7.27.2.9 MII_CTRL1000

#define MII_CTRL1000 0x09
1000BASE-T control.
7.27.2.10  MII_STAT1000

#define MII_STAT1000 0x0a

1000BASE-T status.

7.27.2.11  MII_ESTATUS

#define MII_ESTATUS 0x0f

Extended Status.

7.27.2.12  MII_PHYSTS

#define MII_PHYSTS 0x10

PHY Status register.

7.27.2.13  MII_MICR

#define MII_MICR 0x11

MII Interrupt ctrl register.

7.27.2.14  MII_DCOUNTER

#define MII_DCOUNTER 0x12

Disconnect counter.

7.27.2.15  MII_FCSCOUNTER

#define MII_FCSCOUNTER 0x13

False carrier counter.

7.27.2.16  MII_NWAYTEST

#define MII_NWAYTEST 0x14

N-way auto-neg test reg.
7.27.2.17 MII_RERRCOUNTER

#define MII_RERRCOUNTER 0x15

Receive error counter.

7.27.2.18 MII_SREVISION

#define MII_SREVISION 0x16

Silicon revision.

7.27.2.19 MII_RESV1

#define MII_RESV1 0x17

Reserved.

7.27.2.20 MII_LBRERROR

#define MII_LBRERROR 0x18

Lpback, rx, bypass error.

7.27.2.21 MII_PHYADDR

#define MII_PHYADDR 0x19

PHY address.

7.27.2.22 MII_RESV2

#define MII_RESV2 0x1a

Reserved.

7.27.2.23 MII_TPISTATUS

#define MII_TPISTATUS 0x1b

TPI status for 10Mbps.
7.27.2.24  MII\_NCONFIG

#define MII\_NCONFIG 0x1c

Network interface config.

7.27.2.25  BMCR\_RESV

#define BMCR\_RESV 0x007f

Unused.

7.27.2.26  BMCR\_CTST

#define BMCR\_CTST 0x0080

Collision test.

7.27.2.27  BMCR\_FULLDPLX

#define BMCR\_FULLDPLX 0x0100

Full duplex.

7.27.2.28  BMCR\_ANRESTART

#define BMCR\_ANRESTART 0x0200

Auto negotiation restart.

7.27.2.29  BMCR\_ISOLATE

#define BMCR\_ISOLATE 0x0400

Disconnect DP83840 from MII.

7.27.2.30  BMCR\_PDOWN

#define BMCR\_PDOWN 0x0800

Powerdown.
7.27.2.31 **BMCR_ANENABLE**

```c
#define BMCR_ANENABLE 0x1000
```
Enable auto negotiation.

7.27.2.32 **BMCR_SPEED100**

```c
#define BMCR_SPEED100 0x2000
```
Select 100Mbps.

7.27.2.33 **BMCR_LOOPBACK**

```c
#define BMCR_LOOPBACK 0x4000
```
TXD loopback bit.

7.27.2.34 **BMCR_RESET**

```c
#define BMCR_RESET 0x8000
```
Reset.

7.27.2.35 **BMSR_ERCAP**

```c
#define BMSR_ERCAP 0x0001
```
Ext-reg capability.

7.27.2.36 **BMSR_JCD**

```c
#define BMSR_JCD 0x0002
```
Jabber detected.

7.27.2.37 **BMSR_LSTATUS**

```c
#define BMSR_LSTATUS 0x0004
```
Link status.
7.27.2.38  BMSR_ANEGCAPABLE
#define BMSR_ANEGCAPABLE 0x0008
Able to do auto-negotiation.

7.27.2.39  BMSR_RFAULT
#define BMSR_RFAULT 0x0010
Remote fault detected.

7.27.2.40  BMSR_ANEGCOMPLETE
#define BMSR_ANEGCOMPLETE 0x0020
Auto-negotiation complete.

7.27.2.41  BMSR_MFPRESUPPCAP
#define BMSR_MFPRESUPPCAP 0x0040
Able to suppress preamble.

7.27.2.42  BMSR_RESV
#define BMSR_RESV 0x0780
Unused.

7.27.2.43  BMSR_10HALF
#define BMSR_10HALF 0x0800
Can do 10mbps, half-duplex.

7.27.2.44  BMSR_10FULL
#define BMSR_10FULL 0x1000
Can do 10mbps, full-duplex.
7.27.2.45  **BMSR_100HALF**

#define BMSR_100HALF 0x2000

Can do 100mbps, half-duplex.

7.27.2.46  **BMSR_100FULL**

#define BMSR_100FULL 0x4000

Can do 100mbps, full-duplex.

7.27.2.47  **BMSR_100BASE4**

#define BMSR_100BASE4 0x8000

Can do 100mbps, 4k packets.

7.27.2.48  **ADVERTISE_SLCT**

#define ADVERTISE_SLCT 0x001f

Selector bits.

7.27.2.49  **ADVERTISE_CSMA**

#define ADVERTISE_CSMA 0x0001

Only selector supported.

7.27.2.50  **ADVERTISE_10HALF**

#define ADVERTISE_10HALF 0x0020

Try for 10mbps half-duplex.

7.27.2.51  **ADVERTISE_10FULL**

#define ADVERTISE_10FULL 0x0040

Try for 10mbps full-duplex.
7.27.2.52  ADVERTISE_100HALF

#define ADVERTISE_100HALF 0x0080

Try for 100mbps half-duplex.

7.27.2.53  ADVERTISE_100FULL

#define ADVERTISE_100FULL 0x0100

Try for 100mbps full-duplex.

7.27.2.54  ADVERTISE_100BASE4

#define ADVERTISE_100BASE4 0x0200

Try for 100mbps 4k packets.

7.27.2.55  ADVERTISE_PAUSE_CAP

#define ADVERTISE_PAUSE_CAP 0x0400

Try for pause.

7.27.2.56  ADVERTISE_PAUSE_SYM

#define ADVERTISE_PAUSE_SYM 0x0800

Try for asymmetric pause.

7.27.2.57  ADVERTISE_RESV

#define ADVERTISE_RESV 0x1000

Unused.

7.27.2.58  ADVERTISE_RFAULT

#define ADVERTISE_RFAULT 0x2000

Say we can detect faults.
7.27.2.59 **ADVERTISE_LPACK**

```c
#define ADVERTISE_LPACK 0x4000
```

Ack link partners response.

7.27.2.60 **ADVERTISE_NPAGE**

```c
#define ADVERTISE_NPAGE 0x8000
```

Next page bit.

7.27.2.61 **LPA_SLCT**

```c
#define LPA_SLCT 0x001f
```

Same as advertise selector.

7.27.2.62 **LPA_10HALF**

```c
#define LPA_10HALF 0x0020
```

Can do 10mbps half-duplex.

7.27.2.63 **LPA_10FULL**

```c
#define LPA_10FULL 0x0040
```

Can do 10mbps full-duplex.

7.27.2.64 **LPA_100HALF**

```c
#define LPA_100HALF 0x0080
```

Can do 100mbps half-duplex.

7.27.2.65 **LPA_100FULL**

```c
#define LPA_100FULL 0x0100
```

Can do 100mbps full-duplex.
7.27.2.66  LPA_100BASE4

#define LPA_100BASE4 0x0200

Can do 100mbps 4k packets.

7.27.2.67  LPA_PAUSE_CAP

#define LPA_PAUSE_CAP 0x0400

Can pause.

7.27.2.68  LPA_PAUSE_ASYM

#define LPA_PAUSE_ASYM 0x0800

Can pause asymetrically.

7.27.2.69  LPA_RESV

#define LPA_RESV 0x1000

Unused.

7.27.2.70  LPA_RFAULT

#define LPA_RFAULT 0x2000

Link partner faulted.

7.27.2.71  LPA_LPACK

#define LPA_LPACK 0x4000

Link partner acked us.

7.27.2.72  LPA_NPAGE

#define LPA_NPAGE 0x8000

Next page bit.
7.27.2.73 EXPANSION_NWAY

#define EXPANSION_NWAY 0x0001
Can do N-way auto-nego.

7.27.2.74 EXPANSION_LCWP

#define EXPANSION_LCWP 0x0002
Got new RX page code word.

7.27.2.75 EXPANSION_ENABLENPAGE

#define EXPANSION_ENABLENPAGE 0x0004
This enables npage words.

7.27.2.76 EXPANSION_NPCAPABLE

#define EXPANSION_NPCAPABLE 0x0008
Link partner supports npage.

7.27.2.77 EXPANSION_MFAULTS

#define EXPANSION_MFAULTS 0x0010
Multiple faults detected.

7.27.2.78 EXPANSION_RESV

#define EXPANSION_RESV 0xffe0
Unused.

7.27.2.79 NWAYTEST_RESV1

#define NWAYTEST_RESV1 0x00ff
Unused.

ChibiOS/HAL
7.27.2.80 NWAYTEST_LOOPBACK

#define NWAYTEST_LOOPBACK 0x0100

Enable loopback for N-way.

7.27.2.81 NWAYTEST_RESV2

#define NWAYTEST_RESV2 0xfe00

Unused.
7.28 MMC over SPI Driver

Generic MMC driver.

7.28.1 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`.

7.28.2 Driver State Machine

This driver implements a state machine internally, see the Abstract I/O Block Device module documentation for details.

7.28.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Macros

- `#define _mmc_driver_methods _mmcsd_block_device_methods`
  
  MMC_Driver specific methods.

MMC_SPI configuration options

- `#define MMC_NICE_WAITING TRUE`
  
  Delays insertions.

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.
Data Structures

- struct MMCConfig
  MMC/SD over SPI driver configuration structure.
- struct MMCDriverVMT
  MMCDriver virtual methods table.
- struct MMCDriver
  Structure representing a MMC/SD over SPI driver.

Functions

- static uint8_t crc7 (uint8_t crc, const uint8_t *buffer, size_t len)
  Calculate the MMC standard CRC-7 based on a lookup table.
- static void wait (MMCDriver *mmcp)
  Waits an idle condition.
- static void send_hdr (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)
  Sends a command header.
- static uint8_t recv1 (MMCDriver *mmcp)
  Receives a single byte response.
- static uint8_t recv3 (MMCDriver *mmcp, uint8_t *buffer)
  Receives a three byte response.
- static uint8_t send_command_R1 (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)
  Sends a command an returns a single byte response.
- static uint8_t send_command_R3 (MMCDriver *mmcp, uint8_t cmd, uint32_t arg, uint8_t *response)
  Sends a command which returns a five bytes response (R3).
- static uint8_t read_CxD (MMCDriver *mmcp, uint8_t cmd, uint32_t cxd[4])
  Reads the CSD.
- static void sync (MMCDriver *mmcp)
  Waits that the card reaches an idle state.
- void mmcInit (void)
  MMC over SPI driver initialization.
- void mmcObjectInit (MMCDriver *mmcp)
  Initializes an instance.
- void 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 MMCDriverVMT mmc_vmt
  Virtual methods table.
- static const uint8_t crc7_lookup_table[256]
  Lookup table for CRC-7 (based on polynomial x^7 + x^3 + 1).

7.28.4 Macro Definition Documentation

7.28.4.1 MMC_NICE_WAITING

#define MMC_NICE_WAITING TRUE

Delays insertions.

If enabled this option 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. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

7.28.4.2 _mmc_driver_methods

#define _mmc_driver_methods _mmcsd_block_device_methods

MMCDriver specific methods.

7.28.4.3 mmcIsCardInserted

#define mmcIsCardInserted(
  mmcp)
  mmc_lld_is_card_inserted(mmcp)

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  | mmcp | pointer to the MMCDriver object |

Returns

The card state.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>card not inserted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>card inserted.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.28.4.4 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

<table>
<thead>
<tr>
<th>false</th>
<th>card not inserted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>card inserted.</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 Function Documentation
7.28.5.1  crc7()

static uint8_t 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>in</th>
<th>crc</th>
<th>start value for CRC</th>
</tr>
</thead>
<tbody>
<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

7.28.5.2  wait()

static void wait (  
    MMCDriver * mmcp ) [static]

Waits an idle condition.

Parameters

| in  | mmcp   | pointer to the MMCDriver object |

Function Class:

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

Here is the call graph for this function:

```
osalThreadSuspendS
osalSysUnlock
osalSysLock
spireceive
wait
```

ChibiOS/HAL
7.28.5.3 send_hdr()

static void send_hdr (  
    MMCDriver * mmcp,  
    uint8_t cmd,  
    uint32_t arg ) [static]

Sends a command header.

Parameters

| in | mmcp | pointer to the MMCDriver object |
| in | cmd  | the command id                  |
| in | arg  | the command argument             |

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

Here is the call graph for this function:

![Call Graph for send_hdr()]

7.28.5.4 recvr1()

static uint8_t recvr1 (  
    MMCDriver * mmcp ) [static]

Receives a single byte response.

Parameters

| in | mmcp | pointer to the MMCDriver object |

ChibiOS/HAL
Returns

The response as an uint8_t value.

Return values

0xFF  timed out.

Function Class:

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

Here is the call graph for this function:

```
recvr1 -> spiReceive -> osalSysUnlock

osalSysLock

osalThreadSuspendS
```

### 7.28.5.5 recvr3()

```c
static uint8_t recvr3 (MMCDriver * mmcp,
                       uint8_t * buffer)  [static]
```

Receives a three byte response.

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 four bytes wide buffer</td>
</tr>
</tbody>
</table>

Returns

First response byte as an uint8_t value.
Return values

\texttt{0xFF} \text{ timed out.}

Function Class:

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

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>recvr3</th>
<th>recvr1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>spiReceive</td>
</tr>
</tbody>
</table>
```

7.28.5.6 \texttt{send\_command\_R1()}

```c
static uint8_t send_command_R1 (  
    MMCDriver \* mmcp,  
    uint8_t cmd,  
    uint32_t arg ) [static]
```

Sends a command and returns a single byte response.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the \texttt{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 response as an \texttt{uint8_t} value.

Return values

\texttt{0xFF} \text{ timed out.}

Function Class:

Not an API, this function is for internal use only.
Here is the call graph for this function:

7.28.5.7  send_command_R3()

static uint8_t send_command_R3 (  
   MMCDriver * mmcp,  
   uint8_t cmd,  
   uint32_t arg,  
   uint8_t * response ) [static]

Sends a command which returns a five bytes response (R3).

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>
<tr>
<td>out</td>
<td>response</td>
<td>pointer to four bytes wide uint8_t buffer</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.

Here is the call graph for this function:

```
send_command_R3
recvr3
send_hdr
spiSelect
spiUnselect
recvr1
spiReceive
osalSysLock
osalSysUnlock
crc7
spiSend
wait
```

7.28.5.8 read_CxD()

```c
static bool read_CxD (  
    MMCDriver ∗ mmcp,  
    uint8_t cmd,  
    uint32_t cxd[4] ) [static]
```

Reads the CSD.

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>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.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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:

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

Here is the call graph for this function:

```
read_CxD
recvr1
send_hdr
spiSelect
spiUnselect
spiReceive
osalSysLock
osalSysUnlock
crc7
spiSend
wait
```

### 7.28.5.9 sync()

```c
static void sync (MMCDriver * mmcp) [static]
```

Waits that the card reaches an idle state.

**Parameters**

* in mmcp pointer to the MMCDriver object*
Here is the call graph for this function:

![Call Graph]

### 7.28.5.10 mmcInit()

```c
void mmcInit ( )
```

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.

### 7.28.5.11 mmcObjectInit()

```c
void mmcObjectInit ( MMCDriver * mmcp )
```

Initializes an instance.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out mmcp</code></td>
<td>pointer to the MMCDriver object</td>
</tr>
</tbody>
</table>
Function Class:

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

### 7.28.5.12 mmcStart()

```c
void mmcStart ( 
    MMCDriver * mmcp, 
    const MMCConfig * config )
```

Configures and activates the MMC peripheral.

**Parameters**

- `in mmcp` pointer to the MMCDriver object
- `in config` pointer to the MMCConfig object.

Function Class:

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

### 7.28.5.13 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.

Here is the call graph for this function:

![Call Graph]

7.28.5.14 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

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>the operation succeeded and the driver is now in the MMC_READY state.</th>
</tr>
</thead>
<tbody>
<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.

Here is the call graph for this function:

```
mmConnect
send_command_R1
spiIgnore
spiStart
recvr1
send_hdr
spiSelect
spiUnselect
osalSysLock
osalSysUnlock
osalThreadSuspendS
spi_lld_start
```

### 7.28.5.15 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 | the operation succeeded and the driver is now in the MMC_INSERTED state. |
| 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:

```
bool mmcStartSequentialRead ( 
    MMCDriver * mmcp, 
    uint32_t startblk )
```

Starts a sequential read.

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 read</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>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.

Here is the call graph for this function:

7.28.5.17 mmcSequentialRead()

```c
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

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>the operation succeeded.</th>
</tr>
</thead>
<tbody>
<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.
7.28.5.18 mmcStopSequentialRead()

```c
bool mmcStopSequentialRead ( 
    MMCDriver * mmcp )
```

Stops a sequential read gracefully.

**Parameters**

- *mmcp*: pointer to the **MMCDriver** 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.

7.28.5.19 mmcStartSequentialWrite()

```c
bool mmcStartSequentialWrite ( 
    MMCDriver * mmcp, 
    uint32_t startblk )
```

Starts a sequential write.

**Parameters**

- *mmcp*: pointer to the **MMCDriver** object
- *startblk*: first block to write

**Returns**

The operation status.

**Return values**

- **HAL_SUCCESS**: the operation succeeded.
- **HAL_FAILED**: the operation failed.
7.28 MMC over SPI 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:

```
mmcStartSequentialWrite
  send_hdr
    spiSelect
      spiStart
        crc7
          spiSend
            wait
              osalSysLock
                osalSysUnlock
      spiSelect
        osalSysLock
          spi_lld_start
```

7.28.5.20 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>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>the operation succeeded.</th>
</tr>
</thead>
<tbody>
<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.
Here is the call graph for this function:

```
spiIgnore
spiReceive
spiSend
spiStop
spiUnselect
wait
osalSysLock
osalSysUnlock
osalThreadSuspendS
spi_lld_stop
```

### 7.28.5.21 mmcStopSequentialWrite()

```c
bool mmcStopSequentialWrite (MMCDriver *mmcp )
```

Stops a sequential write gracefully.

**Parameters**

| in | mmcp | pointer to the MMCDriver 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:

![Call Graph Diagram](image)

### 7.28.5.22 mmcSync()

```c
bool mmcSync (MMCDriver * mmcp)
```

Waits for card idle condition.

**Parameters**

| in  | mmcp | pointer to the MMCDriver object |

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th></th>
<th>the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>HAL_SUCCESS</code></td>
<td></td>
</tr>
<tr>
<td><code>HAL_FAILED</code></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:

```
mmcSync
spiStart
sync
osalSysLock
osalSysUnlock
spi_lld_start
spiReceive
spiSelect
spiUnselect
```

### 7.28.5.23 mmcGetInfo()

```c
bool mmcGetInfo (
            MMCDriver * mmcp,
            BlockDeviceInfo * bdip )
```

Returns the media info.

**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>bdip</td>
<td>pointer to a BlockDeviceInfo structure</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>the operation succeeded.</th>
</tr>
</thead>
<tbody>
<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.
**7.28.5.24 mmcErase()**

```c
bool mmcErase ( 
    MMCDriver * mmcp, 
    uint32_t startblk, 
    uint32_t endblk )
```

Erases blocks.

**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>starting block number</td>
</tr>
<tr>
<td>in</td>
<td>endblk</td>
<td>ending block number</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</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.

Here is the call graph for this function:

---

**7.28.6 Variable Documentation**
7.28.6.1 mmc_vmt

const struct MMCDriverVMT mmc_vmt [static]

Initial value:
= {
  (size_t)0,
  (bool (*)(void *))mmc_lld_is_cardInserted,  // not defined
  (bool (*)(void *))mmc_lld_isWriteProtected,
  (bool (*)(void *))mmcConnect,
  (bool (*)(void *))mmcDisconnect,
  mmc_read,
  mmc_write,
  (bool (*)(void *))mmcSync,
  (bool (*)(void *, BlockDeviceInfo *))mmcGetInfo
}

Virtual methods table.

7.28.6.2 crc7_lookup_table

const uint8_t 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, 0x43, 0x4a, 0x75, 0x7c, 0x67, 0x6e, 0x32, 0x3b, 0x22, 0x29,
  0x16, 0x1f, 0x40, 0x7a, 0x73, 0x68, 0x61, 0x5e, 0x57, 0x4c, 0x45,
  0x2b, 0x22, 0x39, 0x30, 0x0f, 0x06, 0x1d, 0x14, 0x63, 0x6a, 0x71, 0x78,
  0x47, 0x4e, 0x55, 0x5c, 0x64, 0x6d, 0x76, 0x7f, 0x00, 0x09, 0x52, 0x5b,
  0x2c, 0x35, 0x3a, 0x37, 0x08, 0x01, 0x1a, 0x13, 0x7d, 0x74, 0x6f, 0x66,
  0x59, 0x4b, 0x42, 0x35, 0x3c, 0x27, 0x2e, 0x11, 0x18, 0x03, 0x0a,
  0x56, 0x5f, 0x44, 0x4d, 0x72, 0x73, 0x60, 0x69, 0x1e, 0x17, 0x0c, 0x05,
  0x3a, 0x33, 0x28, 0x21, 0x4f, 0x46, 0x5d, 0x54, 0x4b, 0x42, 0x49, 0x40,
  0x57, 0x5e, 0x41, 0x44, 0x31, 0x38, 0x41, 0x48, 0x33, 0x3a,
  0x65, 0x6c, 0x77, 0x76, 0x09, 0x08, 0x1b, 0x12, 0x2d, 0x24, 0x3f, 0x36,
  0x58, 0x51, 0x4a, 0x43, 0x7c, 0x75, 0x6e, 0x67, 0x10, 0x19, 0x02, 0x0b,
  0x34, 0x3d, 0x26, 0x2f, 0x73, 0x7a, 0x61, 0x58, 0x57, 0x5e, 0x4b, 0x4c,
  0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x6a, 0x63, 0x78, 0x71,
  0x4e, 0x47, 0x5c, 0x55, 0x22, 0x2b, 0x30, 0x39, 0x06, 0x0f, 0x14, 0x1d,
  0x25, 0x2c, 0x37, 0x3a, 0x01, 0x08, 0x13, 0x1a, 0x6d, 0x64, 0x7f, 0x76,
  0x49, 0x4b, 0x5f, 0x52, 0x35, 0x32, 0x2e, 0x18, 0x11, 0x0a, 0x03,
  0x74, 0x7d, 0x6b, 0x6f, 0x50, 0x59, 0x42, 0x4b, 0x17, 0x1e, 0x05, 0x0c,
  0x33, 0x3a, 0x21, 0x28, 0x5f, 0x56, 0x4d, 0x44, 0x7b, 0x72, 0x69, 0x60,
  0x0e, 0x07, 0x1c, 0x15, 0x2a, 0x23, 0x38, 0x31, 0x44, 0x4f, 0x54, 0x5d,
  0x62, 0x6b, 0x70, 0x79
}

Lookup table for CRC-7 ( based on polynomial \( x^7 + x^3 + 1 \)).
7.29 MMC/SD Block Device

7.29.1 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_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_CONTENT_PROT_APP_SLICE 16U, 16U
• #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_ERASE_BLK_EN_SLICE MMCSD_CSD_20_ERASE_BLK_EN_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_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
  
  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 *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.

7.29.2 Macro Definition Documentation

7.29.2.1 MMCSD_BLOCK_SIZE

#define MMCSD_BLOCK_SIZE 512U

Fixed block size for MMC/SD block devices.
7.29.2.2 MMCSD_R1_ERROR_MASK

#define MMCSD_R1_ERROR_MASK 0xFDFFE008U

Mask of error bits in R1 responses.

7.29.2.3 MMCSD_CMD8_PATTERN

#define MMCSD_CMD8_PATTERN 0x000001AAU

Fixed pattern for CMD8.

7.29.2.4 _mmcsd_block_device_methods

#define _mmcsd_block_device_methods _base_block_device_methods

MMCSDBlockDevice specific methods.

7.29.2.5 _mmcsd_block_device_data

#define _mmcsd_block_device_data

Value:

_value
/** 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.

7.29.2.6 MMCSD_R1_ERROR

#define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)

Evaluates to true if the R1 response contains error flags.
7.29.2.7 MMC_SD_STS

```c
#define MMC_SD_STS( r1 ) (((r1) >> 9U) & 15U)
```

Returns the status field of an R1 response.

Parameters

| in  | r1 | the r1 response |

7.29.2.8 MMC_SD_R1_IS_CARD_LOCKED

```c
#define MMC_SD_R1_IS_CARD_LOCKED( r1 ) ((((r1) >> 21U) & 1U) != 0U)
```

Evaluates to true if the R1 response indicates a locked card.

Parameters

| in  | r1 | the r1 response |

7.29.2.9 mmcsdGetCardCapacity

```c
#define mmcsdGetCardCapacity( ip ) ((ip)-capacity)
```

Returns the card capacity in blocks.

Parameters

| in  | ip | pointer to a MMCSDBlockDevice or derived class |

Returns

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

7.29.3 Function Documentation

7.29.3.1 _mmcsd_get_slice()

```
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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>data</code></td>
<td>pointer to the words array</td>
</tr>
<tr>
<td>in</td>
<td><code>end</code></td>
<td>bit offset of the last bit of the field, inclusive</td>
</tr>
<tr>
<td>in</td>
<td><code>start</code></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.

7.29.3.2 _mmcsd_get_capacity()

```
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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>csd</code></td>
<td>the CSD record</td>
</tr>
</tbody>
</table>
Returns

The card capacity.

Return values

| 0 | CSD format error |

Function Class:

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

Here is the call graph for this function:

```
_chmmcsd_get_capacity _mmcsd_get_slice
```

### 7.29.3.3 _mmcsd_get_capacity_ext()

```c
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

- **ext_csd** the extended CSD record

Returns

The card capacity.

Function Class:

Not an API, this function is for internal use only.
7.29.3.4 _mmcsd_unpack_sdc_cid()

```c
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:

```
  _mmcsd_unpack_sdc_cid
      _mmcsd_get_slice
```

7.29.3.5 _mmcsd_unpack_mmc_cid()

```c
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>

ChibiOS/HAL
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
```

### 7.29.3.6 _mmcsd_unpack_csd_mmc()

```c
void _mmcsd_unpack_csd_mmc (  
    const MMCSDBlockDevice * sdcp,  
    unpacked_mmc_csd_t * csdmmc )
```

Unpacks MMC CSD 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>csdmmc</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
```
7.29.3.7 _mmcsd_unpack_csd_v10()

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:

---

7.29.3.8 _mmcsd_unpack_csd_v20()

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>
Function Class:

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

Here is the call graph for this function:
7.30 Serial NOR Flash Driver

Serial NOR Flash driver.

7.30.1 Detailed Description

Serial NOR Flash driver.

This module implements a generic driver for serial NOR Flash devices.

7.30.2 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_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.

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.
- 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)
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.

7.30.3 Macro Definition Documentation

7.30.3.1 SNOR_BUS_DRIVER

#define SNOR_BUS_DRIVER SNOR_BUS_DRIVER_WSPI

Physical transport interface.

7.30.3.2 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.
7.30.3.3 _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.

7.30.3.4 _snor_flash_methods

#define _snor_flash_methods

Value:

_base_flash_methods
_snor_flash_methods_alone

SNORDriver specific methods with inherited ones.

7.30.4 Function Documentation

7.30.4.1 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.

7.30.4.2 bus_acquire()

void bus_acquire ( 
    BUSDriver * busp,
    const BUSConfig * config )

Bus acquisition and lock.
### 7.30.4.3 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.

### 7.30.4.4 bus_stop()

```c
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.
7.30.4.5 bus_cmd()

```c
void bus_cmd ( 
    BUSDriver ∗ busp, 
    uint32_t cmd )
```

Sends a naked command.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>busp</th>
<th>pointer to the bus driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>instruction code</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.30.4.6 bus_cmd_send()

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>busp</th>
<th>pointer to the bus driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>instruction code</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to receive</td>
</tr>
<tr>
<td>in</td>
<td>p</td>
<td>data buffer</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.30.4.7 bus_cmd_receive()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>busp</th>
<th>pointer to the bus driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>instruction code</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to receive</td>
</tr>
<tr>
<td>out</td>
<td>p</td>
<td>data buffer</td>
</tr>
</tbody>
</table>

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

### 7.30.4.8 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

<table>
<thead>
<tr>
<th>in</th>
<th>busp</th>
<th>pointer to the bus driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>instruction code</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>flash offset</td>
</tr>
</tbody>
</table>

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

### 7.30.4.9 bus_cmd_addr_send()

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>in</th>
<th>busp</th>
<th>pointer to the bus driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>instruction code</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>flash offset</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to receive</td>
</tr>
<tr>
<td>in</td>
<td>p</td>
<td>data buffer</td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

7.30.4.10  bus_cmd_addr_receive()

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

| in  | busp    | pointer to the bus driver |
| in  | cmd     | instruction code          |
| in  | offset  | flash offset              |
| in  | n       | number of bytes to receive|
| out | p       | data buffer               |

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

7.30.4.11  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

| in  | busp    | pointer to the bus driver |
| in  | cmd     | instruction code          |
| in  | dummy   | number of dummy cycles    |
| in  | n       | number of bytes to receive|
| out | p       | data buffer               |
Function Class:
Not an API, this function is for internal use only.

### 7.30.4.12 bus_cmd_addr_dummy_receive()

```c
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>busp</code></td>
<td>pointer to the bus driver</td>
</tr>
<tr>
<td><code>cmd</code></td>
<td>instruction code</td>
</tr>
<tr>
<td><code>offset</code></td>
<td>flash offset</td>
</tr>
<tr>
<td><code>dummy</code></td>
<td>number of dummy cycles</td>
</tr>
<tr>
<td><code>n</code></td>
<td>number of bytes to receive</td>
</tr>
<tr>
<td><code>p</code></td>
<td>data buffer</td>
</tr>
</tbody>
</table>

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

### 7.30.4.13 snorObjectInit()

```c
void snorObjectInit (  
    SNORDriver * devp  
)
```

Initializes an instance.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>devp</code></td>
<td>pointer to the SNORDriver object</td>
</tr>
</tbody>
</table>

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

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 SNORDriver 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.

7.30.4.15  snorStop()

void snorStop (  
    SNORDriver * devp )

Deactivates the SNOR 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.

7.30.4.16  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

| in  | devp | pointer to the SNORDriver object |
| out | addrp | pointer to the memory start address of the mapped flash or NULL |
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:

```
7.30.4.17  snorMemoryUnmap()

void snorMemoryUnmap (  
   SNORDriver * devp )

Leaves the memory Mapping mode.

Parameters

    in   devp  pointer to the SNORDriver object
```
Here is the call graph for this function:

```
bus_acquire

snorMemoryUnmap

bus_release

wspiUnmapFlash

osalSysLock

osalSysUnlock
```

7.30.5 Variable Documentation

7.30.5.1 snor_vmt

const struct SNORDriverVMT snor_vmt [static]

Initial value:

```
= {
    (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_read_sfdp
}
```

Virtual methods table.
7.31 Base Object

7.31.1 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 stream class.

7.31.2 Macro Definition Documentation

7.31.2.1 _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.
### 7.31.2.2 _base_object_data

```c
#define _base_object_data

BaseObject specific data.
```

**Note**

This object defines no data.

### 7.31.2.3 objGetInstance

```c
#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
7.32 PAL Driver

I/O Ports Abstraction Layer.

7.32.1 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.

7.32.2 Implementation Rules

In implementing a PAL Low Level Driver there are some rules/behaviors that should be respected.

7.32.2.1 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.

7.32.2.2 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.
7.32.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

7.32.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

7.32.2.5 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 name = _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 `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.
  ```

### 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 palWriteGroup(port, mask, offset, bits)`
  Writes a group of bits.
- `#define palSetGroupMode(port, mask, offset, mode)`
  Sets 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), 0U, mode)`
  Line mode setup.
- `#define palEnablePadEventI(port, pad, mode)`
  Pad event enable.
- `#define palDisablePadEventI(port, pad)`
  Pad event disable.
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 palEnableLineEventI(line, mode)
Line event enable.
• #define palDisableLineEventI(line)
Line event disable.
• #define pullPadEventEnabledX(port, pad) false
Pad event enable check.
• #define pullLineEventEnabledX(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 iop tidal_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.

7.32.3 Macro Definition Documentation

7.32.3.1 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.
7.32.3.2 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.

7.32.3.3 PAL_MODE_INPUT

#define PAL_MODE_INPUT 2U

Regular input high-Z pad.

7.32.3.4 PAL_MODE_INPUT_PULLUP

#define PAL_MODE_INPUT_PULLUP 3U

Input pad with weak pull up resistor.

7.32.3.5 PAL_MODE_INPUT_PULLDOWN

#define PAL_MODE_INPUT_PULLDOWN 4U

Input pad with weak pull down resistor.

7.32.3.6 PAL_MODE_INPUT_ANALOG

#define PAL_MODE_INPUT_ANALOG 5U

Analog input mode.

7.32.3.7 PAL_MODE_OUTPUT_PUSHPULL

#define PAL_MODE_OUTPUT_PUSHPULL 6U

Push-pull output pad.
7.32.3.8 PAL_MODE_OUTPUT_OPENDRAIN

#define PAL_MODE_OUTPUT_OPENDRAIN 7U
Open-drain output pad.

7.32.3.9 PAL_LOW

#define PAL_LOW 0U
Logical low state.

7.32.3.10 PAL_HIGH

#define PAL_HIGH 1U
Logical high state.

7.32.3.11 PAL_EVENT_MODE_EDGES_MASK

#define PAL_EVENT_MODE_EDGES_MASK 3U
Mask of edges field.

7.32.3.12 PAL_EVENT_MODE_DISABLED

#define PAL_EVENT_MODE_DISABLED 0U
Channel disabled.

7.32.3.13 PAL_EVENT_MODE_RISING_EDGE

#define PAL_EVENT_MODE_RISING_EDGE 1U
Rising edge callback.
7.32.3.14 PAL_EVENT_MODE_FALLING_EDGE

#define PAL_EVENT_MODE_FALLING_EDGE 2U

Falling edge callback.

7.32.3.15 PAL_EVENT_MODE_BOTH_EDGES

#define PAL_EVENT_MODE_BOTH_EDGES 3U

Both edges callback.

7.32.3.16 PAL_USE_CALLBACKS

#define PAL_USE_CALLBACKS TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.32.3.17 PAL_USE_WAIT

#define PAL_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.32.3.18 PAL_PORT_BIT

#define PAL_PORT_BIT(n) ((iportmask_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.

7.32.3.19  PAL_GROUP_MASK

#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.

7.32.3.20  _IOBUS_DATA

#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 |
#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 |

#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.

#define _pal_init_event (e)

Value:

ChibiOS/HAL
do {
    _pal_events[e].cb = NULL;
    _pal_events[e].arg = NULL;
} while (false)

Initializes a PAL event object.

Parameters

\textbf{in} \hspace{0.5cm} \textbf{e} \hspace{0.5cm} event index

Function Class:

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

\subsection{7.32.3.24 \_pal_init\_event \[3/3\]}

\begin{verbatim}
#define _pal_init_event(e) 

do {
    osalThreadQueueObjectInit(&_pal_events[e].threads);
} while (false)
\end{verbatim}

Initializes a PAL event object.

Parameters

\textbf{in} \hspace{0.5cm} \textbf{e} \hspace{0.5cm} event index

Function Class:

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

\subsection{7.32.3.25 \_pal_clear\_event \[1/3\]}

\begin{verbatim}
#define _pal_clear_event(e) 

do {
    osalThreadDequeueAllI(&_pal_events[e].threads, MSG_RESET);
    _pal_events[e].cb = NULL;
    _pal_events[e].arg = NULL;
} while (false)
\end{verbatim}

Clears a PAL event object.
Parameters

in\e event index

Function Class:

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

7.32.3.26 _pal_clear_event [2/3]

#define _pal_clear_event{
  e
}

Value:

do {
  _pal_events[e].cb = NULL;
  _pal_events[e].arg = NULL;
} while (false)

Clears a PAL event object.

Parameters

in\e event index

Function Class:

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

7.32.3.27 _pal_clear_event [3/3]

#define _pal_clear_event{
  e
}

Value:

do {
  osalThreadDequeueAllI(&_pal_events[pad].threads, MSG_RESET);
} while (false)

Clears a PAL event object.

Parameters

in\e event index
Function Class:

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

7.32.3.28 _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

in  e  event index

Function Class:

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

7.32.3.29 _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

in event index

Function Class:

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

7.32.3.30 _pal_isr_code [3/3]

#define _pal_isr_code((e))

Value:

\{
    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

in event index

Function Class:

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

7.32.3.31 palInit

#define palInit() pal_lld_init()

PAL subsystem 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.
7.32.3.32  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.

7.32.3.33  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.
7.32.3.34  palWritePort

#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.

7.32.3.35  palSetPort

#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.
### 7.32.3.36 palClearPort

```c
#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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>port</td>
<td>port identifier</td>
</tr>
<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.

### 7.32.3.37 palTogglePort

```c
#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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>port</td>
<td>port identifier</td>
</tr>
<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.
### 7.32.3.38 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**

- **port** port identifier
- **mask** group mask, a logic AND is performed on the input data
- **offset** group bit offset within the port

**Returns**

The group logic states.

**Function Class:**

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

### 7.32.3.39 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.

7.32.3.40  \texttt{palSetGroupMode}

\begin{verbatim}
#define palSetGroupMode(
    port,
    mask,
    offset,
    mode )
\end{verbatim}

Value:

\begin{verbatim}
do {
    (void) (port);
    (void) (mask);
    (void) (offset);
    (void) (port);
} while (mode)
\end{verbatim}

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 \texttt{osalSysLock()} and \texttt{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>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>mode</td>
<td>group mode</td>
</tr>
</tbody>
</table>

Function Class:

Special function, this function has special requirements see the notes.
7.32.3.41  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**

<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**

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.

7.32.3.42  palWritePad

```c
#define palWritePad(
    port, 
    pad, 
    bit )
```

**Value:**

```
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></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>
<tr>
<td></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.

### 7.32.3.43 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></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:

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

### 7.32.3.44 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.

7.32.3.45 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.

7.32.3.46 palSetPadMode

```c
#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.
7.32.3.47 palReadLine

```c
#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**

- **PAL_LOW** low logic state.
- **PAL_HIGH** high logic state.

**Function Class:**

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

7.32.3.48 palWriteLine

```c
#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

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<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.

7.32.3.49  palSetLine

```c
#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.

7.32.3.50  palClearLine

```c
#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.
7.32 PAL Driver

Parameters

| in | line | line identifier |

Function Class:

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

7.32.3.51 palToggleLine

```c
#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.

7.32.3.52 palSetLineMode

```c
#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

<table>
<thead>
<tr>
<th></th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<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.

7.32.3.53  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></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.

7.32.3.54  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.

### 7.32.3.55 palEnablePadEvent

```c
#define palEnablePadEvent(  
    port, 
    pad, 
    mode )
```

**Value:**

```c
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.

### 7.32.3.56 palDisablePadEvent

```c
#define palDisablePadEvent(  
    port,  
    pad )
```

**Value:**

ChibiOS/HAL
Pad event disable.

This function also disables previously programmed event callbacks.

Parameters

| in | port | port identifier |
| in | pad  | pad number within the port |

Function Class:

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

7.32.3.57 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

| in | line | line identifier |
| in | mode | line event mode |

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.

7.32.3.58 palDisableLineEventI

#define palDisableLineEventI(
    line ) palDisablePadEventI(PAL_PORT(line), PAL_PAD(line))

Line event disable.

This function also disables previously programmed event callbacks.
Parameters

\[ \text{in} \quad \text{line} \quad \text{line identifier} \]

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.

7.32.3.59 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

\[ \text{in} \quad \text{line} \quad \text{line identifier} \]
\[ \text{in} \quad \text{mode} \quad \text{line event mode} \]

Function Class:

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

7.32.3.60 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.

#### 7.32.3.61 `palIsPadEventEnabledX`

```c
#define palIsPadEventEnabledX(
    port,
    pad
) false
```

Pad event enable check.

**Parameters**

| in | port | port identifier |
| in | pad | pad number within the port |

**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.

#### 7.32.3.62 `palIsLineEventEnabledX`

```c
#define palIsLineEventEnabledX(
    line
) pal_lld_ispadeventenabled(PAL_PORT(line), PAL_PAD(line))
```

Line event enable check.

**Parameters**

| in | line | line identifier |
Returns

Line event status.

Return values

| false | if the line event is disabled. |
| true  | if the line event is enabled.  |

Function Class:

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

### 7.32.3.63 palSetPadCallback

```c
#define palSetPadCallback(
    port,
    pad,
    cb,
    arg )
```

**Value:**

```c
do {
    osalSysLock();
    palSetPadCallbackI(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.

### 7.32.3.64 palSetLineCallback

```c
#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>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>int</td>
<td>line identifier</td>
</tr>
<tr>
<td>cb</td>
<td>event callback function</td>
<td></td>
</tr>
<tr>
<td>arg</td>
<td>callback argument</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.

---

### 7.32.3.65 PAL_IOPORTS_WIDTH

```c
#define PAL_IOPORTS_WIDTH 16U
```

Width, in bits, of an I/O port.

---

### 7.32.3.66 PAL_WHOLE_PORT

```c
#define PAL_WHOLE_PORT ((ioparmask_t)0xFFFFU)
```

Whole port mask.

This macro specifies all the valid bits into a port.

---

### 7.32.3.67 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.

---

### 7.32.3.68 PAL_PORT

```c
#define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFFFU))
```

Decodes a port identifier from a line identifier.
7.32.3.69 PAL_PAD

#define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))

Decodes a pad identifier from a line identifier.

7.32.3.70 PAL_NOLINE

#define PAL_NOLINE 0U

Value identifying an invalid line.

7.32.3.71 IOPORT1

#define IOPORT1 0

First I/O port identifier.

Low level drivers can define multiple ports, it is suggested to use this naming convention.

7.32.3.72 pal_lld_init

#define pal_lld_init() _pal_lld_init()

Low level PAL subsystem initialization.

Function Class:

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

7.32.3.73 pal_lld_readport

#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>

ChibiOS/HAL
Returns

The port bits.

Function Class:

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

7.32.3.74  pal_lld_readlatch

#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.

7.32.3.75  pal_lld_writeport

#define pal_lld_writeport(
    port,
    bits )

Value:

    do {
        (void)port;
        (void)bits;
    } while (false)

Writes a bits mask on a I/O port.

Parameters

|  in  | port  | port identifier  |
|  in  | bits  | bits to be written on the specified port |
Function Class:

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

7.32.3.76  pal_lld_setport

#define pal_lld_setport(
    port,
    bits )

Value:

    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>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:

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

7.32.3.77  pal_lld_clearport

#define pal_lld_clearport(
    port,
    bits )

Value:

    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.

### 7.32.3.78 pal_lld_toggleport

```c
#define pal_lld_toggleport(  
  port,  
  bits  )
```

**Value:**

```c
  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.

### 7.32.3.79 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>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<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.

7.32.3.80  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>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<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.
7.32.3.81  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

<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</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>group bit offset within the port</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>group mode</td>
</tr>
</tbody>
</table>

Function Class:

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

7.32.3.82  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

<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

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.

### 7.32.3.83 pal_lld_writepad

```c
#define pal_lld_writepad(
    port,
    pad,
    bit)

Value:

```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.
### 7.32.3.84 pal_lld_setpad

```c
#define pal_lld_setpad(
    port,
    pad )
```

**Value:**
```c
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>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.

### 7.32.3.85 pal_lld_clearpad

```c
#define pal_lld_clearpad(
    port,
    pad )
```

**Value:**
```c
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>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>
7.32 PAL Driver

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

7.32.3.86 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.

7.32.3.87 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.

ChibiOS/HAL
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.

7.32.3.88 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.

7.32.3.89 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

| in  | line    | line identifier |
Function Class:
Not an API, this function is for internal use only.

7.32.4  Typedef Documentation

7.32.4.1  palcallback_t

typedef void(* palcallback_t) (void *arg)

Type of a PAL event callback.

7.32.4.2  iomask_t

typedef uint32_t iomask_t

Digital I/O port sized unsigned type.

7.32.4.3  iomode_t

typedef uint32_t iomode_t

Digital I/O modes.

7.32.4.4  ioline_t

typedef uint32_t ioline_t

Type of an I/O line.

7.32.4.5  iopid_t

typedef uint32_t iopid_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.
7.32.4.6 iopadid_t

typedef uint32_t iopadid_t

Type of an pad identifier.

7.32.5 Function Documentation

7.32.5.1 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.

7.32.5.2 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 <code>IOBus</code> 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.

### 7.32.5.3 `palSetBusMode()`

```c
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 <code>IOBus</code> 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.
7.32.5.4 palSetPadCallbackI()

```c
void palSetPadCallbackI (
    iopordid_t port,
    iopadid_t pad,
    palcallback_t cb,
    void *arg )
```

Associates a callback to a 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>cb</td>
<td>event callback function</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>callback argument</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.

7.32.5.5 palSetLineCallbackI()

```c
void palSetLineCallbackI ( 
    ioline_t line,
    palcallback_t cb,
    void *arg )
```

Associates a callback to a line.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<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:**

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

7.32.5.6 palWaitPadTimeoutS()

```c
msg_t palWaitPadTimeoutS ( 
    iopordid_t port,
```
7.32 PAL Driver

```c
iopadid_t pad,
sysinterval_t timeout )
```

Waits for an edge on the specified port/pad.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>port</strong></td>
<td>port identifier</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>pad</strong></td>
<td>pad number within the port</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>timeout</strong></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_IMMEDIATE</strong> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITE</strong> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The operation state.

**Return values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSG_OK</strong></td>
<td>if an edge has been detected.</td>
</tr>
<tr>
<td><strong>MSG_TIMEOUT</strong></td>
<td>if a timeout occurred before an edge could be detected.</td>
</tr>
<tr>
<td><strong>MSG_RESET</strong></td>
<td>if the event has been disabled while the thread was waiting for an edge.</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.

Here is the call graph for this function:

![Call Graph](call-graph.png)

### 7.32.5.7 `palWaitPadTimeout()`

```c
msg_t palWaitPadTimeout ( 
    ioportid_t port,
    iopadid_t pad,
    sysinterval_t timeout )
```

Waits for an edge on the specified port/pad.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>pad</th>
<th>timeout</th>
<th>The number of ticks before the operation timeouts, the following special values are allowed:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation state.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>MSG_TIMEOUT</th>
<th>MSG_RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>if an edge has been detected.</td>
<td>if a timeout occurred before an edge could be detected.</td>
<td>if the event has been disabled while the thread was waiting for an edge.</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:

7.32.5.8  palWaitLineTimeoutS()

```
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

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if an edge has been detected.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if a timeout occurred before an edge could be detected.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the event has been disabled while the thread was waiting for an edge.</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.

Here is the call graph for this function:

```
palWaitLineTimeoutS osalThreadEnqueueTimeoutS
```

### 7.32.5.9 `palWaitLineTimeout()`

```
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

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if an edge has been detected.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if a timeout occurred before an edge could be detected.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the event has been disabled while the thread was waiting for an edge.</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:

7.32.5.10 _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.

7.32.5.11 _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.
7.33 Peripheral Interfaces

HAL Abstract Peripheral Interfaces.

7.33.1 Detailed Description

HAL Abstract Peripheral Interfaces.

Modules

- Generic Flash Interface

  HAL Generic Flash Driver Interface.
7.34 PWM Driver

Generic PWM Driver.

7.34.1 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`.

7.34.2 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).

![PWM State Machine Diagram]

7.34.3 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) (((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.
7.34 PWM Driver

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 pwmchnmsk_t
  Type of a channels mask.
- typedef uint32_t pwmcnt_t
  Type of a PWM counter.

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.
- void 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.

7.34.4  Macro Definition Documentation

7.34.4.1  PWM_OUTPUT_MASK

#define PWM_OUTPUT_MASK 0x0FU

Standard output modes mask.
### 7.34.4.2 PWM_OUTPUT_DISABLED

```c
#define PWM_OUTPUT_DISABLED 0x00U
```

Output not driven, callback only.

### 7.34.4.3 PWM_OUTPUT_ACTIVE_HIGH

```c
#define PWM_OUTPUT_ACTIVE_HIGH 0x01U
```

Positive PWM logic, active is logic level one.

### 7.34.4.4 PWM_OUTPUT_ACTIVE_LOW

```c
#define PWM_OUTPUT_ACTIVE_LOW 0x02U
```

Inverse PWM logic, active is logic level zero.

### 7.34.4.5 PWM_FRACTION_TO_WIDTH

```c
#define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)
```

**Value:**

\[
\frac{((\text{pwmcnt}_t)((\text{pwmcnt}_t)(\text{pwmp})\rightarrow \text{period}) \times \frac{\text{numerator}}{\text{denominator}})}{\text{pwmcnt}_t}
\]

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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>pwmp</code> pointer to a <code>PWM_Driver</code> object</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>denominator</code> denominator of the fraction</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>numerator</code> numerator of the fraction</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
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.

### 7.34.4.6 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>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>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.

### 7.34.4.7 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`. 
7.34 PWM Driver

Parameters

| in  | pwmp   | pointer to a PWMDriver object |
| in  | percentage | percentage as an integer between 0 and 10000 |

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.

7.34.4.8 pwmChangePeriodI

#define pwmChangePeriodI(
   pwmp, 
   value 
)

Value:

{ 
   {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

| in  | pwmp   | pointer to a PWMDriver object |
| in  | value   | new cycle time in ticks |
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.

### 7.34.4.9  pwmEnableChannelI

```c
#define pwmEnableChannelI(
    pwmp,
    channel,
    width )
```

**Value:**

```c
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**

- **in** `pwmp` pointer to a `PWMDriver` object
- **in** `channel` PWM channel identifier (0...channels-1)
- **in** `width` PWM pulse width as clock pulses number

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.

### 7.34.4.10  pwmDisableChannelI

```c
#define pwmDisableChannelI(
    pwmp,
    channel )
```
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>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>
</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.

### 7.34.4.11 pwmIsChannelEnabledI

```c
#define pwmIsChannelEnabledI(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)
```

Retains a PWM channel status.

Precondition

The PWM unit must have been activated using `pwmStart()`.

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>
</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.

7.34.4.12 pwmEnablePeriodicNotificationI

#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.

7.34.4.13 pwmDisablePeriodicNotificationI

#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.
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
- `pwmp`: pointer to a `PWMDriver` object
- `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.

Disables a channel de-activation edge notification.

Parameters
- `pwmp`: pointer to a `PWMDriver` object
- `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.
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>Parameter</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:

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

### 7.34.16 PWM_CHANNELS

```c
#define PWM_CHANNELS 4
```

Number of PWM channels per PWM driver.

### 7.34.17 PLATFORM_PWM_USE_PWM1

```c
#define PLATFORM_PWM_USE_PWM1 FALSE
```

PWMD1 driver enable switch.

If set to `TRUE` the support for PWM1 is included.

Note

The default is `FALSE`. 
7.34.18 pwm_lld_change_period

#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 \texttt{pwmStart()}. 

Precondition

The PWM unit must have been activated using \texttt{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

\begin{tabular}{|c|c|}
\hline
in & \textit{pwmp} & pointer to a \texttt{PWMDriver} object \\
\hline
in & \textit{period} & new cycle time in ticks \\
\hline
\end{tabular}

Function Class:

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

7.34.5 Typedef Documentation

7.34.5.1 PWMDriver

typedef struct PWMDriver PWMDriver

Type of a structure representing a PWM driver.

7.34.5.2 pwmcallback_t

typedef void(* pwmcallback_t) (PWMDriver *pwmp)

Type of a PWM notification callback.
Parameters

| in | pwmp | pointer to a PWMDriver object |

7.34.5.3  pwmmode_t

typedef uint32_t pwmmode_t

Type of a PWM mode.

7.34.5.4  pwmchannel_t

typedef uint8_t pwmchannel_t

Type of a PWM channel.

7.34.5.5  pwmchnmsk_t

typedef uint32_t pwmchnmsk_t

Type of a channels mask.

7.34.5.6  pwmcnt_t

typedef uint32_t pwmcnt_t

Type of a PWM counter.

7.34.6  Enumeration Type Documentation

7.34.6.1  pwmstate_t

enum pwmstate_t

Driver state machine possible states.
7.34 PWM Driver

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWM_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>PWM_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>PWM_READY</td>
<td>Ready.</td>
</tr>
</tbody>
</table>

7.34.7 Function Documentation

7.34.7.1 pwmInit()

```c
#include "ch.h"
#include "hal.h"

void pwmInit ( void )
{
    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:

```
pwmInit → pwm_lld_init → pwmObjectInit
```

7.34.7.2 pwmObjectInit()

```c
#include "ch.h"
#include "hal.h"

void pwmObjectInit ( PWMDriver * pwmp )
{
}
```

Initializes the standard part of a `PWMDriver` structure.
Parameters

| out  | pwmp | pointer to a PWMDriver object |

Function Class:

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

### 7.34.7.3 pwmStart()

```c
void 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

| in   | pwmp | pointer to a PWMDriver object |
|      | config | pointer to a PWMConfig 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](attachment:image.png)
### 7.34.7.4 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](call_graph.png)

### 7.34.7.5 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.
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>period</td>
<td>new cycle time 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:

```
pwmChangePeriod
    osalSysLock
    osalSysUnlock
```

7.34.7.6 pwmEnableChannel()

```c
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).
### 7.34 PWM Driver

#### 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>
<tr>
<td>in 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:

![Call Graph]

#### 7.34.7.7 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).
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pwmp</td>
<td>PWMDriver pointer</td>
<td>pointer to a PWMDriver object</td>
</tr>
<tr>
<td>in channel</td>
<td>PWM channel identifier (0...channels-1)</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:

```
osalSysLock

pwmDisableChannel

osalSysUnlock
```

7.34.7.8  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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pwmp</td>
<td>PWMDriver pointer</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:

![Call Graph Image]

### 7.34.7.9 `pwmDisablePeriodicNotification()`

```c
void pwmDisablePeriodicNotification (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:

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
```

### 7.34.7.10 `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>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:

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:

```
pwmEnableChannelNotification
  osalSysLock
osalSysUnlock
```

7.34.7.11 pwmDisableChannelNotification()

```c
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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>pwmp</code> pointer to a <code>PWMDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>channel</code> 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
```

### 7.34.7.12 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
```

### 7.34.7.13 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.
Parameters

| in  | pwmp | pointer to a PWMDriver object |

Function Class:

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

### 7.34.7.14 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.

### 7.34.7.15 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pwmp</td>
<td>PWMDriver object</td>
<td>pointer to a PWMDriver object</td>
</tr>
<tr>
<td>in channel</td>
<td>PWM channel identifier (0...channels-1)</td>
<td></td>
</tr>
<tr>
<td>in width</td>
<td>PWM pulse width as clock pulses number</td>
<td></td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.34.7.16 pwm_lld_disable_channel()

```c
define 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pwmp</td>
<td>PWMDriver object</td>
<td>pointer to a PWMDriver object</td>
</tr>
<tr>
<td>in channel</td>
<td>PWM channel identifier (0...channels-1)</td>
<td></td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.34.7.17 pwm_lld_enable_periodic_notification()

```c
define 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.

---

### 7.34.7.18 `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.

---

### 7.34.7.19 `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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><code>pwmp</code></td>
<td>pointer to a <code>PWMDriver</code> object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<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.

7.34.7.20  `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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><code>pwmp</code></td>
<td>pointer to a <code>PWMDriver</code> object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<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.

7.34.8  Variable Documentation
7.34.8.1 PWMD1

PWMDriver PWMD1

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.
7.35  I/O Bytes Queues

7.35.1  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.
### 7.35 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 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.

### 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)**
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.

### 7.35.2 Macro Definition Documentation

#### 7.35.2.1 Q_OK

#define Q_OK MSG_OK

Operation successful.

#### 7.35.2.2 Q_TIMEOUT

#define Q_TIMEOUT MSG_TIMEOUT

Timeout condition.

#### 7.35.2.3 Q_RESET

#define Q_RESET MSG_RESET

Queue has been reset.
7.35.2.4 Q_EMPTY

#define Q_EMPTY MSG_TIMEOUT

Queue empty.

7.35.2.5 Q_FULL

#define Q_FULL MSG_TIMEOUT

Queue full,

7.35.2.6 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.

7.35.2.7 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>qp</code></td>
<td>pointer to <code>io_queue_t</code> structure</td>
</tr>
</tbody>
</table>

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.

### 7.35.2.8 qGetLink

```c
#define qGetLink(
    qp ) ((qp)->q_link)
```

Returns the queue application-defined link.

Note

This function can be called in any context.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>qp</code></td>
<td>pointer to <code>io_queue_t</code> structure</td>
</tr>
</tbody>
</table>

Returns

The application-defined link.

Function Class:

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

### 7.35.2.9 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.
7.35 I/O Bytes Queues

Parameters

<table>
<thead>
<tr>
<th></th>
<th>q</th>
<th>pointer to a io_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>l</td>
<td>The application-defined link.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.35.2.10 iqGetFullI

```c
#define iqGetFullI(iqp) qSpaceI(iqp)
```

Returns the filled space into an input queue.

Parameters

|    |   q   | pointer to an input_queue_t structure |

Returns

The number of full bytes in the queue.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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.

7.35.2.11 iqGetEmptyI

```c
#define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))
```

Returns the empty space into an input queue.

Parameters

|    |   q   | pointer to an input_queue_t structure |

ChibiOS/HAL
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.

### 7.35.2.12 iqIsEmptyI

```c
#define iqIsEmptyI(iqp) ((bool)(qSpaceI(iqp) == 0U))
```

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

Parameters

| in iq | pointer to an input_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.

### 7.35.2.13 iqIsFullI

```c
#define iqIsFullI(iqp) 
{bool}((iqp)->q_wrptr == (iqp)->q_rdptr) && ((iqp)->q_counter != 0U))
```

Evaluates to `true` if the specified input queue is full.

Value:

`/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/` 
`{bool}((iqp)->q_wrptr == (iqp)->q_rdptr) && ((iqp)->q_counter != 0U))` 
`/*lint -restore*/`

ChibiOS/HAL
7.35 I/O Bytes Queues

Parameters

\[
\text{in } iqp \quad \text{pointer to an input_queue_t structure}
\]

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.

7.35.2.14 iqGet

#define iqGet(
   \text{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

\[
\text{in } iqp \quad \text{pointer to an input_queue_t structure}
\]

Returns

A byte value from the queue.

Return values

\[
\text{MSG_RESET} \quad \text{if the queue has been reset.}
\]

Function Class:

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

#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.

7.35.2.16  oqGetEmptyI

#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.

7.35.2.17  oqIsEmptyI

#define oqIsEmptyI(
    oqp )

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.

7.35.2.18  oqIsFullI

#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

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

7.35.2.19 oqPut

#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

| 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_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.

7.35.3 Typedef Documentation
7.35 I/O Bytes Queues

7.35.1 io_queue_t

typedef struct io_queue io_queue_t

Type of a generic I/O queue structure.

7.35.2 qnotify_t

typedef void(∗qnotify_t)(io_queue_t ∗qp)

Queue notification callback type.

Parameters

| in  | qp | the queue pointer |

7.35.3 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.

7.35.4 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.

7.35.4 Function Documentation

7.35.4.1 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>oqp</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.

7.35.4.2 oq_write()

```c
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.

7.35.4.3 iqObjectInit()

```c
void iqObjectInit {
  input_queue_t * iqp,
```
7.35 I/O Bytes Queues

```c
uint8_t * bp,
size_t size,
qnotify_t infy,
void * link );
```

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>iqp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>bp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>size</code></td>
</tr>
<tr>
<td>in</td>
<td><code>infy</code></td>
</tr>
<tr>
<td>in</td>
<td><code>link</code></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:

```
<table>
<thead>
<tr>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>iqObjectInit</td>
</tr>
</tbody>
</table>
```

---

### 7.35.4.4 iqResetI()

```c
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:

```
    iqResetI  ↓
             ↓
osalThreadDequeueAll
```

7.35.4.5 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

- **in iqp** pointer to an `input_queue_t` structure
- **in b** the byte value to be written in the queue

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_OK</code></td>
<td>if the operation has been completed with success.</td>
</tr>
<tr>
<td><code>MSG_TIMEOUT</code></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:

```
iqPutI → osalThreadDequeueNextI
```

### 7.35.4.6 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**

- **inqp** 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.
### 7.35.4.7 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>iqp</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
</tr>
</tbody>
</table>

- iqp: pointer to an `input_queue_t` structure
- 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.

#### 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:
7.35.4.8  iqReadI()

```c
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**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>iqp</td>
<td>pointer to an <code>input_queue_t</code> structure</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, 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](image.png)

7.35.4.9  iqReadTimeout()

```c
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.
Module Documentation

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>
<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:

```
    iq_read
    iqReadTimeout
    osalSysLock
    osalThreadEnqueueTimeoutS
```

7.35.4.10  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.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out oqp</code></td>
<td>pointer to an <code>output_queue_t</code> structure</td>
</tr>
<tr>
<td><code>in bp</code></td>
<td>pointer to a memory area allocated as queue buffer</td>
</tr>
<tr>
<td><code>in size</code></td>
<td>size of the queue buffer</td>
</tr>
<tr>
<td><code>in only</code></td>
<td>pointer to a callback function that is invoked when data is written to the queue. The value can be <code>NULL</code>.</td>
</tr>
<tr>
<td><code>in link</code></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
```

### 7.35.4.11 oqReset()

```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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in oqp</code></td>
<td>pointer to an <code>output_queue_t</code> structure</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:

```
oqResetI
  osalThreadDequeueAllI
```

7.35.4.12 oqPutI()

```c
def 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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oqp</code></td>
<td>pointer to an <code>output_queue_t</code> structure</td>
</tr>
<tr>
<td><code>b</code></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>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_OK</code></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><code>MSG_TIMEOOUT</code></td>
<td>if the queue is full.</td>
</tr>
<tr>
<td><code>MSG_RESET</code></td>
<td>if the queue has been reset.</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.

7.35.4.13 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oqp</code></td>
<td>pointer to an <code>output_queue_t</code> structure</td>
</tr>
<tr>
<td><code>b</code></td>
<td>the byte value to be written in the queue</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 operation status.

**Return values**

- `MSG_OK` if the operation succeeded.
- `MSG_TIMEOUT` if the specified time expired.
- `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.

Here is the call graph for this function:
7.35.4.14  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
```

7.35.4.15  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.
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
```

7.35.4.16  oqWriteTimeout()

```c
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>in</th>
<th>oq</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>
<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:
7.36 RTC Driver

Generic RTC Driver.

7.36.1 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 \texttt{HAL\_USE\_RTC} option must be enabled in \texttt{halconf.h}.

Macros

- \#define \texttt{RTC\_BASE\_YEAR} 1980U
  
  Base year of the calendar.
- \#define \texttt{\_rtc\_driver\_methods\_base\_pers\_storage\_methods}
  
  BasePersistentStorage specific methods.
- \#define \texttt{rtc\_lld\_driver\_fields} uint32\_t dummy
  
  Implementation-specific \texttt{RTCDriver} fields.

Date/Time bit masks for FAT format

- \#define \texttt{RTC\_FAT\_TIME\_SECONDS\_MASK} 0x0000001FU
- \#define \texttt{RTC\_FAT\_TIME\_MINUTES\_MASK} 0x000007E0U
- \#define \texttt{RTC\_FAT\_TIME\_HOURS\_MASK} 0x0000F800U
- \#define \texttt{RTC\_FAT\_DATE\_DAYS\_MASK} 0x001F0000U
- \#define \texttt{RTC\_FAT\_DATE\_MONTHS\_MASK} 0x01E00000U
- \#define \texttt{RTC\_FAT\_DATE\_YEARS\_MASK} 0xFE000000U

Day of week encoding

- \#define \texttt{RTC\_DAY\_CATURDAY} 0U
- \#define \texttt{RTC\_DAY\_MONDAY} 1U
- \#define \texttt{RTC\_DAY\_TUESDAY} 2U
- \#define \texttt{RTC\_DAY\_WEDNESDAY} 3U
- \#define \texttt{RTC\_DAY\_THURSDAY} 4U
- \#define \texttt{RTC\_DAY\_FRIDAY} 5U
- \#define \texttt{RTC\_DAY\_SATURDAY} 6U
- \#define \texttt{RTC\_DAY\_SUNDAY} 7U

Implementation capabilities

- \#define \texttt{RTC\_SUPPORTS\_CALLBACKS} TRUE
  
  Callback support int the driver.
- \#define \texttt{RTC\_ALARMS} 2
  
  Number of alarms available.
- \#define \texttt{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, RTCAAlarm ∗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.
• 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 rtc_event_t
  Type of an RTC event.

7.36.2 Macro Definition Documentation

7.36.2.1 RTC_BASE_YEAR

#define RTC_BASE_YEAR 1980U
Base year of the calendar.

7.36.2.2 _rtc_driver_methods

#define _rtc_driver_methods _base_persistent_storage_methods
BasePersistentStorage specific methods.

7.36.2.3 RTC_SUPPORTS_CALLBACKS

#define RTC_SUPPORTS_CALLBACKS TRUE
Callback support int the driver.

7.36.2.4 RTC_ALARMS

#define RTC_ALARMS 2
Number of alarms available.
7.36.2.5 RTC_HAS_STORAGE

#define RTC_HAS_STORAGE FALSE

Presence of a local persistent storage.

7.36.2.6 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.

7.36.2.7 rtc_lld_driver_fields

#define rtc_lld_driver_fields uint32_t dummy

Implementation-specific RTCDriver fields.

7.36.3 Typedef Documentation

7.36.3.1 RTCDriver

typedef struct RTCDriver RTCDriver

Type of a structure representing an RTC driver.

7.36.3.2 rtcalarm_t

typedef unsigned int rtcalarm_t

Type of an RTC alarm number.
7.36.3.3 rtcb_t

typedef void(* rtcb_t) (RTCDriver *rtcp, rtcevent_t event)
Type of a generic RTC callback.

7.36.4 Enumeration Type Documentation

7.36.4.1 rtcevent_t
enum rtcevent_t
Type of an RTC event.

7.36.5 Function Documentation

7.36.5.1 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:

```
  rtcInit  →  rtc_lld_init  →  rtcObjectInit
```

7.36.5.2 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.

7.36.5.3 rtcSetTime()

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

| in | rtcp | pointer to RTC driver structure |
| in | timespec | pointer to a RTCDateTime structure |

Function Class:

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

Here is the call graph for this function:
### 7.36.5.4 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></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rtcp</td>
</tr>
<tr>
<td>out</td>
<td>timespec</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)

### 7.36.5.5 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.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rtcp</td>
<td>pointer to RTC driver structure</td>
</tr>
<tr>
<td>in</td>
<td>alarm</td>
<td>alarm identifier</td>
</tr>
<tr>
<td>in</td>
<td>alarmspec</td>
<td>pointer to a RTCAlarm structure or NULL</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
rtcSetAlarm —> rtc_lld_set_alarm
```

7.36.5.6 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.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rtcp</td>
<td>pointer to RTC driver structure</td>
</tr>
<tr>
<td>in</td>
<td>alarm</td>
<td>alarm identifier</td>
</tr>
<tr>
<td>out</td>
<td>alarmspec</td>
<td>pointer to a RTCAlarm 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:

```
    rtcGetAlarm  rtc_lld_get_alarm
```

### 7.36.5.7 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rtcp</td>
</tr>
<tr>
<td>in</td>
<td>callback</td>
</tr>
</tbody>
</table>

**Function Class:**

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

### 7.36.5.8 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></th>
<th>timespec</th>
<th>pointer to a <strong>RTCDateTime</strong> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</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.

### 7.36.5.9 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></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 <strong>RTCDateTime</strong> 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.

### 7.36.5.10 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**

<table>
<thead>
<tr>
<th></th>
<th>timespec</th>
<th>pointer to a <strong>RTCDateTime</strong> structure</th>
</tr>
</thead>
</table>
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.

### 7.36.5.11 rtc_lld_init()

```c
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
```

### 7.36.5.12 rtc_lld_set_time()

```c
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.
7.36.5.13 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>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.

7.36.5.14 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

| in | rtcp   | pointer to RTC driver structure. |
| in | alarm  | alarm identifier. Can be 1 or 2. |
| in | alarmspec | pointer to a RTCAlarm structure. |

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

7.36.5.15 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

| in | rtcp   | pointer to RTC driver structure |
| in | alarm  | alarm identifier |
| out| alarmspec | pointer to a RTCAlarm structure |

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

Generic SD Card Driver.

7.37.1 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 `HAL_USE_SDC` option must be enabled in `halconf.h`.

7.37.2 Driver State Machine

This driver implements a state machine internally, see the Abstract I/O Block Device module documentation for details.

7.37.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Macros

- `#define _sdc_driver_methods _mmcsd_block_device_methods`
  
  `SDCDriver` specific methods.

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.

PLATFORM configuration options

- `#define PLATFORM_SDC_USE_SDC1 FALSE`
  PWMDI 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 SDConfig`
  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 bool mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)
  Constructs CMD6 argument for MMC.
- static bool 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.
- void 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.
7.37.4 Macro Definition Documentation

7.37.4.1 SDC_INIT_RETRY

#define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

   Attempts are performed at 10mS intervals.

7.37.4.2 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.

7.37.4.3 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.

7.37.4.4 SDC_INIT_OCR_V20

#define SDC_INIT_OCR_V20 0x50FF8000U

OCR initialization constant for V20 cards.
7.37 SDC Driver

7.37.4.5  SDC_INIT_OCR

#define SDC_INIT_OCR 0x80100000U

OCR initialization constant for non-V20 cards.

7.37.4.6  sdcIsCardInserted

#define sdcIsCardInserted(sdc) ((sdc_lld_is_cardInserted(sdc))

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.

7.37.4.7 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.
7.37 SDC Driver

7.37.4.8 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.

7.37.4.9 _sdc_driver_methods

#define _sdc_driver_methods _mmc_sd_block_device_methods

SDCDriver specific methods.

7.37.5 Typedef Documentation

7.37.5.1 sdcmode_t

typedef uint32_t sdcmode_t

Type of card flags.

7.37.5.2 sdcflags_t

typedef uint32_t sdcflags_t

SDC Driver condition flags type.

7.37.5.3 SDCDriver

typedef struct SDCDriver SDCDriver

Type of a structure representing an SDC driver.
7.37.6 Enumeration Type Documentation

7.37.6.1 mmc_switch_t

enum mmc_switch_t
MMC switch mode.

7.37.6.2 sd_switch_t

enum sd_switch_t
SDC switch mode.

7.37.6.3 sd_switch_function_t

enum sd_switch_function_t
SDC switch function.

7.37.6.4 sdcbusmode_t

enum sdcbusmode_t
Type of SDIO bus mode.

7.37.6.5 sdcbusclk_t

enum sdcbusclk_t
Max supported clock.

7.37.7 Function Documentation

7.37.7.1 mode_detect()

static bool mode_detect(
    SDCDriver * sdcp ) [static]

Detects card mode.
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:

```
mode_detect    sdc_lid_send_cmd_short_crc
```

### 7.37.7.2  mmc_init()

static bool mmc_init (  
    SDCDriver * sdcp )  [static]

Init procedure for MMC.

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:

```
mmc_init  sdc_lld_send_cmd_short
```

7.37.7.3  sdc_init()

```c
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

| HAL_SUCCESS | operation succeeded. |
| HAL_FAILED  | operation failed.   |

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

7.37.7.4  mmc_cmd6_construct()

```c
static uint32_t mmc_cmd6_construct ( 
    mmc_switch_t access, 
    uint32_t idx, 
```
uint32_t value,
uint32_t cmd_set ) [static]

Constructs CMD6 argument for MMC.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>access</th>
<th>EXT_CSD access mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx</td>
<td>EXT_CSD byte number</td>
</tr>
<tr>
<td>in</td>
<td>value</td>
<td>value to be written in target field</td>
</tr>
<tr>
<td>in</td>
<td>cmd_set</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.

7.37.7.5 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>in</th>
<th>mode</th>
<th>switch/test mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>function</td>
<td>function number to be switched</td>
</tr>
<tr>
<td>in</td>
<td>value</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.

7.37.7.6 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

<table>
<thead>
<tr>
<th>in</th>
<th>function</th>
<th>function number to be switched</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>buffer with answer</td>
</tr>
</tbody>
</table>

Returns

extracted answer.

Function Class:

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

7.37.7.7  sdc_cmd6_check_status()

static bool sdc_cmd6_check_status (  
  sd_switch_function_t function,  
  const uint8_t * buf ) [static]

Checks status after switching using CMD6.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>function</th>
<th>function number to be switched</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>buffer with answer</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.

7.37.7.8  sdc_detect_bus_clk()

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.

7.37.7.9 mmc_detect_bus_clk()

static bool mmc_detect_bus_clk (  
  SDCDriver * sdcp,  
  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.
7.37 SDC Driver

7.37.7.10 detect_bus_clk()

```c
static bool detect_bus_clk (  
    SDCDriver * sdcp,  
    sdcbusclk_t * clk ) [static]
```

Reads supported bus clock and switch card 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.

7.37.7.11 sdc_set_bus_width()

```c
static bool sdc_set_bus_width (  
    SDCDriver * sdcp ) [static]
```

Sets bus width 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.

7.37.7.12  mmc_set_bus_width()

static bool mmc_set_bus_width (  
    SDCDriver * sdcp )  [static]

Sets bus width for MMC.

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:

    mmc_set_bus_width  ->  mmc_cmd6_construct

7.37.7.13  _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
```

7.37.7.14 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
```

### 7.37.7.15  sdcObjectInit()

```c
void sdcObjectInit ( SCDriver * sdcp )
```

Initializes the standard part of a SCDriver structure.

**Parameters**

| out | sdcp | pointer to the SCDriver object |

Function Class:

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

### 7.37.7.16  sdcStart()

```c
void sdcStart ( SCDriver * sdcp, const SDCConfig * config )
```

Configures and activates the SDC peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</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>
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:

```
7.37.7.17 sdcStop()

void sdcStop ( SDCDriver * sdcp )

Deactives the SDC peripheral.

Parameters

| in  | sdcp | pointer to the SDCDriver 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:

```
sdcStop
  osalSysLock
  osalSysUnlock
  sdc_lld_stop
```

### 7.37.7.18 sdcConnect()

```c
bool sdcConnect ( SCDriver * 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 SCDriver object

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</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.
7.37.19  sdcDisconnect()

bool sdcDisconnect ( 
    SDCDriver * sdcp )

Brings the driver in a state safe for card removal.

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:

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:

```
    _sdc_wait_for_transfer
       _state
          sdcDisconnect
             osalSysLock
                 osalSysUnlock
                    sdc_lld_stop_clk
                        sdc_lld_send_cmd_short_crc
```

7.37.20  sdcRead()

bool sdcRead ( 
    SDCDriver * sdcp, 
    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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>sdcp</code></td>
<td>pointer to the <code>SDCDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>startblk</code></td>
<td>first block to read</td>
</tr>
<tr>
<td>out</td>
<td><code>buf</code></td>
<td>pointer to the read buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code></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:

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

### 7.37.7.21 sdcWrite()

```c
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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>sdcp</code></td>
<td>pointer to the <code>SDCDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>startblk</code></td>
<td>first block to write</td>
</tr>
<tr>
<td>out</td>
<td><code>buf</code></td>
<td>pointer to the write buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code></td>
<td>number of blocks to write</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.

7.37.7.22 `sdcGetAndClearErrors()`

```c
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:

![Call Graph]

7.37.7.23 `sdcSync()`

```c
bool sdcSync ( 
    SDCDriver * sdcp 
)
```

Waits for card idle condition.
Parameters

| in | sdp | 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_lid_sync
```

7.37.7.24 sdcGetInfo()

```c
bool sdcGetInfo (  
    SDCDriver * sdp,  
    BlockDeviceInfo * bdip )
```

Returns the media info.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bdip</td>
<td>pointer to a BlockDeviceInfo structure</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>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.

### 7.37.7.25 sdcErase()

```c
bool sdcErase (  
    SCDriver * sdcp,  
    uint32_t startblk,  
    uint32_t endblk )
```

Erases the supplied blocks.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in sdcp</td>
<td>pointer to the SCDriver object</td>
</tr>
<tr>
<td>in startblk</td>
<td>starting block number</td>
</tr>
<tr>
<td>in endblk</td>
<td>ending block number</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>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.

### 7.37.7.26 sdc_lld_init()

```c
void sdc_lld_init (  
    void )
```

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
```

### 7.37.7.27 sdc_lld_start()

```c
void sdc_lld_start ( 
    SDCDriver * sdcp )
```

Configures and activates the SDC peripheral.

Parameters
---

| Type | in | sdcp | pointer to the SDCDriver object |

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

### 7.37.7.28 sdc_lld_stop()

```c
void sdc_lld_stop ( 
    SDCDriver * sdcp )
```

Deactivates the SDC peripheral.

Parameters
---

| Type | in | sdcp | pointer to the SDCDriver object |

Function Class:
Not an API, this function is for internal use only.
### 7.37.7.29 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.

### 7.37.7.30 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 |
|     | clk  | the clock mode                |

**Function Class:**

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

### 7.37.7.31 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.

### 7.37.7.32 sdc_lld_set_bus_mode()

```c
void sdc_lld_set_bus_mode (  
    SDCDriver * sdcp,  
    sdcbusmode_t mode  
)
```

Switches the bus to 4 bits mode.

**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>mode</td>
<td>bus mode</td>
</tr>
</tbody>
</table>

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

### 7.37.7.33 sdc_lld_send_cmd_none()

```c
void sdc_lld_send_cmd_none (  
    SDCDriver * sdcp,  
    uint8_t cmd,  
    uint32_t arg  
)
```

Sends an SDIO command with no response expected.

**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>
</tbody>
</table>

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

### 7.37.7.34 sdc_lld_send_cmd_short()

```c
bool sdc_lld_send_cmd_short (  
    SDCDriver * sdcp,  
)
```
7.37 SDC Driver

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>sdcp</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>
<tr>
<td>out</td>
<td>resp</td>
<td>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.

### 7.37.7.35 sdc_lld_send_cmd_short_crc()

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.

**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>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 (one word)</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>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.

### 7.37.7.36  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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdcp pointer to the SDCDriver object</td>
</tr>
<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 (four words)</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>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.
7.37 SDC Driver

7.37.37 sdc_lld_read()

bool sdc_lld_read (  
    SDCDriver * sdp,  
    uint32_t startblk,  
    uint8_t * buf,  
    uint32_t n )

Reads one or more blocks.

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>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.

7.37.38 sdc_lld_write()

bool sdc_lld_write (  
    SDCDriver * sdp,  
    uint32_t startblk,  
    const uint8_t * buf,  
    uint32_t n )

Writes one or more blocks.

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>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</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:
Not an API, this function is for internal use only.

7.37.7.39 sdc_lld_sync()

bool sdc_lld_sync ( SDCCDriver * sdc )

Waits for card idle condition.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdc</td>
<td>pointer to the SDCCDriver object</td>
</tr>
</tbody>
</table>

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Operation</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.

7.37.8 Variable Documentation

7.37.8.1 sdc_vmt

c const struct SDCCDriverVMT 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.

### 7.37.8.2 SDCD1

**SDCDriver** SDCD1

SDCD1 driver identifier.
7.38 Serial Driver

Generic Serial Driver.

7.38.1 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`.

7.38.2 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

- `#define SD_PARITY_ERROR (eventflags_t)32`
  Parity.

- `#define SD_FRAMING_ERROR (eventflags_t)64`
  Framing.

- `#define SD_OVERRUN_ERROR (eventflags_t)128`
  Overflow.

- `#define SD_NOISE_ERROR (eventflags_t)256`
  Line noise.

- `#define SD_BREAK_DETECTED (eventflags_t)512`
  LIN Break.

- `#define SD_QUEUE_FULL_ERROR (eventflags_t)1024`
  Queue full.

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 sdGetI(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 blocking write to a `SerialDriver`.

- `#define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)`
  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, TIME_INFINITE)`
  Direct blocking read from a `SerialDriver`. 
• #define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)
  Direct blocking read from a SerialDriver.

• #define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->iqueue, b, n, t)
  Direct blocking read from a SerialDriver with timeout specification.

• #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 SerialDriver SerialDriver
  Structure representing a serial driver.

Data Structures

• struct SerialDriverVMT
  SerialDriver virtual methods table.
• struct SerialDriver
  Full duplex serial driver class.
• struct SerialConfig
  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.
• void 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.

7.38.3 Macro Definition Documentation

7.38.3.1 SD_PARITY_ERROR

#define SD_PARITY_ERROR (eventflags_t)32

Parity.

7.38.3.2 SD_FRAMING_ERROR

#define SD_FRAMING_ERROR (eventflags_t)64

Framing.

7.38.3.3 SD_OVERRUN_ERROR

#define SD_OVERRUN_ERROR (eventflags_t)128

Overflow.

7.38.3.4 SD_NOISE_ERROR

#define SD_NOISE_ERROR (eventflags_t)256

Line noise.
7.38.3.5  SD_BREAK_DETECTED

#define SD_BREAK_DETECTED (eventflags_t)512

LIN Break.

7.38.3.6  SD_QUEUE_FULL_ERROR

#define SD_QUEUE_FULL_ERROR (eventflags_t)1024

Queue full.

7.38.3.7  SERIAL_DEFAULT_BITRATE

#define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.38.3.8  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.

7.38.3.9  _serial_driver_methods

#define _serial_driver_methods _base_asynchronous_channel_methods

SerialDriver specific methods.
7.38.3.10 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.

7.38.3.11 sdPut

#define sdPut(
    sdp,
    b) oqPut(&(sdp)->queue, 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.

7.38.3.12 sdPutTimeout

#define sdPutTimeout(
    sdp,
    b, t) oqPutTimeout(&(sdp)->queue, 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.
7.38.3.13 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.

7.38.3.14 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.

7.38.3.15 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.
7.38.3.16  sdWriteI

#define sdWriteI(
sdp,
b,
n) oqWriteI(&(sdp)->oqueue, b, n)

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:**

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

7.38.3.17  sdWrite

#define sdWrite(
sdp,
b,
n) oqWriteTimeout(&(sdp)->oqueue, 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.

7.38.3.18  sdWriteTimeout

#define sdWriteTimeout(
sdp,
b,
n,
t) oqWriteTimeout(&(sdp)->oqueue, 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.
7.38.3.19  sdAsynchronousWrite

#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.

7.38.3.20  sdReadI

#define sdReadI(  
    sdp,  
    b,  
    n
)  
  iqReadI(&(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:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.38.3.21  sdRead

#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.
7.38.3.22 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.

7.38.3.23 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.

7.38.3.24 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.
7.38.3.25 _serial_driver_data

#define _serial_driver_data

Value:

_value_base_asynchronous_channel_data

/* Driver state. */
.sdstate_t state;
/* Input queue. */
input_queue_t iqueue;
/* Output queue. */
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.

7.38.4 Typedef Documentation

7.38.4.1 SerialDriver

typedef struct SerialDriver SerialDriver

Structure representing a serial driver.

7.38.5 Enumeration Type Documentation

7.38.5.1 sdstate_t

enum sdstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SD_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>SD_READY</td>
<td>Ready.</td>
</tr>
</tbody>
</table>
7.38.6 Function Documentation

7.38.6.1 sdInit()

```c
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:

```
sdInit ─ sd_lld_init ─ sdObjectInit
```

7.38.6.2 sdObjectInit()

```c
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.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>sdp</th>
<th>pointer to a <code>SerialDriver</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>inotify</td>
<td>pointer to a callback function that is invoked when some data is read from the Queue. The value can be <code>NULL</code>.</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 <code>NULL</code>.</td>
</tr>
</tbody>
</table>
Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.38.6.3 sdStart()

```c
void sdStart (
    SerialDriver * sdp,
    const SerialConfig * config )
```

Configures and starts the driver.

**Parameters**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdp</td>
<td>pointer to a SerialDriver object</td>
</tr>
<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:
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)

7.38.6.4 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:

```
sdStop
iqResetI
oqResetI
osalOsRescheduleS
osalSysLock
osalSysUnlock
sd_lld_stop
osalThreadDequeueAll
```

### 7.38.6.5 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

| in | sdp          | pointer to a SerialDriver structure |
|    | b            | the byte to be written in the driver's Input 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.

Here is the call graph for this function:

![Call Graph](image)

### 7.38.6.6 sdRequestDataI()

```c
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:

![Call Graph Diagram](image-url)

### 7.38.6.7 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.

**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
```

7.38.6.8 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
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)

### 7.38.6.9 sdControl()

```c
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 pointer to a <code>SerialDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td>operation control operation code</td>
</tr>
<tr>
<td>in,out</td>
<td>arg 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><code>MSG_OK</code></td>
<td>in case of success.</td>
</tr>
<tr>
<td><code>MSG_TIMEOUT</code></td>
<td>in case of operation timeout.</td>
</tr>
<tr>
<td><code>MSG_RESET</code></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.
7.38.6.10  sd_lld_init()

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
```

7.38.6.11  sd_lld_start()

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.

7.38.6.12  sd_lld_stop()

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.

7.38.7 Variable Documentation

7.38.7.1 SD1

SerialDriver SD1

USART1 serial driver identifier.

7.38.7.2 default_config

const SerialConfig default_config [static]

Initial value:

```
38400
```

Driver default configuration.
7.39 Serial over USB Driver

Serial over USB Driver.

7.39.1 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.

7.39.2 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 _serial_usb_driver_data SerialDriver specific data`
- `#define _serial_usb_driver_methods _base_asynchronous_channel_methods SerialUSBDriver specific methods`
7.39 Serial over USB Driver

**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 (io_buffers_queue_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.
- `void 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.
Enumerations

• enum sdustate_t { SDU_UNINIT = 0, SDU_STOP = 1, SDU_READY = 2 }

Driver state machine possible states.

7.39.3 Macro Definition Documentation

7.39.3.1 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.

7.39.3.2 SERIAL_USB_BUFFERS_NUMBER

#define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.39.3.3 _serial_usb_driver_data

#define _serial_usb_driver_data

Value:

_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.
7.39 Serial over USB Driver

7.39.3.4 _serial_usb_driver_methods

#define _serial_usb_driver_methods _base_asynchronous_channel_methods

SerialUSBDriver specific methods.

7.39.4 Typedef Documentation

7.39.4.1 SerialUSBDriver

typedef struct SerialUSBDriver SerialUSBDriver

Structure representing a serial over USB driver.

7.39.5 Enumeration Type Documentation

7.39.5.1 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>

7.39.6 Function Documentation

7.39.6.1 ibnotify()

static void ibnotify (  
    io_buffers_queue_t * bqp ) [static]

Notification of empty buffer released into the input buffers queue.
7.39.6.2 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](attachment:image.png)

7.39.6.3 sdulnit()

```c
void sdulnit (  
    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.
7.39 Serial over USB Driver

7.39.6.4 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.

7.39.6.5 sduStart()

```c
void 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 |

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:
### 7.39.6.6 sduStop()

```c
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:

![Call Graph for sduStop()](image)

### 7.39.6.7 sduSuspendHookI()

```c
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.
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.

7.39.6.8  sduWakeupHookI()

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() in 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.

7.39.6.9  sduConfigureHookI()

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.

Here is the call graph for this function:

```
sduConfigureHookI
  ibqResetI
  obqResetI
  osalThreadDequeueAllI
```

7.39.6.10  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

| in  | usbp | pointer to the USBDriver object |

Returns

The hook status.

Return values

| true | Message handled internally. |
| false | Message not handled. |
7.39.6.11  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:

```
sduSOFHookI
   |                   |
   |                   |
   | errorFlushI       |
   |                   |
   |                   |
   |                   |
   |                   |
   obqGetFullBufferI
   obqTryFlushI
   usbStartTransmitI
   usb_lld_start_in
```

7.39.6.12  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 |
| in   | ep   | IN endpoint number |

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

```
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

<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>OUT endpoint number</td>
</tr>
</tbody>
</table>

Here is the call graph for this function:
7.39.6.14  sduInterruptTransmitted()

```c
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bp</strong></td>
<td>pointer to the <strong>USBDriver</strong> object</td>
<td></td>
</tr>
<tr>
<td><strong>ep</strong></td>
<td>endpoint number</td>
<td></td>
</tr>
</tbody>
</table>

7.39.6.15  sduControl()

```c
msg_t sduControl (  
    USBDriver * usbp,  
    unsigned int operation,  
    void * arg )
```

Control operation on a serial USB port.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bp</strong></td>
<td>pointer to a <strong>USBDriver</strong> object</td>
<td></td>
</tr>
<tr>
<td><strong>operation</strong></td>
<td>control operation code</td>
<td></td>
</tr>
<tr>
<td><strong>arg</strong></td>
<td>operation argument</td>
<td></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.
7.40 SIO Driver

Generic SIO Driver.

7.40.1 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.

7.40.2 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).
Macros

- `#define _sio_driver_methods__base_channel_methods` 
  SIODriver specific methods.
- `#define sioGetFlagsX(siop) sio_lld_get_flags(siop)`
  Returns the current set of flags and clears it.
- `#define sioIsRXEmptyX(siop) sio_lld_is_rx_empty(siop)`
  Determines the state of the RX FIFO.
- `#define sioIsTXFullX(siop) sio_lld_is_tx_full(siop)`
  Determines the state of the TX FIFO.
- `#define sioGetAndClearEventsI(siop) sio_lld_get_and_clear_events(siop)`
  Returns the pending SIO events 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 sioAsyncReadI(siop, size, buffer) sio_lld_read(siop, size, buffer)`
  Reads data from the RX FIFO.
- `#define sioAsyncWriteI(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_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_tx_full(siop) false` 
  Determines the state of the TX FIFO.
- `#define sio_lld_is_tx_ongoing(siop) false` 
  Determines the transmission state.

SIO status flags

- `#define SIO_NO_ERROR 0` 
  No pending conditions.
- `#define SIO_PARITY_ERROR 4` 
  Parity error happened.
- `#define SIO_FRAMING_ERROR 8` 
  Framing error happened.
- `#define SIO_OVERRUN_ERROR 16` 
  Overflow happened.
- `#define SIO_NOISE_ERROR 32` 
  Noise on the line.
- `#define SIO_BREAK_DETECTED 64` 
  Break detected.
SIO additional messages

- #define SIO_MSG_IDLE 1
- #define SIO_MSG_ERRORS 2

SIO configuration options

- #define SIO_DEFAULT_BITRATE 38400
  Default bit rate.
- #define SIO_USE_SYNCHRONIZATION TRUE
  Support for thread synchronization API.

Low level driver helper macros

- #define __sio_callback_rx(siop) RX callback.
- #define __sio_callback_rx_idle(siop) RX idle callback.
- #define __sio_callback_tx(siop) TX callback.
- #define __sio_callback_tx_end(siop) TX end callback.
- #define __sio_callback_rx_evt(siop) RX event callback.
- #define __sio_wakeup_rx(siop, msg) Wakes up the RX-waiting thread.
- #define __sio_wakeup_tx(siop, msg) Wakes up the TX-waiting thread.
- #define __sio_wakeup_txend(siop, msg) Wakes up the TXend-waiting thread.

PLATFORM configuration options

- #define PLATFORM_SIO_USE_SIO1 FALSE
  SIO driver enable switch.

Typedefs

- typedef uint8_t sioflags_t
  SIO driver condition flags type.
- 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 struct hal_sio_operation SIOOperation
  Type of structure representing a SIO operation.
- typedef void(* siocb_t)(SIODriver *siop)
  Generic SIO notification callback type.
- typedef uint32_t sio_events_mask_t
  Type of a SIO events mask.
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.
- struct hal_sio_operation
  Structure representing a SIO operation.

Functions

- void sioInit (void)
  SIO Driver initialization.
- void sioObjectInit (SIODriver ∗siop)
  Initializes the standard part of a SIODriver structure.
- bool sioStart (SIODriver ∗siop, const SIOConfig ∗config)
  Configures and activates the SIO peripheral.
- void sioStop (SIODriver ∗siop)
  Deactivates the SIO peripheral.
- void sioStartOperation (SIODriver ∗siop, const SIOOperation ∗operation)
  Starts a SIO operation.
- void sioStopOperation (SIODriver ∗siop)
  Stops an ongoing SIO operation, if any.
- sio_events_mask_t sioGetAndClearEvents (SIODriver ∗siop)
  Returns the pending SIO events 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 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.
- bool sio_lld_start (SIODriver ∗siop)
  Configures and activates the SIO peripheral.
- void sio_lld_stop (SIODriver ∗siop)
  Deactivates the SIO peripheral.
- void sio_lld_start_operation (SIODriver ∗siop)
  Starts a SIO operation.
- void sio_lld_stop_operation (SIODriver ∗siop)
  Stops an ongoing SIO operation, if any.
- sio_events_mask_t sio_lld_get_and_clear_events (SIODriver ∗siop)
  Returns the pending SIO events 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.

Enumerations

- `enum siostate_t { SIO_UNINIT = 0, SIO_STOP = 1, SIO_READY = 2, SIO_ACTIVE = 3 }`
  Driver state machine possible states.

Variables

- `SIODriver SIO1`
  SIO1 driver identifier.

7.40.3 Macro Definition Documentation

7.40.3.1 SIO_NO_ERROR

```c
#define SIO_NO_ERROR 0
```
No pending conditions.

7.40.3.2 SIO_PARITY_ERROR

```c
#define SIO_PARITY_ERROR 4
```
Parity error happened.

7.40.3.3 SIO_FRAMING_ERROR

```c
#define SIO_FRAMING_ERROR 8
```
Framing error happened.
7.40.3.4 SIO_OVERRUN_ERROR

#define SIO_OVERRUN_ERROR 16

Overflow happened.

7.40.3.5 SIO_NOISE_ERROR

#define SIO_NOISE_ERROR 32

Noise on the line.

7.40.3.6 SIO_BREAK_DETECTED

#define SIO_BREAK_DETECTED 64

Break detected.

7.40.3.7 SIO_DEFAULT_BITRATE

#define SIO_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.40.3.8 SIO_USE_SYNCHRONIZATION

#define SIO_USE_SYNCHRONIZATION TRUE

Support for thread synchronization API.

7.40.3.9 _sio_driver_methods

#define _sio_driver_methods _base_channel_methods

SIODriver specific methods.
7.40.3.10  *sioGetFlagsX*

```c
#define sioGetFlagsX(sio) sio_lld_get_flags(sio)
```

Returns the current set of flags and clears it.

7.40.3.11  *sioIsRXEmptyX*

```c
#define sioIsRXEmptyX(sio) sio_lld_is_rx_empty(sio)
```

Determines the state of the RX FIFO.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sio</td>
</tr>
</tbody>
</table>

**Returns**

The RX FIFO state.

**Return values**

- `false` if RX FIFO is not empty
- `true` if RX FIFO is empty

**Function Class:**

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

7.40.3.12  *sioIsTXFullX*

```c
#define sioIsTXFullX(sio) sio_lld_is_tx_full(sio)
```

Determines the state of the TX FIFO.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sio</td>
</tr>
</tbody>
</table>

Returns

The TX FIFO state.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if TX FIFO is not full</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if TX FIFO is full</td>
</tr>
</tbody>
</table>

Function Class:

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

7.40.3.13  sioGetAndClearEventsI

#define sioGetAndClearEventsI(siol)
  siolld_get_and_clear_events(siol)

Return the pending SIO events flags.

Parameters

| in   | siop | pointer to the SIODriver 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.

7.40.3.14  sioGetX

#define sioGetX(siol)
  siolld_get(siol)

Returns one frame from the RX FIFO.

Note

If the FIFO is empty then the returned value is unpredictable.
666 Module Documentation

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.

7.40.3.15  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

| in  | siop | pointer to the SIODriver object |
| in  | data | frame to be written |

Function Class:

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

7.40.3.16  sioAsyncReadI

#define sioAsyncReadI(
   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.
7.40 SIO Driver

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>in</td>
<td>buffer</td>
<td>buffer for the received data</td>
</tr>
</tbody>
</table>

Returns

The number of received frames.

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.

7.40.3.17 sioAsyncWriteI

#define sioAsyncWriteI(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 I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
7.40.3.18 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**

| MSG_OK | in case of success. |
| MSG_TIMEOUT | in case of operation timeout. |
| MSG_RESET | in case of operation reset. |

**Function Class:**

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

7.40.3.19 __sio_callback_rx

```c
#define __sio_callback_rx(
    siop
)
```

**Value:**

```c
if ((siop)->operation->rx_cb != NULL) {
    (siop)->operation->rx_cb(siop);
}
```

RX callback.

**Parameters**

| in | siop | pointer to the SIODriver object |
7.40 SIO Driver

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

7.40.3.20 __sio_callback_rx_idle

#define __sio_callback_rx_idle(
    siop )

Value:
{ if ((siop)->operation->rx_idle_cb != NULL) {
    (siop)->operation->rx_idle_cb(siop);
}
}

RX idle callback.

Parameters

| in   | siop      | pointer to the SIODriver object |

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

7.40.3.21 __sio_callback_tx

#define __sio_callback_tx(
    siop )

Value:
{ if ((siop)->operation->tx_cb != NULL) {
    (siop)->operation->tx_cb(siop);
}
}

TX callback.

Parameters

| in   | siop      | pointer to the SIODriver object |

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

#define __sio_callback_tx_end( 
    siop )

Value:
{}
    if ((siop)->operation->tx_end_cb != NULL) {
        (siop)->operation->tx_end_cb(siop);
    }

TX end callback.

Parameters

    in  siop  pointer to the SIOC_driver object

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

7.40.3.23 __sio_callback_rx_evt

#define __sio_callback_rx_evt( 
    siop )

Value:
{}
    if ((siop)->operation->rx_evt_cb != NULL) {
        (siop)->operation->rx_evt_cb(siop);
    }

RX event callback.

Parameters

    in  siop  pointer to the SIOC_driver object

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

7.40.3.24 __sio_wakeup_rx

#define __sio_wakeup_rx( 
    siop, 
    msg )
Wakes up the RX-waiting thread.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{siop}</td>
<td>pointer to the 	exttt{SIODriver} object</td>
</tr>
<tr>
<td>\texttt{msg}</td>
<td>the wake up message</td>
</tr>
</tbody>
</table>

Function Class:

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

7.40.3.25 \texttt{__sio_wakeup_tx}

\begin{verbatim}
#define __sio_wakeup_tx(
     siop, \\
    msg )
\end{verbatim}

Wakes up the TX-waiting thread.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{siop}</td>
<td>pointer to the 	exttt{SIODriver} object</td>
</tr>
<tr>
<td>\texttt{msg}</td>
<td>the wake up message</td>
</tr>
</tbody>
</table>

Function Class:

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

7.40.3.26 \texttt{__sio_wakeup_txend}

\begin{verbatim}
#define __sio_wakeup_txend( 
     siop, \\
    msg )
\end{verbatim
osalSysLockFromISR();
osalThreadResumeI(&(siop)->sync_txend, msg);
osalSysUnlockFromISR();
}

Wakes up the TXend-waiting thread.

Parameters

| in  | siop | pointer to the SIODriver object |
| in  | msg  | the wake up message            |

Function Class:

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

7.40.3.27 PLATFORM_SIO_USE_SIO1

#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.

7.40.3.28 sio_lld_driver_fields

#define sio_lld_driver_fields uint32_t dummy

Low level fields of the SIO driver structure.

7.40.3.29 sio_lld_config_fields

#define sio_lld_config_fields uint32_t dummy

Low level fields of the SIO configuration structure.

7.40.3.30 sio_lld_is_rx_empty

#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.

7.40.3.31  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.

7.40.3.32  sio_lld_is_tx_ongoing

#define sio_lld_is_tx_ongoing(
    siop ) false
Determines the transmission state.

**Parameters**

| in | sio | 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.

### 7.40.4 Typedef Documentation

#### 7.40.4.1 sioflags_t

```c
typedef uint_fast8_t sioflags_t
```

SIO driver condition flags type.

#### 7.40.4.2 SIODriver

```c
typedef struct hal_sio_driver SIODriver
```

Type of structure representing a SIO driver.

#### 7.40.4.3 SIOConfig

```c
typedef struct hal_sio_config SIOConfig
```

Type of structure representing a SIO configuration.
### 7.40.4.4 SIOOperation

typedef struct hal_sio_operation SIOOperation

Type of structure representing a SIO operation.

### 7.40.4.5 siocb_t

typedef void(* siocb_t) (SIODriver *siop)

Generic SIO notification callback type.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>siop</td>
<td>pointer to the SIODriver object</td>
</tr>
</tbody>
</table>

### 7.40.4.6 sio_events_mask_t

typedef uint32_t sio_events_mask_t

Type of a SIO events mask.

### 7.40.5 Enumeration Type Documentation

#### 7.40.5.1 siostate_t

enum siostate_t

Driver state machine possible states.

**Enumerator**

<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>
<tr>
<td>SIO_ACTIVE</td>
<td>Operation ongoing.</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
7.40.6 Function Documentation

7.40.6.1 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
```

7.40.6.2 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.
7.40 SIO Driver

7.40.6.3 sioStart()

```c
bool sioStart (  
    SIODriver * siop,  
    const SIOConfig * config )
```

Configures and activates the SIO peripheral.

**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>config</td>
<td>pointer to the SIOConfig object, can be NULL if the default configuration is desired</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

| false | if the driver has been correctly started. |
| true  | if an 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:

![Call Graph](image)

7.40.6.4 sioStop()

```c
void sioStop (  
    SIODriver * siop )
```

Deactivates the SIO peripheral.
7.40.6.5 **sioStartOperation()**

```c
void sioStartOperation (  
   SIODriver * siop,  
   const SIOOperation * operation )
```

Starts a SIO operation.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to a SIODriver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>operation</td>
<td>pointer to a SIOOperation structure, can be NULL if callbacks are not required encoding the operation to be performed</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]

7.40.6.6 sioStopOperation()

void sioStopOperation (SIODriver * siop )

Stops an ongoing SIO operation, if any.

Parameters

in  siop  pointer to an SIODriver structure

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]
7.40.6.7  sioGetAndClearEvents()

sio_events_mask_t sioGetAndClearEvents (  
  SIODriver * siop  )

Return the pending SIO events flags.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
</table>

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

osalSysUnlock
```

7.40.6.8  sioAsyncRead()

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:

```
sioAsyncRead
  ↓
osalSysLock

sioAsyncWrite
  ↓
osalSysUnlock
```

7.40.6.9 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
```

### 7.40.6.10 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**

- `siop` pointer to an SIODriver structure
- `timeout` synchronization timeout

**Returns**

The synchronization result.

**Return values**

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if there is data in the RX FIFO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if synchronization timed out.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>MSG_RESET</th>
<th>operation has been stopped while waiting.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_MSG_IDLE</td>
<td>if RX line went idle.</td>
</tr>
<tr>
<td>SIO_MSG_ERRORS</td>
<td>if RX errors occurred during wait.</td>
</tr>
</tbody>
</table>

7.40.6.11  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>in</th>
<th>siop</th>
<th>pointer to an SIODriver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>synchronization timeout</td>
</tr>
</tbody>
</table>

Returns

The synchronization result.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if there is space in the TX FIFO.</th>
</tr>
</thead>
<tbody>
<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>

7.40.6.12  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>siop</th>
<th>pointer to an SIODriver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>synchronization timeout</td>
</tr>
</tbody>
</table>

Returns

The synchronization result.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if TX operation finished.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if synchronization timed out.</td>
</tr>
</tbody>
</table>

Here is the call graph for this function:

```
sioSynchronizeTXEnd
  ↓
osalSysLock
  ↓
osalThreadSuspendTimeoutS
```

7.40.6.13  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:

```
sio_lld_init
  ↓
sioObjectInit
```
7.40.6.14 sio_lld_start()

bool sio_lld_start (    
    SIODriver * siop )

Configures and activates the SIO peripheral.

Parameters

| in | siop | pointer to the SIODriver object |

Returns

- The operation status.

Return values

| false | if the driver has been correctly started. |
| true  | if an error occurred.                   |

Function Class:

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

7.40.6.15 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.

7.40.6.16 sio_lld_start_operation()

void sio_lld_start_operation (    
    SIODriver * siop )

Starts a SIO operation.
Parameters

in  siop  pointer to an SIODriver structure

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

7.40.6.17  sio_lld_stop_operation()

void sio_lld_stop_operation (  
    SIODriver  *  siop  )

Stops an ongoing SIO operation, if any.

Parameters

in  siop  pointer to an SIODriver structure

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

7.40.6.18  sio_lld_get_and_clear_events()

sio_events_mask_t sio_lld_get_and_clear_events (  
    SIODriver  *  siop  )

Return the pending SIO events flags.

Parameters

in  siop  pointer to the SIODriver object

Returns
The pending event flags.

Function Class:
Not an API, this function is for internal use only.
7.40.6.19  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

<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 read</td>
</tr>
</tbody>
</table>

Returns

The number of frames copied from the buffer.

Return values

0  if the TX FIFO is full.

7.40.6.20  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.

7.40.6.21  sio_lld_get()

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

<table>
<thead>
<tr>
<th></th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
</table>

Returns

The frame from RX FIFO.

Function Class:

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

7.40.6.22  sio_lld_put()

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.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
</table>

|    | data | frame to be written             |
Function Class:
Not an API, this function is for internal use only.

7.40.6.23 sio_lld_control()

```c
msg_t sio_lld_control ( 
    SIODriver * siop, 
    unsigned int operation, 
    void * arg )
```

Control operation on a serial port.

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>
<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>Return 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:
Not an API, this function is for internal use only.

7.40.7 Variable Documentation

7.40.7.1 SIOD1

SIODriver SIOD1

SIO1 driver identifier.
7.41 SPI Driver

Generic SPI Driver.

7.41.1 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`.

7.41.2 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).

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_CIRCULAR FALSE`  
  Enables circular transfers 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.
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 spiInit (void)
  SPI Driver initialization.
- void spiObjectInit (SPIDriver ∗spip)
  Initializes the standard part of a SPIDriver structure.
- void spiStart (SPIDriver ∗spip, const SPIConfig ∗config)
  Configures and activates the SPI peripheral.
- void spiStop (SPIDriver ∗spip)
  Deactivates the SPI peripheral.
- void spiSelect (SPIDriver ∗spip)
  Asserts the slave select signal and prepares for transfers.
- void spiUnselect (SPIDriver ∗spip)
  Deasserts the slave select signal.
- void spiStartIgnore (SPIDriver ∗spip, size_t n)
  Ignores data on the SPI bus.
- void spiStartExchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)
  Exchanges data on the SPI bus.
- void spiStartSend (SPIDriver ∗spip, size_t n, const void ∗txbuf)
  Sends data over the SPI bus.
- void spiStartReceive (SPIDriver ∗spip, size_t n, void ∗rxbuf)
  Receives data from the SPI bus.
- void spiAbortI (SPIDriver ∗spip)
  Aborts the ongoing SPI operation.
- void spiIgnore (SPIDriver ∗spip, size_t n)
  Ignores data on the SPI bus.
• void spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  Exchanges data on the SPI bus.
• void spiSend (SPIDriver *spip, size_t n, const void *txbuf)
  Sends data over the SPI bus.
• void spiReceive (SPIDriver *spip, size_t n, void *rxbuf)
  Receives data from the SPI bus.
• void spiAcquireBus (SPIDriver *spip)
  Gains exclusive access to the SPI bus.
• void spiReleaseBus (SPIDriver *spip)
  Releases exclusive access to the SPI bus.
• 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_lldIgnore (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.

Enumerations

• enum spistate_t {
    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.

7.41.3 Macro Definition Documentation
7.41.3.1  SPI_USE_WAIT

#define SPI_USE_WAIT TRUE
Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.41.3.2  SPI_USE_CIRCULAR

#define SPI_USE_CIRCULAR FALSE
Enables circular transfers APIs.

Note
Disabling this option saves both code and data space.

7.41.3.3  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.

7.41.3.4  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.

7.41.3.5  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 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.

### 7.41.3.6 spiSelectI

```c
#define spiSelectI( spip )
```

**Value:**

```c
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.

### 7.41.3.7 spiUnselectI

```c
#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.

7.41.3.8 spiStartIgnoreI

```c
#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

| in | spip | pointer to the SPIDriver object |
| in | n    | number of words to be ignored |

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.

7.41.3.9 spiStartExchangeI

```c
#define spiStartExchangeI(
    spip,
```

ChibiOS/HAL
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>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>

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.

7.41.3.10 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.

7.41.3.11 spiStartReceiveI

```c
#define spiStartReceiveI(spip, n, txbuf)

{ (spip)->state = SPI_ACTIVE;
  spi_lld_receive(spip, n, txbuf); }
```

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.
7.41 SPI 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 receive</td>
</tr>
<tr>
<td>out</td>
<td>rbuf</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.

7.41.3.12 spiPolledExchange

#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.

7.41.3.13 _spi_wakeup_isr

#define _spi_wakeup_isr(
    spip 
)

Value:

osalSysLockFromISR();
osalThreadResumeI(&(spip)->thread, MSG_OK);
osalSysUnlockFromISR();

Wakes up the waiting thread.
Parameters

\texttt{in \textit{spip}} \text{ pointer to the \texttt{SPIDriver} object}

Function Class:

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

7.41.3.14 \_\_spi\_isr\_code

\begin{verbatim}
#define _spi_isr_code(
  spip )

Value:

\begin{verbatim}
  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);
  }
\end{verbatim}

Common ISR code when circular mode is not supported.

This code handles the portable part of the ISR code:

\begin{itemize}
  \item Callback invocation.
  \item Waiting thread wakeup, if any.
  \item Driver state transitions.
\end{itemize}

Note

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

Parameters

\texttt{in \textit{spip}} \text{ pointer to the \texttt{SPIDriver} object}

Function Class:

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

```c
#define _spi_isr_half_code(spip)
```

**Value:**

```c
{  
    if ((spip)->config->end_cb) {  
        (spip)->config->end_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.

### 7.41.3.16 _spi_isr_full_code

```c
#define _spi_isr_full_code(spip)
```

**Value:**

```c
{  
    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   | spi   | pointer to the SPI Driver object |

Function Class:

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

### 7.41.3.17 SPI_SUPPORTS_CIRCULAR

```c
#define SPI_SUPPORTS_CIRCULAR TRUE
```

Circular mode support flag.

### 7.41.3.18 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.

### 7.41.3.19 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.

### 7.41.3.20 spi_lld_config_fields

```c
#define spi_lld_config_fields
```

**Value:**

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

Low level fields of the SPI configuration structure.
7.41 SPI Driver

7.41.4 Typedef Documentation

7.41.4.1 SPIDriver

typedef struct hal_spi_driver SPIDriver

Type of a structure representing an SPI driver.

7.41.4.2 SPIConfig

typedef struct hal_spi_config SPIConfig

Type of a SPI driver configuration structure.

7.41.4.3 spicallback_t

typedef void(* spicallback_t) (SPIDriver *spip)

SPI notification callback type.

Parameters

| in  | spip | pointer to the SPIDriver object triggering the callback |

7.41.5 Enumeration Type Documentation

7.41.5.1 spistate_t

enum spistate_t

Driver state machine possible states.

Enumerator

<table>
<thead>
<tr>
<th>spistate_t</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>
</tbody>
</table>
7.41.6 Function Documentation

7.41.6.1 spiInit()

```c
void spiInit ( )
```

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.

Here is the call graph for this function:

![Call Graph](https://via.placeholder.com/150)

7.41.6.2 spiObjectInit()

```c
void spiObjectInit ( SPIDriver * spip )
```

Initializes the standard part of a `SPIDriver` structure.
Parameters

| out | spip | pointer to the SPIDriver 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:

spiObjectInit ➔ osalMutexObjectInit

7.41.6.3 spiStart()

void spiStart (  
    SPIDriver * spip,  
    const SPIConfig * config )

Configures and activates the SPI peripheral.

Parameters

| in  | spip | pointer to the SPIDriver object |
| in  | config | pointer to the SPIConfig 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:

```
spiStart
osalSysLock
osalSysUnlock
spi_lld_start
```

### 7.41.6.4 spiStop()

```c
void spiStop (SPIDriver * spip )
```

Deactivates the SPI peripheral.

**Parameters**

```
in  spip  pointer to the SPIDriver object
```
Here is the call graph for this function:

```
osalSysLock
spiStop
osalSysUnlock
spi_lld_stop
```

### 7.41.6.5 spiSelect()

```c
void spiSelect (
    SPIDriver * spip
)
```

Asserts the slave select signal and prepares for transfers.

**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.

Here is the call graph for this function:

```
osalSysLock
spiSelect
osalSysUnlock
```
### 7.41.6.6 spiUnselect()

```c
void spiUnselect (SPIDriver * spip)
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

**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.

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.41.6.7 spiStartIgnore()

```c
void spiStartIgnore (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.

**Precondition**
- A slave must have been selected using `spiSelect()` or `spiSelectI()`.

**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:

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:

```
spiStartIgnore
  osalSysLock
osalSysUnlock
```

### 7.41.6.8 spiStartExchange()

```c
void spiStartExchange (  
    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.

**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 SPI Driver 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>

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:

```
spiStartExchange
  osalSysLock
  spiStartSend()
  osalSysUnlock
```

7.41.6.9 spiStartSend()

```c
void spiStartSend (  
    SPIDriver * spip,  
    size_t n,        
    const void * 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spip</code></td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td><code>n</code></td>
<td>number of words to send</td>
</tr>
<tr>
<td><code>txbuf</code></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:

```
spiStartSend → osalSysLock
spicStartSend → osalSysUnlock
```

---

#### 7.41.6.10  spiStartReceive()

```c
void spiStartReceive (  
    SPIDriver * spip,
    size_t n,
    void * 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.
`spiStartReceive` function:

```c
void spiStartReceive (SPIDriver ∗ spip)
```

Aborts the ongoing SPI operation.

Parameters

<table>
<thead>
<tr>
<th>Mode</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>spip</code></td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code></td>
<td>number of words to receive</td>
</tr>
<tr>
<td>out</td>
<td><code>rxbuf</code></td>
<td>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:

```
spiStartReceive
    ↘ osalSysLock
   ↗ osalSysUnlock
```

### 7.41.6.11 spiAbortI()

Aborts the ongoing SPI operation.

Parameters

<table>
<thead>
<tr>
<th>Mode</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>spip</code></td>
<td>pointer to the SPIDriver 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:

![Call Graph Image]

### 7.41.6.12 spiAbort()

```c
void spiAbort ( 
    SPIDriver * spip )
```

Aborts the ongoing SPI operation, if any.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>spip</th>
<th>pointer to the SPIDriver object</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)

7.41.6.13 spiIgnore()

```c
void spiIgnore ( 
    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_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>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>n</td>
<td>number of words to be ignored</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:

```
spiIgnore
  └── osalSysLock
     └── osalSysUnlock
     └── osalThreadSuspendS
```

### 7.41.6.14  spiExchange()

```c
void spiExchange ( 
    SPIDriver * spip, 
    size_t n, 
    const void * txbuf, 
    void * rxbuf )
```

Exchanges data on the SPI bus.

This synchronous function performs a simultaneous transmit/receive operation.

Precondition

In order to use this function the option SPI_USE_WAIT must be enabled.

In order to use this function the driver must have been configured without callbacks (end_cb = NULL).

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>
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:

```
spiExchange
     osalSysLock

osalSysUnlock

osalThreadSuspendS
```

### 7.41.6.15 spiSend()

```c
void spiSend (SPIDriver ∗spip,
             size_t n,
             const void ∗txbuf )
```

Sends data over the SPI bus.

This synchronous function performs a transmit operation.

**Precondition**

- In order to use this function the option SPI_USE_WAIT must be enabled.
- In order to use this function the driver must have been configured without callbacks (end_cb = NULL).

**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>spip</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</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:

```
spiSend
   ↓
osalSysLock

osalSysUnlock
```

7.41.6.16  spiReceive()

```c
void spiReceive ( 
    SPIDriver * spip, 
    size_t  n, 
    void *  rxbuf )
```

Receives data from the SPI bus.

This synchronous function performs a receive operation.

Precondition

In order to use this function the option SPI_USE_WAIT must be enabled.

In order to use this function the driver must have been configured without callbacks (end_cb = NULL).

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:

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:

```
spiReceive
osalSysLock
osalSysUnlock
osalThreadSuspendS
```

7.41.6.17 spiAcquireBus()

```c
void spiAcquireBus ( 
    SPIDriver ∗ spip )
```

Gains exclusive access to the SPI bus.

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

Precondition

In order to use this function the option SPI_USE_MUTUAL_EXCLUSION must be enabled.

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.

Here is the call graph for this function:

```
spiAcquireBus -> osalMutexLock
```

### 7.41.6.18  spiReleaseBus()

```c
void spiReleaseBus ( 
    SPIDriver * spip )
```

Releases exclusive access to the SPI bus.

Precondition
In order to use this function the option `SPI_USE_MUTUAL_EXCLUSION` must be enabled.

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.

Here is the call graph for this function:

```
spiReleaseBus -> osalMutexUnlock
```
7.41.6.19 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.

Here is the call graph for this function:

```
spi_lld_init      
|               |               
| spiObjectInit  | osalMutexObjectInit |
```

7.41.6.20 spi_lld_start()

```c
void spi_lld_start ( SPIDriver * spip )
```

Configures and activates the SPI peripheral.

Parameters

```
in  spip  pointer to the SPIDriver object
```

Function Class:

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

7.41.6.21 spi_lld_stop()

```c
void spi_lld_stop ( SPIDriver * spip )
```

Deactivates the SPI peripheral.
7.41 SPI Driver

Parameters

| in   | *spip* | pointer to the SPIDriver object |

Function Class:

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

### 7.41.6.22 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.

### 7.41.6.23 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.
7.41.6.24  **spi_lld_ignore()**

```c
void 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**

<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>n</td>
<td>number of words to be ignored</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.41.6.25  **spi_lld_exchange()**

```c
void 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>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>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.

7.41.6.26 spi_lld_send()

```c
void 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.

7.41.6.27 spi_lld_receive()

```c
void 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>rbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

Function Class:

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

7.41.6.28  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.

7.41.6.29  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.
7.41 SPI Driver

Parameters

<table>
<thead>
<tr>
<th></th>
<th>spip</th>
<th>pointer to the SPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td></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.

Function Class:

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

7.41.7 Variable Documentation

7.41.7.1 SPID1

SPIDriver SPID1

SPI1 driver identifier.
7.42 ST Driver

Generic System Tick Driver.

7.42.1 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)
7.42 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.

7.42.2 Function Documentation

7.42.2.1 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.

7.42.2.2 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.
7.42.2.3 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:

![Call Graph](call_graph.png)

7.42.2.4 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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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.

Here is the call graph for this function:

```
void stStartAlarm (
    void )
```

**7.42.2.5 stStopAlarm()**

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:

```
void stSetAlarm (    
    systime_t abstime )
```

**7.42.2.6 stSetAlarm()**

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:

```
sfSetAlarm
    st_lld_set_alarm
sfIsAlarmActive
    st_lld_is_alarm_active
```

### 7.42.2.7 stGetAlarm()

```c
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:

```
sfGetAlarm
    st_lld_get_alarm
sfIsAlarmActive
    st_lld_is_alarm_active
```
7.42 ST Driver

7.42.2.8 stIsAlarmActive()

```c
bool stIsAlarmActive ( void )
```

Determines if the alarm zero is active.

Returns

The alarm status.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the alarm is not active.</td>
</tr>
<tr>
<td>true</td>
<td>is the alarm is active.</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:

```
    stIsAlarmActive
       ▸ st_lld_is_alarm_active
```

7.42.2.9 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>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>alarm</strong> alarm channel number (0..ST_LLD_NUM_ALARMS-1)</td>
</tr>
<tr>
<td>in</td>
<td><strong>cb</strong> 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.

7.42.2.10 stBindAlarmN()

```c
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.

7.42.2.11 stStartAlarmN()

```c
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.
7.42 ST Driver

7.42.2.12 stStopAlarmN() 

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>alarm</th>
<th>alarm channel number (0..ST_LLD_NUMALARMS-1)</th>
</tr>
</thead>
</table>

**Function Class:**

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

7.42.2.13 stSetAlarmN() 

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>alarm</th>
<th>alarm channel number (0..ST_LLD_NUMALARMS-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>abstime</td>
<td>the time to be set for the next 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.
### 7.42.2.14 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.

---

### 7.42.2.15 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**: 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.
7.42.2.16  st_lld_init()

void st_lld_init (  
   void  )

Low level ST driver initialization.

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

7.42.2.17  st_lld_get_counter()

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.

7.42.2.18  st_lld_start_alarm()

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.

7.42.19  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.

7.42.20  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.

7.42.21  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.
7.42.22  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

| false | if the alarm is not active. |
| true  | is the alarm is active      |

Function Class:

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

7.42.3  Variable Documentation

7.42.3.1  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).
7.43  TRNG Driver

Generic True Random Numbers Generator Driver.

7.43.1  Detailed Description

Generic True Random Numbers Generator Driver.

This module implements a generic TRNG driver.

Precondition

In order to use the TRNG driver the \texttt{HAL\_USE\_TRNG} option must be enabled in \texttt{halconf.h}.

Macros

- \texttt{#define trng\_lld\_driver\_fields}
  
  Low level fields of the TRNG driver structure.

- \texttt{#define trng\_lld\_config\_fields}
  
  Low level fields of the TRNG configuration structure.

PLATFORM configuration options

- \texttt{#define PLATFORM\_TRNG\_USE\_TRNG1 FALSE}
  
  TRNGD1 driver enable switch.

Typedefs

- \texttt{typedef struct hal\_trng\_driver TRNGDriver}
  
  Type of a structure representing a TRNG driver.

- \texttt{typedef struct hal\_trng\_config TRNGConfig}
  
  Driver configuration structure.

Data Structures

- \texttt{struct hal\_trng\_config}
  
  Driver configuration structure.

- \texttt{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.
• void 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.

7.43.2 Macro Definition Documentation

7.43.2.1 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.
### 7.43.2.2 trng_lld_driver_fields

```c
#define trng_lld_driver_fields

Value:
/* Dummy field, it is not needed.*/
uint32_t dummy
```

Low level fields of the TRNG driver structure.

### 7.43.2.3 trng_lld_config_fields

```c
#define trng_lld_config_fields

Value:
/* Dummy configuration, it is not needed.*/
uint32_t dummy
```

Low level fields of the TRNG configuration structure.

### 7.43.3 Typedef Documentation

#### 7.43.3.1 TRNGDriver

```c
typedef struct hal_trng_driver TRNGDriver
```

Type of a structure representing a TRNG driver.

#### 7.43.3.2 TRNGConfig

```c
typedef struct hal_trng_config TRNGConfig
```

Driver configuration structure.

**Note**

It could be empty on some architectures.

### 7.43.4 Enumeration Type Documentation

#### 7.43.4.1 trngstate_t

```c
enum trngstate_t
```

Driver state machine possible states.
7.43 TRNG Driver

### Enumerator

<table>
<thead>
<tr>
<th>TRNG_UNINIT</th>
<th>Not initialized.</th>
</tr>
</thead>
<tbody>
<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>

#### 7.43.5 Function Documentation

**7.43.5.1 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:

![Call Graph](call_graph.png)

**7.43.5.2 trngObjectInit()**

```c
void trngObjectInit (  
    TRNGDriver * trngp )
```

Initializes the standard part of a TRNGDriver structure.
7.43.5.3 trngStart()

void trngStart (  
    TRNGDriver * trngp,  
    const TRNGConfig * config  
)

Configures and activates the TRNG peripheral.

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:

7.43.5.4 trngStop()

void trngStop (  
    TRNGDriver * trngp  
)

Deactivates the TRNG peripheral.
Parameters

| in  | trngp | pointer to the TRNGDriver 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
trngStop
osalSysUnlock
trng_lld_stop
```

### 7.43.5.5 trngGenerate()

```c
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

| in  | trngp | pointer to the TRNGDriver object |
| in  | size  | size of output buffer |
| out | out   | output buffer |

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 \rightarrow trng_lld_generate
```

7.43.5.6  trng_lld_init()

```
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 \rightarrow trngObjectInit
```

7.43.5.7  trng_lld_start()

```
void trng_lld_start (  
    TRNDriver * trngp  
)
```

Configures and activates the TRNG peripheral.
7.43 TRNG Driver

Parameters

| in  | trngp | pointer to the TRNGDriver object |

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

7.43.5.8 trng_lld_stop()

```c
void trng_lld_stop (
    TRNGDriver * trngp )
```

Deactivates the TRNG peripheral.

Parameters

| in  | trngp | pointer to the TRNGDriver object |

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

7.43.5.9 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

| in  | trngp | pointer to the TRNGDriver object |
| in  | size  | size of output buffer |
| out | out   | output buffer |
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.

7.43.6 Variable Documentation

7.43.6.1 TRNGD1

TRNGDriver TRNGD1

TRNGD1 driver identifier.
7.44 UART Driver

Generic UART Driver.

7.44.1 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.

7.44.2 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).
7.44.2.1 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.

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 UARTDriver 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.

Data Structures

- `struct UARTConfig`
  Driver configuration structure.
- `struct UARTDriver`
  Structure representing an UART driver.

Functions

- `void uartInit (void)`
  UART Driver initialization.
- `void uartObjectInit (UARTDriver *uartp)`
  Initializes the standard part of a UARTDriver structure.
- `void 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 uartStopReceive (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 uarttxstate_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.

7.44.3 Macro Definition Documentation
7.44.3.1 UART_NO_ERROR

#define UART_NO_ERROR 0

No pending conditions.

7.44.3.2 UART_PARITY_ERROR

#define UART_PARITY_ERROR 4

Parity error happened.

7.44.3.3 UART_FRAMING_ERROR

#define UART_FRAMING_ERROR 8

Framing error happened.

7.44.3.4 UART_OVERRUN_ERROR

#define UART_OVERRUN_ERROR 16

Overflow happened.

7.44.3.5 UART_NOISE_ERROR

#define UART_NOISE_ERROR 32

Noise on the line.

7.44.3.6 UART_BREAK_DETECTED

#define UART_BREAK_DETECTED 64

Break detected.
7.44.3.7 UART_USE_WAIT

#define UART_USE_WAIT FALSE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.44.3.8 UART_USE_MUTUAL_EXCLUSION

#define UART_USE_MUTUAL_EXCLUSION FALSE

Enables the uartAcquireBus() and uartReleaseBus() APIs.

Note
Disabling this option saves both code and data space.

7.44.3.9 _uart_wakeup_tx1_isr

#define _uart_wakeup_tx1_isr(uartp)

Value:

\begin{verbatim}
if ((uartp)->early == true) {
    osalSysLockFromISR();
    osalThreadResumeI(&(uartp)->threadtx, MSG_OK);
    osalSysUnlockFromISR();
}
\end{verbatim}

Wakes up the waiting thread in case of early TX complete.

Parameters

\begin{itemize}
\item \textbf{in} \texttt{uartp} \hspace{1cm} pointer to the UARTDriver object
\end{itemize}

Function Class:

Not an API, this function is for internal use only.
7.44.3.10 _uart_wakeup_tx2_isr

#define _uart_wakeup_tx2_isr(
          uartp
)

Value:

\[
\text{if } ((\text{uartp})\rightarrow\text{early} == \text{false}) \{
  \text{osalSysLockFromISR();}
  \text{osalThreadResumeI(\{(uartp)\rightarrow\text{threadtx}, MSG_OK\);}
  \text{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.

7.44.3.11 _uart_wakeup_rx_complete_isr

#define _uart_wakeup_rx_complete_isr(
          uartp
)

Value:

\[
\text{osalSysLockFromISR();}
\text{osalThreadResumeI(\{(uartp)\rightarrow\text{threadrx}, MSG_OK\);}
\text{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.

7.44.3.12 _uart_wakeup_rx_error_isr

#define _uart_wakeup_rx_error_isr(
          uartp
)
Value:
{
  osalSysLockFromISR();
  osalThreadResume1(&(uartp)->threadrx, MSG_RESET);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread in case of RX error.

Parameters

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>uartp</td>
<td>pointer to the UARTDriver object</td>
<td></td>
</tr>
</tbody>
</table>

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

7.44.3.13 _uart_wakeup_rx_cm_isr

#define _uart_wakeup_rx_cm_isr(
    uartp )

Value:
{
  osalSysLockFromISR();
  osalThreadResume1(&(uartp)->threadrx, MSG_TIMEOUT);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread in case of RX character match.

Parameters

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>uartp</td>
<td>pointer to the UARTDriver object</td>
<td></td>
</tr>
</tbody>
</table>

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

7.44.3.14 _uart_wakeup_rx_timeout_isr

#define _uart_wakeup_rx_timeout_isr(
    uartp )

Value:
{
  osalSysLockFromISR();
  osalThreadResume1(&(uartp)->threadrx, MSG_TIMEOUT);
  osalSysUnlockFromISR();
}

Wakes up the waiting thread in case of RX timeout.
Parameters

in  \texttt{uartp}  pointer to the \texttt{UARTDriver} object

Function Class:

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

7.44.3.15  \texttt{_uart tx1_isr_code}

\begin{verbatim}
#define _uart_tx1_isr_code(
   uartp )
\end{verbatim}

Value:

\begin{verbatim}
{ (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);
}
\end{verbatim}

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  \texttt{uartp}  pointer to the \texttt{UARTDriver} object

Function Class:

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

7.44.3.16  \texttt{_uart tx2_isr_code}

\begin{verbatim}
#define _uart_tx2_isr_code(
   uartp )
\end{verbatim}
7.44 UART Driver

Value:
```c
if ((uartp)->config->txend2_cb != NULL) {
    (uartp)->config->txend2_cb(uartp);
    _uart_wakeup_tx2_isr(uartp);
}
```

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

```c
in  uartp  pointer to the UARTDriver object
```

Function Class:

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

7.44.3.17  _uart_rx_complete_isr_code

```c
#define _uart_rx_complete_isr_code(
    uartp )
```

Value:
```c
(uartp)->rxstate = UART_RX_COMPLETE;
if ((uartp)->config->rxend_cb != NULL) {
    (uartp)->config->rxend_cb(uartp);
}
if ((uartp)->rxstate == UART_RX_COMPLETE) {
    (uartp)->rxstate = UART_RX_IDLE;
    uart_enter_rx_idle_loop(uartp);
    _uart_wakeup_rx_complete_isr(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.

7.44.3.18 _uart_rx_error_isr_code

#define _uart_rx_error_isr_code(
    uartp,
    errors )

Value:

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.

7.44.3.19 _uart_rx_idle_code

#define _uart_rx_idle_code(
    uartp )
Value:
```c
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.

7.44.3.20 _uart_timeout_isr_code

```c
#define _uart_timeout_isr_code(
    uartp
)
```

Value:
```c
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.
Module Documentation

7.44.3.21  _uart_rx_char_match_isr_code

#define _uart_rx_char_match_isr_code(
    uartp  )

Value:

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.

7.44.3.22  PLATFORM_UART_USE_UART1

#define PLATFORM_UART_USE_UART1 FALSE

UART driver enable switch.

If set to TRUE the support for UART1 is included.
Note
   The default is FALSE.

7.44.4 Typedef Documentation

7.44.4.1 uartflags_t

typedef uint32_t uartflags_t

UART driver condition flags type.

7.44.4.2 UARTDriver

typedef struct UARTDriver UARTDriver

Type of structure representing an UART driver.

7.44.4.3 uartcb_t

typedef void(* uartcb_t) (UARTDriver *uartp)

Generic UART notification callback type.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
</table>
| uartp     | pointer to the UARTDriver object | UARTDriver *

7.44.4.4 uartccb_t

typedef void(* uartccb_t) (UARTDriver *uartp, uint16_t c)

Character received UART notification callback type.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
</table>
| uartp     | pointer to the UARTDriver object triggering the callback | UARTDriver *
| c         | received character | uint16_t |
7.44.4.5 uartcb_t

typedef void(*uartcb_t)(UARTDriver *uartp, uartflags_t e)

Receive error UART notification callback type.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>uartp</th>
<th>pointer to the UARTDriver object triggering the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e</td>
<td>receive error mask</td>
</tr>
</tbody>
</table>

7.44.5 Enumeration Type Documentation

7.44.5.1 uartstate_t

enum uartstate_t

Driver state machine possible states.

 Enumerator

| UART_UNINIT | Not initialized. |
| UART_STOP   | Stopped.         |
| UART_READY  | Ready.           |

7.44.5.2 uarttxstate_t

enum uarttxstate_t

Transmitter state machine states.

Enumerator

| UART_TX_IDLE    | Not transmitting. |
| UART_TX_ACTIVE  | Transmitting.     |
| UART_TX_COMPLETE| Buffer complete.  |
7.44.5.3  uartrxstate_t

enum uartrxstate_t

Receiver state machine states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_RX_IDLE</td>
<td>Not receiving.</td>
</tr>
<tr>
<td>UART_RX_ACTIVE</td>
<td>Receiving.</td>
</tr>
<tr>
<td>UART_RX_COMPLETE</td>
<td>Buffer complete.</td>
</tr>
</tbody>
</table>

7.44.6  Function Documentation

7.44.6.1  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]

7.44.6.2  uartObjectInit()

void uartObjectInit ( 
    UARTDriver * uarp )

Initializes the standard part of a UARTDriver structure.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>out</th>
<th>uartp</th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
</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:

```
uartObjectInit -> osalMutexObjectInit
```

7.44.6.3 `uartStart()`

```c
void uartStart (UARTDriver * uartp,
               const UARTConfig * config )
```

Configures and activates the UART peripheral.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>uartp</th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>config</td>
<td>pointer to the UARTConfig 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:

```
uartStart
  osalSysLock
osalSysUnlock
uart_lld_start
```

### 7.44.6.4 `uartStop()`

```c
void uartStop ( UARTDriver * uartp )
```

Deactivates the UART peripheral.

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>
</tbody>
</table>
Here is the call graph for this function:

```
uartStop
  ↓
osalSysLock
uartStop
  ↓
osalSysUnlock
uart_lld_stop
```

### 7.44.6.5 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>in</th>
<th>uartp</th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of data frames to send</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</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:

```
osalSysLock
uartStartSend
osalSysUnlock
uart_lld_start_send
```

### 7.44.6.6 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>uartp</td>
<td>pointer to the UARTDriver object</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of data frames 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.

Here is the call graph for this function:

```
uartStartSendI      uart_lld_start_send
```

### 7.44.6.7 uartStopSend()

```c
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.  

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:

```
uartStopSend
osalSysLock
uart_lld_stop_send
```

### 7.44.6.8 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:

![Call Graph]

### 7.44.6.9 `uartStartReceive()`

```c
void uartStartReceive (UARTDriver * usrp, 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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>uartp</code></td>
<td>pointer to the UARTDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code></td>
<td>number of data frames to receive</td>
</tr>
<tr>
<td>in</td>
<td><code>rxbuf</code></td>
<td>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:

```
osalSysLock
uartStartReceive
osalSysUnlock
uart_lld_start_receive
```

7.44.6.10  uartStartReceiveI()

```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>in</th>
<th>uartp</th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of data frames to receive</td>
</tr>
<tr>
<td>out</td>
<td>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.

Here is the call graph for this function:

```
uartStartReceiveI → uart_lld_start_receive
```

### 7.44.6.11  uartStopReceive()

```c
size_t uartStopReceive (UARTDriver * uartp )
```

Stops any ongoing receive operation.

**Note**
Stopping a receive operation also suppresses the receive callbacks.

**Parameters**

| 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:

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:

```
uartStopReceive
    \|-- osalSysLock
    \|-- uart_lld_stop_receive
```

### 7.44.6.12 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**

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

**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:

```
7.44.6.13 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>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>uartp</td>
<td>pointer to the UARTDriver object</td>
</tr>
<tr>
<td>in, out</td>
<td>np</td>
<td>number of data frames to transmit, on exit the number of frames actually transmitted</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>operation 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>MSG_OK</td>
<td>if the operation completed successfully.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation timed out.</td>
</tr>
</tbody>
</table>
7.44 UART 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:

```
osalSysLock
uartSendTimeout
osalThreadSuspendTimeoutS
uart_lld_start_send
```

### 7.44.6.14 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>Parameter</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:

```
osalSysLock
uartSendFullTimeout
osalThreadSuspendTimeoutS
uart_lld_start_send
```

7.44.6.15 `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

| MSG_OK          | if the operation completed successfully. |
| MSG_TIMEOUT     | if the operation timed out.              |
| MSG_RESET       | in case of a receive 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:

```
uartReceiveTimeout
  ↓
osalSysLock
osalThreadSuspendTimeoutS
uart_lld_start_receive
```

7.44.6.16 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 |

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:

7.44.6.17 uartReleaseBus()

```c
define_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:
7.44.6.18  uart_lld_init()

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

7.44.6.19  uart_lld_start()

void uart_lld_start (  
    UARTDriver ∗ uartp  )  

Configures and activates the UART peripheral.

Parameters
   in  uartp  pointer to the UARTDriver object

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

7.44.6.20  uart_lld_stop()

void uart_lld_stop (  
    UARTDriver ∗ uartp  )  

Deactivates the UART peripheral.
Parameters

in uartp pointer to the UARTDriver object

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

7.44.6.21 uart_lld_start_send()

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

in uartp pointer to the UARTDriver object
in n number of data frames to send
in txbuf the pointer to the transmit buffer

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

7.44.6.22 uart_lld_stop_send()

size_t uart_lld_stop_send (UARTDriver * uartp )

Stops any ongoing transmission.

Note
Stopping a transmission also suppresses the transmission callbacks.
7.44 UART Driver

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>uartp</th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
</table>

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.

7.44.6.23 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

<table>
<thead>
<tr>
<th>in</th>
<th>uartp</th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of data frames to send</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.

7.44.6.24 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

| in  | uartp | pointer to the UARTDriver object |

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.

### 7.44.7 Variable Documentation

#### 7.44.7.1 UARTD1

UARTDriver UARTD1

UART1 driver identifier.
7.45 USB Driver

Generic USB Driver.

7.45.1 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`.

7.45.2 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).

```
USB_UNINIT
usbInit()

USB_STOP
Low Power
usbStop()

USB_READY
Clock Enabled
usbStart()

SUSPEND/WAKEUP
>event_cb<

USB_RESET
>event_cb<

GET DESCRIPTOR
>get_descriptor_cb<

Endpoints Activity
>in_cb< or >out_cb<

SET_CONF(0)
>event_cb<

SET_CONF(n)
>event_cb<

USB_ACTIVE
configuration
selected

USB_SELECTED
address
assigned

SET_ADDRESS
>event_cb<

USB_RESET
>event_cb<

SUSPEND/WAKEUP
Valid EP0 Message
>requests_hook_cb<

GET DESCRIPTOR
>get_descriptor_cb<

Valid EP0 Message
>requests_hook_cb<

GET DESCRIPTOR
>get_descriptor_cb<
```

7.45.3 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.
7.45.3.1 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.

7.45.3.2 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 IN endpoints in transaction mode:
7.45 USB Driver

7.45.3.3 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, bFunctionProctol, 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**

• #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)
   
   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, 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 usblInit (void)
  USB Driver initialization.
- void usbObjectInit (USBDriver *usbp)
  Initializes the standard part of a USBDriver structure.
- void 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.
- 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)
7.45 USB Driver

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.
• void usb_lld_get_status_out (USBDriver *usbp, usbep_t ep)
  Returns the status of an OUT endpoint.
• void 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 usbepstatus_t { EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE = 2 }
  Type of an endpoint status.
• enum usbep0state_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.

7.45.4 Macro Definition Documentation

7.45.4.1 USB_DESC_INDEX

```c
#define USB_DESC_INDEX(i) ((uint8_t)(i))
```
Helper macro for index values into descriptor strings.

7.45.4.2 USB_DESC_BYTE

```c
#define USB_DESC_BYTE(b) ((uint8_t)(b))
```
Helper macro for byte values into descriptor strings.

7.45.4.3 USB_DESC_WORD

```c
#define USB_DESC_WORD(w)
```

Value:

```
(uint8_t)((w) & 255U),
(uint8_t)(((w) >> 8) & 255U)
```

Helper macro for word values into descriptor strings.
7.45.4.4 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.

7.45.4.5 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.

7.45.4.6 USB_DESC_CONFIGURATION_SIZE

#define USB_DESC_CONFIGURATION_SIZE 9U

Configuration Descriptor size.
# 7.45.4.7 USB_DESC_CONFIGURATION

```c
#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.

# 7.45.4.8 USB_DESC_INTERFACE_SIZE

```c
#define USB_DESC_INTERFACE_SIZE 9U
```

Interface Descriptor size.

# 7.45.4.9 USB_DESC_INTERFACE

```c
#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.
### 7.45.10  **USB_DESC_INTERFACE_ASSOCIATION_SIZE**

#define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U

Interface Association Descriptor size.

### 7.45.11  **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.

### 7.45.12  **USB_DESC_ENDPOINT_SIZE**

#define USB_DESC_ENDPOINT_SIZE 7U

Endpoint Descriptor size.

### 7.45.13  **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.
7.45.4.14  USB_EP_MODE_TYPE

#define USB_EP_MODE_TYPE 0x0003U
Endpoint type mask.

7.45.4.15  USB_EP_MODE_TYPE_CTRL

#define USB_EP_MODE_TYPE_CTRL 0x0000U
Control endpoint.

7.45.4.16  USB_EP_MODE_TYPE_ISOC

#define USB_EP_MODE_TYPE_ISOC 0x0001U
Isochronous endpoint.

7.45.4.17  USB_EP_MODE_TYPE_BULK

#define USB_EP_MODE_TYPE_BULK 0x0002U
Bulk endpoint.

7.45.4.18  USB_EP_MODE_TYPE_INTR

#define USB_EP_MODE_TYPE_INTR 0x0003U
Interrupt endpoint.

7.45.4.19  USB_USE_WAIT

#define USB_USE_WAIT FALSE
Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.45.4.20  usbGetDriverStateI

#define usbGetDriverStateI(
    usbp ) ((usbp)->state)
Returns the driver state.
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.

7.45.4.21  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.

7.45.4.22  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.
### 7.45.4.23 usbGetFrameNumberX

```c
#define usbGetFrameNumberX(usbp) ush_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.

### 7.45.4.24 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.
7.45.4.25  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.

7.45.4.26  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.

### 7.45.4.27 usbSetupTransfer

```c
#define usbSetupTransfer(
    usbp,
    buf,
    n,
    endcb )
```

**Value:**

```c
\{ 
    (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 <strong>USBDriver</strong> 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.

### 7.45.4.28 usbReadSetup

```c
#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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usb</td>
<td>pointer to the USBDriver object</td>
</tr>
<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.

### 7.45.4.29 _usb_isr_invoke_event_cb

```c
#define _usb_isr_invoke_event_cb(
    usbp,
    evt)
```

Value:

```
if (((usbp)->config->event_cb) != NULL) {
    (usbp)->config->event_cb(usbp, evt);
}
```

Common ISR code, usb event callback.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usb</td>
<td>pointer to the USBDriver object</td>
</tr>
<tr>
<td>in</td>
<td>evt</td>
<td>USB event code</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.45.4.30 _usb_isr_invoke_sof_cb

```c
#define _usb_isr_invoke_sof_cb(
    usbp)
```

Value:

```
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.

### 7.45.4.31 _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.

### 7.45.4.32 _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

<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.

7.45.4.33  _usb_isr_invoke_out_cb

```c
#define _usb_isr_invoke_out_cb(
    usbp,
    ep )
```

Value:

```c
{ (usbp)->receiving & ~(1 << (ep)); \
  if ((usbp)->epc[ep]->out_cb != NULL) { \
    (usbp)->epc[ep]->out_cb(usbp, ep); \
  } osalSysLockFromISR(); \
  osalThreadResumeI(&(usbp)->epc[ep]->out_state->thread, \
    usbGetReceiveTransactionSizeX(usbp, ep)); \
  osalSysUnlockFromISR(); }
```

Common ISR code, OUT endpoint event.

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.

7.45.4.34  USB_MAX_ENDPOINTS

```c
#define USB_MAX_ENDPOINTS 4
```

Maximum endpoint address.
### 7.45.4.35 USB_EP0_STATUS_STAGE

```c
#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW
```

Status stage handling method.

### 7.45.4.36 USB_SET_ADDRESS_MODE

```c
#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS
```

The address can be changed immediately upon packet reception.

### 7.45.4.37 USB_SET_ADDRESS_ACK_HANDLING

```c
#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW
```

Method for set address acknowledge.

### 7.45.4.38 PLATFORM_USB_USE_USB1

```c
#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`.

### 7.45.4.39 usb_lld_get_frame_number

```c
#define usb_lld_get_frame_number(
               usbp ) 0
```

Returns the current frame number.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usbp</code></td>
<td>pointer to the USBDriver object</td>
</tr>
</tbody>
</table>
Returns

The current frame number.

Function Class:

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

### 7.45.4.40 usb_lld_get_transaction_size

```c
#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>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>
<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.

### 7.45.4.41 usb_lld_connect_bus

```c
#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.
7.45.4.42 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.

7.45.4.43 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.

7.45.5 Typedef Documentation

7.45.5.1 USBDriver

typedef struct USBDriver USBDriver

Type of a structure representing an USB driver.

7.45.5.2 usbep_t

typedef uint8_t usbep_t

Type of an endpoint identifier.

7.45.5.3 usbcallback_t

typedef void(* usbcallback_t) (USBDriver *usbp)

Type of an USB generic notification callback.
7.45 USB Driver

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>pointer to the USBDriver object triggering the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usb</td>
<td>pointer to the USBDriver object triggering the callback</td>
</tr>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

7.45.5.4 usbepcallback_t

typedef void(* usbepcallback_t) (USBDriver *usbp, usbep_t ep)

Type of an USB endpoint callback.

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>pointer to the USBDriver object triggering the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usb</td>
<td>pointer to the USBDriver object triggering the callback</td>
</tr>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

7.45.5.5 usbeventcb_t

typedef void(* usbeventcb_t) (USBDriver *usbp, usbevent_t event)

Type of an USB event notification callback.

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>pointer to the USBDriver object triggering the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usb</td>
<td>pointer to the USBDriver object triggering the callback</td>
</tr>
<tr>
<td>in</td>
<td>event</td>
<td>event type</td>
</tr>
</tbody>
</table>

7.45.5.6 usbreqhandler_t

typedef bool(* usbreqhandler_t) (USBDriver *usbp)

Type of a requests handler callback.

The request is encoded in the usb_setup buffer.

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>pointer to the USBDriver object triggering the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usb</td>
<td>pointer to the USBDriver object triggering the callback</td>
</tr>
</tbody>
</table>
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>

7.45.6 Enumeration Type Documentation

7.45.6.1 usbstate_t

```c
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>

7.45.6.2 usbepstatus_t

```c
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>
### 7.45.6.3 `usbep0state_t`

```c
enum usbep0state_t
```

Type of an endpoint zero state machine states.

**Enumerator**

<table>
<thead>
<tr>
<th><code>USB_EP0_STP_WAITING</code></th>
<th>Waiting for SETUP data.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USB_EP0_IN_TX</code></td>
<td>Transmitting.</td>
</tr>
<tr>
<td><code>USB_EP0_IN_WAITING_TX0</code></td>
<td>Waiting transmit 0.</td>
</tr>
<tr>
<td><code>USB_EP0_IN_SENDING_STS</code></td>
<td>Sending status.</td>
</tr>
<tr>
<td><code>USB_EP0_OUT_WAITING_STS</code></td>
<td>Waiting status.</td>
</tr>
<tr>
<td><code>USB_EP0_OUT_RX</code></td>
<td>Receiving.</td>
</tr>
<tr>
<td><code>USB_EP0_ERROR</code></td>
<td>Error, EP0 stalled.</td>
</tr>
</tbody>
</table>

### 7.45.6.4 `usbevent_t`

```c
enum usbevent_t
```

Type of an enumeration of the possible USB events.

**Enumerator**

<table>
<thead>
<tr>
<th><code>USB_EVENT_RESET</code></th>
<th>Driver has been reset by host.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>USB_EVENT_ADDRESS</code></td>
<td>Address assigned.</td>
</tr>
<tr>
<td><code>USB_EVENT_CONFIGURED</code></td>
<td>Configuration selected.</td>
</tr>
<tr>
<td><code>USB_EVENT_UNCONFIGURED</code></td>
<td>Configuration removed.</td>
</tr>
<tr>
<td><code>USB_EVENT_SUSPEND</code></td>
<td>Entering suspend mode.</td>
</tr>
<tr>
<td><code>USB_EVENT_WAKEUP</code></td>
<td>Leaving suspend mode.</td>
</tr>
<tr>
<td><code>USB_EVENT_STALLED</code></td>
<td>Endpoint 0 error, stalled.</td>
</tr>
</tbody>
</table>
### 7.45.7 Function Documentation

#### 7.45.7.1 set_address()

```c
static void set_address (USBDriver ∗ usbp) [static]
```

SET ADDRESS transaction callback.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usbp</code></td>
<td>pointer to the USBDriver object</td>
</tr>
</tbody>
</table>

Here is the call graph for this function:

```
set_address → usb_lld_set_address
```

#### 7.45.7.2 default_handler()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usbp</code></td>
<td>pointer to the USBDriver object</td>
</tr>
</tbody>
</table>

**Returns**

The request handling exit code.
Return values

<table>
<thead>
<tr>
<th>false</th>
<th>Request not recognized by the handler or error.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Request handled.</td>
</tr>
</tbody>
</table>

### 7.45.7.3 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](call_graph.png)

### 7.45.7.4 usbObjectInit()

```c
void usbObjectInit (   
      USBDriver * usbp )
```

Initializes the standard part of a `USBDriver` structure.

**Parameters**

- **out** `usbp` pointer to the `USBDriver` object
Function Class:

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

### 7.45.7.5 usbStart()

```c
void 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>

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)

### 7.45.7.6 usbStop()

```c
void usbStop (  
    USBDriver * usbp )
```

Deactivates the USB peripheral.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>usbp</code></td>
<td>pointer to the <code>USBDriver</code> 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:

```
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>osalSysLock</td>
</tr>
<tr>
<td>usbStop</td>
</tr>
<tr>
<td>osalThreadResumeI</td>
</tr>
<tr>
<td>usb_lld_stop</td>
</tr>
</tbody>
</table>
```

### 7.45.7.7 `usbInitEndpointI()`

```c
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>usbp</code></td>
<td>pointer to the <code>USBDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>ep</code></td>
<td>endpoint number</td>
</tr>
<tr>
<td>in</td>
<td><code>epcp</code></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:

![Call Graph](image)

### 7.45.7.8 usbdDisableEndpointsI()

```c
void usbdDisableEndpointsI (  
    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_CONFIGURATION** 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:

![Call Graph](image)
### 7.45.7.9 `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>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>

**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](usbStartReceiveI -> usb_lld_start_out)

### 7.45.7.10 `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>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>

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](usbStartTransmitI -> usb_lld_start_in)

7.45.7.11  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>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>
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:

```
    osalSysLock
    usbReceive
    osalSysUnlock
```

7.45.7.12 usbTransmit()

```c
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
        osalSysUnlock
```

### 7.45.7.13 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
```

7.45.7.14  usbStallTransmitI()

```c
bool usbStallTransmitI (    
    USBDriver ∗ usbp,    
    usbep_t ep )
```

Stalls 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>

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:

```
usbStallTransmitI usb_lld_stall_in
```

7.45.7.15  `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 USBDriver object
```

Function Class:

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

7.45.7.16  `_usb_reset()`

```c
void _usb_reset (  
    USBDriver * usbp )
```

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.
7.45 USB Driver

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</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:

```
osalSysLockFromISR
osalThreadResume
_usb_reset
```

7.45.7.17  _usb_suspend()

```c
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>Parameter</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:

```
 publicly defined
 _usb_suspend

 osalSysLockFromISR

 osalThreadResume

 secretive

 void _usb_wakeup (USBDriver ∗ usbp )

 USB wake-up routine.

 This function must be invoked when an USB bus wake-up condition is detected.

 Parameters

 in usbp pointer to the USBDriver object

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

 7.45.7.19 _usb_ep0setup()

 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>in</th>
<th>usb</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number, always zero</td>
</tr>
</tbody>
</table>

Function Class:

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

7.45.7.20 _usb_ep0in()

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>in</th>
<th>usb</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number, always zero</td>
</tr>
</tbody>
</table>

Function Class:

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

7.45.7.21 _usb_ep0out()

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>in</th>
<th>usb</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number, always zero</td>
</tr>
</tbody>
</table>
7.45.7.22  usb_lld_init()

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:

7.45.7.23  usb_lld_start()

void usb_lld_start (  
    USBDriver * usbp  )

Configures and activates the USB peripheral.
7.45 USB Driver

Parameters

| in  | usbp         | pointer to the USBDriver object |

Function Class:

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

7.45.7.24 usb_lld_stop()

void usb_lld_stop (  
    USBDriver * usbp  
)

Deactivates the USB peripheral.

Parameters

| in  | usbp         | pointer to the USBDriver object |

Function Class:

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

7.45.7.25 usb_lld_reset()

void usb_lld_reset (  
    USBDriver * usbp  
)

USB low level reset routine.

Parameters

| in  | usbp         | pointer to the USBDriver object |
Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

![Call Graph](image)

### 7.45.7.26 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.

### 7.45.7.27 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.

7.45.7.28  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.

7.45.7.29  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.     |
7.45.7.30  **usb_lld_get_status_in()**

```c
usbepstatus_t usb_lld_get_status_in(
    USBDriver * usbp,
    usbep_t ep)
```

Returns the status of 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>

**Returns**

The endpoint status.

**Return values**

<table>
<thead>
<tr>
<th>EP_STATUS_DISABLED</th>
<th>The endpoint is not active.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP_STATUS_STALLED</td>
<td>The endpoint is stalled.</td>
</tr>
<tr>
<td>EP_STATUS_ACTIVE</td>
<td>The endpoint is active.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.45.7.31  **usb_lld_read_setup()**

```c
void usb_lld_read_setup(
    USBDriver * usbp,
    usbep_t ep,
    uint8_t * buf)
```

Reads a setup packet from the dedicated packet buffer.

**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.
7.45 USB Driver

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.

7.45.7.32 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.

7.45.7.33 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>

Function Class:

Not an API, this function is for internal use only.
7.45.7.34  **usb_lld_start_out()**

```c
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 <strong>USBDriver</strong> 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.

7.45.7.35  **usb_lld_start_in()**

```c
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 <strong>USBDriver</strong> 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.

7.45.7.36  **usb_lld_stall_out()**

```c
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 <strong>USBDriver</strong> 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.

### 7.45.7.37  usb_lld_stall_in()

```c
void usb_lld_stall_in (    
    USBDriver * usbp,    
    usbep_t ep )
```

Brings an IN endpoint in the stalled state.

**Parameters**

<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.

### 7.45.7.38  usb_lld_clear_out()

```c
void usb_lld_clear_out (    
    USBDriver * usbp,    
    usbep_t ep )
```

Brings an OUT endpoint in the active state.

**Parameters**

<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.

### 7.45.7.39  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>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.

### 7.45.8 Variable Documentation

#### 7.45.8.1 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.

#### 7.45.8.2 USBD1

USBDriver USBD1

USB1 driver identifier.

#### 7.45.8.3 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.

#### 7.45.8.4 in

```c
USBInEndpointState { ... } in
```

IN EP0 state.
7.45 USB Driver

7.45.8.5 out

USBOutEndpointState { ... } out

OUT EP0 state.

7.45.8.6 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.
7.46 USB CDC Header

USB CDC Support Header.

7.46.1 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_PARM 0x30U
- define CDC_GET_RINGER_PARM 0x31U
- define CDC_SET_OPERATION_PARM 0x32U
- define CDC_GET_OPERATION_PARM 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.
7.47 WDG Driver

Generic WDG Driver.

7.47.1 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 wdgResetI(wp) wdg_lld_reset(wp)`

  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.

- `void wdgStart (WDGDriver *wp, const WDGConfig *config)`

  Configures and activates the WDG peripheral.

- `void wdgStop (WDGDriver *wp)`

  Deactivates the WDG peripheral.

- `void wdgReset (WDGDriver *wp)`

  Resets WDG's counter.

- `void wdg_lld_init (void)`

  Low level WDG driver initialization.

- `void wdg_lld_start (WDGDriver *wp)`

  Configures and activates the WDG peripheral.

- `void wdg_lld_stop (WDGDriver *wp)`

  Deactivates the WDG peripheral.

- `void wdg_lld_reset (WDGDriver *wp)`

  Reloads WDG's counter.
Enumerations

- enum wdgstate_t { WDG_UNINIT = 0, WDG_STOP = 1, WDG_READY = 2 }

Driver state machine possible states.

7.47.2 Macro Definition Documentation

7.47.2.1 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.

7.47.2.2 PLATFORM_WDG_USE_WDG1

#define PLATFORM_WDG_USE_WDG1 FALSE

WDG1 driver enable switch.

Note

The default is FALSE.

7.47.3 Typedef Documentation

7.47.3.1 WDGDriver

typedef struct WDGDriver WDGDriver

Type of a structure representing an WDG driver.
7.47.4 Enumeration Type Documentation

7.47.4.1 wdgstate_t

def wdgstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDG_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>WDG_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>WDG_READY</td>
<td>Ready.</td>
</tr>
</tbody>
</table>

7.47.5 Function Documentation

7.47.5.1 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:

[Call graph image]
7.47.5.2  wdgStart()

```c
void wdgStart (  
    WDGDriver * wdgp,  
    const WDGConfig * config )
```

Configures and activates the WDG peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>wdgp</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>config</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)

7.47.5.3  wdgStop()

```c
void wdgStop (  
    WDGDriver * wdgp )
```

Deactivates the WDG peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>wdgp</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
```

### 7.47.5.4 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
```
7.47.5.5  wdg_lld_init()

void wdg_lld_init (  
    void )  

Low level WDG driver initialization.

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

7.47.5.6  wdg_lld_start()

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.

7.47.5.7  wdg_lld_stop()

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.
7.47.5.8  wdg_lld_reset()

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.
7.48 WSPI Driver

Generic WSPI Driver.

7.48.1 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`
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.
- `void 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_t *rxbuf)
    Sends a command then receives data over the WSPI bus.
• void wspiMapFlash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t_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_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_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_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.
7.48.2.1 WSPI_USE_WAIT

#define WSPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.48.2.2 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.

7.48.2.3 `wspiStartCommandI`

#define wspiStartCommandI(

    wspip,
    cmdp )

Value:

Value:  

    {  
        osalDbgAssert((cmdp)->cfg & WSPI_CFG_DATA_MODE_MASK) ==  
        WSPI_CFG_DATA_MODE_NONE,  
        "data mode specified"};  
    (wspip)->state = WSPI_SEND;  
    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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
</tr>
<tr>
<td>in</td>
<td>cmdp</td>
</tr>
</tbody>
</table>
7.48.2.4  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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
</tr>
<tr>
<td>in</td>
<td>cmdp</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</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.

7.48.2.5  wspiStartReceiveI

#define wspiStartReceiveI(
    wspip,
    cmdp,
    n,
    rxbuf )
Value:
{
    osalDbgAssert(((cmdp)->cfg & WSPI_CFG_DATA_MODE_MASK) !=
        WSPI_CFG_DATA_MODE_NONE,
        "data mode required");
    (wspip)->state = WSPI_RECEIVE;
    wspi_lld_receive(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

<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>in</td>
<td>n</td>
<td>number of bytes to receive or zero if no data phase</td>
</tr>
<tr>
<td>out</td>
<td>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.

7.48.2.6 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

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

7.48.2.7 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.

7.48.2.8 _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               |
Function Class:

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

7.48.2.9 _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.

7.48.2.10 _wspi_error_code

#define _wspi_error_code(
    wspip
)

Value:

ChibiOS/HAL
if ((wspip)->config->error_cb) {
    (wspip)->state = WSPI_COMPLETE;
    if ((wspip)->state == WSPI_COMPLETE) {
        (wspip)->state = WSPI_READY;
    } 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.

7.48.2.11 PLATFORM_WSPI_USE_WSPI1

#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.

7.48.2.12 wspi_lld_driver_fields

#define wspi_lld_driver_fields

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

Low level fields of the WSPI driver structure.
7.48.2.13  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.

7.48.3  Typedef Documentation

7.48.3.1  WSPIDriver

typedef struct hal_wspi_driver WSPIDriver

Type of a structure representing an WSPI driver.

7.48.3.2  WSPIConfig

typedef struct hal_wspi_config WSPIConfig

Type of a structure representing an WSPI driver configuration.

7.48.3.3  wspicallback_t

typedef void(* wspicallback_t) (WSPIDriver *wspip)

Type of a WSPI notification callback.

Parameters

| in  | wspip | pointer to the WSPIDriver object triggering the callback |

7.48.4  Enumeration Type Documentation

7.48.4.1  wspistate_t

enum wspistate_t

ChibiOS/HAL
Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSPI_UNINIT</td>
<td>Not initialized.</td>
</tr>
<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>

7.48.5 Function Documentation

7.48.5.1 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:

```
``
7.48.5.2  wspiObjectInit()

void wspiObjectInit (  
    WSPIDriver * wspin )

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:

[Diagram]

7.48.5.3  wspiStart()

```c
void wspiStart (  
    WSPIDriver * wspip,
    const WSPICConfig * config  
)
```

Configures and activates the WSPI peripheral.

Parameters

| in   | wspip | pointer to the WSPIDriver object |
| in   | config | pointer to the WSPICConfig 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:

7.48.5.4 wspiStop()

```c
void wspiStop ( WSPIDriver * wspip )
```

Deactivates the WSPI peripheral.

**Note**
Deactivating the peripheral also enforces a release of the slave select line.

**Parameters**

<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:

![Call Graph Diagram]

7.48.5.5  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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>wspip</code> pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>cmdp</code> 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
```

7.48.5.6  wsplStartSend()

```c
void wsplStartSend ( 
    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>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>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:

```
wspiStartSend
    |       |
    |       |
osalSysLock  osalSysUnlock
```

### 7.48.5.7 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><code>wspip</code> pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>cmdp</code> pointer to the command descriptor</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code> number of bytes to send</td>
</tr>
<tr>
<td>out</td>
<td><code>rxbuf</code> 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:

![Call Graph Diagram]

7.48.5.8 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>wspip</code></td>
<td>pointer to the <code>WSPIDriver</code> object</td>
</tr>
<tr>
<td>in <code>cmdp</code></td>
<td>pointer to the command descriptor</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>false</code></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><code>true</code></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.

7.48.5.9  wspiSend()

```c
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>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>
<tr>
<td><code>n</code></td>
<td>number of bytes to send</td>
</tr>
<tr>
<td><code>txbuf</code></td>
<td>the pointer to the transmit buffer</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>false</code></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><code>true</code></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.

7.48.5.10  wspiReceive()

```c
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 == NULL`).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>wspip</th>
<th>pointer to the <code>WSPIDriver</code> 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>nbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

- `false` if the operation succeeded.
- `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.

### 7.48.5.11  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>in</th>
<th>wspip</th>
<th>pointer to the <code>WSPIDriver</code> 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 <code>NULL</code></td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 7.48.5.12 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**

| 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:

```
    wspiUnmapFlash
        |__________osalSysLock__________|
        |                                |
        v                                |
    osalSysUnlock
```

### 7.48.5.13 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:

```
| wspiAcquireBus | osalMutexLock |
```

### 7.48.5.14 `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:

```
| wspiReleaseBus | osalMutexUnlock |
```
### 7.48.5.15 wspi_lld_init()

```c
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:

```
| wspi_lld_init | wspiObjectInit | osalMutexObjectInit |
```

### 7.48.5.16 wspi_lld_start()

```c
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.

### 7.48.5.17 wspi_lld_stop()

```c
void wspi_lld_stop ( WSPIDriver * wspip )
```

Deactivates the WSPI peripheral.
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>
</tbody>
</table>

Function Class:

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

### 7.48.5.18 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

<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:

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

### 7.48.5.19 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.
Parameters

| in  | wspip | pointer to the WSPIDriver object |
| in  | cmdp  | pointer to the command descriptor |
| in  | n     | number of bytes to send |
| in  | txbuf | the pointer to the transmit buffer |

Function Class:

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

7.48.5.20  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

| in  | wspip | pointer to the WSPIDriver object |
| in  | cmdp  | pointer to the command descriptor |
| in  | n     | number of bytes to send |
| out | rxbuf | the pointer to the receive buffer |

Function Class:

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

7.48.5.21  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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>wspip</strong></td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>cmdp</strong></td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>addrp</strong></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.

### 7.48.5.22 `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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>wspip</strong></td>
<td>pointer to the WSPIDriver object</td>
</tr>
</tbody>
</table>

### Function Class:

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

### 7.48.6 Variable Documentation

#### 7.48.6.1 WSPID1

```
WSPIDriver WSPID1
```

WSPID1 driver identifier.
7.49 HAL

Hardware Abstraction Layer.

7.49.1 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.

7.49.2 HAL Device Drivers Architecture

The HAL contains several kind of modules:

- Normal Device Drivers
- Complex Device Drivers
- Interfaces
- Inner Code

7.49.3 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.
7.49.3.1 Diagram

7.49.4 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.

7.49.5 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.

7.49.6 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.
7.50 Configuration

HAL Configuration.

7.50.1 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 described 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`
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_NICE_WAITING TRUE`
  Delays insertions.

- `#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.
7.50 Configuration

- 
  #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_CIRCULAR FALSE
  Enables circular transfers APIs.
- 
  #define spi_USE_MUTUAL EXCLUSION TRUE
  Enables the spiAcquireBus() and spiReleaseBus() APIs.
- 
  #define spiSELECT_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.

7.50.2 Macro Definition Documentation

7.50.2.1 HAL_USE_PAL

#define HAL_USE_PAL TRUE

Enables the PAL subsystem.

7.50.2.2 HAL_USE_ADC

#define HAL_USE_ADC TRUE

Enables the ADC subsystem.
7.50.2.3 HAL_USE_CAN

#define HAL_USE_CAN TRUE

Enables the CAN subsystem.

7.50.2.4 HAL_USE_CRY

#define HAL_USE_CRY TRUE

Enables the cryptographic subsystem.

7.50.2.5 HAL_USE_DAC

#define HAL_USE_DAC TRUE

Enables the DAC subsystem.

7.50.2.6 HAL_USE_EFL

#define HAL_USE_EFL TRUE

Enables the EFlash subsystem.

7.50.2.7 HAL_USE_GPT

#define HAL_USE_GPT TRUE

Enables the GPT subsystem.

7.50.2.8 HAL_USE_I2C

#define HAL_USE_I2C TRUE

Enables the I2C subsystem.
7.50.2.9  HAL_USE_I2S

#define HAL_USE_I2S TRUE
Enables the I2S subsystem.

7.50.2.10  HAL_USE_ICU

#define HAL_USE_ICU TRUE
Enables the ICU subsystem.

7.50.2.11  HAL_USE_MAC

#define HAL_USE_MAC TRUE
Enables the MAC subsystem.

7.50.2.12  HAL_USE_MMC_SPI

#define HAL_USE_MMC_SPI TRUE
Enables the MMC_SPI subsystem.

7.50.2.13  HAL_USE_PWM

#define HAL_USE_PWM TRUE
Enables the PWM subsystem.

7.50.2.14  HAL_USE_RTC

#define HAL_USE_RTC TRUE
Enables the RTC subsystem.
7.50.2.15 HAL_USE_SDC

#define HAL_USE_SDC TRUE

Enables the SDC subsystem.

7.50.2.16 HAL_USE_SERIAL

#define HAL_USE_SERIAL TRUE

Enables the SERIAL subsystem.

7.50.2.17 HAL_USE_SERIAL_USB

#define HAL_USE_SERIAL_USB TRUE

Enables the SERIAL over USB subsystem.

7.50.2.18 HAL_USE_SIO

#define HAL_USE_SIO TRUE

Enables the SIO subsystem.

7.50.2.19 HAL_USE_SPI

#define HAL_USE_SPI TRUE

Enables the SPI subsystem.

7.50.2.20 HAL_USE_TRNG

#define HAL_USE_TRNG TRUE

Enables the TRNG subsystem.
7.50.2.21 HAL_USE_UART

#define HAL_USE_UART TRUE

Enables the UART subsystem.

7.50.2.22 HAL_USE_USB

#define HAL_USE_USB TRUE

Enables the USB subsystem.

7.50.2.23 HAL_USE_WDG

#define HAL_USE_WDG TRUE

Enables the WDG subsystem.

7.50.2.24 HAL_USE_WSPI

#define HAL_USE_WSPI TRUE

Enables the WSPI subsystem.

7.50.2.25 PAL_USE_CALLBACKS

#define PAL_USE_CALLBACKS FALSE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.
7.50.2.26 PAL_USE_WAIT

#define PAL_USE_WAIT FALSE

Enables synchronous APIs.

Note
   Disabling this option saves both code and data space.

7.50.2.27 ADC_USE_WAIT

#define ADC_USE_WAIT TRUE

Enables synchronous APIs.

Note
   Disabling this option saves both code and data space.

7.50.2.28 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.

7.50.2.29 CAN_USE_SLEEP_MODE

#define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

7.50.2.30 CAN_ENFORCE_USE_CALLBACKS

#define CAN_ENFORCE_USE_CALLBACKS FALSE

Enforces the driver to use direct callbacks rather than OSAL events.
7.50.2.31 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.

7.50.2.32 HAL_CRY_ENFORCE_FALLBACK

#define HAL_CRY_ENFORCE_FALLBACK FALSE

Makes the driver forcibly use the fall-back implementations.

7.50.2.33 DAC_USE_WAIT

#define DAC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.50.2.34 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.
7.50.2.35  I2C_USE_MUTUAL_EXCLUSION

#define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.50.2.36  MAC_USE_ZERO_COPY

#define MAC_USE_ZERO_COPY TRUE

Enables the zero-copy API.

7.50.2.37  MAC_USE_EVENTS

#define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.50.2.38  MMC_NICE_WAITING

#define MMC_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. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

7.50.2.39  SDC_INIT_RETRY

#define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.
7.50.2.40 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.

7.50.2.41 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.

7.50.2.42 SDC_INIT_OCR_V20

#define SDC_INIT_OCR_V20 0x50FF8000U

OCR initialization constant for V20 cards.

7.50.2.43 SDC_INIT_OCR

#define SDC_INIT_OCR 0x80100000U

OCR initialization constant for non-V20 cards.

7.50.2.44 SERIAL_DEFAULT_BITRATE

#define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.
7.50.2.45 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.

7.50.2.46 SIO_DEFAULT_BITRATE

#define SIO_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.50.2.47 SIO_USE_SYNCHRONIZATION

#define SIO_USE_SYNCHRONIZATION TRUE

Support for thread synchronization API.

7.50.2.48 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.
7.50.2.49  SERIAL_USB_BUFFERS_NUMBER

#define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.50.2.50  SPI_USE_WAIT

#define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.50.2.51  SPI_USE_CIRCULAR

#define SPI_USE_CIRCULAR FALSE

Enables circular transfers APIs.

Note

Disabling this option saves both code and data space.

7.50.2.52  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.
7.50.2.53 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.

7.50.2.54 UART_USE_WAIT

#define UART_USE_WAIT TRUE

Enables synchronous APIs.

Note
   Disabling this option saves both code and data space.

7.50.2.55 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.

7.50.2.56 USB_USE_WAIT

#define USB_USE_WAIT TRUE

Enables synchronous APIs.

Note
   Disabling this option saves both code and data space.
7.50.2.57  WSPI_USE_WAIT

#define WSPI_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.50.2.58  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.
7.51 Normal Drivers

HAL Normal Drivers.

7.51.1 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
  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.
7.52 Complex Drivers

HAL Complex Drivers.

7.52.1 Detailed Description

HAL Complex Drivers.

Modules

- 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.
7.53 Interfaces and Classes

HAL Interfaces and Classes.

7.53.1 Detailed Description

HAL Interfaces and Classes.

Modules

- Blocks
- Streams
- Base Object
7.54 Inner Code

HAL Inner Code.

7.54.1 Detailed Description

HAL Inner Code.

Modules

- I/O Buffers Queues
- MMC/SD Block Device
- I/O Bytes Queues
7.55  Support Code

HAL Support Code.

7.55.1  Detailed Description

HAL Support Code.

Modules

- MII/RMII Header
  MII/RMII Support Header.
- USB CDC Header
  USB CDC Support Header.
7.56 OSAL

Operating System Abstraction Layer.

7.56.1 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) ∗ (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 systemctl_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 (*)(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 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 osalTimelsInRangeX (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_halt_msg
  Pointer to a halt error message.

- const char *osal_halt_msg
  Pointer to a halt error message.

7.56.2 Macro Definition Documentation

7.56.2.1 OSAL_ST_RESOLUTION

#define OSAL_ST_RESOLUTION 32

Size in bits of the systick_t type.

7.56.2.2 OSAL_ST_FREQUENCY

#define OSAL_ST_FREQUENCY 1000

Required systick frequency or resolution.
7.56.2.3 OSAL_ST_MODE

#define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC
Systick mode required by the underlying OS.

7.56.2.4 OSAL_IRQ_PRIORITY_LEVELS

#define OSAL_IRQ_PRIORITY_LEVELS 16U
Total priority levels.
Implementation not mandatory.

7.56.2.5 OSAL_IRQ_MAXIMUM_PRIORITY

#define OSAL_IRQ_MAXIMUM_PRIORITY 0U
Highest IRQ priority for HAL drivers.
Implementation not mandatory.

7.56.2.6 OSAL_DBG_ENABLE_ASSERTS

#define OSAL_DBG_ENABLE_ASSERTS FALSE
Enables OSAL assertions.

7.56.2.7 OSAL_DBG_ENABLE_CHECKS

#define OSAL_DBG_ENABLE_CHECKS FALSE
Enables OSAL functions parameters checks.

7.56.2.8 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.
7.56 OSAL

Parameters

| in  | c               | the condition to be verified to be true |
| in  | remark          | a remark string                           |

Function Class:

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

7.56.2.9 osalDbgCheck

#define osalDbgCheck(
    c
)

Value:

do {
    /*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.

7.56.2.10 osalDbgCheckClassI

#define osalDbgCheckClassI( )

I-Class state check.

Note

Implementation is optional.
7.56.2.11 osalDbgCheckClassS

#define osalDbgCheckClassS( )

S-Class state check.

Note
   Implementation is optional.

7.56.2.12 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.

7.56.2.13 OSAL_IRQ_PROLOGUE

#define OSAL_IRQ_PROLOGUE( )

IRQ prologue code.
This macro must be inserted at the start of all IRQ handlers.

7.56.2.14 OSAL_IRQ_EPILOGUE

#define OSAL_IRQ_EPILOGUE( )

IRQ epilogue code.
This macro must be inserted at the end of all IRQ handlers.

7.56.2.15 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
7.56.2.16 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.

7.56.2.17 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

\textbf{in} \texttt{msecs} \hspace{1em} 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.

7.56.2.18 \hspace{1em} \textsc{OSAL\_US2I}

\#define \textsc{OSAL\_US2I}(  
\quad \text{\texttt{usecs}} )

\textbf{Value:}

\begin{align*}
\langle\text{sysinterval\_t} \rangle((\langle\text{time\_conv\_t}\rangle(\text{\texttt{usecs}}) \times \\
(\langle\text{time\_conv\_t}\rangle\text{\texttt{OSAL\_ST\_FREQUENCY}}) + \\
(\langle\text{time\_conv\_t}\rangle999999) / (\langle\text{time\_conv\_t}\rangle1000000))
\end{align*}

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

\textbf{in} \texttt{usecs} \hspace{1em} 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.
7.56 OSAL

7.56.2.19 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.

7.56.2.20 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.

7.56.2.21 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.
7.56.2.22 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>in</th>
<th>freq</th>
<th>clock frequency, in Hz, of the realtime counter</th>
</tr>
</thead>
<tbody>
<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.

7.56.2.23 OSAL_MS2RTC

#define OSAL_MS2RTC(
    freq,
    msec ) (rtcnt_t)(((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>in</th>
<th>freq</th>
<th>clock frequency, in Hz, of the realtime counter</th>
</tr>
</thead>
<tbody>
<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.

7.56.2.24 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 \( \text{freq} \geq 1000000 \).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>freq</th>
<th>clock frequency, in Hz, of the realtime counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usec</td>
<td>number of microseconds</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.

7.56.2.25 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.

7.56.2.26 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.

7.56.2.27 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.

7.56.3 Typedef Documentation

7.56.3.1 syssts_t

typedef uint32_t syssts_t
Type of a system status word.

7.56.3.2 msg_t

typedef int32_t msg_t
Type of a message.

7.56.3.3 systime_t

typedef uint32_t systime_t
Type of system time counter.

7.56.3.4 sysinterval_t

typedef uint32_t sysinterval_t
Type of system time interval.

7.56.3.5 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.
7.56.3.6 rtcnt_t

typedef uint32_t rtcnt_t

Type of realtime counter.

7.56.3.7 thread_reference_t

typedef void* thread_reference_t

Type of a thread reference.

7.56.3.8 eventflags_t

typedef uint32_t eventflags_t

Type of an event flags mask.

7.56.3.9 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.

7.56.3.10 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.
7.56.3.11 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 them mechanism can be simulated.

7.56.4 Function Documentation

7.56.4.1 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.

7.56.4.2 osalSysHalt()

void osalSysHalt (  
    const char * reason  
)

System halt with error message.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>reason</th>
<th>the halt message pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></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:

```
osalSysHalt  osalSysDisable
```

### 7.56.4.3 osalSysPolledDelayX()

```c
void osalSysPolledDelayX (  
    rtcnt_t cycles )
```

Polled delay.

**Note**

The real delay is always few cycles in excess of the specified value.

**Parameters**

| in  | cycles | number of cycles |

Function Class:

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

### 7.56.4.4 osalOsTimerHandlerI()

```c
void osalOsTimerHandlerI (  
    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.
7.56.4.5  osalOsRescheduleS()

```c
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.

7.56.4.6  osalOsGetSystemTimeX()

```c
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.

7.56.4.7  osalThreadSleepS()

```c
void osalThreadSleepS ( 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>• <strong>TIME_INFINITE</strong> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_IMMEDIATE</strong> this value is not allowed.</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.

### 7.56.4.8 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>• <strong>TIME_INFINITE</strong> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_IMMEDIATE</strong> 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.

### 7.56.4.9 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.

### 7.56.4.10 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>in</th>
<th>trp</th>
<th>a pointer to a thread reference 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>• <strong>TIME_INFINITE</strong> the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_IMMEDIATE</strong> the thread is not enqueued and the function returns</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>MSG_TIMEOUT</strong> as if a timeout occurred.</td>
</tr>
</tbody>
</table>

Returns

The wake up message.

**Return values**

| MSG_TIMEOUT | if the operation timed out. |

Function Class:

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

### 7.56.4.11 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.

7.56.4.12 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.

7.56.4.13 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

| IN | tqp     | Pointer to the threads queue object |
| IN | timeout | The timeout in system ticks, the special values are handled as follow: |
|    |         | • **TIME_INFINITE** the thread enters an infinite sleep state. |
|    |         | • **TIME_IMMEDIATE** the thread is not enqueued and the function returns MSG_TIMEOUT as if a timeout occurred. |

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.

7.56.4.14 osalThreadDequeueNextI()

```c
void osalThreadDequeueNextI ( 
    threads_queue_t * tqp, 
    msg_t msg )
```

Dequeues and wakes up one thread from the queue, if any.

Parameters

| IN | tqp | Pointer to the threads queue object |
| IN | msg | The message code |

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.

7.56.4.15 osalThreadDequeueAllI()

```c
void osalThreadDequeueAllI ( 
    threads_queue_t * tqp, 
    msg_t msg )
```
Dequeues and wakes up all threads from the queue.
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.

### 7.56.4.16 osalEventBroadcastFlagsI()

```c
void osalEventBroadcastFlagsI ( 
    event_source_t * esp, 
    eventflags_t flags )
```

Add flags to an event source object.

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>flags</td>
<td>flags to be ORed to the flags mask</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.

### 7.56.4.17 osalEventBroadcastFlags()

```c
void osalEventBroadcastFlags ( 
    event_source_t * esp, 
    eventflags_t flags )
```

Add flags to an event source object.

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>flags</td>
<td>flags to be ORed to the flags mask</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](image)

### 7.56.4.18 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.
7.56.4.19  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.

7.56.4.20  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.

7.56.4.21  osalSysDisable()

```c
static void osalSysDisable ( void ) [inline], [static]
```

Disables interrupts globally.
Function Class:
Special function, this function has special requirements see the notes.

7.56.4.22 osalSysEnable()

static void osalSysEnable ( void ) [inline], [static]

Enables interrupts globally.

Function Class:
Special function, this function has special requirements see the notes.

7.56.4.23 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.

7.56.4.24 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.
7.56.4.25 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.

7.56.4.26 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.

7.56.4.27 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.
7.56.4.28 osalSysRestoreStatusX()

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.

7.56.4.29 osalTimeAddX()

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.

7.56.4.30 osalTimeDiffX()

static sysinterval_t osalTimeDiffX (  
    systime_t start,  
    systime_t end ) [inline], [static]

Subtracts two system times returning an interval.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>start</th>
<th>first system time</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>end</td>
<td>second system time</td>
</tr>
</tbody>
</table>

Returns

The interval representing the time difference.

Function Class:

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

7.56.4.31 osalTimeIsInRangeX()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the time to be verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>start</td>
<td>the start of the time window (inclusive)</td>
</tr>
<tr>
<td>in</td>
<td>end</td>
<td>the end of the time window (non inclusive)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>true</th>
<th>current time within the specified time window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>current time not within the specified time window.</td>
</tr>
</tbody>
</table>

Function Class:

This is an **X-Class API**, this function can be invoked from any context.
7.56.4.32  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.

7.56.4.33  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.

7.56.4.34  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.
7.56.5 Variable Documentation

7.56.5.1 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.

7.56.5.2 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.
Chapter 8

Data Structure Documentation

8.1 BaseAsynchronousChannel Struct Reference

Base asynchronous channel class.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseAsynchronousChannel:
Collaboration diagram for BaseAsynchronousChannel:

![Collaboration Diagram]

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

The `BaseAsynchronousChannelVMT` virtual methods table.

```cpp
#include <hal_channels.h>
```

Inheritance 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:
Data Structure Documentation

Collaboration diagram for BaseBlockDevice:

```
BaseObjectVMT

+ vmt

BaseObject

+ vmt

BaseBlockDeviceVMT

+ vmt

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:
Collaboration diagram for 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.
BaseChannelVMT Struct Reference

BaseChannel virtual methods table.

#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:

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 stream class.

#include <hal_objects.h>
Inheritance diagram for BaseObject:

Collaboration diagram for BaseObject:

**Data Fields**

- const struct `BaseObjectVMT * vmt`
  
  *Virtual Methods Table.*

**8.11.1 Detailed Description**

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

**8.11.2 Field Documentation**
8.12 BaseObjectVMT Struct Reference

8.12.1 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:

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:
Collaboration diagram for BasePersistentStorage:

```
BaseObjectVMT

+ vmt

BaseObject

BasePersistentStorageVMT

+ vmt

BasePersistentStorage
```

### 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:

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:

![Collaboration Diagram]

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:

Collaboration diagram for BaseSequentialStreamVMT:

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

```c
uint32_t BlockDeviceInfo::blk_num
```

Total number of blocks.

8.18 BufferedStreamAdapter Struct Reference

Buffered stream adapter object.

```c
#include <bufstreams.h>
```

Inherits BufferedStream.

Collaboration diagram for BufferedStreamAdapter:

![Collaboration diagram](image)

Data Fields

- const struct BufferedStreamAdapterVMT * vmt
  
  *Virtual Methods Table.*

8.18.1 Detailed Description

Buffered stream adapter object.

8.18.2 Field Documentation
8.18.2.1 vmt

const struct BufferedStreamAdapterVMT* BufferedStreamAdapter::vmt

Virtual Methods Table.

8.19 BufferedStreamAdapterVMT Struct Reference

BufferedStreamAdapter virtual methods table.

#include <bufstreams.h>

Collaboration diagram for BufferedStreamAdapterVMT:

8.19.1 Detailed Description

BufferedStreamAdapter virtual methods table.

8.20 CANConfig Struct Reference

Driver configuration structure.

#include <hal_can_lld.h>

Collaboration diagram for CANConfig:
8.20.1 Detailed Description

Driver configuration structure.

8.21 CANDriver Struct Reference

Structure representing an CAN driver.

```c
#include <hal_can_lld.h>
```

Collaboration diagram for CANDriver:

![Collaboration diagram for CANDriver]

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 rxfull_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.
8.21 CANDriver Struct Reference

8.21.1 Detailed Description

Structure representing an CAN driver.

8.21.2 Field Documentation

8.21.2.1 state

```
canstate_t CANDriver::state
```

Driver state.

8.21.2.2 config

```
const CANConfig* CANDriver::config
```

Current configuration data.

8.21.2.3 txqueue

```
threads_queue_t CANDriver::txqueue
```

Transmission threads queue.

8.21.2.4 rxqueue

```
threads_queue_t CANDriver::rxqueue
```

Receive threads queue.
8.21.2.5  rxfull_event

event_source_t CANDriver::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.

8.21.2.6  txempty_event

event_source_t CANDriver::txempty_event

One or more transmission mailbox become available.

Note
The flags associated to the listeners will indicate which transmit mailboxes become empty.

8.21.2.7  error_event

event_source_t CANDriver::error_event

A CAN bus error happened.

Note
The flags associated to the listeners will indicate the error(s) that have occurred.

8.21.2.8  sleep_event

event_source_t CANDriver::sleep_event

Entering sleep state event.
8.21.2.9  wakeup_event

`event_source_t CANDriver::wakeup_event`

Exiting sleep state event.

8.22  CANRxFrame Struct Reference

CAN received frame.

```c
#include <hal_can_lld.h>
```

Collaboration diagram for CANRxFrame:

```
CANRxFrame
+ FMI
+ TIME
+ DLC
+ RTR
+ IDE
+ SID
+ EID
+ _align1
+ @8
+ data8
+ data16
+ data32
+ @10
```

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.
8.22.1 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.

8.22.2 Field Documentation

8.22.2.1 FMI

uint8_t CANRxFrame::FMI

Filter id.

8.22.2.2 TIME

uint16_t CANRxFrame::TIME

Time stamp.
8.22.2.3 DLC

```
8.22.2.3 DLC
```

uint8_t CANRxFrame::DLC

Data length.

8.22.2.4 RTR

```
8.22.2.4 RTR
```

uint8_t CANRxFrame::RTR

Frame type.

8.22.2.5 IDE

```
8.22.2.5 IDE
```

uint8_t CANRxFrame::IDE

Identifier type.

8.22.2.6 SID

```
8.22.2.6 SID
```

uint32_t CANRxFrame::SID

Standard identifier.

8.22.2.7 EID

```
8.22.2.7 EID
```

uint32_t CANRxFrame::EID

Extended identifier.

8.22.2.8 data8

```
8.22.2.8 data8
```

uint8_t CANRxFrame::data8[8]

Frame data.
8.22.2.9 data16

uint16_t CANRxFrame::data16[4]

Frame data.

8.22.2.10 data32

uint32_t CANRxFrame::data32[2]

Frame data.

8.23 CANTxFrame Struct Reference

CAN transmission frame.

#include <hal_can_lld.h>

Collaboration diagram for CANTxFrame:
Data Fields

- uint8_t DLC \( \text{4} \)
  
  Data length.

- uint8_t RTR \( \text{1} \)

  Frame type.

- uint8_t IDE \( \text{1} \)

  Identifier type.

- uint32_t SID \( \text{11} \)

  Standard identifier.

- uint32_t EID \( \text{29} \)

  Extended identifier.

- uint8_t data8 \( [8] \)

  Frame data.

- uint16_t data16 \( [4] \)

  Frame data.

- uint32_t data32 \( [2] \)

  Frame data.

8.23.1 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.

8.23.2 Field Documentation

8.23.2.1 DLC

uint8_t CANTxFrame::DLC

Data length.
8.23.2.2  RTR

uint8_t CANTxFrame::RTR

Frame type.

8.23.2.3  IDE

uint8_t CANTxFrame::IDE

Identifier type.

8.23.2.4  SID

uint32_t CANTxFrame::SID

Standard identifier.

8.23.2.5  EID

uint32_t CANTxFrame::EID

Extended identifier.

8.23.2.6  data8

uint8_t CANTxFrame::data8[8]

Frame data.

8.23.2.7  data16

uint16_t CANTxFrame::data16[4]

Frame data.
8.23.2.8 data32

```c
uint32_t CANTxFrame::data32[2]
```

Frame data.

8.24 cdc_linecoding_t Struct Reference

Type of Line Coding structure.

```c
#include <hal_usb_cdc.h>
```

Collaboration diagram for cdc_linecoding_t:

```
cdc_linecoding_t
+ dwDTERate
+ bCharFormat
+ bParityType
+ bDataBits
```

8.24.1 Detailed Description

Type of Line Coding structure.

8.25 CRYConfig Struct Reference

Driver configuration structure.

```c
#include <hal_crypto_lld.h>
```

Collaboration diagram for CRYConfig:

```
CRYConfig
+ dummy
```

ChibiOS/HAL
8.25.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.26 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.

8.26.1 Detailed Description

Structure representing an CRY driver.

8.26.2 Field Documentation
8.26.2.1 state

```
crystate_t CRYDriver::state
```

Driver state.

8.26.2.2 config

```
const CRYConfig* CRYDriver::config
```

Current configuration data.

8.27 EFlashDriverVMT Struct Reference

EFlash virtual methods table.

```
#include <hal_efl.h>
```

Inheritance diagram for EFlashDriverVMT:
Collaboration diagram for EFlashDriverVMT:

8.27.1 Detailed Description

_EFlash_ virtual methods table.

8.28 event_source Struct Reference

Events source object.

#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.

8.28.1 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.

8.28.2 Field Documentation

8.28.2.1 flags

volatile eventflags_t event_source::flags

Stored event flags.

8.28.2.2 cb

eventcallback_t event_source::cb

Event source callback.

8.28.2.3 param

void* event_source::param

User defined field.
Base file stream class.

```c
#include <hal_files.h>
```

Inheritance diagram for FileStream:
Data Fields

- const struct FileStreamVMT * vmt

Virtual Methods Table.

8.29.1 Detailed Description

Base file stream class.

This class represents a generic file data stream.

8.29.2 Field Documentation

8.29.2.1 vmt

const struct FileStreamVMT* FileStream::vmt

Virtual Methods Table.
8.30 FileStreamVMT Struct Reference

FileStream virtual methods table.

#include <hal_files.h>

Inheritance diagram for FileStreamVMT:

![Inheritance Diagram]

Collaboration diagram for FileStreamVMT:

![Collaboration Diagram]

8.30.1 Detailed Description

FileStream virtual methods table.
8.31 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.

8.31.1 Detailed Description

Type of a flash device descriptor.
8.31.2 Field Documentation

8.31.2.1 attributes

```
uint32_t flash_descriptor_t::attributes
```

Device attributes.

8.31.2.2 page_size

```
uint32_t flash_descriptor_t::page_size
```

Size of write page.

8.31.2.3 sectors_count

```
flash_sector_t flash_descriptor_t::sectors_count
```

Number of sectors in the device.

8.31.2.4 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.

8.31.2.5 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.
8.31.2.6 address

uint8_t* flash_descriptor_t::address

Flash address if memory mapped or zero.

Note

Conventionally, non memory mapped devices have address NULL.

8.31.2.7 size

uint32_t flash_descriptor_t::size

Flash size.

8.32 flash_sector_descriptor_t Struct Reference

Flash sector descriptor.

#include <hal_flash.h>

Collaboration diagram for flash_sector_descriptor_t:

---

Data Fields

- flash_offset_t offset
  Sector offset.
- uint32_t size
  Sector size.

8.32.1 Detailed Description

Flash sector descriptor.
8.32.2 Field Documentation

8.32.2.1 offset

```
flash_offset_t flash_sector_descriptor_t::offset
```

Sector offset.

8.32.2.2 size

```
uint32_t flash_sector_descriptor_t::size
```

Sector size.

8.33 GPTConfig Struct Reference

Driver configuration structure.

```
#include <hal_gpt_lld.h>
```

Collaboration diagram for GPTConfig:

![Collaboration diagram](image)

Data Fields

- `gptfreq_t frequency`
  
  Timer clock in Hz.

- `gptcallback_t callback`
  
  Timer callback pointer.
8.33.1 Detailed Description

Driver configuration structure.

Note
It could be empty on some architectures.

8.33.2 Field Documentation

8.33.2.1 frequency

gptfreq_t GPTConfig::frequency

Timer clock in Hz.

Note
The low level can use assertions in order to catch invalid frequency specifications.

8.33.2.2 callback

gptcallback_t GPTConfig::callback

Timer callback pointer.

Note
This callback is invoked on GPT counter events.
8.34 GPTDriver Struct Reference

Structure representing a GPT driver.

#include <hal_gpt_lld.h>

Collaboration diagram for GPTDriver:

Data Fields

- gptstate_t state
  
  Driver state.

- const GPTConfig * config
  
  Current configuration data.

8.34.1 Detailed Description

Structure representing a GPT driver.

8.34.2 Field Documentation

8.34.2.1 state

gptstate_t GPTDriver::state

Driver state.
8.35 hal_adc_config Struct Reference

Driver configuration structure.

#include <hal_adc.h>

Collaboration diagram for hal_adc_config:

8.35.1 Detailed Description

Driver configuration structure.

8.36 hal_adc_configuration_group Struct Reference

Conversion group configuration structure.

#include <hal_adc.h>
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`.
- `adcerrorcallback_t error_cb`
  Error callback or `NULL`.

8.36.1 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.
8.36.2 Field Documentation

8.36.2.1 circular

bool hal_adc_configuration_group::circular

Enables the circular buffer mode for the group.

8.36.2.2 num_channels

adc_channels_num_t hal_adc_configuration_group::num_channels

Number of the analog channels belonging to the conversion group.

8.36.2.3 end_cb

adccallback_t hal_adc_configuration_group::end_cb

Callback function associated to the group or NULL.

8.36.2.4 error_cb

adcerrorcallback_t hal_adc_configuration_group::error_cb

Error callback or NULL.
8.37 hal_adc_driver Struct Reference

Structure representing an ADC driver.

```c
#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.
8.37 hal_adc_driver Struct Reference

- const ADCConversionGroup * grpp
  Current conversion group pointer or NULL.
- thread_reference_t thread
  Waiting thread.
- mutex_t mutex
  Mutex protecting the peripheral.

8.37.1 Detailed Description

Structure representing an ADC driver.

8.37.2 Field Documentation

8.37.2.1 state

adcstate_t hal_adc_driver::state

Driver state.

8.37.2.2 config

const ADCConfig* hal_adc_driver::config

Current configuration data.

8.37.2.3 samples

adcsample_t* hal_adc_driver::samples

Current samples buffer pointer or NULL.

8.37.2.4 depth

size_t hal_adc_driver::depth

Current samples buffer depth or 0.
8.37.2.5 grpp

const ADCConversionGroup* hal_adc_driver::grpp

Current conversion group pointer or NULL.

8.37.2.6 thread

thread_reference_t hal_adc_driver::thread

Waiting thread.

8.37.2.7 mutex

mutex_t hal_adc_driver::mutex

Mutex protecting the peripheral.

8.38 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
```

8.38.1 Detailed Description

Driver configuration structure.
8.39 hal_dac_conversion_group Struct Reference

DAC Conversion group structure.

#include <hal_dac.h>

Collaboration diagram for hal_dac_conversion_group:

```
hal_dac_conversion_group
  + num_channels
  + dac_lld_conversion_group_fields
hal_dac_driver
  + state
  + samples
  + depth
  + thread
  + mutex
  + dac_lld_driver_fields
  +group
  +error_cb
  +end_cb
hal_dac_config
  + dac_lld_config_fields
  +config
```

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`.

8.39.1 Detailed Description

DAC Conversion group structure.

8.39.2 Field Documentation
8.39.2.1 num_channels

```cpp
tuint32_t hal_dac_conversion_group::num_channels
```

Number of DAC channels.

8.39.2.2 end_cb

```cpp
daccallback_t hal_dac_conversion_group::end_cb
```

Operation complete callback or NULL.

8.39.2.3 error_cb

```cpp
dacerrorcallback_t hal_dac_conversion_group::error_cb
```

Error handling callback or NULL.

8.40 hal_dac_driver Struct Reference

Structure representing a DAC driver.

```cpp
#include <hal_dac.h>
```
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.
8.40.1 Detailed Description

Structure representing a DAC driver.

8.40.2 Field Documentation

8.40.2.1 state

dacstate_t hal_dac_driver::state

Driver state.

8.40.2.2 grpp

const DACConversionGroup* hal_dac_driver::grpp

Conversion group.

8.40.2.3 samples

dacsample_t* hal_dac_driver::samples

Samples buffer pointer.

8.40.2.4 depth

size_t hal_dac_driver::depth

Samples buffer size.

8.40.2.5 config

const DACConfig* hal_dac_driver::config

Current configuration data.
8.40.2.6 thread

```c
thread_reference_t hal_dac_driver::thread
```

Waiting thread.

8.40.2.7 mutex

```c
mutex_t hal_dac_driver::mutex
```

Mutex protecting the bus.

8.41 hal_efl_config Struct Reference

Type of a structure representing a flash driver configuration.

```c
#include <hal_efl.h>
```

Collaboration diagram for hal_efl_config:

```
hal_efl_config
+ efl_lld_config_fields
```

8.41.1 Detailed Description

Type of a structure representing a flash driver configuration.
8.42 hal_efl_driver Struct Reference

Structure representing an embedded flash driver.

#include <hal_efl.h>

Inheritance diagram for hal_efl_driver:
Data Fields

- const struct EFlashDriverVMT *vmt

SNORDriver Virtual Methods Table.

8.42.1 Detailed Description

Structure representing an embedded flash driver.

8.42.2 Field Documentation

8.42.2.1 vmt

const struct EFlashDriverVMT* hal_efl_driver::vmt

SNORDriver Virtual Methods Table.
8.43  **hal_i2s_config Struct Reference**

Driver configuration structure.

```c
#include <hal_i2s.h>
```

Collaboration diagram for hal_i2s_config:

![Collaboration diagram](image)

**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.*

**8.43.1  Detailed Description**

Driver configuration structure.

**8.43.2  Field Documentation**
8.43.2.1 tx_buffer

const void* hal_i2s_config::tx_buffer

Transmission buffer pointer.

Note
Can be NULL if TX is not required.

8.43.2.2 rx_buffer

void* hal_i2s_config::rx_buffer

Receive buffer pointer.

Note
Can be NULL if RX is not required.

8.43.2.3 size

size_t hal_i2s_config::size

TX and RX buffers size as number of samples.

8.43.2.4 end_cb

i2scallback_t hal_i2s_config::end_cb

Callback function called during streaming.
8.44  **hal_i2s_driver Struct Reference**

Structure representing an I2S driver.

```
#include <hal_i2s.h>
```

Collaboration diagram for hal_i2s_driver:

```
hal_i2s_driver
+ state
+ i2s_lld_driver_fields
hal_i2s_config
+ tx_buffer
+ rx_buffer
+ size
+ i2s_lld_config_fields
```

### Data Fields

- **i2sstate_t state**
  
  Driver state.

- **const I2SConfig * config**
  
  Current configuration data.

### 8.44.1 Detailed Description

Structure representing an I2S driver.

### 8.44.2 Field Documentation

#### 8.44.2.1 state

```
i2sstate_t hal_i2s_driver::state
```

Driver state.
8.44.2.2 config

const I2SConfig* hal_i2s_driver::config

Current configuration data.

8.45 hal_sio_config Struct Reference

Driver configuration structure.

#include <hal_sio.h>

Collaboration diagram for hal_sio_config:

hal_sio_config
    + sio_lld_config_fields

8.45.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.46 hal_sio_driver Struct Reference

Structure representing a SIO driver.

#include <hal_sio.h>
Collaboration diagram for hal_sio_driver:

Data Fields

- const struct sio_driver_vmt *vmt
  Virtual Methods Table.
- siostate_t state
  Driver state.
- const SIOConfig *config
  Current configuration data.
- const SIOOperation *operation
  Current configuration data.
- thread_reference_t sync_rx
8.46.1 Detailed Description

Structure representing a SIO driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.46.2 Field Documentation

8.46.2.1 vmt

const struct sio_driver_vmt* hal_sio_driver::vmt

Virtual Methods Table.

8.46.2.2 state

siostate_t hal_sio_driver::state

Driver state.

8.46.2.3 config

const SIOConfig* hal_sio_driver::config

Current configuration data.

8.46.2.4 operation

const SIOOperation* hal_sio_driver::operation

Current configuration data.
8.46.2.5 sync_rx

thread_reference_t hal_sio_driver::sync_rx

Synchronization point for RX.

8.46.2.6 sync_tx

thread_reference_t hal_sio_driver::sync_tx

Synchronization point for TX.

8.46.2.7 sync_txend

thread_reference_t hal_sio_driver::sync_txend

Synchronization point for TX-end.

8.47 hal_sio_operation Struct Reference

Structure representing a SIO operation.

#include <hal_sio.h>

Collaboration diagram for hal_sio_operation:
Data Fields

- `siocb_t rx_cb`
  Receive non-empty callback.
- `siocb_t rx_idle_cb`
  Receive idle callback.
- `siocb_t tx_cb`
  Transmission buffer non-full callback.
- `siocb_t tx_end_cb`
  Physical end of transmission callback.
- `siocb_t rx_evt_cb`
  Receive event callback.

8.47.1 Detailed Description

Structure representing a SIO operation.

8.47.2 Field Documentation

8.47.2.1 rx_cb

```
siocb_t hal_sio_operation::rx_cb
```

Receive non-empty callback.

**Note**

Can be `NULL`.

8.47.2.2 rx_idle_cb

```
siocb_t hal_sio_operation::rx_idle_cb
```

Receive idle callback.

**Note**

Can be `NULL`.
8.47.2.3 tx_cb

siocb_t hal_sio_operation::tx_cb

Transmission buffer non-full callback.

Note
Can be NULL.

8.47.2.4 tx_end_cb

siocb_t hal_sio_operation::tx_end_cb

Physical end of transmission callback.

Note
Can be NULL.

8.47.2.5 rx_evt_cb

siocb_t hal_sio_operation::rx_evt_cb

Receive event callback.

Note
Can be NULL.

8.48 hal_spi_config Struct Reference

Driver configuration structure.

#include <hal_spi.h>
Data Fields

- **bool circular**
  *Enables the circular buffer mode.*
- **spicallback_t end_cb**
  *Operation complete callback or NULL.*
- **ioinline_t ssline**
  *The chip select line.*
- **ioportid_t ssport**
  *The chip select port.*
- **ioportmask_t ssmask**
  *The chip select port mask.*
- **uint_fast8_t sspad**
  *The chip select pad number.*

### 8.48.1 Detailed Description

Driver configuration structure.

### 8.48.2 Field Documentation
8.48.2.1  circular

    bool hal_spi_config::circular

Enables the circular buffer mode.

8.48.2.2  end_cb

    spicallback_t hal_spi_config::end_cb

Operation complete callback or NULL.

8.48.2.3  ssline

    ioline_t hal_spi_config::ssline

The chip select line.

8.48.2.4  ssport

    ioportid_t hal_spi_config::ssport

The chip select port.

8.48.2.5  ssmask

    ioportmask_t hal_spi_config::ssmask

The chip select port mask.

8.48.2.6  sspad

    uint_fast8_t hal_spi_config::sspad

The chip select pad number.
8.49 hal_spi_driver Struct Reference

Structure representing an SPI driver.

#include <hal_spi.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.

8.49.1 Detailed Description

Structure representing an SPI driver.

8.49.2 Field Documentation
8.49.2.1 state

spistate_t hal_spi_driver::state

Driver state.

8.49.2.2 config

const SPIConfig* hal_spi_driver::config

Current configuration data.

8.49.2.3 thread

thread_reference_t hal_spi_driver::thread

Waiting thread.

8.49.2.4 mutex

mutex_t hal_spi_driver::mutex

Mutex protecting the peripheral.

8.50 hal_trng_config Struct Reference

Driver configuration structure.

#include <hal_trng.h>

Collaboration diagram for hal_trng_config:
8.50.1 Detailed Description

Driver configuration structure.

8.51 hal_trng_driver Struct Reference

Structure representing a TRNG driver.

#include <hal_trng.h>

Collaboration diagram for hal_trng_driver:

```
hal_trng_config
 + trng_lld_config_fields

+config

hal_trng_driver
 + state
 + trng_lld_driver_fields
```

Data Fields

- trngstate_t state
  
  Driver state.
- const TRNGConfig * config
  
  Current configuration data.

8.51.1 Detailed Description

Structure representing a TRNG driver.

8.51.2 Field Documentation
8.51.2.1 state

trngstate_t hal_trng_driver::state

Driver state.

8.51.2.2 config

const TRNGConfig* hal_trng_driver::config

Current configuration data.

8.52 hal_wspi_config Struct Reference

Driver configuration structure.

#include <hal_wspi.h>

Collaboration diagram for hal_wspi_config:

Data Fields

- wspicallback_t end_cb
  
  Operation complete callback or NULL.

- wspicallback_t error_cb
  
  Operation error callback or NULL.
8.52.1 Detailed Description

Driver configuration structure.

8.52.2 Field Documentation

8.52.2.1 end_cb

wspicallback_t hal_wspi_config::end_cb

Operation complete callback or NULL.

8.52.2.2 error_cb

wspicallback_t hal_wspi_config::error_cb

Operation error callback or NULL.

8.53 hal_wspi_driver Struct Reference

Structure representing an WSPI driver.

#include <hal_wspi.h>

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.

8.53.1 Detailed Description

Structure representing an WSPI driver.

8.53.2 Field Documentation

8.53.2.1 state

`wspistate_t hal_wspi_driver::state`

Driver state.

8.53.2.2 config

`const WSPIConfig* hal_wspi_driver::config`

Current configuration data.

8.53.2.3 thread

`thread_reference_t hal_wspi_driver::thread`

Waiting thread.

8.53.2.4 mutex

`mutex_t hal_wspi_driver::mutex`

Mutex protecting the peripheral.
8.54 halclkcfg_t Struct Reference

Type of a clock configuration structure.

#include <hal_lld.h>

Collaboration diagram for halclkcfg_t:

8.54.1 Detailed Description

Type of a clock configuration structure.

8.55 HMACSHA256Context Struct Reference

Type of a HMAC_SHA256 context.

#include <hal_crypto_lld.h>

Collaboration diagram for HMACSHA256Context:

8.55.1 Detailed Description

Type of a HMAC_SHA256 context.
8.56 HMACSHA512Context Struct Reference

Type of a HMAC_SHA512 context.

#include <hal_crypto_lld.h>

Collaboration diagram for HMACSHA512Context:

8.56.1 Detailed Description

Type of a HMAC_SHA512 context.

8.57 I2CConfig Struct Reference

Type of I2C driver configuration structure.

#include <hal_i2c_lld.h>

Collaboration diagram for I2CConfig:

8.57.1 Detailed Description

Type of I2C driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.
Structure representing an I2C driver.

```
#include <hal_i2c_lld.h>
```

Collaboration diagram for I2CDriver:

---

**Data Fields**

- `i2cstate_t state`
  
  *Driver state.*

- `const I2CConfig * config`
  
  *Current configuration data.*

- `i2cflags_t errors`
  
  *Error flags.*

---

**8.58.1 Detailed Description**

Structure representing an I2C driver.

---

**8.58.2 Field Documentation**
8.58.2.1 state

\texttt{i2cstate_t I2CDriver::state}

Driver state.

8.58.2.2 config

\texttt{const I2CConfig* I2CDriver::config}

Current configuration data.

8.58.2.3 errors

\texttt{i2cflags_t I2CDriver::errors}

Error flags.

8.59 ICUConfig Struct Reference

Driver configuration structure.

\texttt{#include <hal_icu_lld.h>}

Collaboration diagram for ICUConfig:

\begin{center}
\includegraphics[width=0.5\textwidth]{diagram.png}
\end{center}
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.

8.59.1 Detailed Description

Driver configuration structure.

Note
It could be empty on some architectures.

8.59.2 Field Documentation

8.59.2.1 mode

icumode_t ICUConfig::mode

Driver mode.

8.59.2.2 frequency

icufreq_t ICUConfig::frequency

Timer clock in Hz.

Note
The low level can use assertions in order to catch invalid frequency specifications.
8.59.2.3 width_cb

`icuCallback_t ICUConfig::width_cb`

Callback for pulse width measurement.

8.59.2.4 period_cb

`icuCallback_t ICUConfig::period_cb`

Callback for cycle period measurement.

8.59.2.5 overflow_cb

`icuCallback_t ICUConfig::overflow_cb`

Callback for timer overflow.

8.60 ICUDriver Struct Reference

Structure representing an ICU driver.

```c
#include <hal_icu_lld.h>
```

Collaboration diagram for ICUDriver:
Data Fields

- `icustate_t state`
  Driver state.
- `const ICUConfig * config`
  Current configuration data.

8.60.1 Detailed Description

Structure representing an ICU driver.

8.60.2 Field Documentation

8.60.2.1 state

`icustate_t ICUDriver::state`

Driver state.

8.60.2.2 config

`const ICUConfig* ICUDriver::config`

Current configuration data.

8.61 io_buffers_queue Struct Reference

Structure of a generic buffers queue.

#include <hal_buffers.h>
Collaboration diagram for io_buffers_queue:

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**
### 8.61.1 Detailed Description

Structure of a generic buffers queue.

### 8.61.2 Field Documentation

#### 8.61.2.1 waiting

```c
threads_queue_t io_buffers_queue::waiting
```

Queue of waiting threads.

#### 8.61.2.2 suspended

```c
bool io_buffers_queue::suspended
```

Queue suspended state flag.

#### 8.61.2.3 bcounter

```c
volatile size_t io_buffers_queue::bcounter
```

Active buffers counter.

#### 8.61.2.4 bwrptr

```c
uint8_t* io_buffers_queue::bwrptr
```

Buffer write pointer.
8.61.2.5  **brdptr**

`uint8_t* io_buffers_queue::brdptr`

Buffer read pointer.

8.61.2.6  **btop**

`uint8_t* io_buffers_queue::btop`

Pointer to the buffers boundary.

8.61.2.7  **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.

8.61.2.8  **bn**

`size_t io_buffers_queue::bn`

Number of buffers.

8.61.2.9  **buffers**

`uint8_t* io_buffers_queue::buffers`

Queue of buffer objects.
8.61.2.10  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.

8.61.2.11  top

```
uint8_t* io_buffers_queue::top
```

Boundary for R/W sequential access.

8.61.2.12  notify

```
bqnotify_t io_buffers_queue::notify
```

Data notification callback.

8.61.2.13  link

```
void* io_buffers_queue::link
```

Application defined field.
8.62 io_queue Struct Reference

Generic I/O queue structure.

#include <hal_queues.h>

Collaboration diagram for io_queue:

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.
8.62.1 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.

8.62.2 Field Documentation

8.62.2.1 q_waiting

```c
threads_queue_t io_queue::q_waiting
```

Queue of waiting threads.

8.62.2.2 q_counter

```c
volatile size_t io_queue::q_counter
```

Resources counter.

8.62.2.3 q_buffer

```c
uint8_t* io_queue::q_buffer
```

Pointer to the queue buffer.

8.62.2.4 q_top

```c
uint8_t* io_queue::q_top
```

Pointer to the first location after the buffer.
8.62.2.5 q_wrptr

```c
uint8_t* io_queue::q_wrptr
```
Write pointer.

8.62.2.6 q_rdptr

```c
uint8_t* io_queue::q_rdptr
```
Read pointer.

8.62.2.7 q_notify

```c
qnotify_t io_queue::q_notify
```
Data notification callback.

8.62.2.8 q_link

```c
void* io_queue::q_link
```
Application defined field.

8.63 IOBus Struct Reference

I/O bus descriptor.

```c
#include <hal_pal.h>
```

Collaboration diagram for IOBus:

```
IOBus
+ portid
+ mask
+ offset
```
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.

8.63.1 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.

8.63.2 Field Documentation

8.63.2.1 portid

`ioportid_t IOBus::portid`

Port identifier.

8.63.2.2 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.

8.63.2.3 offset

`uint_fast8_t IOBus::offset`

Offset, within the port, of the least significant bit of the bus.
8.64 MACConfig Struct Reference

Driver configuration structure.

#include <hal_mac_lld.h>

Collaboration diagram for MACConfig:

![Collaboration Diagram]

Data Fields

- uint8_t * mac_address

  MAC address.

8.64.1 Detailed Description

Driver configuration structure.

8.64.2 Field Documentation

8.64.2.1 mac_address

uint8_t* MACConfig::mac_address

MAC address.
8.65 MACDriver Struct Reference

Structure representing a MAC driver.

#include <hal_mac_lld.h>

Collaboration diagram for MACDriver:

Data Fields

- **macstate_t state**
  
  Driver state.
- **const MACConfig * config**
  
  Current configuration data.
- **threads_queue_t tdqueue**
  
  Transmit semaphore.
- **threads_queue_t rdqueue**
  
  Receive semaphore.
- **event_source_t rdevent**
  
  Receive event.

8.65.1 Detailed Description

Structure representing a MAC driver.

8.65.2 Field Documentation
8.65.2.1 state

macstate_t MACDriver::state

Driver state.

8.65.2.2 config

const MACConfig* MACDriver::config

Current configuration data.

8.65.2.3 tdqueue

threads_queue_t MACDriver::tdqueue

Transmit semaphore.

8.65.2.4 rdqueue

threads_queue_t MACDriver::rdqueue

Receive semaphore.

8.65.2.5 rdevent

event_source_t MACDriver::rdevent

Receive event.

8.66 MACReceiveDescriptor Struct Reference

Structure representing a receive descriptor.

#include <hal_mac_lld.h>

Collaboration diagram for MACReceiveDescriptor:

```
+--- MACReceiveDescriptor
|   +--- offset
|   +--- size
```
Data Fields

- `size_t offset`
  Current read offset.
- `size_t size`
  Available data size.

8.66.1 Detailed Description

Structure representing a receive descriptor.

8.66.2 Field Documentation

8.66.2.1 offset

`size_t MACReceiveDescriptor::offset`

Current read offset.

8.66.2.2 size

`size_t MACReceiveDescriptor::size`

Available data size.

8.67 MACTransmitDescriptor Struct Reference

Structure representing a transmit descriptor.

```
#include <hal_mac_lld.h>
```

Collaboration diagram for MACTransmitDescriptor:
Data Fields

- size_t offset
  Current write offset.
- size_t size
  Available space size.

8.67.1 Detailed Description

Structure representing a transmit descriptor.

8.67.2 Field Documentation

8.67.2.1 offset

size_t MACTransmitDescriptor::offset

Current write offset.

8.67.2.2 size

size_t MACTransmitDescriptor::size

Available space size.

8.68 MemoryStream Struct Reference

Memory stream object.

#include <memstreams.h>
Inheritance diagram for MemoryStream:
Collaboration diagram for MemoryStream:

![Collaboration Diagram](image)

### Data Fields

- const struct MemStreamVMT* vmt

  *Virtual Methods Table.*

### 8.68.1 Detailed Description

Memory stream object.

### 8.68.2 Field Documentation

#### 8.68.2.1 vmt

const struct MemStreamVMT* MemoryStream::vmt

*Virtual Methods Table.*
8.69 MemStreamVMT Struct Reference

MemStream virtual methods table.

#include <memstreams.h>

Collaboration diagram for MemStreamVMT:

![Collaboration diagram for MemStreamVMT](image1)

8.69.1 Detailed Description

MemStream virtual methods table.

8.70 mfs_bank_header_t Union Reference

Type of a bank header.

#include <hal_mfs.h>

Collaboration diagram for mfs_bank_header_t:

![Collaboration diagram for mfs_bank_header_t](image2)
8.70.1 Detailed Description

Type of a bank header.

Note

The header resides in the first 16 bytes of a bank.

8.70.2 Field Documentation

8.70.2.1 magic1

`uint32_t mfs_bank_header_t::magic1`

Bank magic 1.

8.70.2.2 magic2

`uint32_t mfs_bank_header_t::magic2`

Bank magic 2.

8.70.2.3 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.

8.70.2.4 reserved1

`uint16_t mfs_bank_header_t::reserved1`

Reserved field.
8.70.2.5  crc

uint16_t mfs_bank_header_t::crc

Header CRC.

8.71  mfs_data_header_t Union Reference

Type of a data block header.

#include <hal_mfs.h>

Collaboration diagram for mfs_data_header_t:

```
+ mfs_data_header_t
  + magic1
  + magic2
  + id
  + crc
  + size
  + fields
  + hdr8
  + hdr32
```

8.71.1  Detailed Description

Type of a data block header.

This structure is placed before each written data block.

8.71.2  Field Documentation

8.71.2.1  magic1

uint32_t mfs_data_header_t::magic1

Data header magic 1.
8.71.2.2 magic2

`uint32_t mfs_data_header_t::magic2`

Data header magic 2.

8.71.2.3 id

`uint16_t mfs_data_header_t::id`

Record identifier.

8.71.2.4 crc

`uint16_t mfs_data_header_t::crc`

Data CRC.

8.71.2.5 size

`uint32_t mfs_data_header_t::size`

Data size.

Note

The next record is located at `MFS_ALIGN_NEXT(size)`.

8.72 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:
Data Fields

- `flash_offset_t offset`
  
  Written header offset.

- `size_t size`
  
  Written data size.

- `mfs_id_t id`
  
  Record identifier.

8.72.1 Detailed Description

Type of a buffered write/erase operation within a transaction.

8.72.2 Field Documentation

8.72.2.1 offset

```c
flash_offset_t mfs_transaction_op_t::offset
```

Written header offset.

8.72.2.2 size

```c
size_t mfs_transaction_op_t::size
```

Written data size.

8.72.2.3 id

```c
mfs_id_t mfs_transaction_op_t::id
```

Record identifier.
8.73 MFSCfg Struct Reference

Type of a MFS configuration structure.

#include <hal_mfs.h>

Collaboration diagram for MFSCfg:

![Collaboration Diagram]

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.*
8.73 MFSCfg Struct Reference

8.73.1 Detailed Description

Type of a MFS configuration structure.

8.73.2 Field Documentation

8.73.2.1 flashp

BaseFlash* MFSCfg::flashp

Flash driver associated to this MFS instance.

8.73.2.2 erased

uint32_t MFSCfg::erased

Erased value.

8.73.2.3 bank_size

flash_offset_t MFSCfg::bank_size

Banks size.

8.73.2.4 bank0_start

flash_sector_t MFSCfg::bank0_start

Base sector index for bank 0.

8.73.2.5 bank0_sectors

flash_sector_t MFSCfg::bank0_sectors

Number of sectors for bank 0.

Note

The total size of bank0 sectors must be greater or equal to bank_size.
8.73.2.6 bank1_start

flash_sector_t MFSConfig::bank1_start

Base sector index for bank 1.

8.73.2.7 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.

8.74 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`
  
  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.

- `union {
  
  } buffer`
  
  Transient buffer.

8.74.1 Detailed Description

Type of an MFS instance.

8.74.2 Field Documentation

8.74.2.1 state

`mfs_state_t MFSDriver::state`

Driver state.
8.74.2.2  config

const MFSConfig * MFSDriver::config

Current configuration data.

8.74.2.3  current_bank

mfs_bank_t MFSDriver::current_bank

Bank currently in use.

8.74.2.4  current_counter

uint32_t MFSDriver::current_counter

Usage counter of the current bank.

8.74.2.5  next_offset

flash_offset_t MFSDriver::next_offset

Pointer to the next free position in the current bank.

8.74.2.6  used_space

flash_offset_t MFSDriver::used_space

Used space in the current bank without considering erased records.

8.74.2.7  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.
8.74.2.8  tr_next_offset

`flash_offset_t` MFSDriver::tr_next_offset

Next write offset for current transaction.

8.74.2.9  tr_limit_offset

`flash_offset_t` MFSDriver::tr_limit_offset

Maximum offset for the transaction.

8.74.2.10  tr_nops

`uint32_t` MFSDriver::tr_nops

Number of buffered operations in current transaction.

8.74.2.11  tr_ops

`mfs_transaction_op_t` MFSDriver::tr_ops[MFS_CFG_TRANSACTION_MAX]

Buffered operations in current transaction.

8.74.2.12  buffer

`union { ... }` MFSDriver::buffer

Transient buffer.
8.75 MMCConfig Struct Reference

MMC/SD over SPI driver configuration structure.

```c
#include <hal_mmc_spi.h>
```

Collaboration diagram for MMCConfig:

---

**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.*

**8.75.1 Detailed Description**

MMC/SD over SPI driver configuration structure.
8.75.2 Field Documentation

8.75.2.1 spip

SpiDriver* MMCConfig::spip

SPI driver associated to this MMC driver.

8.75.2.2 lscfg

const SPIConfig* MMCConfig::lscfg

SPI low speed configuration used during initialization.

8.75.2.3 hscfg

const SPIConfig* MMCConfig::hscfg

SPI high speed configuration used during transfers.

8.76 MMCDriver Struct Reference

Structure representing a MMC/SD over SPI driver.

#include <hal_mmc_spi.h>
Inheritance diagram for MMCDriver:
Collaboration diagram for MMCDriver:

Data Fields

- const struct MMCDriverVMT * vmt
  Virtual Methods Table.
- const _mmcsd_block_device_data MMCConfig * config
  Current configuration data.
- bool block_addresses
  Addresses use blocks instead of bytes.

8.76.1 Detailed Description

Structure representing a MMC/SD over SPI driver.

8.76.2 Field Documentation
8.76.2.1 vmt

const struct MMCDriverVMT* MMCDriver::vmt

Virtual Methods Table.

8.76.2.2 config

const _mmcsd_block_device_data MMCConfig* MMCDriver::config

Current configuration data.

8.76.2.3 block_addresses

bool MMCDriver::block_addresses

Addresses use blocks instead of bytes.

8.77 MMCDriverVMT Struct Reference

MMCDriver virtual methods table.

#include <hal_mmc_spi.h>

Inheritance diagram for MMCDriverVMT:
Collaboration diagram for MMCDriverVMT:

![Collaboration diagram](image)

### 8.77.1 Detailed Description

**MMCDriver** virtual methods table.

### 8.78 MMCSDBlockDevice Struct Reference

MCC/SD block device class.

```
#include <hal_mmcsd.h>
```
Inheritance diagram for MMCSDBlockDevice:
Collaboration diagram for MMCSDBlockDevice:

Data Fields

- `const struct MMCSDBlockDeviceVMT * vmt`
  Virtual Methods Table.

8.78.1 Detailed Description

MCC/SD block device class.

This class represents a block-accessible, MMC/SD device.

8.78.2 Field Documentation

8.78.2.1 vmt

`const struct MMCSDBlockDeviceVMT * MMCSDBlockDevice::vmt`

Virtual Methods Table.
8.79  MMCSDBlockDeviceVMT Struct Reference

MMCSDBlockDevice virtual methods table.

#include <hal_mmcsd.h>

Inheritance diagram for MMCSDBlockDeviceVMT:

Collaboration diagram for MMCSDBlockDeviceVMT:
8.79.1 Detailed Description

`MMCSDBlockDevice` virtual methods table.

8.80 NullStream Struct Reference

Null stream object.

#include <nullstreams.h>

Inheritance diagram for NullStream:
Collaboration diagram for NullStream:

![Collaboration Diagram](image)

**Data Fields**

- const struct NullStreamVMT * vmt

  *Virtual Methods Table.*

### 8.80.1 Detailed Description

Null stream object.

### 8.80.2 Field Documentation

#### 8.80.2.1 vmt

const struct NullStreamVMT * NullStream::vmt

*Virtual Methods Table.*
8.81  NullStreamVMT Struct Reference

NullStream virtual methods table.
#include <nullstreams.h>

Collaboration diagram for NullStreamVMT:

```
Collaboration diagram for NullStreamVMT:

<table>
<thead>
<tr>
<th>NullStreamVMT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

8.81.1 Detailed Description

NullStream virtual methods table.

8.82  PALConfig Struct Reference

Generic I/O ports static initializer.
#include <hal_pal_lld.h>

Collaboration diagram for PALConfig:

```
Collaboration diagram for PALConfig:

<table>
<thead>
<tr>
<th>PALConfig</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

8.82.1 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.
8.83  palevent_t Struct Reference

Type of a PAL event record.

#include <hal_pal.h>

Collaboration diagram for palevent_t:

![Collaboration Diagram]

**Data Fields**

- **threads_queue_t threads**
  
  Threads queued for an event.

- **palcallback_t cb**
  
  Event callback.

- **void * arg**
  
  Event callback argument.

**8.83.1 Detailed Description**

Type of a PAL event record.

**8.83.2 Field Documentation**

**8.83.2.1 threads**

threads_queue_t palevent_t::threads

Threads queued for an event.
8.83.2.2 cb

*palcallback_t* palevent_t::cb

Event callback.

8.83.2.3 arg

*void* palevent_t::arg

Event callback argument.

8.84 PWMChannelConfig Struct Reference

Type of a PWM driver channel configuration structure.

```c
#include <hal_pwm_lld.h>
```

Collaboration diagram for PWMChannelConfig:
Data Fields

- **pwmmode_t mode**
  
  Channel active logic level.

- **pwmcallback_t callback**
  
  Channel callback pointer.

8.84.1 Detailed Description

Type of a PWM driver channel configuration structure.

8.84.2 Field Documentation

8.84.2.1 mode

```c
pwmmode_t PWMChannelConfig::mode
```

Channel active logic level.

8.84.2.2 callback

```c
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.
8.85 PWMConfig Struct Reference

Type of a PWM driver configuration structure.

#include <hal_pwm_lld.h>

Collaboration diagram for PWMConfig:

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.

8.85.1 Detailed Description

Type of a PWM driver configuration structure.
8.85.2 Field Documentation

8.85.2.1 frequency

```c
uint32_t PWMConfig::frequency
```

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.85.2.2 period

```c
pwmcnt_t PWMConfig::period
```

PWM period in ticks.

Note

The low level can use assertions in order to catch invalid period specifications.

8.85.2.3 callback

```c
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.

8.85.2.4 channels

```c
PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]
```

Channels configurations.
8.86  PWMDriver Struct Reference

Structure representing a PWM driver.

#include <hal_pwm_lld.h>

Collaboration diagram for PWMDriver:

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.
8.86.1 Detailed Description

Structure representing a PWM driver.

8.86.2 Field Documentation

8.86.2.1 state

\texttt{pwmstate_t PWMDriver::state}

Driver state.

8.86.2.2 config

\texttt{const PWMConfig* PWMDriver::config}

Current driver configuration data.

8.86.2.3 period

\texttt{pwmcnt_t PWMDriver::period}

Current PWM period in ticks.

8.86.2.4 enabled

\texttt{pwmchnmsk_t PWMDriver::enabled}

Mask of the enabled channels.

8.86.2.5 channels

\texttt{pwmchannel_t PWMDriver::channels}

Number of channels in this instance.
8.87  RTCAlarm Struct Reference

Type of a structure representing an RTC alarm time stamp.

#include <hal_rtc_lld.h>

Collaboration diagram for RTCAlarm:

8.87.1  Detailed Description

Type of a structure representing an RTC alarm time stamp.

8.88  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.

8.88.1 Detailed Description

Type of a structure representing an RTC date/time stamp.

8.88.2 Field Documentation

8.88.2.1 year

`uint32_t RTCDateTime::year`

Years since 1980.

8.88.2.2 month

`uint32_t RTCDateTime::month`

Months 1..12.
### 8.88.2.3 dstflag

uint32_t RTCDateTime::dstflag

DST correction flag.

### 8.88.2.4 dayofweek

uint32_t RTCDateTime::dayofweek

Day of week 1..7.

### 8.88.2.5 day

uint32_t RTCDateTime::day

Day of the month 1..31.

### 8.88.2.6 millisecond

uint32_t RTCDateTime::millisecond

Milliseconds since midnight.

### 8.89 RTCDriver Struct Reference

Structure representing an RTC driver.

#include <hal_rtc.h>
Collaboration diagram for RTCDriver:

![Collaboration Diagram](attachment:image.png)

**Data Fields**

- const struct RTCDriverVMT * vmt
  
  *Virtual Methods Table.*

**8.89.1 Detailed Description**

Structure representing an RTC driver.

**8.89.2 Field Documentation**

**8.89.2.1 vmt**

*const struct RTCDriverVMT* * RTCDriver::vmt

*Virtual Methods Table.*
**8.90 RTCDriverVMT Struct Reference**

*RTCDriver* virtual methods table.

```c
#include <hal rtc.h>
```

Inheritance diagram for RTCDriverVMT:
Collaboration diagram for RTCDriverVMT:

```
FileStream

BaseSequentialStream

FileStreamVMT

+ vmt
```

**Additional Inherited Members**

**8.90.1 Detailed Description**

RTCDriver virtual methods table.

**8.91 SDCCConfig Struct Reference**

Driver configuration structure.

```c
#include <hal_sdc_lld.h>
```

Collaboration diagram for SDCCConfig:

```
SDCCConfig
+ bus_width
```
Data Fields

- `sdcbusmode_t bus_width

  Bus width.

8.91.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.91.2 Field Documentation

8.91.2.1 bus_width

`sdcbusmode_t SDCConfig::bus_width

Bus width.

8.92 SDCDriver Struct Reference

Structure representing an SDC driver.

#include <hal_sdc_lld.h>
Collaboration diagram for SDCDriver:

Data Fields

- 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.

8.92.1 Detailed Description

Structure representing an SDC driver.
8.92.2 Field Documentation

8.92.2.1 vmt

const struct SDCDriverVMT * SDCDriver::vmt

Virtual Methods Table.

8.92.2.2 config

const _mmcsd_block_device_data SDCConfig* SDCDriver::config

Current configuration data.

8.92.2.3 cardmode

dcmode_t SDCDriver::cardmode

Various flags regarding the mounted card.

8.92.2.4 errors

sdcflags_t SDCDriver::errors

Errors flags.

8.92.2.5 rca

uint32_t SDCDriver::rca

Card RCA.

8.92.2.6 buf

uint8_t SDCDriver::buf[MMCSD_BLOCK_SIZE]

Buffer for internal operations.
8.93  SDCDriverVMT Struct Reference

SDCDriver virtual methods table.

#include <hal_sdc_lld.h>

Inheritance diagram for SDCDriverVMT:
Collaboration diagram for SDCDriverVMT:

8.93.1 Detailed Description

SDCDriver virtual methods table.

8.94 SerialConfig Struct Reference

PLATFORM Serial Driver configuration structure.

#include <hal_serial_lld.h>

Collaboration diagram for SerialConfig:
Data Fields

- uint32_t speed
  
  Bit rate.

8.94.1 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.

8.94.2 Field Documentation

8.94.2.1 speed

uint32_t SerialConfig::speed

Bit rate.

8.95 SerialDriver Struct Reference

Full duplex serial driver class.

#include <hal_serial.h>
Inheritance diagram for SerialDriver:
Collaboration diagram for SerialDriver:

Data Fields

- const struct SerialDriverVMT * vmt
  Virtual Methods Table.

8.95.1 Detailed Description

Full duplex serial driver class.

This class extends BaseAsynchronousChannel by adding physical I/O queues.

8.95.2 Field Documentation

8.95.2.1 vmt

const struct SerialDriverVMT* SerialDriver::vmt

Virtual Methods Table.
SerialDriver virtual methods table.

#include <hal_serial.h>

Inheritance diagram for SerialDriverVMT:
Collaboration diagram for SerialDriverVMT:

8.96.1 Detailed Description

SerialDriver virtual methods table.

8.97 SerialUSBCfg 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."

- **usbp_t bulk_in**
  
  "Bulk IN endpoint used for outgoing data transfer."

- **usbp_t bulk_out**
  
  "Bulk OUT endpoint used for incoming data transfer."

- **usbp_t int_in**
  
  "Interrupt IN endpoint used for notifications."

8.97.1 Detailed Description

Serial over USB Driver configuration structure.

An instance of this structure must be passed to * aduStart () in order to configure and start the driver operations.

8.97.2 Field Documentation
8.97.2.1 usbp

USBDriver* SerialUSBConfig::usbp

USB driver to use.

8.97.2.2 bulk_in

usbep_t SerialUSBConfig::bulk_in

Bulk IN endpoint used for outgoing data transfer.

8.97.2.3 bulk_out

usbep_t SerialUSBConfig::bulk_out

Bulk OUT endpoint used for incoming data transfer.

8.97.2.4 int_in

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:
Data Fields

- const struct SerialUSBDriverVMT * vmt
  
  Virtual Methods Table.

8.98.1 Detailed Description

Full duplex serial driver class.

This class extends BaseAsynchronousChannel by adding physical I/O queues.

8.98.2 Field Documentation

8.98.2.1 vmt

const struct SerialUSBDriverVMT * SerialUSBDriver::vmt

Virtual Methods Table.
**SerialDriver** virtual methods table.

```
#include <hal_serial_usb.h>
```

Inheritance diagram for SerialUSB driver VMT:
Collaboration diagram for SerialUSBDriverVMT:

```
+-------------------------
| BaseChannelVMT          |
|                         |
+-------------------------

+-------------------------
| BaseAsynchronousChannelVMT |
|                         |
+-------------------------

+-------------------------
| SerialUSBDriverVMT      |

8.99.1 Detailed Description

SerialDriver virtual methods table.

8.100 SHA1Context Struct Reference

Type of a SHA1 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA1Context:
8.100.1  Detailed Description

Type of a SHA1 context.

8.101  SHA256Context Struct Reference

Type of a SHA256 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA256Context:

```
Collaboration diagram for SHA256Context:

SHA256Context
+ dummy
```

8.101.1  Detailed Description

Type of a SHA256 context.

8.102  SHA512Context Struct Reference

Type of a SHA512 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA512Context:

```
Collaboration diagram for SHA512Context:

SHA512Context
+ dummy
```
8.102.1 Detailed Description

Type of a SHA512 context.

8.103 sio_driver_vmt Struct Reference

SIODriver virtual methods table.

#include <hal_sio.h>

Inheritance diagram for sio_driver_vmt:
Collaboration diagram for sio_driver_vmt:

8.103.1 Detailed Description

`SIODriver` virtual methods table.

8.104 SNORConfig Struct Reference

Type of a SNOR configuration structure.

```c
#include <hal_serial_nor.h>
```

Collaboration diagram for SNORConfig:
8.104.1 Detailed Description

Type of a SNOR configuration structure.

8.105 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.

- uint8_t device_id [20]
  
  Device ID and unique ID.

8.105.1 Detailed Description

Type of SNOR flash class.

8.105.2 Field Documentation
8.105.2.1 vmt

const struct SNORDriverVMT* SNORDriver::vmt

SNORDriver Virtual Methods Table.

8.105.2.2 config

const _base_flash_data SNORConfig* SNORDriver::config

Current configuration data.

8.105.2.3 device_id

uint8_t SNORDriver::device_id[20]

Device ID and unique ID.

8.106 SNORDriverVMT Struct Reference

SNOR virtual methods table.

#include <hal_serial_nor.h>

Inheritance diagram for SNORDriverVMT:
Collaboration diagram for SNORDriverVMT:

8.106.1 Detailed Description

SNOR virtual methods table.

8.107 threads_queue_t Struct Reference

Type of a thread queue.

#include <osal.h>

Collaboration diagram for threads_queue_t:

8.107.1 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.
8.108 UARTConfig Struct Reference

Driver configuration structure.

```c
#include <hal_uart_lld.h>
```

Collaboration diagram for UARTConfig:

```
UARTConfig
  +config
  +rxend_cb
  +rxchar_cb
  +rxerr_cb
  +txend2_cb
  +txend1_cb

UARTDriver
  + state
  + bstate
  + rstate
  + early
  + threadrx
  + threadtx
  + mutex
```

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.*

- `uartccb_t rxchar_cb`
  
  *Character received while out if the UART_RECEIVE state.*

- `uartecb_t rxerr_cb`
  
  *Receive error callback.*

8.108.1 Detailed Description

Driver configuration structure.

**Note**

Implementations may extend this structure to contain more, architecture dependent, fields.
8.108.2 Field Documentation

8.108.2.1 txend1_cb

```c
uartcb_t UARTConfig::txend1_cb
```

End of transmission buffer callback.

8.108.2.2 txend2_cb

```c
uartcb_t UARTConfig::txend2_cb
```

Physical end of transmission callback.

8.108.2.3 rxend_cb

```c
uartcb_t UARTConfig::rxend_cb
```

Receive buffer filled callback.

8.108.2.4 rxchar_cb

```c
uartccb_t UARTConfig::rxchar_cb
```

Character received while out if the UART_RECEIVE state.

8.108.2.5 rxerr_cb

```c
uartecb_t UARTConfig::rxerr_cb
```

Receive error callback.
8.109 UARTDriver Struct Reference

Structure representing an UART driver.

```c
#include <hal_uart_lld.h>
```

Collaboration diagram for UARTDriver:

![Collaboration Diagram](image)

**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.
8.109 UARTDriver Struct Reference

8.109.1 Detailed Description

Structure representing an UART driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.109.2 Field Documentation

8.109.2.1 state

 typedef uartstate_t UARTDriver::state

Driver state.

8.109.2.2 txstate

 typedef uartxstate_t UARTDriver::txstate

Transmitter state.

8.109.2.3 rxstate

 typedef uartrxstate_t UARTDriver::rxstate

Receiver state.

8.109.2.4 config

 typedef const UARTConfig* UARTDriver::config

Current configuration data.
8.109.2.5 early

bool UARTDriver::early

Synchronization flag for transmit operations.

8.109.2.6 threadrx

thread_reference_t UARTDriver::threadrx

Waiting thread on RX.

8.109.2.7 threadtx

thread_reference_t UARTDriver::threadtx

Waiting thread on TX.

8.109.2.8 mutex

mutex_t UARTDriver::mutex

Mutex protecting the peripheral.

8.110 unpacked_mmc_cid_t Struct Reference

Unpacked CID register from MMC.

#include <hal_mmcsd.h>

Collaboration diagram for unpacked_mmc_cid_t:
### 8.110.1 Detailed Description

Unpacked CID register from MMC.

### 8.111 unpacked_mmc_csd_t Struct Reference

Unpacked CSD register from MMC.

#include `<hal_mmcsd.h`

Collaboration diagram for unpacked_mmc_csd_t:

![Collaboration diagram for unpacked_mmc_csd_t](image)

### 8.111.1 Detailed Description

Unpacked CSD register from MMC.

### 8.112 unpacked_sdc_cid_t Struct Reference

Unpacked CID register from SDC.

#include `<hal_mmcsd.h`

ChibiOS/HAL
8.112.1 Detailed Description

Unpacked CID register from SDC.

8.113 unpacked_sdc_csd_10_t Struct Reference

Unpacked CSD v1.0 register from SDC.

#include <hal_mmcsd.h>

Collaboration diagram for unpacked_sdc_csd_10_t:
8.114 unpacked_sdc_csd_20_t Struct Reference

Unpacked CSD v2.0 register from SDC.

#include <hal_mmcsd.h>

Collaboration diagram for unpacked_sdc_csd_20_t:

```
unpacked_sdc_csd_20_t
+ csd_structure
+ taac
+ nsac
+ tran_speed
+ ccc
+ read_bl_len
+ read_bl_partial
+ write_blk_misalign
+ read_blk_misalign
+ dsr_imp
and 14 more...
```

8.114.1 Detailed Description

Unpacked CSD v2.0 register from SDC.

8.115 USBConfig Struct Reference

Type of an USB driver configuration structure.

#include <hal_usb_lld.h>
Collaboration diagram for USBConfig:

```
<table>
<thead>
<tr>
<th>USBConfig</th>
<th>+ get_descriptor_cb</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+config</td>
</tr>
<tr>
<td></td>
<td>+requests_hook_cb</td>
</tr>
<tr>
<td></td>
<td>+event_cb</td>
</tr>
<tr>
<td></td>
<td>+sof_cb</td>
</tr>
<tr>
<td>USBDriver</td>
<td>+ state</td>
</tr>
<tr>
<td></td>
<td>+ transmitting</td>
</tr>
<tr>
<td></td>
<td>+ receiving</td>
</tr>
<tr>
<td></td>
<td>+ in_params</td>
</tr>
<tr>
<td></td>
<td>+ out_params</td>
</tr>
<tr>
<td></td>
<td>+ ep0state</td>
</tr>
<tr>
<td></td>
<td>+ ep0next</td>
</tr>
<tr>
<td></td>
<td>+ ep0n</td>
</tr>
<tr>
<td></td>
<td>+ setup</td>
</tr>
<tr>
<td></td>
<td>+ status</td>
</tr>
<tr>
<td></td>
<td>+ address</td>
</tr>
<tr>
<td></td>
<td>+ configuration</td>
</tr>
<tr>
<td></td>
<td>+ saved_state</td>
</tr>
<tr>
<td></td>
<td>+config</td>
</tr>
<tr>
<td></td>
<td>+requests_hook_cb</td>
</tr>
<tr>
<td></td>
<td>+event_cb</td>
</tr>
<tr>
<td></td>
<td>+sof_cb</td>
</tr>
<tr>
<td></td>
<td>ep0endcb</td>
</tr>
<tr>
<td></td>
<td>out_cb</td>
</tr>
<tr>
<td></td>
<td>in_cb</td>
</tr>
<tr>
<td></td>
<td>setup_cb</td>
</tr>
<tr>
<td></td>
<td>epc</td>
</tr>
<tr>
<td>USBEndpointConfig</td>
<td>+ ep_mode</td>
</tr>
<tr>
<td></td>
<td>+ in_maxsize</td>
</tr>
<tr>
<td></td>
<td>+ out_maxsize</td>
</tr>
</tbody>
</table>
```

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.
8.115 Detailed Description

Type of an USB driver configuration structure.

8.115.2 Field Documentation

8.115.2.1 event_cb

```
usbeventcb_t USBConfig::event_cb
```

USB events callback.

This callback is invoked when an USB driver event is registered.

8.115.2.2 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.

8.115.2.3 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.

8.115.2.4 sof_cb

```
usbcallback_t USBConfig::sof_cb
```

Start Of Frame callback.
8.116 USBDescriptor Struct Reference

Type of an USB descriptor.

#include <hal_usb.h>

Collaboration diagram for USBDescriptor:

![Collaboration Diagram]

Data Fields

- `size_t ud_size`
  
  Descriptor size in unicode characters.

- `const uint8_t * ud_string`
  
  Pointer to the descriptor.

8.116.1 Detailed Description

Type of an USB descriptor.

8.116.2 Field Documentation

8.116.2.1 ud_size

`size_t USBDescriptor::ud_size`

Descriptor size in unicode characters.

8.116.2.2 ud_string

`const uint8_t* USBDescriptor::ud_string`

Pointer to the descriptor.
# USBDriver Struct Reference

Structure representing an USB driver.

```c
#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`

---

ChibiOS/HAL
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.

8.117.1 Detailed Description

Structure representing an USB driver.

8.117.2 Field Documentation

8.117.2.1 state

`usbstate_t USBDriver::state`

Driver state.

8.117.2.2 config

`const USBConfig* USBDriver::config`

Current configuration data.

8.117.2.3 transmitting

`uint16_t USBDriver::transmitting`

Bit map of the transmitting IN endpoints.
8.117.2.4 receiving

```c
uint16_t USBDriver::receiving
```

Bit map of the receiving OUT endpoints.

8.117.2.5 epc

```c
const USBEndpointConfig* USBDriver::epc[USB_MAX_ENDPOINTS+1]
```

Active endpoints configurations.

8.117.2.6 in_params

```c
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.

8.117.2.7 out_params

```c
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.

8.117.2.8 ep0state

```c
usbep0state_t USBDriver::ep0state
```

Endpoint 0 state.
8.117.2.9 ep0next

`uint8_t* USBDriver::ep0next`

Next position in the buffer to be transferred through endpoint 0.

8.117.2.10 ep0n

`size_t USBDriver::ep0n`

Number of bytes yet to be transferred through endpoint 0.

8.117.2.11 ep0endcb

`usbcallback_t USBDriver::ep0endcb`

Endpoint 0 end transaction callback.

8.117.2.12 setup

`uint8_t USBDriver::setup[8]`

Setup packet buffer.

8.117.2.13 status

`uint16_t USBDriver::status`

Current USB device status.

8.117.2.14 address

`uint8_t USBDriver::address`

Assigned USB address.
8.117.2.15 configuration

uint8_t USBDriver::configuration

Current USB device configuration.

8.117.2.16 saved_state

usbstate_t USBDriver::saved_state

State of the driver when a suspend happened.

8.118 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`

ChibiOS/HAL
8.118 USBEndpointConfig Struct Reference

8.118.1 Detailed Description

Type of an USB endpoint configuration structure.

Note

Platform specific restrictions may apply to endpoints.

8.118.2 Field Documentation

8.118.2.1 ep_mode

uint32_t USBEndpointConfig::ep_mode

Type and mode of the endpoint.

8.118.2.2 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.
8.118.2.3  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.

8.118.2.4  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.

8.118.2.5  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.

8.118.2.6  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.

8.118.2.7  in_state

USBInEndpointState* USBEndpointConfig::in_state

USBEndpointState associated to the IN endpoint.

This structure maintains the state of the IN endpoint.

8.118.2.8  out_state

USBOutEndpointState* USBEndpointConfig::out_state

USBEndpointState associated to the OUT endpoint.

This structure maintains the state of the OUT endpoint.
8.119 USBInEndpointState Struct Reference

Type of an IN endpoint state structure.

#include <hal_usb_lld.h>

Collaboration diagram for USBInEndpointState:

![Collaboration Diagram]

**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.

8.119.1 Detailed Description

Type of an IN endpoint state structure.

8.119.2 Field Documentation

8.119.2.1 txsize

*size_t USBInEndpointState::txsize*

Requested transmit transfer size.
8.119.2.2 txcnt

size_t USBInEndpointState::txcnt

Transmitted bytes so far.

8.119.2.3 txbuf

const uint8_t* USBInEndpointState::txbuf

Pointer to the transmission linear buffer.

8.119.2.4 thread

thread_reference_t USBInEndpointState::thread

Waiting thread.

8.120 USBOutEndpointState Struct Reference

Type of an OUT endpoint state structure.

#include <hal_usb_lld.h>

Collaboration diagram for USBOutEndpointState:
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.

8.120.1 Detailed Description

Type of an OUT endpoint state structure.

8.120.2 Field Documentation

8.120.2.1 rxsize

`size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

8.120.2.2 rxcnt

`size_t USBOutEndpointState::rxcnt`

Received bytes so far.

8.120.2.3 rxbuf

`uint8_t * USBOutEndpointState::rxbuf`

Pointer to the receive linear buffer.

8.120.2.4 thread

`thread_reference_t USBOutEndpointState::thread`

Waiting thread.
8.121 WDGConfig Struct Reference

Driver configuration structure.

#include <hal_wdg_lld.h>

Collaboration diagram for WDGConfig:

![Collaboration diagram for WDGConfig]

8.121.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.122 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.

8.122.1 Detailed Description

Structure representing an WDG driver.

8.122.2 Field Documentation

8.122.2.1 state

```c
wdgstate_t WDGDriver::state
```

Driver state.

8.122.2.2 config

```c
const WDGConfig* WDGDriver::config
```

Current configuration data.

8.123 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.

8.123.1 Detailed Description

Type of a WSPI command descriptor.

8.123.2 Field Documentation

8.123.2.1 cfg

```c
uint32_t wspi_command_t::cfg
```
Transfer configuration field.

8.123.2.2 cmd

```c
uint32_t wspi_command_t::cmd
```
Command phase data.

8.123.2.3 addr

```c
uint32_t wspi_command_t::addr
```
Address phase data.

8.123.2.4 alt

```c
uint32_t wspi_command_t::alt
```
Alternate phase data.

8.123.2.5 dummy

```c
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.

```c
#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.

```c
#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.

```c
#include <stdarg.h>
```
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.

```c
#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.

9.7 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 "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 "halBuffers.h"
#include "hal_queues.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 0**
  
  Stable release flag.

**ChibiOS/HAL version identification**

- **#define HAL_VERSION "7.2.0"**
HAL version string.
- #define CH_HAL_MAJOR 7
  HAL version major number.
- #define CH_HAL_MINOR 2
  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.

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 a ADCDriver structure.
- void 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 adclsBufferComplete(adcp) ((bool)((adcp)->state == ADC_COMPLETE))
  
  Buffer state.
Low level driver helper macros

- #define _adc_reset_i(adcp) osalThreadResume(&(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.
- void 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.
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.

**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.

9.12.1 Detailed Description

PLATFORM ADC subsystem low level driver header.
I/O Buffers code.

```c
#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 (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 (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.

9.13.1 Detailed Description
I/O Buffers code.

9.14 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)(bqSpace(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 *ibqp)
  
  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 *obqGetEmptyBufferI (output_buffers_queue_t *obqp, size_t *sizep)
  
  Gets the next empty 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 (outputBuffers_queue_t *obqp, uint8_t b, sysinterval_t timeout)**
  
  Output queue write with timeout.

- **size_t obqWriteTimeout (outputBuffers_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.

### 9.14.1 Detailed Description

I/O Buffers macros and structures.

### 9.15 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.

- **void 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.
9.15.1 Detailed Description

CAN Driver code.

9.16 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, 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.
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.
• void 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.

9.16.1 Detailed Description

CAN Driver macros and structures.

9.17 hal_can_lld.c File Reference

PLATFORM CAN subsystem low level driver source.

#include "hal.h"
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)
  
  Forcibly leaves the sleep mode.

Variables

- CANDriver CAND1
  
  CAN1 driver identifier.

9.17.1 Detailed Description

PLATFORM CAN subsystem low level driver source.

9.18 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 CANConfig
  
  Driver configuration structure.
- struct CANDriver
  
  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 CANDriver 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.

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.

9.18.1 Detailed Description

PLATFORM CAN subsystem low level driver header.
9.19  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 \\ ` 
  `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.

Macro Functions (BaseAsynchronousChannel)

• #define chnGetEventSource(ip) (&((ip)->event))
  Returns the I/O condition event source.
• #define chnAddFlags(ip, flags)
  Adds status flags to the listeners’s flags mask.

9.19.1 Detailed Description

I/O channels access.

This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

9.20 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.
• void 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_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 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_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.

• 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.

9.20.1 Detailed Description

Cryptographic Driver code.

9.21 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.

- void **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-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-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-GCM.

- 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.

- `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 macros and structures.
9.22  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, 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, const 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.

- 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.

Variables

• CRYDriver CRYD1
  CRY1 driver identifier.

9.22.1 Detailed Description

PLATFORM cryptographic subsystem low level driver source.

9.23 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_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, const uint8_t *tag_in)

Initialization of the DES transient key.

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

Encryption of a single block using (T)DES.

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

Encryption operation using (T)DES-EBC.

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

Decryption operation using (T)DES-EBC.

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

Encryption operation using (T)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)

Decryption 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)

Hash initialization using SHA1.

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

Hash update using SHA1.

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

Hash finalization using SHA1.

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

Hash initialization using SHA256.

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

Hash update 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_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.

### 9.23.1 Detailed Description

PLATFORM cryptographic subsystem low level driver header.

### 9.24 hal_dac.c File Reference

DAC Driver code.

```c
#include "hal.h"
```
Functions

- void dacInit (void)
  DAC Driver initialization.
- void dacObjectInit (DACDriver *dcp)
  Initializes the standard part of a DACDriver structure.
- void 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.
- msg_t dacConvert (DACDriver *dcp, const DACConversionGroup *grpp, dacsample_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.

9.24.1 Detailed Description

DAC Driver code.

9.25 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_wait_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.

- void 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.

9.25.1 Detailed Description

DAC Driver macros and structures.

9.26 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.

9.26.1 Detailed Description

PLATFORM DAC subsystem low level driver source.

9.27 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 *daccp)
  *Configures and activates the DAC peripheral.*
- void dac_lld_stop (DACDriver *daccp)
  *Deactivates the DAC peripheral.*
- void dac_lld_put_channel (DACDriver *daccp, dacchannel_t channel, dacsample_t sample)
  *Outputs a value directly on a DAC channel.*
- void dac_lld_start_conversion (DACDriver *daccp)
  *Starts a DAC conversion.*
- void dac_lld_stop_conversion (DACDriver *daccp)
  *Stops an ongoing conversion.*

9.27.1 Detailed Description

PLATFORM DAC subsystem low level driver header.

9.28 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.*
- void eflStart (EFlashDriver *eflp, const EFlashConfig *config)
  *Configures and starts the driver.*
- void eflStop (EFlashDriver *eflp)
  *Stops the driver.*

9.28.1 Detailed Description

Embedded Flash Driver code.

9.29 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_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.
- void 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 macros and structures.

9.30 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.

9.30.1 Detailed Description

PLATFORM Embedded Flash subsystem low level driver source.

9.31 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.

9.31.1 Detailed Description

PLATFORM Embedded Flash subsystem low level driver header.

9.32 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 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.`

9.32.1 Detailed Description

Data files.
This header defines abstract interfaces useful to access generic data files in a standardized way.

9.33 hal_flash.c File Reference

Generic flash driver class code.

```
#include "hal.h"
#include "hal_flash.h"
```
9.34 hal_flash.h File Reference

Generic flash driver class code.

9.34 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
  ChibiOS/HAL
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.

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.

9.34.1 Detailed Description

Generic flash driver class header.
9.35 hal_gpt.c File Reference

GPT Driver code.

#include "hal.h"

Functions

- void gptInit (void)
  
  GPT Driver initialization.

- void gptObjectInit (GPTDriver *gptp)
  
  Initializes the standard part of a GPTDriver structure.

- void 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.

9.35.1 Detailed Description

GPT Driver code.

9.36 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.

Functions

• `void gptInit (void)`  
  GPT Driver initialization.

• `void gptObjectInit (GPTDriver *gptp)`  
  Initializes the standard part of a GPTDriver structure.

• `void 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.
9.36.1 Detailed Description

GPT Driver macros and structures.

9.37 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.

9.37.1 Detailed Description

PLATFORM GPT subsystem low level driver source.

9.38 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.

9.38.1 Detailed Description

PLATFORM GPT subsystem low level driver header.

9.39  `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.
- void 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.

9.39.1 Detailed Description

I2C Driver code.

9.40 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 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.
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.

Functions

- `void i2cInit (void)`
  I2C Driver initialization.

- `void i2cObjectInit (I2CDriver *i2cp)`
  Initializes the standard part of a `I2CDriver` structure.

- `void 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 macros and structures.

9.41 hal_i2c_lld.c File Reference

PLATFORM I2C subsystem low level driver source.

#include "hal.h"

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_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.

Variables

- I2CDriver I2CD1
  I2C1 driver identifier.

9.41.1 Detailed Description

PLATFORM I2C subsystem low level driver source.

9.42 hal_i2c_lld.h File Reference

PLATFORM I2C subsystem low level driver header.

Data Structures

- struct I2CConfig
  Type of I2C driver configuration structure.
- struct I2CDriver
  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 I2CDriver 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.

9.42.1 Detailed Description

PLATFORM I2C subsystem low level driver header.

9.43 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.
- void 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.

9.43.1 Detailed Description

I2S Driver code.

9.44 hal_i2s.h File Reference

I2S Driver macros and structures.

#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 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.
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

- 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.
- void 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 macros and structures.

9.45 hal_i2s_lld.c File Reference

PLATFORM I2S subsystem low level driver source.

#include "hal.h"
Functions

- void i2s_lld_init (void)
  \(\text{Low level I2S driver initialization.}\)
- void i2s_lld_start (I2SDriver *i2sp)
  \(\text{Configures and activates the I2S peripheral.}\)
- void i2s_lld_stop (I2SDriver *i2sp)
  \(\text{Deactivates the I2S peripheral.}\)
- void i2s_lld_start_exchange (I2SDriver *i2sp)
  \(\text{Starts a I2S data exchange.}\)
- void i2s_lld_stop_exchange (I2SDriver *i2sp)
  \(\text{Stops the ongoing data exchange.}\)

Variables

- I2SDriver i2SD1
  \(\text{i2S2 driver identifier.}\)

9.45.1 Detailed Description

PLATFORM I2S subsystem low level driver source.

9.46 hal_i2s_lld.h File Reference

PLATFORM I2S subsystem low level driver header.

Macros

- #define i2s_lld_driver_fields
  \(\text{Low level fields of the I2S driver structure.}\)
- #define i2s_lld_config_fields
  \(\text{Low level fields of the I2S configuration structure.}\)

PLATFORM configuration options

- #define PLATFORM_I2S_USE_I2S1 FALSE
  \(\text{i2SD1 driver enable switch.}\)

Functions

- void i2s_lld_init (void)
  \(\text{Low level I2S driver initialization.}\)
- void i2s_lld_start (I2SDriver *i2sp)
  \(\text{Configures and activates the I2S peripheral.}\)
- void i2s_lld_stop (I2SDriver *i2sp)
  \(\text{Deactivates the I2S peripheral.}\)
- void i2s_lld_start_exchange (I2SDriver *i2sp)
  \(\text{Starts a I2S data exchange.}\)
- void i2s_lld_stop_exchange (I2SDriver *i2sp)
  \(\text{Stops the ongoing data exchange.}\)
9.46.1 Detailed Description

PLATFORM I2S subsystem low level driver header.

9.47 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.
- void 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.

9.47.1 Detailed Description

ICU Driver code.

9.48 hal_icu.h File Reference

ICU Driver macros and structures.

#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 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.

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.
- `void 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 macros and structures.

9.49 hal_icu_lld.c File Reference

PLATFORM ADC subsystem low level driver source.

#include "hal.h"

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.
9.49.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.50 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.

9.50.1 Detailed Description

PLATFORM ICU subsystem low level driver header.

9.51 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,
  BLK_SYNCING = 8 }
  Driver state machine possible states.

9.51.1 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.

9.52 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.
9.52.1 Detailed Description

PLATFORM HAL subsystem low level driver source.

9.53 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_USE_CLOCK_MANAGEMENT
  
  Specifies implementation of dynamic clock management.

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.

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 header.
9.54  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.
- void macStart (MACDriver *macp, const MACConfig *config)
  Configures and activates the MAC peripheral.
- void macStop (MACDriver *macp)
  Deactivates the MAC peripheral.
- msg_t macWaitTransmitDescriptor (MACDriver *macp, MACTransmitDescriptor *tdp, sysinterval_t timeout)
  Allocates a transmission descriptor.
- void macReleaseTransmitDescriptor (MACTransmitDescriptor *tdp)
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t macWaitReceiveDescriptor (MACDriver *macp, MACReceiveDescriptor *rdp, sysinterval_t timeout)
  Waits for a received frame.
- void macReleaseReceiveDescriptor (MACReceiveDescriptor *rdp)
  Releases a receive descriptor.
- bool macPollLinkStatus (MACDriver *macp)
  Updates and returns the link status.

9.54.1  Detailed Description

MAC Driver code.

9.55  hal_mac.h File Reference

MAC Driver macros and structures.

#include "hal_mac_lld.h"
Macros

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.

Macro Functions

• #define macGetReceiveEventSource(macp) (&(macp)->rdevent)
  Enables the zero-copy API.
• #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 MACDriver MACDriver
  Type of a structure representing a MAC driver.

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.
• void macStart (MACDriver *macp, const MACConfig *config)
  Configures and activates the MAC peripheral.
• void macStop (MACDriver *macp)
  Deactivates the MAC peripheral.
• msg_t macWaitTransmitDescriptor (MACDriver *macp, MACTransmitDescriptor *tdp, sysinterval_t timeout)
  Allocates a transmission descriptor.
• void macReleaseTransmitDescriptor (MACTransmitDescriptor *tdp)
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
• msg_t macWaitReceiveDescriptor (MACDriver *macp, MACReceiveDescriptor *rdp, sysinterval_t timeout)
  Waits for a received frame.
• void macReleaseReceiveDescriptor (MACReceiveDescriptor *rdp)
  Releases a receive descriptor.
• bool macPollLinkStatus (MACDriver *macp)
  Updates and returns the link status.
9.56 hal_mac_lld.c File Reference

PLATFORM MAC subsystem low level driver source.

```c
#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.

9.56.1 Detailed Description

PLATFORM MAC subsystem low level driver source.
9.57  

hal_mac_lld.h File Reference

PLATFORM MAC subsystem low level driver header.

Data Structures

- struct MACConfig
  
  Driver configuration structure.

- struct MACDriver
  
  Structure representing a MAC driver.

- struct MACTransmitDescriptor
  
  Structure representing a transmit descriptor.

- struct MACReceiveDescriptor
  
  Structure representing a receive descriptor.

Macros

- #define MAC_SUPPORTS_ZERO_COPY TRUE
  
  This implementation supports the zero-copy mode API.

PLATFORM configuration options

- #define PLATFORM_MAC_USE_MAC1 FALSE
  
  MAC driver enable switch.

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 (MACDriver *macp, 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 (MACDriver *macp, MACTransmitDescriptor *tdp, size_t size)
  
  Writes to a transmit descriptor's stream.

- size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor *rdp, size_t size)
  
  Reads from a receive descriptor's stream.

- uint8_t *mac_lld_get_next_transmit_buffer (MACReceiveDescriptor *rdp, size_t size)
  
  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)
  
  Returns a pointer to the next receive buffer in the descriptor chain.
9.57.1 Detailed Description

PLATFORM MAC subsystem low level driver header.

9.58 hal_mfs.c File Reference

Managed Flash Storage module code.

```c
#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.
• static mfs_error_t mfs_garbage_collect (MFSDriver ∗msp)  
  Enforces a garbage collection.
• static mfs_error_t mfs_try_mount (MFSDriver ∗msp)  
  Performs a flash partition mount attempt.
• mfs_error_t mfs_mount (MFSDriver ∗msp)  
  Configures and activates a MFS driver.
• void mfsObjectInit (MFSDriver ∗msp)  
  Initializes an instance.
• mfs_error_t mfsStart (MFSDriver ∗msp, const MFSCfg ∗config)  
  Configures and activates a MFS driver.
• void mfsStop (MFSDriver ∗msp)  
  Deactivates a MFS driver.
• mfs_error_t mfsErase (MFSDriver ∗msp)  
  Destroys the state of the managed storage by erasing the flash.
• mfs_error_t mfsReadRecord (MFSDriver ∗msp, mfs_id_t id, size_t ∗np, uint8_t ∗buffer)  
  Retrieves and reads a data record.
• mfs_error_t mfsWriteRecord (MFSDriver ∗msp, mfs_id_t id, size_t n, const uint8_t ∗buffer)  
  Creates or updates a data record.
• mfs_error_t mfsEraseRecord (MFSDriver ∗msp, mfs_id_t id)  
  Erases a data record.
• mfs_error_t mfsPerformGarbageCollection (MFSDriver ∗msp)  
  Enforces a garbage collection operation.
• mfs_error_t mfsStartTransaction (MFSDriver ∗msp, size_t size)  
  Puts the driver in transaction mode.
• mfs_error_t mfsCommitTransaction (MFSDriver ∗msp)  
  A transaction is committed and finalized atomically.
• mfs_error_t mfsRollbackTransaction (MFSDriver ∗msp)  
  A transaction is rolled back atomically.

9.58.1 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.

9.59 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.
- 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.
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)
  Initializes an instance.
- mfs_error_t mfsStart (MFSDriver ∗mfsp, const MFSCConfig ∗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.

9.59.1 Detailed Description

Managed Flash Storage module header.

9.60 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_Dounter 0x12
- \#define MII_FCSCOUNTER 0x13
- \#define MII_NWAYTEST 0x14
- \#define MII_RERRCOUNTER 0x15
- \#define MII_SREV 0x16
- \#define MII_RESV1 0x17
- \#define MII_LBERR 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 0x0078
- \#define BMSR_10HALF 0x0080
- \#define BMSR_10FULL 0x0100
- \#define BMSR_100HALF 0x0200
- \#define BMSR_100FULL 0x0400
- \#define BMSR_100BASE 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
9.60.1 Detailed Description

MII macros and structures.

9.61 hal_mmc_spi.c File Reference

MMC over SPI driver code.

#include <string.h>
#include "hal.h"

Functions

- static uint8_t crc7 (uint8_t crc, const uint8_t *buffer, size_t len)
  Calculate the MMC standard CRC-7 based on a lookup table.
- static void wait (MMCDriver *mmcp)
  Waits an idle condition.
- static void send_hdr (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)
  Sends a command header.
- static uint8_t recvr1 (MMCDriver *mmcp)
  Receives a single byte response.
- static uint8_t recvr3 (MMCDriver *mmcp, uint8_t *buffer)
  Receives a three byte response.
- static uint8_t send_command_R1 (MMCDriver *mmcp, uint8_t cmd, uint32_t arg)
  Sends a command an returns a single byte response.
- static uint8_t send_command_R3 (MMCDriver *mmcp, uint8_t cmd, uint32_t arg, uint8_t *response)
  Sends a command which returns a five bytes response (R3).
- static bool read_CxD (MMCDriver *mmcp, uint8_t cmd, uint32_t cxd[4])
  Reads the CSD.
- static void sync (MMCDriver *mmcp)
  Waits that the card reaches an idle state.
- void mmcInit (void)
  MMC over SPI driver initialization.
- void mmcObjectInit (MMCDriver *mmcp)
  Initializes an instance.
- void 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 MMCDriverVMT mmc_vmt
  Virtual methods table.

- static const uint8_t crc7_lookup_table [256]
  Lookup table for CRC-7 (based on polynomial $x^7 + x^3 + 1$).

### 9.61.1 Detailed Description

MMC over SPI driver code.

### 9.62 hal_mmc_spi.h File Reference

MMC over SPI driver header.

**Data Structures**

- struct MMCConfig
  MMC/SD over SPI driver configuration structure.

- struct MMCDriverVMT
  MMCDriver virtual methods table.

- struct MMCDriver
  Structure representing a MMC/SD over SPI driver.

**Macros**

- `#define _mmc_driver_methods _mmcsd_block_device_methods`
  MMCDriver specific methods.

**MMC_SPI configuration options**

- `#define MMC_NICE_WAITING TRUE`
  Delays insertions.

**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.
Functions

- **void mmcInit (void)**
  
  MMC over SPI driver initialization.

- **void mmcObjectInit (MMCDriver *mmcp)**
  
  Initializes an instance.

- **void 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.

9.62.1 Detailed Description

MMC over SPI driver header.

9.63 hal_mmcsd.c File Reference

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 *csdmcc)`

  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.

9.63.1 Detailed Description

MMC/SD cards common code.

9.64 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_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_20_CRC_SLICE 7U, 1U
• #define MMCSD_CSD_20_FILE_FORMAT_SLICE 11U, 10U
• #define MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE 13U, 13U
• #define MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE 12U, 12U
• #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, 22U
• #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_NAA_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

ChibiOS/HAL
• #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_ERASE_BLK_EN_SLICE MMCSD_CSD_20_ERASE_BLK_EN_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_PNM5_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.

**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 *clidmmc)`
  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.

### 9.64.1 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.

### 9.65 hal_objects.h File Reference

Base object.

**Data Structures**

- `struct BaseObjectVMT
  BaseObject virtual methods table.`
- `struct BaseObject
  Base stream class.`
Macros

- `#define _base_object_methods
  BaseObject specific methods.`
- `#define _base_object_data
  BaseObject specific data.`

Macro Functions (BaseObject)

- `#define objGetInstance(type, ip) (((size_t)(ip)) - (ip)>vmt->instance_offset)
  Returns the instance pointer starting from an interface pointer.`

9.65.1 Detailed Description

Base object.

This header defines a base object that is the root for the inheritance system.

9.66 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 palSetPadCallback (ioportid_t port, iopadid_t pad, palcallback_t cb, void *arg)
  Associates a callback to a port/pad.`
- `void palSetLineCallback (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.`
9.66.1 Detailed Description

I/O Ports Abstraction Layer code.

9.67 hal_pal.h File Reference

I/O Ports Abstraction Layer macros, types and structures.

#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 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) palClearPort(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 palEnablePadEvent(port, pad, mode)
  Pad event enable.
• #define palDisablePadEvent(port, pad)
  Pad event disable.
• #define palEnablePadEvent(port, pad, mode)
  Pad event enable.
• #define palDisablePadEvent(port, pad)
  Pad event disable.
• #define palEnableLineEvent(line, mode) palEnablePadEvent(PAL_PORT(line), PAL_PAD(line), mode)
  Line event enable.
• #define palDisableLineEvent(line) palDisablePadEvent(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))
9.68 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.
9.68.1 Detailed Description

PLATFORM PAL subsystem low level driver source.

9.69 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 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_readport(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.

```c
#define pal_lld_setpadmode(port, pad, mode)
Pad mode setup.
```

```c
#define pal_lld_get_pad_event(port, pad) &_pal_events[0]; (void)(port); (void)pad
Returns a PAL event structure associated to a pad.
```

```c
#define pal_lld_get_line_event(line) &_pal_events[0]; (void)line
Returns a PAL event structure associated to a line.
```

### Port related definitions

```c
#define PAL_IOPORTS_WIDTH 16U
Width, in bits, of an I/O port.
```

```c
#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)
Whole port mask.
```

### Line handling macros

```c
#define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad)))
Forms a line identifier.
```

```c
#define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFF0U))
Decodes a port identifier from a line identifier.
```

```c
#define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x00000FU))
Decodes a pad identifier from a line identifier.
```

```c
#define PAL_NOLINE 0U
Value identifying an invalid line.
```

### Typedefs

```c
typedef uint32_t ioportmask_t
Digital I/O port sized unsigned type.
```

```c
typedef uint32_t iomode_t
Digital I/O modes.
```

```c
typedef uint32_t ioline_t
Type of an I/O line.
```

```c
typedef uint32_t ioportid_t
Port Identifier.
```

```c
typedef uint32_t iopadid_t
Type of an pad identifier.
```

### Functions

```c
void _pal_lld_init (void)
STM32 I/O ports configuration.
```

```c
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 header.
9.70 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.

9.70.1 Detailed Description

Generic persistent storage class header.
9.71 hal_pwm.c File Reference

PWM Driver code.

#include "hal.h"

Functions

- void pwmInit (void)
  
  PWM Driver initialization.
- void pwmObjectInit (PWMDriver *pwmp)
  
  Initializes the standard part of a PWMDriver structure.
- void 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.

9.71.1 Detailed Description

PWM Driver code.

9.72 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)-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.

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.

Functions

- `void pwmInit (void)
  PWM Driver initialization.
- `void pwmObjectInit (PWMDriver *pwmp)
  Initializes the standard part of a PWMDriver structure.
- `void 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 macros and structures.

9.73 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.

9.73.1 Detailed Description

PLATFORM PWM subsystem low level driver source.

9.74 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 PWMConfig
   Type of a PWM driver 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 pwmcnt_t`
  Type of a PWM counter.
- `typedef uint32_t pwmcnmsk_t`
  Type of a channels mask.

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.

9.74.1 Detailed Description

PLATFORM PWM subsystem low level driver header.
9.75 hal_queues.c File Reference

I/O Queues code.

```c
#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.

9.75.1 Detailed Description

I/O Queues code.
9.76 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) ((bool)(qSpaceI(iqp) == 0U))`  
  Evaluates to `true` if the specified input queue is full.
- `#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) ((bool)(qSpaceI(oqp) == 0U))`  
  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.

9.76.1 Detailed Description

I/O Queues macros and structures.
9.77  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 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.

9.77.1  Detailed Description

RTC Driver code.

9.78  hal rtc.h File Reference

RTC Driver macros and structures.

#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_CATURDAY 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`

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 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.
RTC Driver macros and structures.

## Detailed Description

PLATFORM RTC subsystem low level driver source.

```c
#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.

## Detailed Description

PLATFORM RTC subsystem low level driver source.

## 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 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 header.
SDC Driver code.

```c
#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.
- void 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.

Variables

- static const struct SDCDriverVMT sdc_vmt
  Virtual methods table.

9.81.1 Detailed Description

SDC Driver code.

9.82 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.

- void 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.

- bool _sdc_wait_for_transfer_state (SDCDriver *sdcp)
  Wait for the card to complete pending operations.

9.82.1 Detailed Description

SDC Driver macros and structures.

9.83 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 *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.

Variables

- `SDCDriver SDCD1`
  
  SDCD1 driver identifier.

9.83.1 Detailed Description

PLATFORM SDC subsystem low level driver source.

9.84 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_short_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_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)

Reads one or more blocks.

- bool sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)

- bool sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)

Waits for card idle condition.

9.84.1 Detailed Description

PLATFORM SDC subsystem low level driver header.

9.85 hal_serial.c File Reference

Serial Driver code.

#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.

- void 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.

9.85.1 Detailed Description

Serial Driver code.
9.86  hal_serial.h File Reference

Serial Driver macros and structures.

#include "hal_serial_lld.h"

Data Structures

- struct SerialDriverVMT
  SerialDriver virtual methods table.
- struct SerialDriver
  Full duplex serial driver class.

Macros

- #define _serial_driver_methods _base_asynchronous_channel_methods
  SerialDriver specific methods.

Serial status flags

- #define SD_PARITY_ERROR (eventflags_t)32
  Parity.
- #define SD_FRAMING_ERROR (eventflags_t)64
  Framing.
- #define SD_OVERRUN_ERROR (eventflags_t)128
  Overflow.
- #define SD_NOISE_ERROR (eventflags_t)256
  Line noise.
- #define SD_BREAK_DETECTED (eventflags_t)512
  LIN Break.
- #define SD_QUEUE_FULL_ERROR (eventflags_t)1024
  Queue full.

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 sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->oqueue, b, t)
  Direct write to a SerialDriver with timeout specification.
• #define sdGet(sdp) iqGet(&(sdp)->iqueue)
  Direct read from a SerialDriver.
• #define sdGetI(sdp) iqGetI(&(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) oqWrite(&(sdp)->oqueue, b, n)
  Direct blocking write to a SerialDriver.
• #define sdWriteI(sdp, b, n) oqWriteI(&(sdp)->oqueue, b, n)
  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 sdRead(sdp, b, n) iqRead(&(sdp)->iqueue, b, n)
  Direct blocking read from a SerialDriver.
• #define sdReadI(sdp, b, n) iqReadI(&(sdp)->iqueue, b, n, TIME_INFINITE)
  Direct blocking read from a SerialDriver.
• #define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->iqueue, b, n, t)
  Direct blocking read from a SerialDriver with timeout specification.
• #define sdAsynchronousRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)
  Direct non-blocking read from a SerialDriver.

Typedefs

• typedef struct SerialDriver SerialDriver
  Structure representing a serial driver.

Enumerations

• enum sdstate_t { SD_UNINIT = 0, SD_STOP = 1, SD_READY = 2 }
  Driver state machine possible states.

Functions

• void sdInit (void)
  Serial Driver initialization.
• void sdObjectInit (SerialDriver *sdp, qnotify_t inotify, qnotify_t onotify)
  Initializes a generic serial driver object.
• void 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 macros and structures.

9.87 hal_serial_lld.c File Reference

PLATFORM serial subsystem low level driver source.

#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.

9.87.1 Detailed Description

PLATFORM serial subsystem low level driver source.

9.88 hal_serial_lld.h File Reference

PLATFORM serial subsystem low level driver header.

Data Structures

• struct SerialConfig
  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.

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.

9.88.1 Detailed Description

PLATFORM serial subsystem low level driver header.

9.89 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)
  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.

9.89.1 Detailed Description

Serial NOR serial flash driver code.

9.90 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_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.

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, uint32_t dummy, 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)`
  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.
9.90.1 Detailed Description

Serial NOR driver header.

9.91 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 *sdup)
  Initializes a generic full duplex driver object.
- void 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.

9.91.1 Detailed Description

Serial over USB Driver code.
Serial over USB Driver macros and structures.

```c
#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.
• void 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.

9.92.1 Detailed Description

Serial over USB Driver macros and structures.

9.93 hal_sio.c File Reference

SIO Driver code.

#include "hal.h"
Functions

- `void sioInit (void)`
  SIO Driver initialization.
- `void sioObjectInit (SIODriver *siop)`
  Initializes the standard part of a SIODriver structure.
- `bool sioStart (SIODriver *siop, const SIOConfig *config)`
  Configures and activates the SIO peripheral.
- `void sioStop (SIODriver *siop)`
  Deactivates the SIO peripheral.
- `void sioStartOperation (SIODriver *siop, const SIOOperation *operation)`
  Starts a SIO operation.
- `void sioStopOperation (SIODriver *siop)`
  Stops an ongoing SIO operation, if any.
- `sio_events_mask_t sioGetAndClearEvents (SIODriver *siop)`
  Return the pending SIO events 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 sioSynchronizeTX (SIODriver *siop, sysinterval_t timeout)`
  Synchronizes with TX FIFO space availability.
- `msg_t sioSynchronizeTXEnd (SIODriver *siop, sysinterval_t timeout)`
  Synchronizes with TX completion.

9.93.1 Detailed Description

SIO Driver code.

9.94 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.
- `struct hal_sio_operation`
  Structure representing a SIO operation.
Macros

- `#define _sio_driver_methods _base_channel_methods` _SIODriver specific methods._
- `#define sioGetFlagsX(siop) sio_lld_get_flags(siop)`
  Returns the current set of flags and clears it.
- `#define siolSRXEmptyX(siop) sio_lld_is_rx_empty(siop)`
  Determines the state of the RX FIFO.
- `#define siolSTXFullX(siop) sio_lld_is_tx_full(siop)`
  Determines the state of the TX FIFO.
- `#define sioGetAndClearEventsI(siop) sio_lld_get_and_clear_events(siop)`
  Return the pending SIO events 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 sioAsyncReadI(siop, size, buffer) sio_lld_read(siop, size, buffer)`
  Reads data from the RX FIFO.
- `#define sioAsyncWriteI(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.

SIO status flags

- `#define SIO_NO_ERROR 0` _No pending conditions._
- `#define SIO_PARITY_ERROR 4` _Parity error happened._
- `#define SIO_FRAMING_ERROR 8` _Framing error happened._
- `#define SIO_OVERRUN_ERROR 16` _Overflow happened._
- `#define SIO_NOISE_ERROR 32` _Noise on the line._
- `#define SIO_BREAK_DETECTED 64` _Break detected._

SIO additional messages

- `#define SIO_MSG_IDLE 1` __SIO_MSG_ERRORS 2_`

SIO configuration options

- `#define SIO_DEFAULT_BITRATE 38400` _Default bit rate._
- `#define SIO_USE_SYNCHRONIZATION TRUE`
Support for thread synchronization API.

Low level driver helper macros

- `#define __sio_callback_rx(siop)`  
  RX callback.
- `#define __sio_callback_rx_idle(siop)`  
  RX idle callback.
- `#define __sio_callback_tx(siop)`  
  TX callback.
- `#define __sio_callback_tx_end(siop)`  
  TX end callback.
- `#define __sio_callback_rx_evt(siop)`  
  RX event callback.
- `#define __sio_wakeup_rx(siop, msg)`  
  Wakes up the RX-waiting thread.
- `#define __sio_wakeup_tx(siop, msg)`  
  Wakes up the TX-waiting thread.
- `#define __sio_wakeup_txend(siop, msg)`  
  Wakes up the TXend-waiting thread.

Typedefs

- `typedef uint_fast8_t sioflags_t`  
  SIO driver condition flags type.
- `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 struct hal_sio_operation SIOOperation`  
  Type of structure representing a SIO operation.
- `typedef void(__sio_t) (SIODriver *siop)`  
  Generic SIO notification callback type.

Enumerations

- `enum siostate_t { SIO_UNINIT = 0, SIO_STOP = 1, SIO_READY = 2, SIO_ACTIVE = 3 }`  
  Driver state machine possible states.

Functions

- `void sioInit (void)`  
  SIO Driver initialization.
- `void sioObjectInit (SIODriver *siop)`  
  Initializes the standard part of a SIODriver structure.
- `bool sioStart (SIODriver *siop, const SIOConfig *config)`  
  Configures and activates the SIO peripheral.
- `void sioStop (SIODriver *siop)`  
  Deactivates the SIO peripheral.
- `void sioStartOperation (SIODriver *siop, const SIOOperation *operation)`  
  Starts a SIO operation.
• **void** sioStopOperation (SIODriver *siop)
  
  Stops an ongoing SIO operation, if any.

• **sio_events_mask_t** sioGetAndClearEvents (SIODriver *siop)
  
  Return the pending SIO events 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** 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 macros and structures.

### 9.95 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.

• **bool** sio_lld_start (SIODriver *siop)
  
  Configures and activates the SIO peripheral.

• **void** sio_lld_stop (SIODriver *siop)
  
  Deactivates the SIO peripheral.

• **void** sio_lld_start_operation (SIODriver *siop)
  
  Starts a SIO operation.

• **void** sio_lld_stop_operation (SIODriver *siop)
  
  Stops an ongoing SIO operation, if any.

• **sio_events_mask_t** sio_lld_get_and_clear_events (SIODriver *siop)
  
  Return the pending SIO events 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.
Variables

- **SIODriver SIOD1**
  
  SIO1 driver identifier.

### 9.95.1 Detailed Description

PLATFORM SIO subsystem low level driver source.

### 9.96 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_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.

### Typedefs

- **typedef uint32_t sio_events_mask_t**
  
  Type of a SIO events mask.
Functions

- void sio_lld_init (void)
  Low level SIO driver initialization.
- bool sio_lld_start (SIODriver *siop)
  Configures and activates the SIO peripheral.
- void sio_lld_stop (SIODriver *siop)
  Deactivates the SIO peripheral.
- void sio_lld_start_operation (SIODriver *siop)
  Starts a SIO operation.
- void sio_lld_stop_operation (SIODriver *siop)
  Stops an ongoing SIO operation, if any.
- sio_events_mask_t sio_lld_get_and_clear_events (SIODriver *siop)
  Return the pending SIO events 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.

9.96.1 Detailed Description

PLATFORM SIO subsystem low level driver header.

9.97 hal_spi.c File Reference

SPI Driver code.

#include "hal.h"

Functions

- void spInit (void)
  SPI Driver initialization.
- void spObjectInit (SPIDriver *spip)
  Initializes the standard part of a SPIDriver structure.
- void spStart (SPIDriver *spip, const SPIConfig *config)
  Configures and activates the SPI peripheral.
- void spStop (SPIDriver *spip)
  Deactivates the SPI peripheral.
- void spSelect (SPIDriver *spip)
  Asserts the slave select signal and prepares for transfers.
• void spiUnselect (SPIDriver *spip)
  Deasserts the slave select signal.
• void spiStartIgnore (SPIDriver *spip, size_t n)
  Ignores data on the SPI bus.
• void spiStartExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  Exchanges data on the SPI bus.
• void spiStartSend (SPIDriver *spip, size_t n, const void *txbuf)
  Sends data over the SPI bus.
• void spiStartReceive (SPIDriver *spip, size_t n, void *rxbuf)
  Receives data from the SPI bus.
• void spiAbortI (SPIDriver *spip)
  Aborts the ongoing SPI operation.
• void spiAbort (SPIDriver *spip)
  Aborts the ongoing SPI operation, if any.
• void spilgnore (SPIDriver *spip, size_t n)
  Ignores data on the SPI bus.
• void spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  Exchanges data on the SPI bus.
• voidspiSend (SPIDriver *spip, size_t n, const void *txbuf)
  Sends data over the SPI bus.
• voidspiReceive (SPIDriver *spip, size_t n, void *rxbuf)
  Receives data from the SPI bus.
• voidspiAcquireBus (SPIDriver *spip)
  Gains exclusive access to the SPI bus.
• voidspiReleaseBus (SPIDriver *spip)
  Releases exclusive access to the SPI bus.

### 9.97.1 Detailed Description

SPI Driver code.

### 9.98 hal_spi.h File Reference

SPI 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`  
  `@brief Legacy mode.`
- `#define SPI_SELECT_MODE_PAD`  
  `@brief Fastest mode.`
- `#define SPI_SELECT_MODE_PORT`  
  `@brief Packed mode.`
- `#define SPI_SELECT_MODE_LINE`  
  `@brief LLD-defined mode.`

SPI configuration options

- `#define SPI_USE_WAIT TRUE`  
  `Enables synchronous APIs.`
- `#define SPI_USE_CIRCULAR FALSE`  
  `Enables circular transfers 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
}

Driver state machine possible states.

Functions

- void spiInit (void)
  SPI Driver initialization.
- void spiObjectInit (SPIDriver *spip)
  Initializes the standard part of a SPIDriver structure.
- void spiStart (SPIDriver *spip, const SPIConfig *config)
  Configures and activates the SPI peripheral.
- void spiStop (SPIDriver *spip)
  Deactivates the SPI peripheral.
- void spiSelect (SPIDriver *spip)
  Asserts the slave select signal and prepares for transfers.
- void spiUnselect (SPIDriver *spip)
  Deasserts the slave select signal.
- void spiStartIgnore (SPIDriver *spip, size_t n)
  Ignores data on the SPI bus.
- void spiStartExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  Exchanges data on the SPI bus.
- void spiStartSend (SPIDriver *spip, size_t n, const void *txbuf)
  Sends data over the SPI bus.
- void spiStartReceive (SPIDriver *spip, size_t n, void *rxbuf)
  Receives data from the SPI bus.
- void spiAbortI (SPIDriver *spip)
  Aborts the ongoing SPI operation.
- void spiAbort (SPIDriver *spip)
  Aborts the ongoing SPI operation, if any.
- void spiIgnore (SPIDriver *spip, size_t n)
  Ignores data on the SPI bus.
- void spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
  Exchanges data on the SPI bus.
- void spiSend (SPIDriver *spip, size_t n, const void *txbuf)
  Sends data over the SPI bus.
- void spiReceive (SPIDriver *spip, size_t n, void *rxbuf)
  Receives data from the SPI bus.
- void spiAcquireBus (SPIDriver *spip)
  Gains exclusive access to the SPI bus.
- void spiReleaseBus (SPIDriver *spip)
  Releases exclusive access to the SPI bus.

9.98.1 Detailed Description

SPI Driver macros and structures.
9.99  hal_spi_lld.c File Reference

PLATFORM SPI 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.

9.99.1 Detailed Description

PLATFORM SPI subsystem low level driver source.

9.100  hal_spi_lld.h File Reference

PLATFORM SPI 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.

9.100.1 Detailed Description

PLATFORM SPI subsystem low level driver header.

9.101 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.

- 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.

Variables

- **st_callback_t** **st_callbacks** [ST_LLD_NUM_ALARMS]
  
  Callback pointers for each alarm.

9.101.1 Detailed Description

ST Driver code.

9.102 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 stIsAlarmActive (void)
  
  Determines if the alarm zero is active.

9.102.1 Detailed Description

ST Driver macros and structures.

This header is designed to be include-able without having to include other files from the HAL.

9.103 hal_st_lld.c File Reference

PLATFORM ST subsystem low level driver source.

#include "hal.h"

Functions

- void st_lld_init (void)
  
  _Low level ST driver initialization._

9.103.1 Detailed Description

PLATFORM ST subsystem low level driver source.

9.104 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.

9.104.1 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.

9.105 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.

9.105.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

9.106 hal_trng.c File Reference

TRNG Driver code.

#include "hal.h"
Functions

- void trngInit (void)
  TRNG Driver initialization.
- void trngoObjectInit (TRNGDriver *trngp)
  Initializes the standard part of a TRNGDriver structure.
- void 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.

9.106.1 Detailed Description

TRNG Driver code.

9.107 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.
- void 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.

9.107.1 Detailed Description

TRNG Driver macros and structures.

9.108 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.

9.108.1 Detailed Description

PLATFORM TRNG subsystem low level driver source.
9.109  **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.

9.109.1  **Detailed Description**

PLATFORM TRNG subsystem low level driver header.

9.110  **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.

- **void 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 uartStartReceive1 (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 uartStopReceive1 (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.

## 9.110.1 Detailed Description

UART Driver code.

## 9.111 hal_uart.h File Reference

UART Driver macros and structures.

```c
#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.
- void 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.

9.111.1 Detailed Description

UART Driver macros and structures.
9.112  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.

9.112.1  Detailed Description

PLATFORM UART subsystem low level driver source.

9.113  hal_uart_lld.h File Reference

PLATFORM UART subsystem low level driver header.

Data Structures

- struct UARTConfig
  
  Driver configuration structure.

- struct UARTDriver
  
  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 UARTDriver 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.

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.

9.113.1 Detailed Description

PLATFORM UART subsystem low level driver header.

9.114 hal_usb.c File Reference

USB Driver code.

```c
#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.
- void 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.

9.114.1 Detailed Description

USB Driver code.
9.115  hal_usb.h File Reference

USB Driver macros and structures.

#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
- #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, 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
• \#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) \((\text{usbp})->\text{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) \(((\text{usbp})->\text{transmitting} \& (\text{unsigned})1U \ll (\text{unsigned})(\text{ep}))) != 0U)
  
  Returns the status of an IN endpoint.
• \#define usbGetReceiveStatusI(usbp, ep) \(((\text{usbp})->\text{receiving} \& (\text{unsigned})1U \ll (\text{unsigned})(\text{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 usbepstatus_t {  
  EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE = 2 }

  Type of an endpoint status.

- enum usbep0state_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.

Functions

- void usbInit (void)  
  USB Driver initialization.

- void usbdInit (USBDriver *usbp)  
  Initializes the standard part of a USBDriver structure.

- void usbStart (USBDriver *usbp, const USBConfig *config)  
  Configures and activates the USB peripheral.

- void usbStop (USBDriver *usbp)  
  Deactivates the USB peripheral.

- void usbInitEndpointl (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.

Variables
• const typedef USBDescriptor *(*usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)
  Type of an USB descriptor-retrieving callback.

9.115.1 Detailed Description

USB Driver macros and structures.

9.116 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

9.116.1 Detailed Description

USB CDC macros and structures.

9.117 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.
9.117.1 Detailed Description

PLATFORM USB subsystem low level driver source.

9.117.2 Variable Documentation

9.117.2.1 in

USBInEndpointState in

IN EP0 state.

9.117.2.2 out

USBOutEndpointState out

OUT EP0 state.

9.118 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_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_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.

9.118.1 Detailed Description

PLATFORM USB subsystem low level driver header.

9.119 hal_wdg.c File Reference

WDG Driver code.

#include "hal.h"

Functions

• void wdgInit (void)
  WDG Driver initialization.
• void 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.

9.119.1 Detailed Description

WDG Driver code.

9.120 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.
• void 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.

9.120.1 Detailed Description

WDG Driver macros and structures.

9.121 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.

9.121.1 Detailed Description

WDG Driver subsystem low level driver source template.
9.122 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

- `#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.

9.122.1 Detailed Description

WDG Driver subsystem low level driver header template.

9.123 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.

- **void 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.

### 9.123.1 Detailed Description

WSPI Driver code.

### 9.124 hal_wspi.h File Reference

WSPI Driver macros and structures.

```c
#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 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.

Enumerations

- `enum wspistate_t {
  WSPI_UNINIT = 0, WSPI_STOP = 1, WSPI_READY = 2, WSPI_SEND = 3,
  WSPI_RECEIVE = 4, WSPICOMPLETE = 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.
- `void 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.
9.125 hal_wspi_lld.c File Reference

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.

9.124.1 Detailed Description

WSPI Driver macros and structures.

9.125 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 *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.
Variables

- $\text{WSPIDriver WSPID1}$
  
  WSPID1 driver identifier.

9.125.1 Detailed Description

PLATFORM WSPI subsystem low level driver source.

9.126 hal_wspi_lld.h File Reference

PLATFORM WSPI subsystem low level driver header.

Macros

- \#define \texttt{wspi\_lld\_driver\_fields}
  
  Low level fields of the WSPI driver structure.

- \#define \texttt{wspi\_lld\_config\_fields}
  
  Low level fields of the WSPI configuration structure.

WSPI implementation capabilities

- \#define \texttt{WSPI\_SUPPORTS\_MEMMAP} TRUE

- \#define \texttt{WSPI\_DEFAULT\_CFG\_Masks} TRUE

Configuration options

- \#define \texttt{PLATFORM\_WSPI\_USE\_WSPI1} FALSE
  
  WSPID1 driver enable switch.

Functions

- void \texttt{wspi\_lld\_init} (void)
  
  Low level WSPI driver initialization.

- void \texttt{wspi\_lld\_start} (WSPIDriver *wspip)
  
  Configures and activates the WSPI peripheral.

- void \texttt{wspi\_lld\_stop} (WSPIDriver *wspip)
  
  Deactivates the WSPI peripheral.

- void \texttt{wspi\_lld\_command} (WSPIDriver *wspip, const wspi\_command\_t *cmdp)
  
  Sends a command without data phase.

- void \texttt{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 \texttt{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 \texttt{wspi\_lld\_map\_flash} (WSPIDriver *wspip, const wspi\_command\_t *cmdp, uint8\_t **addrp)
  
  Maps in memory space a WSPI flash device.

- void \texttt{wspi\_lld\_unmap\_flash} (WSPIDriver *wspip)
  
  Unmaps from memory space a WSPI flash device.
9.126.1 Detailed Description

PLATFORM WSPI subsystem low level driver header.

9.127 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

ChibiOS/HAL
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 fail-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_NICE_WAITING TRUE`  
  Delays insertions.

- `#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_CIRCULAR FALSE
  Enables circular transfers 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.
• \#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.

9.127.1 Detailed Description

HAL configuration header.

HAL configuration file, this file allows to enable or disable the various device drivers from your application. You may also use this file in order to override the device drivers default settings.

9.128 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.

9.128.1 Detailed Description

Memory streams code.

9.129 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.

9.129.1 Detailed Description

Memory streams structures and macros.

9.130 nullstreams.c File Reference

Null streams code.

#include "hal.h"
#include "nullstreams.h"
9.131.1 Detailed Description

Null streams code.

9.131 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.

9.131.1 Detailed Description

Null streams structures and macros.

9.132 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 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.

Variables

- **const char *osal_halt_msg**
  Pointer to a halt error message.
9.132.1 Detailed Description

OSAL module code.

9.133 osal.h File Reference

OSAL module header.

`#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) \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.

Sleepmacros 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.

9.133.1 Detailed Description

OSAL module header.
Index

_IOBUS_DATA
  PAL Driver, 480
__CHIBIOS_HAL__
  HAL Driver, 24
__sio_callback_rx
  SIO Driver, 668
__sio_callback_rx_evt
  SIO Driver, 670
__sio_callback_rx_idle
  SIO Driver, 669
__sio_callback_tx
  SIO Driver, 669
__sio_callback_tx_end
  SIO Driver, 669
__sio_wakeup_rx
  SIO Driver, 670
__sio_wakeup_tx
  SIO Driver, 671
__sio_wakeup_txend
  SIO Driver, 671
__adc_isr_error_code
  ADC Driver, 40
__adc_isr_full_code
  ADC Driver, 39
__adc_isr_half_code
  ADC Driver, 38
__can_error_isr
  CAN Driver, 89
__can_rx_full_isr
  CAN Driver, 88
__can_tx_empty_isr
  CAN Driver, 88
__can_wakeup_isr
  CAN Driver, 89
__dac_isr_error_code
  DAC Driver, 191
__dac_isr_full_code
  DAC Driver, 190
__base_block_device_data
  Abstract I/O Block Device, 354
__base_block_data
  Abstract I/O Channel Interface, 320
__base_block_device_methods
  Abstract I/O Block Device, 354
__base_buffered_stream_data
  Abstract Streams Interface, 310
__base_buffered_stream_methods
  Abstract Streams Interface, 310
__base_channel_data
  Abstract I/O Channel Interface, 314
__base_channel_methods
  Abstract I/O Channel Interface, 314
__base_channel_methods
  Abstract I/O Channel Interface, 314
__base_channel_methods
  Abstract I/O Channel Interface, 314
__base_channel_methods
  Abstract I/O Channel Interface, 314
__base_flash_data
  Generic Flash Interface, 224
__base_flash_methods
  Generic Flash Interface, 223
__base_flash_methods_alone
  Generic Flash Interface, 223
__base_object_data
  Base Object, 468
__base_object_methods
  Base Object, 468
__base_pers_storage_methods
  Abstract Persistent Storage Interface, 331
__base_pers_storage_methods_alone
  Abstract Persistent Storage Interface, 331
__base_persistent_storage_data
  Abstract Persistent Storage Interface, 331
__base_sequential_stream_data
  Abstract Streams Interface, 307
__base_sequential_stream_methods
  Abstract Streams Interface, 307
__buffered_stream_adapter_data
  Buffered Streams Class, 339
__can_error_isr
  CAN Driver, 89
__can_rx_full_isr
  CAN Driver, 88
__can_tx_empty_isr
  CAN Driver, 88
__can_wakeup_isr
  CAN Driver, 89
__dac_isr_error_code
  DAC Driver, 191
__dac_isr_full_code
  DAC Driver, 190
__dac_isr_half_code
  DAC Driver, 189
__dac_reset_i
  DAC Driver, 188
__dac_reset_s
  DAC Driver, 188
__dac_timeout_isr
  DAC Driver, 189
__dac_wait_s
  DAC Driver, 187
__dac_wakeup_isr
  DAC Driver, 189
__efl_driver_data
  EFL Driver, 210
_efl_flash_methods
  EFL Driver, 209

_efl_flash_methods_alone
  EFL Driver, 209

_file_stream_data
  Abstract Files Interface, 324

_file_stream_methods
  Abstract Files Interface, 323

gpt_isr_invoke_cb
  GPT Driver, 235

i2c_wakeup_error_isr
  I2C Driver, 255

i2c_wakeup_isr
  I2C Driver, 254

i2s_isr_full_code
  I2S Driver, 271

i2s_isr_half_code
  I2S Driver, 270

_icu_isr_invoke_overflow_cb
  ICU Driver, 288

_icu_isr_invoke_period_cb
  ICU Driver, 287

_icu_isr_invoke_width_cb
  ICU Driver, 287

_memory_stream_data
  Memory Streams Class, 335

mmc_driver_methods
  MMC over SPI Driver, 421

mmc(sd)_block_device_data
  MMC/SD Block Device, 448

mmc(sd)_block_device_methods
  MMC/SD Block Device, 448

mmc(sd)_get_capacity
  MMC/SD Block Device, 450

mmc(sd)_get_capacity_ext
  MMC/SD Block Device, 451

mmc(sd)_get_slice
  MMC/SD Block Device, 450

mmc(sd)_unpack_csd_mmc
  MMC/SD Block Device, 453

mmc(sd)_unpack_csd_v10
  MMC/SD Block Device, 453

mmc(sd)_unpack_csd_v20
  MMC/SD Block Device, 454

mmc(sd)_unpack_mmc_cid
  MMC/SD Block Device, 452

mmc(sd)_unpack_sdc_cid
  MMC/SD Block Device, 451

null_stream_data
  Null Streams Class, 337

pal_clear_event
  PAL Driver, 482, 483

pal_init_event
  PAL Driver, 481, 482

pal_isr_code
  PAL Driver, 484, 485

pal_lld_init
  PAL Driver, 522

_pal_lld_setgroupmode
  PAL Driver, 522

_rtc_driver_methods
  RTC Driver, 581

_sdc_driver_methods
  SDC Driver, 599

_sdc_wait_for_transfer_state
  SDC Driver, 608

_serial_driver_data
  Serial Driver, 635

_serial_driver_methods
  Serial Driver, 630

_serial_usb_driver_data
  Serial over USB Driver, 648

_serial_usb_driver_methods
  Serial over USB Driver, 648

_sio_driver_methods
  SIO Driver, 663

_snor_flash_methods
  Serial NOR Flash Driver, 459

_snor_flash_methods_alone
  Serial NOR Flash Driver, 458

_spi_isr_code
  SPI Driver, 700

_spi_isr_full_code
  SPI Driver, 701

_spi_isr_half_code
  SPI Driver, 700

_spi_wakeup_isr
  SPI Driver, 699

_uart_rx_char_match_isr_code
  UART Driver, 760

_uart_rx_complete_isr_code
  UART Driver, 757

_uart_rx_error_isr_code
  UART Driver, 758

_uart_rx_idle_code
  UART Driver, 758

_uart_timeout_isr_code
  UART Driver, 759

_uart_tx1_isr_code
  UART Driver, 756

_uart_tx2_isr_code
  UART Driver, 756

_uart_wakeup_rx_cm_isr
  UART Driver, 756

_uart_wakeup_rx_complete_isr
  UART Driver, 756

_uart_wakeup_rx_cm_isr
  UART Driver, 755

_uart_wakeup_rx_complete_isr
  UART Driver, 754

_uart_wakeup_rx_error_isr
  UART Driver, 754

_uart_wakeup_rx_timeout_isr
  UART Driver, 755

_uart_wakeup_tx1_isr
  UART Driver, 753

_uart_wakeup_tx2_isr
  UART Driver, 753

_usb_ep0in
  USB Driver, 821

ChibiOS/HAL
INDEX

_index 1281
_usb_ep0out
  USB Driver, 821
_usb_ep0setup
  USB Driver, 820
_usb_isr_invoke_event_cb
  USB Driver, 799
_usb_isr_invoke_in_cb
  USB Driver, 800
_usb_isr_invoke_out_cb
  USB Driver, 801
_usb_isr_invoke_setup_cb
  USB Driver, 800
_usb_isr_invoke_sof_cb
  USB Driver, 799
_usb_reset
  USB Driver, 818
_usb_suspend
  USB Driver, 819
_usb_wakeup
  USB Driver, 820
wsspi_error_code
  WSPI Driver, 849
_wspi_isr_code
  WSPI Driver, 849
_wspi_wakeup_isr
  WSPI Driver, 848

Abstract Files Interface, 322
  _file_stream_data, 324
  _file_stream_methods, 323
FILE_EOF, 323
FILE_ERROR, 323
FILE_OK, 323
fileoffset_t, 329
fileStreamClose, 326
fileStreamGet, 326
fileStreamGetError, 327
fileStreamGetPosition, 328
fileStreamGetSize, 328
fileStreamPut, 325
fileStreamRead, 325
fileStreamGetPosition, 329
fileStreamWrite, 324

Abstract I/O Block Device, 353
  _base_block_device_data, 354
  _base_block_device_methods, 354
BLK_ACTIVE, 361
BLK_CONNECTING, 361
BLK_DISCONNECTING, 361
BLK_READING, 361
BLK_READY, 361
BLK_STOP, 361
BLK_SYNCHING, 361
BLK_UNINIT, 361
BLK_WRITING, 361
blkConnect, 357
blkDisconnect, 358
blkGetDriverState, 355
blkGetInfo, 360
  blkIsInserted, 356
  blkIsTransferring, 355
  blkIsWriteProtected, 356
  blkRead, 358
  blkstate_t, 361
  blkSync, 359
  blkWrite, 359

Abstract I/O Channel Interface, 312
  _base_asynchronous_channel_data, 320
  _base_asynchronous_channel_methods, 320
  _base_channel_data, 314
  _base_channel_methods, 314
CHN_CONNECTED, 319
CHN_CTL_INVALID, 313
CHN_CTL_NOP, 313
CHN_CTL_TX_WAIT, 314
CHN_DISCONNECTED, 319
CHN_INPUT_AVAILABLE, 319
CHN_NO_ERROR, 319
CHN_OUTPUT_EMPTY, 319
CHN_TRANSMISSION_END, 320
chnAddFlagsI, 321
chnControl, 318
chnGetEventSource, 320
chnGetTimeout, 315
chnPutTimeout, 314
chnRead, 317
chnReadTimeout, 317
chnWrite, 316
chnWriteTimeout, 316

Abstract Persistent Storage Interface, 330
  _base_pers_storage_methods_alone, 331
  _base_pers_storage_methods, 331
getBasePersistentStorage, 331
ps_error_t, 333
ps_offset_t, 333
psGetStorageSize, 331
psRead, 332
psWrite, 332

Abstract Streams Interface, 306
  _base_buffered_stream_data, 310
  _base_buffered_stream_methods, 310
  _base_sequential_stream_data, 307
  _base_sequential_stream_methods, 307
streamGet, 309
streamPut, 308
streamRead, 308
streamUnget, 310
streamWrite, 307

ADC Driver, 32
  _adc_isr_error_code, 40
  _adc_isr_full_code, 39
  _adc_isr_half_code, 38
  _adc_reset_i, 37
  _adc_reset_s, 37
  _adc_timeout_isr, 38
  _adc_wakeup_isr, 37

ChibiOS/HAL
ADC_ACTIVE, 44
adc_channels_num_t, 43
ADC_COMPLETE, 44
ADC_ERR_AWD, 41
ADC_ERR_DMAFAILURE, 41
ADC_ERR_OVERFLOW, 41
ADC_ERROR, 44
adc_lld_config_fields, 42
adc_lld_configuration_group_fields, 42
adc_lld_driver_fields, 41
adc_lld_init, 53
adc_lld_start, 53
adc_lld_start_conversion, 54
adc_lld_stop, 53
adc_lld_stop_conversion, 54
ADC_READY, 44
ADC_STOP, 44
ADC_UNINIT, 44
ADC_USE_MUTUAL_EXCLUSION, 36
ADC_USE_WAIT, 36
adcAcquireBus, 51
adccallback_t, 43
ADCConfig, 42
ADCConversionGroup, 42
adcConvert, 50
ADCD1, 54
ADCDriver, 42
adcerror_t, 44
adcerrorcallback_t, 43
adcInit, 44
adcsBufferComplete, 36
adcObjectInit, 45
adcReleaseBus, 52
adcsample_t, 43
adcStart, 45
adcStartConversion, 47
adcStartConversionl, 48
adcstate_t, 44
adcStop, 46
adcStopConversion, 49
adcStopConversionl, 50
PLATFORM_ADC_USE_ADC1, 41
ADC_ACTIVE
ADC Driver, 44
adc_channels_num_t
ADC Driver, 43
ADC_COMPLETE
ADC Driver, 44
ADC_ERR_AWD
ADC Driver, 41
ADC_ERR_DMAFAILURE
ADC Driver, 41
ADC_ERR_OVERFLOW
ADC Driver, 41
ADC_ERROR
ADC Driver, 44
adc_lld_config_fields
ADC Driver, 42
adc_lld_configuration_group_fields
ADC Driver, 42
adc_lld_driver_fields
ADC Driver, 41
adc_lld_init
ADC Driver, 53
adc_lld_start
ADC Driver, 53
adc_lld_start_conversion
ADC Driver, 54
adc_lld_stop
ADC Driver, 53
adc_lld_stop_conversion
ADC Driver, 54
ADC_READY
ADC Driver, 44
ADC_STOP
ADC Driver, 44
ADC_UNINIT
ADC Driver, 44
ADC_USE_MUTUAL_EXCLUSION
ADC Driver, 36
Configuration, 878
ADC_USE_WAIT
ADC Driver, 36
Configuration, 878
adcAcquireBus
ADC Driver, 51
adccallback_t
ADC Driver, 43
ADCConfig
ADC Driver, 42
ADCConversionGroup
ADC Driver, 42
adcConvert
ADC Driver, 50
ADCD1
ADC Driver, 54
ADCDriver
ADC Driver, 42
adcerror_t
ADC Driver, 44
adcerrorcallback_t
ADC Driver, 43
adcInit
ADC Driver, 44
adcsBufferComplete
ADC Driver, 36
adcObjectInit
ADC Driver, 45
adcReleaseBus
ADC Driver, 52
adcsample_t
ADC Driver, 43
adcStart
ADC Driver, 45
adcStartConversion
ADC Driver, 47
INDEX

BUFFERS_CHUNKS_SIZE
I/O Buffers Queues, 58
BUFFSTREAM_BUFFER_SIZE
Buffered Streams Class, 339
bufstreams.c, 1113
bufstreams.h, 1113
bulk_in
SerialUSBConfig, 1076
bulk_out
SerialUSBConfig, 1076
bus_acquire
Serial NOR Flash Driver, 459
bus_cmd
Serial NOR Flash Driver, 460
bus_cmd_addr
Serial NOR Flash Driver, 462
bus_cmd_addr_dummy_receive
Serial NOR Flash Driver, 464
bus_cmd_addr_receive
Serial NOR Flash Driver, 463
bus_cmd_addr_send
Serial NOR Flash Driver, 462
bus_cmd_dummy_receive
Serial NOR Flash Driver, 463
bus_cmd_receive
Serial NOR Flash Driver, 461
bus_cmd_send
Serial NOR Flash Driver, 461
bus_release
Serial NOR Flash Driver, 460
bus_stop
Serial NOR Flash Driver, 460
bus_width
SDCConfig, 1065
bwrptr
ioBuffers_queue, 1015
callback
GPTConfig, 975
PWMChannelConfig, 1054
PWMConfig, 1056
CAN Driver, 83
__can_error_isr, 89
__can_rx_full_isr, 88
__can_tx_empty_isr, 88
__can_wakeup_isr, 89
CAN_ANY_MAILBOX, 87
CAN_BUS_OFF_ERROR, 86
can_callback_t, 90
CAN_ENFORCE_USE_CALLBACKS, 87
CAN_FRAMING_ERROR, 87
CAN_LIMIT_ERROR, 87
CAN_LIMIT_WARNING, 87
can_abort, 105
can_init, 101
can_is_rx_nonempty, 104
can_is_tx_empty, 103
can_receive, 105
can_sleep, 105
can_start, 101
can_stop, 103
can_transmit, 104
can_wakeup, 106
CAN_MAILBOX_TO_MASK, 87
CAN_OVERFLOW_ERROR, 87
CAN_READY, 91
CAN_RX_MAILBOXES, 89
CAN_SLEEP, 91
CAN_STARTING, 91
CAN_STOP, 91
CAN_STOPPING, 91
CAN_TX_MAILBOXES, 89
CAN_UNINIT, 91
CAN_USE_SLEEP_MODE, 87
CAND1, 106
CANDriver, 90
canInit, 91
canmbx_t, 90
canObjectInit, 92
canReceive, 88
canReceiveTimeout, 98
canSleep, 99
canStart, 92
canstate_t, 91
canStop, 93
canTransmit, 88
canTransmitTimeout, 96
canTryAbortX, 96
canTryReceiveI, 95
canTryTransmitI, 94
canWakeup, 100
PLATFORM_CAN_USE_CAN1, 90
CAN_ANY_MAILBOX
CAN Driver, 87
CAN_BUS_OFF_ERROR
CAN Driver, 86
can_callback_t
CAN Driver, 90
CAN_ENFORCE_USE_CALLBACKS
CAN Driver, 87
Configuration, 878
CAN_FRAMING_ERROR
CAN Driver, 87
CAN_LIMIT_ERROR
CAN Driver, 86
CAN_LIMIT_WARNING
CAN Driver, 86
can_abort
CAN Driver, 105
can_init
CAN Driver, 101
can_is_rx_nonempty
CAN Driver, 104
can_is_tx_empty
CAN Driver, 103
can_receive
CAN Driver, 105
ChibiOS/HAL
can_lld_sleep
  CAN Driver, 105
can_lld_start
  CAN Driver, 101
can_lld_stop
  CAN Driver, 103
can_lld_transmit
  CAN Driver, 104
can_lld_wakeup
  CAN Driver, 106
CAN_MAILBOX_TO_MASK
  CAN Driver, 87
CAN_OVERFLOW_ERROR
  CAN Driver, 87
CAN_READY
  CAN Driver, 91
CAN_RX_MAILBOXES
  CAN Driver, 89
CAN_SLEEP
  CAN Driver, 91
CAN_STARTING
  CAN Driver, 91
CAN_STOP
  CAN Driver, 91
CAN_STOPPING
  CAN Driver, 91
CAN_TX_MAILBOXES
  CAN Driver, 89
CAN_UNINIT
  CAN Driver, 91
CAN_USE_SLEEP_MODE
  CAN Driver, 91
  Configuration, 878
CANConfig, 953
CAND1
  CAN Driver, 106
CANDriver, 954
  CAN Driver, 90
canInit
  CAN Driver, 91
  config, 955
error_event, 956
txfull_event, 955
rxqueue, 955
sleep_event, 956
state, 955
txempty_event, 956
txqueue, 955
wakeup_event, 956
canInit
  CAN Driver, 91
canmbx_t
  CAN Driver, 90
canObjectInit
  CAN Driver, 92
canReceive
  CAN Driver, 88
canReceiveTimeout
  CAN Driver, 98
CANRxFrame, 957
data16, 959
data32, 960
data8, 959
DLC, 958
EID, 959
FMI, 958
IDE, 959
RTR, 959
SID, 959
TIME, 958
canSleep
  CAN Driver, 99
canStart
  CAN Driver, 92
canstate_t
  CAN Driver, 91
canStop
  CAN Driver, 93
canTransmit
  CAN Driver, 88
canTransmitTimeout
  CAN Driver, 96
canTryAbortX
  CAN Driver, 96
canTryReceiveI
  CAN Driver, 95
canTryTransmitI
  CAN Driver, 94
CANTxFrame, 960
data16, 962
data32, 962
data8, 962
DLC, 961
EID, 962
IDE, 962
RTR, 961
SID, 962
canWakeup
  CAN Driver, 100
cardmode
  SDCDriver, 1067
cb
  event_source, 967
palevent_t, 1052
cdc_linecoding_t, 963
cfg
  wspi_command_t, 1112
CH_HAL_MAJOR
  HAL Driver, 25
CH_HAL_MINOR
  HAL Driver, 25
CH_HAL_PATCH
  HAL Driver, 25
CH_HAL_STABLE
  HAL Driver, 24
channels
  PWMConfig, 1056
PWMDriver, 1058
CHN_CONNECTED
Abstract I/O Channel Interface, 319

CHN_CTL_INVALID
Abstract I/O Channel Interface, 313

CHN_CTL_NOP
Abstract I/O Channel Interface, 313

CHN_CTL_TX_WAIT
Abstract I/O Channel Interface, 314

CHN_DISCONNECTED
Abstract I/O Channel Interface, 319

CHN_INPUT_AVAILABLE
Abstract I/O Channel Interface, 319

CHN_NO_ERROR
Abstract I/O Channel Interface, 319

CHN_OUTPUT_EMPTY
Abstract I/O Channel Interface, 319

CHN_TRANSMISSION_END
Abstract I/O Channel Interface, 320

chnAddFlagsI
Abstract I/O Channel Interface, 321

chnControl
Abstract I/O Channel Interface, 318

chnGetEventSource
Abstract I/O Channel Interface, 320

chnGetTimeout
Abstract I/O Channel Interface, 315

chnPutTimeout
Abstract I/O Channel Interface, 314

chnRead
Abstract I/O Channel Interface, 317

chnReadTimeout
Abstract I/O Channel Interface, 317

chnWrite
Abstract I/O Channel Interface, 316

chnWriteTimeout
Abstract I/O Channel Interface, 316

chprintf
Output Formatter Utility, 342

chprintf.c, 1114
chprintf.h, 1114

CHPRINTF_USE_FLOAT
Output Formatter Utility, 341

chscanf
Input Formatter Utility, 348

chscanf.c, 1115
chscanf.h, 1116

CHSCANSF_USE_FLOAT
Input Formatter Utility, 347

chssprintf
Output Formatter Utility, 343

chssscanf
Input Formatter Utility, 349

chvprintf
Output Formatter Utility, 341

chvscanf
Input Formatter Utility, 347

chvssprintf
Output Formatter Utility, 345
HAL_USE_TRNG, 876
HAL_USE_UART, 876
HAL_USE_USB, 877
HAL_USE_WDG, 877
HAL_USE_WSPI, 877
I2C_USE_MUTUAL_Exclusion, 879
MAC_USE_EVENTS, 880
MAC_USE_ZERO_COPY, 880
MMC_NICE_WAITING, 880
PAL_USE_CALLBACKS, 877
PAL_USE_WAIT, 877
SDC_INIT_OCR, 881
SDC_INIT_OCR_V20, 881
SDC_INIT_RETRY, 880
SDC_MMC_SUPPORT, 880
SDC_NICE_WAITING, 881
SERIAL_BUFFERS_SIZE, 881
SERIAL_DEFAULT_BITRATE, 881
SERIAL_USB_BUFFERS_NUMBER, 882
SIO_DEFAULT_BITRATE, 882
SIO_USE_SYNCHRONIZATION, 882
SPI_SELECT_MODE, 883
SPI_USE_CIRCULAR, 883
SPI_USE_MUTUAL_Exclusion, 883
SPI_USE_WAIT, 883
UART_USE_MUTUAL_Exclusion, 884
UART_USE_WAIT, 884
USB_USE_WAIT, 884
WSPI_USE_MUTUAL_Exclusion, 885
WSPI_USE_WAIT, 884
configuration
USBDriver, 1102
counter
mfs_bank_header_t, 1030
crc
mfs_bank_header_t, 1030
mfs_data_header_t, 1032
crc7
MMC over SPI Driver, 422
crc7_lookup_table
MMC over SPI Driver, 442
cry_algo_aes
Crypto Driver, 115
cry_algo_des
Crypto Driver, 115
cry_algo_hmac
Crypto Driver, 115
CRY_ERR_AUTH_FAILED
Crypto Driver, 115
CRY_ERR_INV_ALGO
Crypto Driver, 114
CRY_ERR_INV_KEY_ID
Crypto Driver, 115
CRY_ERR_INV_KEY_SIZE
Crypto Driver, 114
CRY_ERR_INV_KEY_TYPE
Crypto Driver, 115
CRY_ERR_OP_FAILURE
Crypto Driver, 115
cry_lld_aes_loadkey
Crypto Driver, 155
cry_lld_decrypt_AES
Crypto Driver, 157
cry_lld_decrypt_AES_CBC
Crypto Driver, 160
cry_lld_decrypt_AES_CFB
Crypto Driver, 162
cry_lld_decrypt_AES_CTR
Crypto Driver, 163
cry_lld_decrypt_AES_ECB
Crypto Driver, 158
cry_lld_decrypt_AES_GCM
Crypto Driver, 165
cry_lld_decrypt_DES
Crypto Driver, 168
cry_lld_decrypt_DES_CBC
Crypto Driver, 171
cry_lld_decrypt_DES_ECB
Crypto Driver, 170
cry_lld_des_loadkey
Crypto Driver, 166
cry_lld_encrypt_AES
Crypto Driver, 156
cry_lld_encrypt_AES_CBC
Crypto Driver, 159
cry_lld_encrypt_AES_CFB
Crypto Driver, 161
cry_lld_encrypt_AES_CTR
Crypto Driver, 163
cry_lld_encrypt_AES_GCM
Crypto Driver, 157
cry_lld_encrypt_AES_CBC
Crypto Driver, 164
cry_lld_encrypt_DES
Crypto Driver, 167
cry_lld_encrypt_DES_CBC
Crypto Driver, 171
cry_lld_encrypt_DES_ECB
Crypto Driver, 169
cry_lld_hmac_loadkey
Crypto Driver, 178
cry_lld_HMACSHA256_final
Crypto Driver, 180
cry_lld_HMACSHA256_init
Crypto Driver, 179
cry_lld_HMACSHA256_update
Crypto Driver, 179
cry_lld_HMACSHA512_final
Crypto Driver, 182
cry_lld_HMACSHA512_init
Crypto Driver, 181
cry_lld_HMACSHA512_update
Crypto Driver, 181
cry_lld_init
Crypto Driver, 154
DAC Driver, 184
    _dac_isr_error_code, 191
    _dac_isr_full_code, 190
    _dac_isr_half_code, 189
    _dac_reset_i, 188
    _dac_reset_s, 188
    _dac_timeout_isr, 189
    _dac_wait_s, 187
    _dac_wakeup_isr, 189
DAC_ACTIVE, 194
DAC_COMPLETE, 194
DAC_ERR_DMAFAILURE, 195
DAC_ERR_UNDERFLOW, 195
DAC_ERROR, 194
dac_lld_config_fields, 192
dac_lld_conversion_group_fields, 192
dac_lld_driver_fields, 192
dac_lld_init, 204
dac_lld_put_channel, 206
dac_lld_start, 205
dac_lld_start_conversion, 206
dac_lld_stop, 205
dac_lld_stop_conversion, 207
DAC_MAX_CHANNELS, 191
DAC_READY, 194
DAC_STOP, 194
DAC_UNINIT, 194
DAC_USE_MUTUAL_EXCLUSION, 186
DAC_USE_WAIT, 186
dacAcquireBus, 203
daccallback_t, 193
dacchannel_t, 194
DACConfig, 193
DACConversionGroup, 193
dacConvert, 202
DACD1, 207
DACDriver, 193
dacerror_t, 194
dacerrorcallback_t, 193
daclnt, 195
daclBufferComplete, 187
daclObjectInit, 195
daclPutChannelX, 198
daclReleaseBus, 204
daclsample_t, 194
daclStart, 196
daclStartConversion, 199
daclStartConversionL, 199
daclstate_t, 194
daclStop, 197
daclStopConversion, 200
daclStopConversionL, 201
PLATFORM_DAC_USE_DAC1, 192
DAC_ACTIVE
    DAC Driver, 194
DAC_COMPLETE
    DAC Driver, 194
DAC_ERR_DMAFAILURE
    DAC Driver, 195
DAC_ERR_UNDERFLOW
    DAC Driver, 195
DAC_ERROR
    DAC Driver, 194
dac_lld_config_fields
    DAC Driver, 192
dac_lld_conversion_group_fields
    DAC Driver, 192
dac_lld_driver_fields
    DAC Driver, 192
dac_lld_init
    DAC Driver, 204
dac_lld_put_channel
    DAC Driver, 206
dac_lld_start
    DAC Driver, 205
dac_lld_start_conversion
    DAC Driver, 206
dac_lld_stop
    DAC Driver, 205
dac_lld_stop_conversion
    DAC Driver, 207
DAC_MAX_CHANNELS
    DAC Driver, 191
DAC_READY
    DAC Driver, 194
DAC_STOP
    DAC Driver, 194
DAC_UNINIT
    DAC Driver, 194
DAC_USE_MUTUAL_EXCLUSION
    Configuration, 879
    DAC Driver, 186
DAC_USE_WAIT
    Configuration, 879
    DAC Driver, 186
dacAcquireBus
    DAC Driver, 203
daclcallback_t
    DAC Driver, 193
daclchannel_t
    DAC Driver, 194
DACConfig
    DAC Driver, 193
DACConversionGroup
    DAC Driver, 193
daclConvert
    DAC Driver, 202
DACD1
    DAC Driver, 207
DACDriver
    DAC Driver, 193
daclerror_t
    DAC Driver, 194
daclerrorcallback_t
    DAC Driver, 193
daclnt
DAC Driver, 195
daclIsBufferComplete
DAC Driver, 187
daclObjectInit
DAC Driver, 195
daclPutChannelX
DAC Driver, 198
daclReleaseBus
DAC Driver, 204
daclsample_t
DAC Driver, 194
daclStart
DAC Driver, 196
daclStartConversion
DAC Driver, 199
daclStartConversionI
DAC Driver, 199
daclstate_t
DAC Driver, 194
daclStop
DAC Driver, 197
daclStopConversion
DAC Driver, 200
daclStopConversionI
DAC Driver, 201
data16
CANRxFrame, 959
CANTxFrame, 962
data32
CANRxFrame, 960
CANTxFrame, 962
data8
CANRxFrame, 959
CANTxFrame, 962
day
RTCDateTime, 1061
dayofweek
RTCDateTime, 1061
default_config
Serial Driver, 645
default_handler
USB Driver, 808
depth
hal_adc_driver, 981
hal_dac_driver, 986
descriptors
MFSDriver, 1038
detect_bus_clk
SDC Driver, 606
device_id
SNORDriver, 1086
DLC
CANRxFrame, 958
CANTxFrame, 961
dstflag
RTCDateTime, 1060
dummy
wspi_command_t, 1112
early
UARTDriver, 1091
EFL Driver, 208
_efl_driver_data, 210
_efl_flash_methods, 209
_efl_flash_methods_alone, 209
efl_lld_config_fields, 210
efl_lld_driver_fields, 210
efl_lld_get_descriptor, 214
efl_lld_init, 213
efl_lld_program, 215
efl_lld_query_erase, 217
efl_lld_read, 215
efl_lld_start, 213
efl_lld_start_erase_all, 216
efl_lld_start_erase_sector, 217
efl_lld_stop, 214
efl_lld_verify_erase, 218
EFlashConfig, 211
EFlashDriver, 211
EFLD1, 218
eflInit, 211
eflObjectInit, 211
eflStart, 212
eflStop, 212
PLATFORM_EFL_USE_EFL1, 210
efl_lld_config_fields
EFL Driver, 210
efl_lld_driver_fields
EFL Driver, 210
efl_lld_get_descriptor
EFL Driver, 214
efl_lld_init
EFL Driver, 213
efl_lld_program
EFL Driver, 215
efl_lld_query_erase
EFL Driver, 217
efl_lld_read
EFL Driver, 215
efl_lld_start
EFL Driver, 213
efl_lld_start_erase_all
EFL Driver, 216
efl_lld_start_erase_sector
EFL Driver, 217
efl_lld_stop
EFL Driver, 214
efl_lld_verify_erase
EFL Driver, 218
EFlashConfig
EFL Driver, 211
EFlashDriver
EFL Driver, 211
EFlashDriverVMT, 965
EFLD1
EFL Driver, 218
eflInit

ChibiOS/HAL
EFL Driver, 211
eflObjectInit
EFL Driver, 211
eflStart
EFL Driver, 212
eflStop
EFL Driver, 212
EID
CANRxFrame, 959
CANTxFrame, 962
enabled
PWMDriver, 1058
end_cb
hal_adc_configuration_group, 979
hal_dac_conversion_group, 984
hal_i2s_config, 991
hal_spi_config, 1000
hal_vspl_config, 1005
ep0_state
USB Driver, 830
ep0config
USB Driver, 831
ep0endcb
USBDriver, 1102
ep0
USBDriver, 1102
ep0next
USBDriver, 1101
ep0state
USBDriver, 1101
ep_mode
USBEndpointConfig, 1105
EP_STATUS_ACTIVE
USB Driver, 806
EP_STATUS_DISABLED
USB Driver, 806
EP_STATUS_STALLED
USB Driver, 806
epc
USBDriver, 1101
erased
MFSConfig, 1035
error_cb
hal_adc_configuration_group, 979
hal_dac_conversion_group, 984
hal_vspl_config, 1005
error_event
CANDriver, 956
errors
i2CDriver, 1010
SDCDriver, 1067
ETHD1
MAC Driver, 381
event_cb
USBConfig, 1097
event_source, 966
cb, 967
flags, 967
param, 967
event_source_t
OSAL, 909
eventcallback_t
OSAL, 909
eventflags_t
OSAL, 909
EXPANSION_ENABLENPAGE
MII/RMII Header, 417
EXPANSION_LCW
MII/RMII Header, 417
EXPANSION_MFAULTS
MII/RMII Header, 417
EXPANSION_NPCAPABLE
MII/RMII Header, 417
EXPANSION_NWAY
MII/RMII Header, 416
EXPANSION_RESV
MII/RMII Header, 417
FILE_EOF
Abstract Files Interface, 323
FILE_ERROR
Abstract Files Interface, 323
FILE_OK
Abstract Files Interface, 323
fileoffset_t
Abstract Files Interface, 329
FileStream, 968
vmt, 969
fileStreamClose
Abstract Files Interface, 326
fileStreamGet
Abstract Files Interface, 326
fileStreamGetError
Abstract Files Interface, 327
fileStreamGetPosition
Abstract Files Interface, 328
fileStreamGetSize
Abstract Files Interface, 328
fileStreamPut
Abstract Files Interface, 325
fileStreamRead
Abstract Files Interface, 325
fileStreamGetPosition
Abstract Files Interface, 329
FileStreamVMT, 970
fileStreamWrite
Abstract Files Interface, 324
flags
event_source, 967
FLASH_ATTR_ECC_CAPABLE
Generic Flash Interface, 222
FLASH_ATTR_ECC_ZERO_LINE_CAPABLE
Generic Flash Interface, 223
FLASH_ATTR_ERASED_IS_ONE
Generic Flash Interface, 222
FLASH_ATTR_MEMORY_MAPPED
Generic Flash Interface, 222
FLASH_ATTR_REWRITABLE
   Generic Flash Interface, 222
FLASH_ATTR_SUSPEND_ERASE_CAPABLE
   Generic Flash Interface, 223
flash_descriptor_t, 971
   address, 972
   attributes, 972
   page_size, 972
   sectors, 972
   sectors_count, 972
   size, 972
flash_error_t
   Generic Flash Interface, 229
flash_offset_t
   Generic Flash Interface, 228
flash_sector_descriptor_t, 973
   offset, 974
   size, 974
flash_sector_t
   Generic Flash Interface, 229
flash_state_t
   Generic Flash Interface, 229
flashGetDescriptor
   Generic Flash Interface, 224
flashGetSectorOffset
   Generic Flash Interface, 230
flashGetSectorSize
   Generic Flash Interface, 230
flashp
   MFSConfig, 1035
flashProgram
   Generic Flash Interface, 225
flashQueryErase
   Generic Flash Interface, 227
flashRead
   Generic Flash Interface, 225
flashStartEraseAll
   Generic Flash Interface, 226
flashStartEraseSector
   Generic Flash Interface, 227
flashVerifyErase
   Generic Flash Interface, 228
flashWaitErase
   Generic Flash Interface, 229
FMI
   CANRxFrame, 958
frequency
   GPTConfig, 975
   ICUConfig, 1011
   PWMConfig, 1056
Generic Flash Interface, 220
   _base_flash_data, 224
   _base_flash_methods, 223
   _base_flash_methods_alone, 223
FLASH_ATTR_ECC_CAPABLE, 222
FLASH_ATTR_ECC_ZERO_LINE_CAPABLE, 223
FLASH_ATTR_ERASED_IS_ONE, 222
FLASH_ATTR_MEMORY_MAPPED, 222
FLASH_ATTR_REWRITABLE, 222
FLASH_ATTR_SUSPEND_ERASE_CAPABLE, 223
flash_error_t, 229
flash_offset_t, 228
flash_sector_t, 229
flash_state_t, 229
flashGetDescriptor, 224
flashGetSectorOffset, 230
flashGetSectorSize, 230
flashProgram, 225
flashQueryErase, 227
flashRead, 225
flashStartEraseAll, 226
flashStartEraseSector, 227
flashVerifyErase, 228
flashWaitErase, 229
getBaseFlash, 224
get_descriptor_cb
   USBConfig, 1097
getBaseFlash
   Generic Flash Interface, 224
getBasePersistentStorage
   Abstract Persistent Storage Interface, 331
GPT Driver, 231
   _gpt_isr_invoke_cb, 235
   GPT_CONTINUOUS, 238
   gpt_lld_change_interval, 236
   gpt_lld_init, 247
   gpt_lld_polled_delay, 248
   gpt_lld_start, 247
   gpt_lld_start_timer, 248
   gpt_lld_stop, 247
   gpt_lld_stop_timer, 248
   GPT_ONESHOT, 238
   GPT_READY, 238
   GPT_STOP, 238
   GPT_UNINIT, 238
gptcallback_t, 237
gptChangeInterval, 241
gptChangeIntervalV, 234
gptcnt_t, 238
GPTD1, 249
GPTDriver, 237
gptfreq_t, 237
gptGetCounterX, 235
gptGetIntervalX, 234
gptInit, 238
gptObjectInit, 239
gptPolledDelay, 246
   gptStart, 239
   gptStartContinuous, 242
   gptStartContinuousV, 242
   gptStartOneShot, 243
gptStartOneShotV, 244
gptstate_t, 238
gptStop, 240
<table>
<thead>
<tr>
<th>Function/Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gptStartContinuous</code></td>
<td>GPT Driver, 242</td>
</tr>
<tr>
<td><code>gptStartContinuousI</code></td>
<td>GPT Driver, 242</td>
</tr>
<tr>
<td><code>gptStartOneShot</code></td>
<td>GPT Driver, 243</td>
</tr>
<tr>
<td><code>gptStartOneShotI</code></td>
<td>GPT Driver, 244</td>
</tr>
<tr>
<td><code>gptstate_t</code></td>
<td>GPT Driver, 238</td>
</tr>
<tr>
<td><code>grpp</code></td>
<td>hal_adc_driver, 981</td>
</tr>
<tr>
<td></td>
<td>hal_dac_driver, 986</td>
</tr>
<tr>
<td><strong>HAL, 868</strong></td>
<td></td>
</tr>
<tr>
<td><strong>HAL Driver, 23</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CHIBIOS_HAL</strong>, 24</td>
<td></td>
</tr>
<tr>
<td>CH_HAL_MAJOR, 25</td>
<td></td>
</tr>
<tr>
<td>CH_HAL_MINOR, 25</td>
<td></td>
</tr>
<tr>
<td>CH_HAL_PATCH, 25</td>
<td></td>
</tr>
<tr>
<td>CH_HAL_STABLE, 24</td>
<td></td>
</tr>
<tr>
<td>HAL_FAILED, 25</td>
<td></td>
</tr>
<tr>
<td>hal_lld_clock_switch_mode, 30</td>
<td></td>
</tr>
<tr>
<td>hal_lld_get_clock_point, 31</td>
<td></td>
</tr>
<tr>
<td>hal_lld_init, 30</td>
<td></td>
</tr>
<tr>
<td>HAL_LLD_USE_CLOCK_MANAGEMENT, 26</td>
<td></td>
</tr>
<tr>
<td>HAL_SUCCESS, 25</td>
<td></td>
</tr>
<tr>
<td>HAL_VERSION, 25</td>
<td></td>
</tr>
<tr>
<td>halclkpt_t, 26</td>
<td></td>
</tr>
<tr>
<td>halClockGetPointX, 29</td>
<td></td>
</tr>
<tr>
<td>halClockSwitchMode, 28</td>
<td></td>
</tr>
<tr>
<td>halfreq_t, 26</td>
<td></td>
</tr>
<tr>
<td>halInit, 26</td>
<td></td>
</tr>
<tr>
<td>hal.c, 1116</td>
<td></td>
</tr>
<tr>
<td>hal.h, 1117</td>
<td></td>
</tr>
<tr>
<td>hal_adc.c, 1118</td>
<td></td>
</tr>
<tr>
<td>hal_adc.h, 1119</td>
<td></td>
</tr>
<tr>
<td>hal_adc_config, 977</td>
<td></td>
</tr>
<tr>
<td>hal_adc_configuration_group, 977</td>
<td></td>
</tr>
<tr>
<td>circular, 979</td>
<td></td>
</tr>
<tr>
<td>end_cb, 979</td>
<td></td>
</tr>
<tr>
<td>error_cb, 979</td>
<td></td>
</tr>
<tr>
<td>num_channels, 979</td>
<td></td>
</tr>
<tr>
<td>hal_adc_driver, 980</td>
<td></td>
</tr>
<tr>
<td>config, 981</td>
<td></td>
</tr>
<tr>
<td>depth, 981</td>
<td></td>
</tr>
<tr>
<td>grpp, 981</td>
<td></td>
</tr>
<tr>
<td>mutex, 982</td>
<td></td>
</tr>
<tr>
<td>samples, 981</td>
<td></td>
</tr>
<tr>
<td>state, 981</td>
<td></td>
</tr>
<tr>
<td>thread, 982</td>
<td></td>
</tr>
<tr>
<td>hal_adc_lld.c, 1121</td>
<td></td>
</tr>
<tr>
<td>hal_adc_lld.h, 1122</td>
<td></td>
</tr>
<tr>
<td>Function/Type</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>hal_wdg.h</td>
<td>1259</td>
</tr>
<tr>
<td>hal_wdg_lld.c</td>
<td>1260</td>
</tr>
<tr>
<td>hal_wdg_lld.h</td>
<td>1261</td>
</tr>
<tr>
<td>hal_wspi.c</td>
<td>1261</td>
</tr>
<tr>
<td>hal_wspi.h</td>
<td>1262</td>
</tr>
<tr>
<td>hal_wspi_config</td>
<td>1004</td>
</tr>
<tr>
<td>end_cb</td>
<td>1005</td>
</tr>
<tr>
<td>error_cb</td>
<td>1005</td>
</tr>
<tr>
<td>hal_wspi_driver</td>
<td>1005</td>
</tr>
<tr>
<td>config</td>
<td>1006</td>
</tr>
<tr>
<td>mutex</td>
<td>1006</td>
</tr>
<tr>
<td>state</td>
<td>1006</td>
</tr>
<tr>
<td>thread</td>
<td>1006</td>
</tr>
<tr>
<td>hal_wspi_lld.c</td>
<td>1265</td>
</tr>
<tr>
<td>hal_wspi_lld.h</td>
<td>1266</td>
</tr>
<tr>
<td>halclkcfg_t</td>
<td>1007</td>
</tr>
<tr>
<td>halclkpt_t</td>
<td>26</td>
</tr>
<tr>
<td>DispatchQueueX</td>
<td>26</td>
</tr>
<tr>
<td>DispatchSwitchMode</td>
<td>28</td>
</tr>
<tr>
<td>halconf.h</td>
<td>1267</td>
</tr>
<tr>
<td>halfreq_t</td>
<td>26</td>
</tr>
<tr>
<td>halInit</td>
<td>26</td>
</tr>
<tr>
<td>HAL Driver, 26</td>
<td></td>
</tr>
<tr>
<td>HMACSHA256Context, 1007</td>
<td></td>
</tr>
<tr>
<td>HMACSHA512Context, 1008</td>
<td></td>
</tr>
<tr>
<td>hscfg</td>
<td>1041</td>
</tr>
<tr>
<td>I/O Buffers Queues, 56</td>
<td></td>
</tr>
<tr>
<td>BQ_BUFFER_SIZE, 58</td>
<td></td>
</tr>
<tr>
<td>bqGetLinkX</td>
<td>59</td>
</tr>
<tr>
<td>bqIsSuspendedX</td>
<td>60</td>
</tr>
<tr>
<td>bqNotify_t</td>
<td>64</td>
</tr>
<tr>
<td>bqResumeX</td>
<td>61</td>
</tr>
<tr>
<td>bqSetLinkX</td>
<td>60</td>
</tr>
<tr>
<td>bqSizeX</td>
<td>58</td>
</tr>
<tr>
<td>bqSpaceI</td>
<td>59</td>
</tr>
<tr>
<td>bqSuspendI</td>
<td>61</td>
</tr>
<tr>
<td>BUFFERS_CHUNKS_SIZE, 58</td>
<td></td>
</tr>
<tr>
<td>ibqGetEmptyBuffer1</td>
<td>66</td>
</tr>
<tr>
<td>ibqGetFullBufferTimeout</td>
<td>68</td>
</tr>
<tr>
<td>ibqGetFullBufferTimeoutS</td>
<td>69</td>
</tr>
<tr>
<td>ibqGetTimeout</td>
<td>71</td>
</tr>
<tr>
<td>ibqIsEmpty1</td>
<td>62</td>
</tr>
<tr>
<td>ibqIsFull1</td>
<td>62</td>
</tr>
<tr>
<td>ibqObjectInit</td>
<td>65</td>
</tr>
<tr>
<td>ibqPostFullBuffer1</td>
<td>67</td>
</tr>
<tr>
<td>ibqReadTimeout</td>
<td>72</td>
</tr>
<tr>
<td>ibqReleaseEmptyBuffer1</td>
<td>70</td>
</tr>
<tr>
<td>ibqReleaseEmptyBufferS1</td>
<td>71</td>
</tr>
<tr>
<td>ibqReset1</td>
<td>66</td>
</tr>
<tr>
<td>input_buffers_queue_t</td>
<td>65</td>
</tr>
<tr>
<td>io_buffers_queue_t</td>
<td>64</td>
</tr>
<tr>
<td>obqFlush</td>
<td>82</td>
</tr>
<tr>
<td>obqGetEmptyBufferTimeout</td>
<td>76</td>
</tr>
<tr>
<td>obqGetFullBuffer1</td>
<td>74</td>
</tr>
<tr>
<td>obqIsEmpty1</td>
<td>63</td>
</tr>
<tr>
<td>obqIsFull1</td>
<td>64</td>
</tr>
<tr>
<td>obqObjectInit</td>
<td>73</td>
</tr>
<tr>
<td>obqPostFullBuffer1</td>
<td>78</td>
</tr>
<tr>
<td>obqPostFullBufferS1</td>
<td>79</td>
</tr>
<tr>
<td>obqPutTimeout</td>
<td>79</td>
</tr>
<tr>
<td>obqReleaseEmptyBuffer1</td>
<td>75</td>
</tr>
<tr>
<td>obqReset1</td>
<td>74</td>
</tr>
<tr>
<td>obqTryFlush1</td>
<td>81</td>
</tr>
<tr>
<td>obqWriteTimeout</td>
<td>80</td>
</tr>
<tr>
<td>output_buffers_queue_t</td>
<td>65</td>
</tr>
<tr>
<td>I/O Bytes Queues, 554</td>
<td></td>
</tr>
<tr>
<td>input_queue_t</td>
<td>565</td>
</tr>
<tr>
<td>io_queue_t</td>
<td>564</td>
</tr>
<tr>
<td>iq_read</td>
<td>565</td>
</tr>
<tr>
<td>iqGet</td>
<td>561</td>
</tr>
<tr>
<td>iqGetEmpty</td>
<td>559</td>
</tr>
<tr>
<td>iqGetFull</td>
<td>559</td>
</tr>
<tr>
<td>iqGetI</td>
<td>569</td>
</tr>
<tr>
<td>iqGetTimeout</td>
<td>569</td>
</tr>
<tr>
<td>iqIsEmpty</td>
<td>560</td>
</tr>
<tr>
<td>iqIsFull</td>
<td>560</td>
</tr>
<tr>
<td>iqObjectInit</td>
<td>566</td>
</tr>
<tr>
<td>iqPut</td>
<td>568</td>
</tr>
<tr>
<td>iqRead</td>
<td>570</td>
</tr>
<tr>
<td>iqReadTimeout</td>
<td>571</td>
</tr>
<tr>
<td>iqReset</td>
<td>567</td>
</tr>
<tr>
<td>oq_write</td>
<td>566</td>
</tr>
<tr>
<td>oqGetEmpty</td>
<td>562</td>
</tr>
<tr>
<td>oqGetFull</td>
<td>561</td>
</tr>
<tr>
<td>oqGetI</td>
<td>575</td>
</tr>
<tr>
<td>oqIsEmpty</td>
<td>563</td>
</tr>
<tr>
<td>oqIsFull</td>
<td>563</td>
</tr>
<tr>
<td>oqObjectInit</td>
<td>572</td>
</tr>
<tr>
<td>oqPut</td>
<td>564</td>
</tr>
<tr>
<td>oqPutI</td>
<td>574</td>
</tr>
<tr>
<td>oqPutTimeout</td>
<td>574</td>
</tr>
<tr>
<td>oqReset</td>
<td>573</td>
</tr>
<tr>
<td>oqWrite</td>
<td>576</td>
</tr>
<tr>
<td>oqWriteTimeout</td>
<td>577</td>
</tr>
<tr>
<td>output_queue_t</td>
<td>565</td>
</tr>
<tr>
<td>Q_EMPTY</td>
<td>556</td>
</tr>
<tr>
<td>Q_FULL</td>
<td>557</td>
</tr>
<tr>
<td>Q_OK</td>
<td>556</td>
</tr>
<tr>
<td>Q_RESET</td>
<td>556</td>
</tr>
<tr>
<td>Q_TIMEOUT</td>
<td>556</td>
</tr>
<tr>
<td>qGetLink</td>
<td>558</td>
</tr>
<tr>
<td>qNotify_t</td>
<td>565</td>
</tr>
<tr>
<td>qSetLink</td>
<td>558</td>
</tr>
<tr>
<td>qSizeI</td>
<td>557</td>
</tr>
<tr>
<td>qSpaceI</td>
<td>557</td>
</tr>
<tr>
<td>I2C Driver, 250</td>
<td></td>
</tr>
<tr>
<td>_i2c_wakeup_error_isr</td>
<td>255</td>
</tr>
<tr>
<td>_i2c_wakeup_isr</td>
<td>254</td>
</tr>
<tr>
<td>I2C_ACK_FAILURE, 253</td>
<td></td>
</tr>
<tr>
<td>I2C_ACTIVE_RX, 257</td>
<td></td>
</tr>
</tbody>
</table>
INDEX 1299

I2C_ACTIVE_TX, 257
I2C_ARBITRATION_LOST, 253
I2C_BUS_ERROR, 253
i2c_lld_get_errors, 256
i2c_lld_init, 264
i2c_lld_master_receive_timeout, 265
i2c_lld_master_transmit_timeout, 265
i2c_lld_start, 264
i2c_lld_stop, 264
I2C_LOCKED, 257
I2C_NO_ERROR, 253
I2C_PEC_ERROR, 254
I2C_READY, 257
I2C_SMB_ALERT, 254
I2C_STOP, 257
I2C_TIMEOUT, 254
I2C_UNINIT, 257
I2C_USE_MUTUAL_EXCLUSION, 254
i2cAcquireBus, 262
i2caddr_t, 256
I2CD1, 266
I2CDriver, 257
i2cflags_t, 257
i2cGetErrors, 260
i2cInit, 258
i2cMasterReceive, 255
i2cMasterReceiveTimeout, 261
i2cMasterTransmit, 255
i2cMasterTransmitTimeout, 261
i2cObjectInit, 258
i2cReleaseBus, 263
i2cStart, 259
i2cstate_t, 257
i2cStop, 259
PLATFORM_I2C_USE_I2C1, 256
I2C_ACK_FAILURE
I2C Driver, 253
I2C_ACTIVE_RX
I2C Driver, 257
I2C_ACTIVE_TX
I2C Driver, 257
I2C_ARBITRATION_LOST
I2C Driver, 253
I2C_BUS_ERROR
I2C Driver, 253
i2c_lld_get_errors
I2C Driver, 256
i2c_lld_init
I2C Driver, 264
i2c_lld_master_receive_timeout
I2C Driver, 265
i2c_lld_master_transmit_timeout
I2C Driver, 265
i2c_lld_start
I2C Driver, 264
i2c_lld_stop
I2C Driver, 264
I2C_LOCKED
I2C Driver, 257
I2C_NO_ERROR
I2C Driver, 253
I2C_PEC_ERROR
I2C Driver, 254
I2C_READY
I2C Driver, 257
I2C_SMB_ALERT
I2C Driver, 254
I2C_STOP
I2C Driver, 257
I2C_TIMEOUT
I2C Driver, 254
I2C_UNINIT
I2C Driver, 257
I2C_USE_MUTUAL_EXCLUSION
Configuration, 879
I2C Driver, 254
i2cAcquireBus
I2C Driver, 262
i2caddr_t
I2C Driver, 256
I2CConfig, 1008
I2CD1
I2C Driver, 266
I2CDriver, 1009
config, 1010
errors, 1010
I2C Driver, 257
state, 1009
i2cflags_t
I2C Driver, 257
i2cGetErrors
I2C Driver, 260
i2cInit
I2C Driver, 258
i2cMasterReceive
I2C Driver, 255
i2cMasterReceiveTimeout
I2C Driver, 261
i2cMasterTransmit
I2C Driver, 255
i2cMasterTransmitTimeout
I2C Driver, 261
i2cObjectInit
I2C Driver, 258
i2cReleaseBus
I2C Driver, 263
i2cStart
I2C Driver, 259
i2cstate_t
I2C Driver, 257
i2cStop
I2C Driver, 259
I2S Driver, 267
_i2s_isr_full_code, 271
_i2s_isr_half_code, 270
I2S_ACTIVE, 273
I2S_COMPLETED, 273
_i2s_lld_config_fields, 272
_i2s_lld_driver_fields, 272
_i2s_lld_init, 277
_i2s_lld_start, 277
_i2s_lld_start_exchange, 278
_i2s_lld_stop, 278
_i2s_lld_stop_exchange, 279
I2S_READY, 273
I2S_STOP, 273
I2S_UNINIT, 273
i2scallback_t, 272
I2SCfg, 272
I2SD1, 279
I2SDriver, 272
i2sInit, 273
i2sIsBufferComplete, 269
i2sObjectInit, 274
i2sStart, 274
i2sStartExchange, 276
i2sStartExchangeI, 269
i2sstream_t, 273
I2SDriver, 272
I2SStop, 275
i2sStopExchange, 276
i2sStopExchangeI, 270
I2S_READY, 273
I2S_STOP, 273
I2S_UNINIT, 273
i2scallback_t, 272
I2SCfg, 272
I2SD1, 279
I2SDriver, 272
i2sInit, 273
i2sIsBufferComplete, 269
i2sObjectInit, 274
i2sStart, 274
i2sStartExchange, 276
i2sStartExchangeI, 269
i2sstream_t, 273
I2SDriver, 272
I2SStop, 275
i2sStopExchange, 276
i2sStopExchangeI, 270
ibnotify
Serial over USB Driver, 649
ibqGetEmptyBuffer
I/O Buffers Queues, 66
ibqGetFullBufferTimeout
I/O Buffers Queues, 68
ibqGetFullBufferTimeoutS
I/O Buffers Queues, 69
ibqGetTimeout
I/O Buffers Queues, 71
ibqIsEmpty
I/O Buffers Queues, 62
ibqIsFull
I/O Buffers Queues, 62
ibqObjectInit
I/O Buffers Queues, 65
ibqPostFullBuffer
I/O Buffers Queues, 67
ibqReadTimeout
I/O Buffers Queues, 72
ibqReleaseEmptyBuffer
I/O Buffers Queues, 70
ibqReleaseEmptyBufferS
I/O Buffers Queues, 71
ibqReset
I/O Buffers Queues, 66
ICU Driver, 280
__icu_isr_invoke_overflow_cb, 288
__icu_isr_invoke_period_cb, 287
__icu_isr_invoke_width_cb, 287
ICU_ACTIVE, 291
ICU_INPUT_ACTIVE_HIGH, 291
ICU_INPUT_ACTIVE_LOW, 291
icu_lld_are_notifications_enabled, 289
icu_lld_disable_notifications, 302
ChibiOS/HAL
icu_lld_enable_notifications, 301
icu_lld_get_period, 289
icu_lld_get_width, 288
icu_lld_init, 299
icu_lld_start, 299
icu_lld_start_capture, 300
icu_lld_stop, 300
icu_lld_stop_capture, 301
icu_lld_wait_capture, 300
ICU_READY, 291
ICU_STOP, 291
ICU_UNINIT, 291
ICU_WAITING, 291
icuAreNotificationsEnabledX, 285
icucallback_t, 290
icucnt_t, 290
ICUD1, 302
icuDisableNotifications, 298
icuDisableNotificationsI, 285
ICUDriver, 290
icuEnableNotifications, 297
icuEnableNotificationsI, 284
icufreq_t, 290
icuGetPeriodX, 286
icuGetWidthX, 286
icuInit, 291
icumode_t, 291
icuObjectInit, 292
icuStart, 292
icuStartCapture, 295
icuStartCaptureI, 283
icustate_t, 291
icuStop, 294
icuStopCapture, 297
icuStopCaptureI, 284
icuWaitCapture, 295
PLATFORM_ICU_USE_ICU1, 288
ICU_ACTIVE
ICU Driver, 291
ICU_INPUT_ACTIVE_HIGH
ICU Driver, 291
ICU_INPUT_ACTIVE_LOW
ICU Driver, 291
icu_lld_are_notifications_enabled
ICU Driver, 289
icu_lld_disable_notifications
ICU Driver, 302
icu_lld_enable_notifications
ICU Driver, 301
icu_lld_get_period
ICU Driver, 289
icu_lld_get_width
ICU Driver, 288
icu_lld_init
ICU Driver, 299
icu_lld_start
ICU Driver, 299
icu_lld_start_capture
ICU Driver, 299
ICU Driver, 300
ICU Driver, 300
ICU Driver, 301
ICU Driver, 300
ICU READY
ICU Driver, 291
ICU STOP
ICU Driver, 291
ICU UNINIT
ICU Driver, 291
ICU WAITING
ICU Driver, 291
icuAreNotificationsEnabledX
ICU Driver, 291
icucallback_t
ICU Driver, 285
icucnt_t
ICU Driver, 290
ICUCfg, 1010
frequency, 1011
mode, 1011
overflow_cb, 1012
period_cb, 1012
width_cb, 1011
ICUD1
ICU Driver, 302
icuDisableNotifications
ICU Driver, 298
icuDisableNotificationsI
ICU Driver, 285
ICUDriver, 1012
config, 1013
ICU Driver, 290
state, 1013
icuEnableNotifications
ICU Driver, 297
icuEnableNotificationsI
ICU Driver, 284
icufreq_t
ICU Driver, 290
icuGetPeriodX
ICU Driver, 286
icuGetWidthX
ICU Driver, 286
icuInit
ICU Driver, 291
icumode_t
ICU Driver, 291
icuObjectInit
ICU Driver, 292
icuStart
ICU Driver, 292
icuStartCapture
ICU Driver, 295
icuStartCaptureI
ICU Driver, 295
ICU Driver, 283
icustate_t
   ICU Driver, 291
icuStop
   ICU Driver, 294
icuStopCapture
   ICU Driver, 297
icuStopCaptureI
   ICU Driver, 284
icuWaitCapture
   ICU Driver, 295
id
   mfs_data_header_t, 1032
   mfs_transaction_op_t, 1033
IDE
   CANRxFrame, 959
   CANTxFrame, 962
in
   hal_usb_lld.c, 1257
   USB Driver, 830
in_cb
   USBEndpointConfig, 1105
in_maxsize
   USBEndpointConfig, 1106
in_params
   USBDriver, 1101
in_state
   USBEndpointConfig, 1106
Inner Code, 890
Input Formatter Utility, 347
   chscanf, 348
   CHSCANF_USE_FLOAT, 347
   chsnscanf, 349
   chvscanf, 347
   chvsnscanf, 351
input_buffers_queue_t
   I/O Buffers Queues, 65
input_queue_t
   I/O Bytes Queues, 565
int_in
   SerialUSBConfig, 1076
Interfaces and Classes, 889
io_buffers_queue, 1013
   bcounter, 1015
   bn, 1016
   brdptr, 1015
   bsize, 1016
   btop, 1016
   buffers, 1016
   bwrptr, 1015
   link, 1017
   notify, 1017
   ptr, 1016
   suspended, 1015
   top, 1017
   waiting, 1015
io_buffers_queue_t
   I/O Buffers Queues, 64
io_queue, 1018
   q_buffer, 1019
   q_counter, 1019
   q_link, 1020
   q_notify, 1020
   q_rdptr, 1020
   q_top, 1019
   q_waiting, 1019
   q_wrptr, 1019
io_queue_t
   I/O Bytes Queues, 564
IOBus, 1020
   mask, 1021
   offset, 1021
   portid, 1021
IOBUS_DECL
   PAL Driver, 481
ioline_t
   PAL Driver, 515
iomode_t
   PAL Driver, 515
iopadid_t
   PAL Driver, 515
IOPORT1
   PAL Driver, 505
ioportid_t
   PAL Driver, 515
ioportmask_t
   PAL Driver, 515
iq_read
   I/O Bytes Queues, 565
iqGet
   I/O Bytes Queues, 561
iqGetEmptyI
   I/O Bytes Queues, 559
iqGetFullI
   I/O Bytes Queues, 559
iqGetI
   I/O Bytes Queues, 569
iqGetTimeout
   I/O Bytes Queues, 569
iqIsEmptyI
   I/O Bytes Queues, 560
iqIsFullI
   I/O Bytes Queues, 560
iqIsFullI
   I/O Bytes Queues, 560
iqObjectInit
   I/O Bytes Queues, 566
iqPutI
   I/O Bytes Queues, 568
iqRead
   I/O Bytes Queues, 570
iqReadTimeout
   I/O Bytes Queues, 571
iqResetI
   I/O Bytes Queues, 567
link
   io_buffers_queue, 1017
LPA_100BASE4
MAC Driver, 364
macInit
MAC Driver, 368
macObjectInit
MAC Driver, 369
macPollLinkStatus
MAC Driver, 374
macReadReceiveDescriptor
MAC Driver, 365
MACReceiveDescriptor, 1024
offset, 1025
size, 1025
macReleaseReceiveDescriptor
MAC Driver, 373
macReleaseTransmitDescriptor
MAC Driver, 372
macStart
MAC Driver, 369
macstate_t
MAC Driver, 368
macStop
MAC Driver, 370
MACTransmitDescriptor, 1025
offset, 1026
size, 1026
macWaitReceiveDescriptor
MAC Driver, 372
macWaitTransmitDescriptor
MAC Driver, 371
macWriteTransmitDescriptor
MAC Driver, 365
magic1
mfs_bank_header_t, 1030
mfs_data_header_t, 1031
magic2
mfs_bank_header_t, 1030
mfs_data_header_t, 1031
Managed Flash Storage Driver, 382
ALIGNED_DHDR_SIZE, 385
ALIGNED_REC_SIZE, 384
ALIGNED_SIZEOF, 385
mfs_bank_check_header, 391
mfs_bank_erase, 390
mfs_bank_get_state, 392
mfs_bank_scan_records, 392
mfs_bank_state_t, 388
mfs_bank_t, 387
mfs_bank_verify_erase, 390
mfs_bank_write_header, 391
MFS_CFG_BUFFER_SIZE, 386
MFS_CFG_MAX_RECORDS, 385
MFS_CFG_MAX_REPAIR_ATTEMPTS, 386
MFS_CFG_MEMORY_ALIGNMENT, 386
MFS_CFG_STRONG_CHECKING, 386
MFS_CFG_TRANSACTION_MAX, 387
MFS_CFG_WRITE_VERIFY, 386
mfs_error_t, 387
mfs_flash_copy, 389
mfs_flash_read, 388
mfs_flash_write, 388
mfs_garbage_collect, 393
mfs_id_t, 387
mfs_mount, 394
mfs_state_t, 387
mfs_try_mount, 394
mfsCommitTransaction, 402
mfsErase, 396
mfsEraseRecord, 398
mfsObjectInit, 395
mfsPerformGarbageCollection, 400
mfsReadRecord, 397
mfsRollbackTransaction, 402
mfsStart, 395
mfsStartTransaction, 401
mfsStop, 396
mfsWriteRecord, 398
PAIR, 385
RET_ON_ERROR, 385
mask
IOBus, 1021
Memory Streams Class, 335
_memory_stream_data, 335
msObjectInit, 335
MemoryStream, 1026
vmt, 1028
memstreams.c, 1269
memstreams.h, 1270
MemStreamVMT, 1029
mfs_bank_check_header
Managed Flash Storage Driver, 391
mfs_bank_erase
Managed Flash Storage Driver, 390
mfs_bank_get_state
Managed Flash Storage Driver, 392
mfs_bank_header_t, 1029
counter, 1030
crc, 1030
magic1, 1030
magic2, 1030
reserved1, 1030
mfs_bank_scan_records
Managed Flash Storage Driver, 392
mfs_bank_state_t
Managed Flash Storage Driver, 388
mfs_bank_t
Managed Flash Storage Driver, 387
mfs_bank_verify_erase
Managed Flash Storage Driver, 390
mfs_bank_write_header
Managed Flash Storage Driver, 391
MFS_CFG_BUFFER_SIZE
Managed Flash Storage Driver, 386
MFS_CFG_MAX_RECORDS
Managed Flash Storage Driver, 385
MFS_CFG_MAX_REPAIR_ATTEMPTS
Managed Flash Storage Driver, 386
MFS_CFG_MEMORY_ALIGNMENT
Managed Flash Storage Driver, 386
MFS_CFG_STRONG_CHECKING
Managed Flash Storage Driver, 386
MFS_CFG_TRANSACTION_MAX
Managed Flash Storage Driver, 386
MFS_CFG_WRITE_VERIFY
MFS_CFG_MEMORY_ALIGNMENT
  Managed Flash Storage Driver, 386
MFS_CFG_STRONG_CHECKING
  Managed Flash Storage Driver, 386
MFS_CFG_TRANSACTION_MAX
  Managed Flash Storage Driver, 387
MFS_CFG_WRITEVERIFY
  Managed Flash Storage Driver, 386
mfs_data_header_t, 1031
  crc, 1032
  id, 1032
  magic1, 1031
  magic2, 1031
  size, 1032
mfs_error_t
  Managed Flash Storage Driver, 387
mfs_flash_copy
  Managed Flash Storage Driver, 389
mfs_flash_read
  Managed Flash Storage Driver, 388
mfs_flash_write
  Managed Flash Storage Driver, 388
mfs_garbage_collect
  Managed Flash Storage Driver, 393
mfs_id_t
  Managed Flash Storage Driver, 387
mfs_mount
  Managed Flash Storage Driver, 394
mfs_state_t
  Managed Flash Storage Driver, 387
mfs_transaction_op_t, 1032
  id, 1033
  offset, 1033
  size, 1033
mfs_try_mount
  Managed Flash Storage Driver, 394
mfsCommitTransaction
  Managed Flash Storage Driver, 402
MFSConfig, 1034
  bank0_sectors, 1035
  bank0_start, 1035
  bank1_sectors, 1036
  bank1_start, 1035
  bank_size, 1035
  erased, 1035
  flashp, 1035
MFSDriver, 1036
  buffer, 1039
  config, 1037
  current_bank, 1038
  current_counter, 1038
  descriptors, 1038
  next_offset, 1038
  state, 1037
  tr_limit_offset, 1039
  tr_next_offset, 1038
  tr_nops, 1039
  tr_ops, 1039
  used_space, 1038
mfsErase
  Managed Flash Storage Driver, 396
mfsEraseRecord
  Managed Flash Storage Driver, 398
mfsObjectInit
  Managed Flash Storage Driver, 395
mfsPerformGarbageCollection
  Managed Flash Storage Driver, 400
mfsReadRecord
  Managed Flash Storage Driver, 397
mfsRollbackTransaction
  Managed Flash Storage Driver, 402
mfsStart
  Managed Flash Storage Driver, 395
mfsStartTransaction
  Managed Flash Storage Driver, 401
mfsStop
  Managed Flash Storage Driver, 396
mfsWriteRecord
  Managed Flash Storage Driver, 398
MII/RMII Header, 404
  ADVERTISE_100BASE4, 414
  ADVERTISE_100FULL, 414
  ADVERTISE_100HALF, 413
  ADVERTISE_10FULL, 413
  ADVERTISE_10HALF, 413
  ADVERTISE_CSMA, 413
  ADVERTISE_LPACK, 414
  ADVERTISE_NPAGE, 415
  ADVERTISE_PAUSEASYM, 414
  ADVERTISE_PAUSE_CAP, 414
  ADVERTISE_RESV, 414
  ADVERTISE_RFAULT, 414
  ADVERTISE_SLCT, 413
  BMCR_ANENABLE, 410
  BMCR_ANRESTART, 410
  BMCR_CTST, 410
  BMCR_FULLDPLX, 410
  BMCR_ISOLATE, 410
  BMCR_LOOPBACK, 411
  BMCR_PDOWN, 410
  BMCR_RESET, 411
  BMCR_RESV, 410
  BMCR_SPEED100, 411
  BMSR_100BASE4, 413
  BMSR_100FULL, 413
  BMSR_100HALF, 412
  BMSR_10FULL, 412
  BMSR_10HALF, 412
  BMSR_ANEGCAPABLE, 411
  BMSR_ANEGCOMPLETE, 412
  BMSR_ERCAP, 411
  BMSR_JCD, 411
  BMSR_LSTATUS, 411
  BMSR_MFPRESUPPCAP, 412
  BMSR_RESV, 412
  BMSR_RFAULT, 412
<table>
<thead>
<tr>
<th>EXPANSION_ENABLENPAGE, 417</th>
<th>MII_EXPANSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPANSION_LCW, 417</td>
<td>MII_RMII Header, 407</td>
</tr>
<tr>
<td>EXPANSION_MFAULTS, 417</td>
<td>MII_FCSCOUNTER</td>
</tr>
<tr>
<td>EXPANSION_NPCAPABLE, 417</td>
<td>MII_RMII Header, 408</td>
</tr>
<tr>
<td>EXPANSION_NWAY, 416</td>
<td>MII_LBRERROR</td>
</tr>
<tr>
<td>EXPANSION_RESV, 417</td>
<td>MII_RMII Header, 409</td>
</tr>
<tr>
<td>LPA_100BASE4, 415</td>
<td>MII_LPA</td>
</tr>
<tr>
<td>LPA_100FULL, 415</td>
<td>MII_RMII Header, 407</td>
</tr>
<tr>
<td>LPA_10FULL, 415</td>
<td>MII_MICR</td>
</tr>
<tr>
<td>LPA_10HALF, 415</td>
<td>MII_RMII Header, 408</td>
</tr>
<tr>
<td>LPA_LPACK, 416</td>
<td>MII_NCONFIG</td>
</tr>
<tr>
<td>LPA_NPAGE, 416</td>
<td>MII_RMII Header, 409</td>
</tr>
<tr>
<td>LPA_PAUSEASYM, 416</td>
<td>MII_NWAYTEST</td>
</tr>
<tr>
<td>LPA_PAUSE_CAP, 416</td>
<td>MII_RMII Header, 408</td>
</tr>
<tr>
<td>LPA_RESV, 416</td>
<td>MII_PHYADDR</td>
</tr>
<tr>
<td>LPA_RFAULT, 416</td>
<td>MII_RMII Header, 409</td>
</tr>
<tr>
<td>LPA_SLCT, 415</td>
<td>MII_PHYSTS</td>
</tr>
<tr>
<td>MII_ADVERTISE, 407</td>
<td>MII_RMII Header, 406</td>
</tr>
<tr>
<td>MII_ANNPTR, 407</td>
<td>MII_PHYSID1</td>
</tr>
<tr>
<td>MII_BMCR, 406</td>
<td>MII_RMII Header, 407</td>
</tr>
<tr>
<td>MII_BMSR, 406</td>
<td>MII_PHYSID2</td>
</tr>
<tr>
<td>MII_CTRL1000, 407</td>
<td>MII_RMII Header, 408</td>
</tr>
<tr>
<td>MII_DCOUNTER, 408</td>
<td>MII_PHYSTS</td>
</tr>
<tr>
<td>MII_ESTATUS, 408</td>
<td>MII_RMII Header, 408</td>
</tr>
<tr>
<td>MII_EXPANSION, 407</td>
<td>MII_RESV1</td>
</tr>
<tr>
<td>MII_FCSCOUNTER, 408</td>
<td>MII_RMII Header, 409</td>
</tr>
<tr>
<td>MII_LBRERROR, 409</td>
<td>MII_RESV2</td>
</tr>
<tr>
<td>MII_LPA, 407</td>
<td>MII_RMII Header, 409</td>
</tr>
<tr>
<td>MII_MICR, 408</td>
<td>MII_SREVISION</td>
</tr>
<tr>
<td>MII_NCONFIG, 409</td>
<td>MII_RMII Header, 409</td>
</tr>
<tr>
<td>MII_NWAYTEST, 408</td>
<td>MII_STAT1000</td>
</tr>
<tr>
<td>MII_PHYADDR, 409</td>
<td>MII_RMII Header, 407</td>
</tr>
<tr>
<td>MII_PHYSID1, 406</td>
<td>MII_TPISTATUS</td>
</tr>
<tr>
<td>MII_PHYSID2, 407</td>
<td>MMC over SPI Driver, 419</td>
</tr>
<tr>
<td>MII_PHYSTS, 408</td>
<td>_mmc_driver_methods, 421</td>
</tr>
<tr>
<td>MII_RERRCOUNTER, 408</td>
<td>crc7, 422</td>
</tr>
<tr>
<td>MII_RESV1, 409</td>
<td>crc7_lookup_table, 442</td>
</tr>
<tr>
<td>MII_RESV2, 409</td>
<td>MMC_NICE_WAITING, 421</td>
</tr>
<tr>
<td>MII_SREVISON, 409</td>
<td>mmc_vmt, 441</td>
</tr>
<tr>
<td>MII_STAT1000, 407</td>
<td>mmcConnect, 432</td>
</tr>
<tr>
<td>MII_TPISTATUS, 409</td>
<td>mmcDisconnect, 433</td>
</tr>
<tr>
<td>NWAYTEST_LOOPBACK, 417</td>
<td>mmcErase, 440</td>
</tr>
<tr>
<td>NWAYTEST_RESV1, 417</td>
<td>mmcGetInfo, 440</td>
</tr>
<tr>
<td>NWAYTEST_RESV2, 418</td>
<td>mmcInit, 430</td>
</tr>
<tr>
<td>Millisecond</td>
<td>mmcIsCardInserted, 421</td>
</tr>
<tr>
<td>RTCDateTime, 1061</td>
<td>mmcIsWriteProtected, 422</td>
</tr>
<tr>
<td>MMC over SPI Driver, 419</td>
<td>mmcObjectInit, 430</td>
</tr>
<tr>
<td>_mmc_driver_methods, 421</td>
<td>mmcSequentialRead, 435</td>
</tr>
<tr>
<td>crc7, 422</td>
<td>mmcSequentialWrite, 437</td>
</tr>
<tr>
<td>crc7_lookup_table, 442</td>
<td>mmcStart, 431</td>
</tr>
<tr>
<td>MMC_NICE_WAITING, 421</td>
<td>mmcStartSequentialRead, 434</td>
</tr>
<tr>
<td>mmc_vmt, 441</td>
<td>mmcStartSequentialWrite, 436</td>
</tr>
<tr>
<td>MMC over SPI Driver, 419</td>
<td>mmcStop, 431</td>
</tr>
<tr>
<td>_mmc_driver_methods, 421</td>
<td>mmcStopSequentialRead, 435</td>
</tr>
</tbody>
</table>
MMC/SD Block Device, 443
  _mmcsd_block_device_data, 448
  _mmcsd_block_device_methods, 448
  _mmcsd_get_capacity, 450
  _mmcsd_get_capacity_ext, 451
  _mmcsd_get_slice, 450
  _mmcsd_unpack_csd_mmc, 453
  _mmcsd_unpack_csd_v10, 453
  _mmcsd_unpack_csd_v20, 454
  _mmcsd_unpack_mmc_cid, 452
  _mmcsd_unpack_sdc_cid, 451
  MMCSD_BLOCK_SIZE, 447
  MMCSD_CMD8_PATTERN, 448
  MMCSD_R1_ERROR, 448
  MMCSD_R1_ERROR_MASK, 447
  MMCSD_R1_IS_CARD_LOCKED, 449
  MMCSD_R1_STS, 449
  mmcsdGetCardCapacity, 449

mmc_cmd6_construct
SDC Driver, 602

mmc_detect_bus_clk
SDC Driver, 606

mmc_init
SDC Driver, 601

MMC_NICE_WAITING
  Configuration, 880
  MMC over SPI Driver, 421

mmc_set_bus_width
SDC Driver, 608

mmc_switch_t
SDC Driver, 600

mmc_vmt
MMC over SPI Driver, 441

MMCConfig, 1040
  hscfg, 1041
  lscfg, 1041
  spip, 1041

mmcConnect
MMC over SPI Driver, 432

mmcDisconnect
MMC over SPI Driver, 433

MMCConfig, 1041
  block_addresses, 1044
  config, 1044
  vmt, 1043

MMCConfigVMT, 1044

mmcErase
MMC over SPI Driver, 440

mmcGetInfo
MMC over SPI Driver, 440

mmcInit
MMC over SPI Driver, 430

mmcIsCardInserted
MMC over SPI Driver, 421

mmcIsWriteProtected
MMC over SPI Driver, 422

mmcObjectInit
MMC over SPI Driver, 430

MMCSD_BLOCK_SIZE
  MMC/SD Block Device, 447

MMCSD_CMD8_PATTERN
  MMC/SD Block Device, 448

MMCSD_R1_ERROR
  MMC/SD Block Device, 448

MMCSD_R1_ERROR_MASK
  MMC/SD Block Device, 447

MMCSD_R1_IS_CARD_LOCKED
  MMC/SD Block Device, 449

MMCSD_R1_STS
  MMC/SD Block Device, 449

mmcsdGetCardCapacity
MMC over SPI Driver, 435

mmcSequentialRead
MMC over SPI Driver, 437

mmcSequentialWrite
MMC over SPI Driver, 431

mmcStart
MMC over SPI Driver, 431

mmcStartSequentialRead
MMC over SPI Driver, 434

mmcStartSequentialWrite
MMC over SPI Driver, 436

mmcStop
MMC over SPI Driver, 431

mmcStopSequentialRead
MMC over SPI Driver, 435

mmcStopSequentialWrite
MMC over SPI Driver, 438

mmcSync
MMC over SPI Driver, 439

mode
  ICUConfig, 1011
  PWMChannelConfig, 1054

mode_detect
  SDC Driver, 600

month
  RTCDateTime, 1060

msg_t
  OSAL, 908

msgObjectInit
MMC over SPI Driver, 435

Memory Streams Class, 335

mutex
  hal_adc_driver, 982
hal_dac_driver, 987
hal_spi_driver, 1002
hal_wspi_driver, 1006
UARTDriver, 1092
mutex_t
OSAL, 909
next_offset
MFSDriver, 1038
Normal Drivers, 886
notify
io_buffers_queue, 1017
Null Streams Class, 337
_null_stream_data, 337
nullObjectInit, 337
nullObjectInit
Null Streams Class, 337
NullStream, 1049
vmt, 1050
nullstreams.c, 1270
nullstreams.h, 1271
NullStreamVMT, 1051
num_channels
hal_adc_configuration_group, 979
hal_dac_conversion_group, 983
NWAYTEST_LOOPBACK
MII/RMII Header, 417
NWAYTEST_RESV1
MII/RMII Header, 417
NWAYTEST_RESV2
MII/RMII Header, 418
objGetInstance
Base Object, 469
obnotify
Serial over USB Driver, 650
obqFlush
I/O Buffers Queues, 82
obqGetEmptyBufferTimeout
I/O Buffers Queues, 76
obqGetEmptyBufferTimeoutS
I/O Buffers Queues, 77
obqGetFullBuffer
I/O Buffers Queues, 74
obqIsEmpty
I/O Buffers Queues, 63
obqIsFull
I/O Buffers Queues, 64
obqObjectInit
I/O Buffers Queues, 73
obqPostFullBuffer
I/O Buffers Queues, 78
obqPostFullBufferS
I/O Buffers Queues, 79
obqPutTimeout
I/O Buffers Queues, 79
obqReleaseEmptyBuffer
I/O Buffers Queues, 75
obqReset
I/O Buffers Queues, 74
I/O Buffers Queues, 81
obqWriteTimeout
I/O Buffers Queues, 80
offset
flash_sector_descriptor_t, 974
I OBus, 1021
MACReceiveDescriptor, 1025
MACTransmitDescriptor, 1026
mfs_transaction_op_t, 1033
operation
hal_sio_driver, 995
oq_write
I/O Bytes Queues, 566
oqGetEmpty
I/O Bytes Queues, 562
oqGetFull
I/O Bytes Queues, 561
oqGet
I/O Bytes Queues, 575
oqIsEmpty
I/O Bytes Queues, 563
oqIsFull
I/O Bytes Queues, 563
oqObjectInit
I/O Bytes Queues, 572
oqPut
I/O Bytes Queues, 564
oqPut
I/O Bytes Queues, 574
oqPutTimeout
I/O Bytes Queues, 574
oqReset
I/O Bytes Queues, 573
oqWrite
I/O Bytes Queues, 576
oqWriteTimeout
I/O Bytes Queues, 577
OSAL, 892
event_source_t, 909
eventcallback_t, 909
eventflags_t, 909
msg_t, 908
mutex_t, 909
OSAL_DBG_ENABLE_ASSERTS, 898
OSAL_DBG_ENABLE_CHECKS, 898
osal_halt_msg, 926
OSAL_I2MS, 903
OSAL_I2S, 902
OSAL_I2US, 904
OSAL_IRQ_EPILOGUE, 900
OSAL_IRQ_HANDLER, 900
OSAL_IRQHandler, 900
OSAL_IRQ_IS_VALID_PRIORITY, 900
OSAL_IRQ_MAXIMUM_PRIORITY, 898
OSAL_IRQ_PRIORITY_LEVELS, 898
OSAL_IRQ_PROLOGUE, 900
OSAL_MS2I, 901
OSAL_MS2RTC, 905
OSAL_S2I, 901
OSAL_S2RTC, 904
OSAL_ST_FREQUENCY, 897
OSAL_ST_MODE, 897
OSAL_ST_RESOLUTION, 897
OSAL_US2I, 902
OSAL_US2RTC, 906
osalDbgAssert, 898
osalDbgCheck, 899
osalDbgCheckClassI, 899
osalDbgCheckClassS, 899
osalEventBroadcastFlags, 918
osalEventBroadcastFlagsI, 918
osalEventObjectInit, 925
osalEventSetCallback, 919
osalInit, 910
osalMutexLock, 919
osalMutexObjectInit, 925
osalMutexUnlock, 920
osalOsGetSystemTimeX, 912
osalOsRescheduleS, 911
osalOsTimerHandler, 911
osalSysDisable, 920
osalSysEnable, 921
osalSysGetStatusAndLockX, 922
osalSysHalt, 910
osalSysLock, 921
osalSysLockFromISR, 921
osalSysPolledDelayX, 911
osalSysRestoreStatusX, 922
osalSysUnlock, 921
osalSysUnlockFromISR, 922
osalThreadDequeueAllI, 916
osalThreadDequeueNextI, 916
osalThreadEnqueueTimeoutS, 915
osalThreadQueueObjectInit, 924
osalThreadResumeI, 914
osalThreadResumeS, 915
osalThreadSleep, 913
osalThreadSleepMicroseconds, 907
osalThreadSleepMilliseconds, 907
osalThreadSleepS, 912
osalThreadSleepSeconds, 906
osalThreadsuspendS, 913
osalThreadsuspendTimeoutS, 914
osalTimeAddX, 923
osalTimeDiffX, 923
osalTimeIsInRangeX, 924
rtcnt_t, 908
sysinterval_t, 908
syssts_t, 908
systime_t, 908
thread_reference_t, 909
time_conv_t, 908
osal.c, 1271
osal.h, 1273
OSAL_DBG_ENABLE_ASSERTS
OSAL, 898
OSAL_DBG_ENABLE_CHECKS
OSAL, 898
osal_msg
OSAL, 926
OSAL_I2MS
OSAL, 903
OSAL_I2S
OSAL, 902
OSAL_I2US
OSAL, 904
OSAL_IRQ_EPILOGUE
OSAL, 900
OSAL_IRQ_HANDLER
OSAL, 900
OSAL_IRQ_IS_VALID_PRIORITY
OSAL, 900
OSAL_IRQ_MAXIMUM_PRIORITY
OSAL, 898
OSAL_IRQ_PRIORITY_LEVELS
OSAL, 898
OSAL_IRQ_PROLOGUE
OSAL, 900
OSAL_MS2I
OSAL, 901
OSAL_MS2RTC
OSAL, 905
OSAL_S2I
OSAL, 901
OSAL_S2RTC
OSAL, 904
OSAL_ST_FREQUENCY
OSAL, 897
OSAL_ST_MODE
OSAL, 897
OSAL_ST_RESOLUTION
OSAL, 897
OSAL_US2I
OSAL, 902
OSAL_US2RTC
OSAL, 906
osalDbgAssert
OSAL, 898
osalDbgCheck
OSAL, 899
osalDbgCheckClassI
OSAL, 899
osalDbgCheckClassS
OSAL, 899
osalEventBroadcastFlags
OSAL, 918
osalEventBroadcastFlagsI
OSAL, 918
osalEventObjectInit
OSAL, 925
osalEventSetCallback
OSAL, 919
osalInit
OSAL, 910
osalMutexLock
OSAL, 919
osalMutexObjectInit
OSAL, 925
osalMutexUnlock
OSAL, 920
osalOsGetSystemTimeX
OSAL, 912
osalOsRescheduleS
OSAL, 911
osalOsTimerHandlerI
OSAL, 911
osalSysDisable
OSAL, 920
osalSysEnable
OSAL, 921
osalSysGetStatusAndLockX
OSAL, 922
osalSysHalt
OSAL, 910
osalSysLock
OSAL, 910
osalSysLockFromISR
OSAL, 921
osalSysPolledDelayX
OSAL, 911
osalSysRestoreStatusX
OSAL, 922
osalSysUnlock
OSAL, 921
osalSysUnlockFromISR
OSAL, 922
osalThreadDequeueAllI
OSAL, 916
osalThreadDequeueNextI
OSAL, 916
osalThreadEnqueueTimeoutS
OSAL, 915
osalThreadQueueObjectInit
OSAL, 924
osalThreadResumeI
OSAL, 914
osalThreadResumeS
OSAL, 915
osalThreadSleep
OSAL, 913
osalThreadSleepMicroseconds
OSAL, 907
osalThreadSleepMilliseconds
OSAL, 907
osalThreadSleepS
OSAL, 912
osalThreadSleepSeconds
OSAL, 906
osalThreadSuspendS
OSAL, 913
osalThreadSuspendTimeoutS
OSAL, 914
osalTimeAddX
OSAL, 923
osalTimeDiffX
OSAL, 923
osalTimeIsInRangeX
OSAL, 924
osalTimeAddX
hal_usb_lld.c, 1257
USB Driver, 830
out_cb
USBEndpointConfig, 1106
out_maxsize
USBEndpointConfig, 1106
out_params
USBDriver, 1101
out_state
USBEndpointConfig, 1106
Output Formatter Utility, 341
chprintf, 342
CHPRINTF_USE_FLOAT, 341
chsprintf, 343
chvprintf, 341
chvsnprintf, 345
output_buffers_queue_t
I/O Buffers Queues, 65
output Queue_t
I/O Bytes Queues, 565
overflow_cb
ICUConfig, 1012
page_size
flash descriptor_t, 972
PAIR
Managed Flash Storage Driver, 385
PAL Driver, 470
_IOBUS_DATA, 480
_pal_clear_event, 482, 483
_pal_init_event, 481, 482
_pal_isr_code, 484, 485
_pal_lld_init, 522
_pal_lld_setgroupmode, 522
IOBUS_DECL, 481
ioline_t, 515
iomode_t, 515
io_padid_t, 515
IOPORT1, 505
ioportid_t, 515
ioportmask_t, 515
PAL_EVENT_MODE_BOTH_EDGES, 479
PAL_EVENT_MODE_DISABLED, 478
PAL_EVENT_MODE_EDGES_MASK, 478
PAL_EVENT_MODE_FALLING_EDGE, 478
PAL_EVENT_MODE_RISING_EDGE, 478
PAL_GROUP_MASK, 480
PAL_HIGH, 478
PAL_IOPORTS_WIDTH, 504
PAL_LINE, 504
pal_lld_clearpad, 512

ChibiOS/HAL
PAL Driver, 511
pal_lld_setpadmode
PAL Driver, 513
pal_lld_setport
PAL Driver, 507
pal_lld_togglepad
PAL Driver, 513
pal_lld_toggleport
PAL Driver, 508
pal_lld_writegroup
PAL Driver, 509
pal_lld_writepad
PAL Driver, 511
pal_lld_writeport
PAL Driver, 506
PAL_LOW
PAL Driver, 478
PAL_MODE_INPUT
PAL Driver, 477
PAL_MODE_INPUT_ANALOG
PAL Driver, 477
PAL_MODE_INPUT_PULLDOWN
PAL Driver, 477
PAL_MODE_INPUT_PULLUP
PAL Driver, 477
PAL_MODE_OUTPUT_OPENDRAIN
PAL Driver, 477
PAL_MODE_OUTPUT_PUSH_PULL
PAL Driver, 477
PAL_MODE_RESET
PAL Driver, 476
PAL_MODE_UNCONNECTED
PAL Driver, 476
PAL_NOLINE
PAL Driver, 505
PAL_PAD
PAL Driver, 504
PAL_PORT
PAL Driver, 504
PAL_PORT_BIT
PAL Driver, 479
PAL_USE_CALLBACKS
Configuration, 877
PAL Driver, 479
PAL_USE_WAIT
Configuration, 877
PAL Driver, 479
PAL_WHOLE_PORT
PAL Driver, 504
palcallback_t
PAL Driver, 515
palClearLine
PAL Driver, 496
palClearPad
PAL Driver, 492
palClearPort
PAL Driver, 487
PALConfig, 1051
palDisableLineEvent
PAL Driver, 501
palDisableLineEventI
PAL Driver, 500
palDisablePadEvent
PAL Driver, 499
palDisablePadEventI
PAL Driver, 498
palEnableLineEvent
PAL Driver, 501
palEnableLineEventI
PAL Driver, 500
palEnablePadEvent
PAL Driver, 499
palEnablePadEventI
PAL Driver, 498
palevent_t, 1052
arg, 1053
cb, 1052
threads, 1052
palInit
PAL Driver, 485
pallsLineEventEnabledX
PAL Driver, 502
pallsPadEventEnabledX
PAL Driver, 502
palReadBus
PAL Driver, 516
palReadGroup
PAL Driver, 488
palReadLatch
PAL Driver, 486
palReadLine
PAL Driver, 495
palReadPad
PAL Driver, 490
palReadPort
PAL Driver, 485
palSetBusMode
PAL Driver, 517
palSetGroupMode
PAL Driver, 490
palSetLine
PAL Driver, 496
palSetLineCallback
PAL Driver, 503
palSetLineCallbackI
PAL Driver, 518
palSetLineMode
PAL Driver, 497
palSetPad
PAL Driver, 492
palSetPadCallback
PAL Driver, 503
palSetPadCallbackI
PAL Driver, 517
palSetPadMode
PAL Driver, 493
palSetPort
PAL Driver, 487
palToggleLine
PAL Driver, 497
palTogglePad
PAL Driver, 493
palTogglePort
PAL Driver, 488
palWaitLineTimeout
PAL Driver, 521
palWaitLineTimeoutS
PAL Driver, 520
palWaitPadTimeout
PAL Driver, 519
palWaitPadTimeoutS
PAL Driver, 518
palWriteBus
PAL Driver, 516
palWriteGroup
PAL Driver, 489
palWriteLine
PAL Driver, 495
palWritePad
PAL Driver, 491
palWritePort
PAL Driver, 486
param
event_source, 967
period
PWMConfig, 1056
PWMDriver, 1058
period_cb
ICUConfig, 1012
Peripheral Interfaces, 524
PLATFORM_ADC_USE_ADC1
ADC Driver, 41
PLATFORM_CAN_USE_CAN1
CAN Driver, 90
PLATFORM_CRY_USE_CRY1
Crypto Driver, 113
PLATFORM_DAC_USE_DAC1
DAC Driver, 192
PLATFORM_EFL_USE_EFL1
EFL Driver, 210
PLATFORM_GPT_USE_GPT1
GPT Driver, 236
PLATFORM_I2C_USE_I2C1
I2C Driver, 256
PLATFORM_I2S_USE_I2S1
I2S Driver, 271
PLATFORM_ICU_USE_ICU1
ICU Driver, 288
PLATFORM_MAC_USE_MAC1
MAC Driver, 367
PLATFORM_PWM_USE_PWM1
PWM Driver, 536
PLATFORM_RTC_USE_RTC1
RTC Driver, 582
PLATFORM_SDC_USE_SDC1
SDC Driver, 598
PLATFORM_SERIAL_USE_USART1
Serial Driver, 835
PLATFORM_SIO_USE_SIO1
SIO Driver, 672
PLATFORM_SPI_USE_SPI1
SPI Driver, 702
PLATFORM_TRNG_USE_TRNG1
TRNG Driver, 739
PLATFORM_UART_USE_UART1
UART Driver, 760
PLATFORM_USB_USE_USB1
USB Driver, 802
PLATFORM_WDG_USE_WDG1
WDG Driver, 835
PLATFORM_WSPI_USE_WSPI1
WSPI Driver, 850
portid
IOBus, 1021
ps_error_t
Abstract Persistent Storage Interface, 333
ps_offset_t
Abstract Persistent Storage Interface, 333
psGetStorageSize
Abstract Persistent Storage Interface, 331
psRead
Abstract Persistent Storage Interface, 332
psWrite
Abstract Persistent Storage Interface, 332
ptr
io_buffers_queue, 1016
PWM Driver, 525
PLATFORM_PWM_USE_PWM1, 536
PWM_CHANNELS, 536
PWM_FRACTION_TO_WIDTH, 530
pwm_lld_change_period, 536
pwm_lld_change_period, 536
pwm_lld_disable_channel, 550
pwm_lld_disable_channel, 550
pwm_lld_disable_periodic_notification, 551
pwm_lld_disable_periodic_notification, 551
pwm_lld_enable_channel, 549
pwm_lld_enable_channel, 549
pwm_lld_enable_periodic_notification, 550
pwm_lld_enable_periodic_notification, 550
pwm_lld_init, 548
pwm_lld_init, 548
pwm_lld_start, 548
pwm_lld_start, 548
pwm_lld_stop, 549
pwm_lld_stop, 549
PWM_OUTPUT_ACTIVE_HIGH, 529
PWM_OUTPUT_ACTIVE_LOW, 529
PWM_OUTPUT_DISABLED, 528
PWM_OUTPUT_Mask, 528
PWM_PERCENTAGE_TO_WIDTH, 530
PWM_READY, 539
PWM_STOP, 539
PWM_UNINIT, 539
pwmcallback_t, 537
pwmChangePeriod, 541
pwmChangePerioddl, 531
pwmchannel_t, 538
pwmchnmsk_t, 538
pwmcnt_t, 538
PWMD1, 552
pwmDisableChannel, 543
pwmDisableChannel, 532
pwmDisableChannelNotification, 547
pwmDisableChannelNotificationI, 535
pwmDisablePeriodicNotification, 545
pwmDisablePeriodicNotificationI, 534
PWMDriver, 537
pwmEnableChannel, 542
pwmEnableChannel, 532
pwmEnableChannelNotification, 546
pwmEnableChannelNotificationI, 535
pwmEnablePeriodicNotification, 544
pwmEnablePeriodicNotificationI, 534
pwmlnit, 539
pwmIsChannelEnabledI, 533
pwmmode_t, 538
pwmObjectInit, 539
pwmStart, 540
pwmstate_t, 538
pwmStop, 540
PWM_CHANNELS
PWM Driver, 536
PWM_DEGREES_TO_WIDTH
PWM Driver, 530
PWM_FRACTION_TO_WIDTH
PWM Driver, 529
pwm_lld_change_period
PWM Driver, 536
pwm_lld_disable_channel
PWM Driver, 550
pwm_lld_disable_channel_notification
PWM Driver, 552
pwm_lld_disable_periodic_notification
PWM Driver, 551
pwm_lld_enable_channel
PWM Driver, 549
pwm_lld_enable_channel_notification
PWM Driver, 551
pwm_lld_enable_periodic_notification
PWM Driver, 550
pwm_lld_init
PWM Driver, 548
pwm_lld_start
PWM Driver, 548
pwm_lld_stop
PWM Driver, 549
PWM_OUTPUT_ACTIVE_HIGH
PWM Driver, 529
PWM_OUTPUT_ACTIVE_LOW
PWM Driver, 529
PWM_OUTPUT_DISABLED
PWM Driver, 528
PWM_OUTPUT_MASK
PWM Driver, 528

PWM_PERCENTAGE_TO_WIDTH
PWM Driver, 530
PWM READY
PWM Driver, 539
PWM STOP
PWM Driver, 539
PWM UNINIT
PWM Driver, 539
pwmcallback_t
PWM Driver, 537
pwmChangePeriod
PWM Driver, 541
pwmChangePeriodI
PWM Driver, 531
pwmchannel_t
PWM Driver, 538
PWMChannelConfig, 1053
callback, 1054
mode, 1054
pwmlmsChannelEnabledI
PWM Driver, 538
pwmcnt_t
PWM Driver, 538
PWMConfig, 1055
callback, 1056
channels, 1056
frequency, 1056
period, 1056
PWMD1
PWM Driver, 552
pwmDisableChannel
PWM Driver, 543
pwmDisableChannell
PWM Driver, 532
pwmDisableChannelNotification
PWM Driver, 547
pwmDisableChannelNotificationI
PWM Driver, 535
pwmDisablePeriodicNotification
PWM Driver, 545
pwmDisablePeriodicNotificationI
PWM Driver, 534
PWMDriver, 1057
channels, 1058
cfg, 1058
enabled, 1058
period, 1058
PWM Driver, 537
state, 1058
pwmEnableChannel
PWM Driver, 542
pwmEnableChannell
PWM Driver, 532
pwmEnableChannelNotification
PWM Driver, 546
pwmEnableChannelNotificationI
PWM Driver, 535
pwmEnablePeriodicNotification
PWM Driver, 544
pwmEnablePeriodicNotification
PWM Driver, 534
pwmInit
PWM Driver, 539
pwmIsChannelEnabled
PWM Driver, 533
pwm_mode_t
PWM Driver, 538
pwmObjectInit
PWM Driver, 539
pwmStart
PWM Driver, 540
pwm_state_t
PWM Driver, 538
pwmStop
PWM Driver, 540
q_buffer
io_queue, 1019
q_counter
io_queue, 1019
Q_EMPTY
I/O Bytes Queues, 556
Q_FULL
I/O Bytes Queues, 557
q_link
io_queue, 1020
q_notify
io_queue, 1020
Q_OK
I/O Bytes Queues, 556
q_rdptr
io_queue, 1020
Q_RESET
I/O Bytes Queues, 556
Q_TIMEOUT
I/O Bytes Queues, 556
q_top
io_queue, 1019
q_waiting
io_queue, 1019
q_wrptr
io_queue, 1019
qGetLink
I/O Bytes Queues, 558
qnotify_t
I/O Bytes Queues, 565
qSetLink
I/O Bytes Queues, 558
qSizeX
I/O Bytes Queues, 557
qSpaceI
I/O Bytes Queues, 557
rca
SDCDriver, 1067
rdqueue
MACDriver, 1024
read_CxD
MMC over SPI Driver, 428
receiving
USBDriver, 1100
rectrl
MMC over SPI Driver, 424
rectrl3
MMC over SPI Driver, 425
requests_hook_cb
USBCfg, 1097
reserved1
mfs_bank_header_t, 1030
RET_ON_ERROR
Managed Flash Storage Driver, 385
RTC Driver, 579
PLATFORM_RTC_USE_RTC1, 582
RTC_ALARMS, 581
RTC_BASE_YEAR, 581
RTC_HAS_STORAGE, 581
rtc_lld_driver_fields, 582
rtc_lld_get_alarm, 591
rtc_lld_get_time, 590
rtc_lld_init, 589
rtc_lld_set_alarm, 590
rtc_lld_set_time, 589
RTC_SUPPORTS_CALLBACKS, 581
rtcalarm_t, 582
rtc_callback, 582
 rtcConvertDateTimeToFAT, 588
rtcConvertDateTimeToStructTm, 587
rtcGetAlarm, 582
rtcGetEvent, 583
rtcGetTime, 584
rtcInit, 583
rtcObjectInit, 583
rtcSetAlarm, 585
rtcSetCallback, 587
rtcSetTime, 584
RTC_ALARMS
RTC Driver, 581
RTC_BASE_YEAR
RTC Driver, 581
RTC_HAS_STORAGE
RTC Driver, 581
rtc_lld_driver_fields
RTC Driver, 582
rtc_lld_get_alarm
RTC Driver, 591
rtc_lld_get_time
RTC Driver, 590
rtc_lld_init
RTC Driver, 589
rtc_lld_set_alarm
RTC Driver, 590
RTC Driver, 590
rtc_lld_set_time
RTC Driver, 589
RTC_SUPPORTED_CALLBACKS
RTC Driver, 581
RTCAlarm, 1059
rtcalarm_t
RTC Driver, 582
rtccb_t
RTC Driver, 582
rtcConvertDateTimeToFAT
RTC Driver, 588
rtcConvertDateTimeToStructTm
RTC Driver, 587
rtcConvertStructTmToDateTm
RTC Driver, 588
RTCDateTime, 1059
day, 1061
dayOfWeek, 1061
dstFlag, 1060
millisecond, 1061
month, 1060
year, 1060
RTCDriver, 1061
RTC Driver, 582
vmt, 1062
RTCDriverVMT, 1063
rtcevent_t
RTC Driver, 583
rtcGetAlarm
RTC Driver, 586
rtcGetTime
RTC Driver, 584
rtcInit
RTC Driver, 583
rtcnt_t
OSAL, 908
rtcObjectInit
RTC Driver, 583
rtcSetAlarm
RTC Driver, 585
rtcSetCallback
RTC Driver, 587
rtcSetTime
RTC Driver, 584
RTR
CANRxFrame, 959
CntTxFrame, 961
rx_buffer
hal_i2s_config, 991
rx_cb
hal_sio_operation, 997
rx_evt_cb
hal_sio_operation, 998
rx_idle_cb
hal_sio_operation, 997
rxbuf
USBOutEndpointState, 1109
rxchar_cb
UARTConfig, 1089
rxcnt
USBOutEndpointState, 1109
rxend_cb
UARTConfig, 1089
rxerr_cb
UARTConfig, 1089
rxfull_event
CANDriver, 955
rxqueue
CANDriver, 955
rxsizes
USBOutEndpointState, 1109
rxstate
UARTDriver, 1091
samples
hal_adc_driver, 981
hal_dac_driver, 986
saved_state
USBDevice, 1103
SD1
Serial Driver, 645
SD_BREAK_DETECTED
Serial Driver, 629
SD_FRAMING_ERROR
Serial Driver, 629
sd_lld_init
Serial Driver, 643
sd_lld_start
Serial Driver, 644
sd_lld_stop
Serial Driver, 644
SD_NOISE_ERROR
Serial Driver, 629
SD_OVERRUN_ERROR
Serial Driver, 629
SD_PARITY_ERROR
Serial Driver, 629
SD_QUEUE_FULL_ERROR
Serial Driver, 630
SD_READY
Serial Driver, 636
SD_STOP
Serial Driver, 636
sd_switch_function_t
SDC Driver, 592
sd_switch_t
SDC Driver, 600
SD_UNINIT
Serial Driver, 636
sdAsynchronousRead
Serial Driver, 635
sdAsynchronousWrite
Serial Driver, 633
SDC Driver, 592
_sdc_driver_methods, 599
_sdc_wait_for_transfer_state, 608
detect_bus_clk, 606
mmc_cmd6_construct, 602
mmc_detect_bus_clk, 606
mmc_init, 601
mmc_set_bus_width, 608
mmc_switch_t, 600
mode_detect, 600
PLATFORM_SDC_USE_SDC1, 598
sd_switch_function_t, 600
sd_switch_t, 600
sdc_cmd6_check_status, 605
sdc_cmd6_construct, 604
sdc_cmd6_extract_info, 604
sdc_detect_bus_clk, 605
sdc_init, 602
SDC_INIT_OCR, 596
SDC_INIT_OCR_V20, 596
SDC_INIT_RETRY, 596
sdc_lld_init, 617
sdc_lld_read, 622
sdc_lld_send_cmd_long_crc, 622
sdc_lld_send_cmd_none, 620
sdc_lld_send_cmd_short, 620
sdc_lld_send_cmd_short_crc, 621
sdc_lld_set_bus_mode, 620
sdc_lld_set_data_clk, 619
sdc_lld_start, 618
sdc_lld_start_clk, 618
sdc_lld_stop, 618
sdc_lld_stop_clk, 619
sdc_lld_sync, 624
sdc_lld_write, 623
SDC_MMC_SUPPORT, 596
SDC_NICE_WAITING, 596
sdc_set_bus_width, 607
sdc_vmt, 624
sdcbusclk_t, 600
sdcbusmode_t, 600
sdcConnect, 612
SDCD1, 625
sdcDisconnect, 612
SDCDriver, 599
sdcErase, 617
sdcflags_t, 599
sdcGetAndClearErrors, 615
sdcGetInfo, 616
sdcInit, 609
sdcsCardInserted, 597
sdcsWriteProtected, 598
sdcmode_t, 599
sdcObjectInit, 610
sdcRead, 613
sdcStart, 610
sdcStop, 611
sdcSync, 615
sdcWrite, 614
sdc_cmd6_check_status
SDC Driver, 605
sdc_cmd6_construct
SDC Driver, 604
sdc_cmd6_extract_info
SDC Driver, 604
sdc_connect
SDC Driver, 605
sdc_init
SDC Driver, 602
SDC_INIT_OCR
Configuration, 881
SDC Driver, 596
SDC_INIT_OCR_V20
Configuration, 881
SDC Driver, 596
SDC_INIT_RETRY
Configuration, 880
SDC Driver, 596
sdc_lld_init
SDC Driver, 617
sdc_lld_read
SDC Driver, 622
sdc_lld_send_cmd_long_crc
SDC Driver, 622
sdc_lld_send_cmd_none
SDC Driver, 620
sdc_lld_send_cmd_short
SDC Driver, 620
sdc_lld_send_cmd_short_crc
SDC Driver, 621
sdc_lld_set_bus_mode
SDC Driver, 620
sdc_lld_set_data_clk
SDC Driver, 619
sdc_lld_start
SDC Driver, 618
sdc_lld_start_clk
SDC Driver, 618
sdc_lld_stop
SDC Driver, 618
sdc_lld_stop_clk
SDC Driver, 619
sdc_lld_sync
SDC Driver, 624
sdc_lld_write
SDC Driver, 623
SDC_MMC_SUPPORT
Configuration, 880
SDC Driver, 596
SDC_NICE_WAITING
Configuration, 881
SDC Driver, 596
sdc_set_bus_width
SDC Driver, 607
sdc_vmt
SDC Driver, 624
sdcbusclk_t
SDC Driver, 600
sdcbusmode_t
sdInit
SDC Driver, 637
sdObjectInit
Serial Driver, 637
sdPut
Serial Driver, 631
sdPutl
Serial Driver, 630
sdPutTimeout
Serial Driver, 631
sdPutWouldBlock
Serial Driver, 641
sdRead
Serial Driver, 634
sdReadl
Serial Driver, 634
sdReadTimeout
Serial Driver, 634
sdRequestDataI
Serial Driver, 640
sdStart
Serial Driver, 638
sdstate_t
Serial Driver, 636
sdStop
Serial Driver, 638
SDU_READY
Serial over USB Driver, 649
SDU_STOP
Serial over USB Driver, 649
SDU_UNINIT
Serial over USB Driver, 649
sduConfigureHookI
Serial over USB Driver, 653
sduControl
Serial over USB Driver, 657
sduDataReceived
Serial over USB Driver, 656
sduDataTransmitted
Serial over USB Driver, 655
sdlInit
Serial over USB Driver, 650
sdlInterruptTransmitted
Serial over USB Driver, 656
sduObjectInit
Serial over USB Driver, 650
sduRequestsHook
Serial over USB Driver, 654
sduSOFHookI
Serial over USB Driver, 654
sduStart
Serial over USB Driver, 651
sdstate_t
Serial over USB Driver, 649
sdStop
Serial over USB Driver, 651
sduSuspendHookI
Serial over USB Driver, 652
sduWakeupHookI
    Serial over USB Driver, 653
sdWrite
    Serial Driver, 633
sdWritel
    Serial Driver, 632
sdWriteTimeout
    Serial Driver, 633
sectors
    flash_descriptor_t, 972
sectors_count
    flash_descriptor_t, 972
sectors_size
    flash_descriptor_t, 972
send_command_R1
    MMC over SPI Driver, 426
send_command_R3
    MMC over SPI Driver, 427
send_hdr
    MMC over SPI Driver, 424
Serial Driver, 626
_serial_driver_data, 635
_serial_driver_methods, 630
default_config, 645
PLATFORM_SERIAL_USE_USART1, 635
SD1, 645
SD_BREAK_DETECTED, 629
SD_FRAMING_ERROR, 629
sd_lld_init, 643
sd_lld_start, 644
sd_lld_stop, 644
SD_NOISE_ERROR, 629
SD_OVERRUN_ERROR, 629
SD_PARITY_ERROR, 629
SD_QUEUE_FULL_ERROR, 630
SD_READY, 636
SD_STOP, 636
SD_UNINIT, 636
sdAsynchronousRead, 635
sdAsynchronousWrite, 633
sdControl, 643
sdGet, 632
sdGetI, 631
sdGetTimeout, 632
sdGetWouldBlock, 642
sdIncomingData1, 639
sdInit, 637
sdObjectInit, 637
sdPut, 631
sdPutI, 630
sdPutTimeout, 631
sdGetWouldBlock, 641
sdRead, 634
sdReadI, 634
sdReadTimeout, 634
sdRequestData1, 640
sdStart, 638
sdstate_t, 636
sdStop, 638
sdWrite, 633
sdWritel, 632
sdWriteTimeout, 633
SERIAL_BUFFERS_SIZE, 630
SERIAL_DEFAULT_BITRATE, 630
SerialDriver, 636
Serial NOR Flash Driver, 456
_snor_flash_methods, 459
_snor_flash_methods_alone, 458
bus_acquire, 459
bus_cmd, 460
bus_cmd_addr, 462
bus_cmd_addr_dummy_receive, 464
bus_cmd_addr_receive, 463
bus_cmd_addr_send, 462
bus_cmd_dummy_receive, 463
bus_cmd_receive, 461
bus_cmd_send, 461
bus_release, 460
bus_stop, 460
SNOR_BUS_DRIVER, 458
snor_get_descriptor, 459
SNOR_SHARED_BUS, 458
snor_vmt, 467
snorMemoryMap, 465
snorMemoryUnmap, 466
snorObjectInit, 464
snorStart, 464
snorStop, 465
Serial over USB Driver, 646
_serial_usb_driver_data, 648
_serial_usb_driver_methods, 648
ibnotify, 649
obnotify, 650
SDU_READY, 649
SDU_STOP, 649
SDU_UNINIT, 649
sdConfigureHook1, 653
sdControl, 657
sdDataReceived, 656
sdDataTransmitted, 655
sdInit, 650
sdInterruptTransmitted, 656
sdObjectInit, 650
sdRequestsHook, 654
sdSOFHook1, 654
sdStart, 651
sdstate_t, 649
sdStop, 651
sdSuspendHook1, 652
sdWakeupHook1, 653
SERIAL_USB Buffers_NUMBER, 648
SERIAL_USB BUFFERS_SIZE, 648
SerialUSBDriver, 649
SERIAL_BUFFERS_SIZE
Configuration, 881
Serial Driver, 630
ChibiOS/HAL
INDEX

SERIAL_DEFAULT_BITRATE
  Configuration, 881
  Serial Driver, 630
SERIAL_USB_BUFFERS_NUMBER
  Configuration, 882
  Serial over USB Driver, 648
SERIAL_USB_BUFFERS_SIZE
  Configuration, 882
  Serial over USB Driver, 648
SerialConfig, 1069
  speed, 1070
SerialDriver, 1070
  Serial Driver, 636
  vmt, 1072
SerialDriverVMT, 1073
SerialUSBCfg, 1074
  bulk_in, 1076
  bulk_out, 1076
  int_in, 1076
  usbp, 1075
SerialUSBDriver, 1077
  Serial over USB Driver, 649
  vmt, 1078
SerialUSBDriverVMT, 1079
  set_address
    USB Driver, 808
setup
  USBDriver, 1102
setup_cb
    USBEndpointConfig, 1105
SHA1Context, 1080
SHA256Context, 1081
SHA512Context, 1081
SID
  CANRxFrame, 959
  CANTxFrame, 962
SIO Driver, 658
  _sio_callback_rx, 668
  _sio_callback_rx_evt, 670
  _sio_callback_rx_idle, 669
  _sio_callback_tx, 669
  _sio_callback_tx_end, 669
  _sio_wakeup_rx, 670
  _sio_wakeup_tx, 671
  _sio_wakeup_txend, 671
  _sio_driver_methods, 663
PLATFORM_SIO_USE_SIO1, 672
SIO_ACTIVE, 675
SIO_BREAK_DETECTED, 663
SIO_DEFAULT_BITRATE, 663
sio_events_mask_t, 675
SIO_FRAMING_ERROR, 662
sio_lld_config_fields, 672
sio_lld_control, 689
sio_lld_driver_fields, 672
sio_lld_get, 688
sio_lld_get_and_clear_events, 686
sio_lld_init, 684
sio_lld_is_rx_empty, 672
sio_lld_is_tx_full, 673
sio_lld_is_tx_ongoing, 673
sio_lld_put, 688
sio_lld_read, 686
sio_lld_start, 684
sio_lld_start_operation, 685
sio_lld_stop, 685
sio_lld_stop_operation, 686
sio_lld_write, 687
SIO_NO_ERROR, 662
SIO_NOISE_ERROR, 663
SIO_OVERRUN_ERROR, 662
SIO_PARITY_ERROR, 662
SIO_READY, 675
SIO_STOP, 675
SIO_UNINIT, 675
SIO_USE_SYNCHRONIZATION, 663
sioAsyncRead, 680
sioAsyncReadI, 666
sioAsyncWrite, 681
sioAsyncWriteI, 667
siocb_t, 675
SIOConfig, 674
sioControlX, 667
SIOD1, 689
SIODriver, 674
sioflags_t, 674
sioGetAndClearEvents, 679
sioGetAndClearEventsI, 665
sioGetFlagsX, 663
sioGetX, 665
sioInit, 676
sioIsRXXEmptyX, 664
sioIsTXFullX, 664
sioObjectInit, 676
SIOOperation, 674
sioPutX, 666
sioStart, 676
sioStartOperation, 678
siostate_t, 675
sioStop, 677
sioStopOperation, 679
sioSynchronizeRX, 682
sioSynchronizeTX, 683
sioSynchronizeTXEnd, 683
SIO_ACTIVE
  SIO Driver, 675
SIO_BREAK_DETECTED
  SIO Driver, 663
SIO_DEFAULT_BITRATE
  Configuration, 882
  SIO Driver, 663
sio_driver_vmt, 1082
sio_events_mask_t
  SIO Driver, 675
SIO_FRAMING_ERROR
  SIO Driver, 662
sio_lld_config_fields
SIO Driver, 672
sio_lld_control
SIO Driver, 689
sio_lld_driver_fields
SIO Driver, 672
sio_lld_get
SIO Driver, 688
sio_lld_get_and_clear_events
SIO Driver, 686
sio_lld_init
SIO Driver, 684
sio_lld_is_rx_empty
SIO Driver, 672
sio_lld_is_tx_full
SIO Driver, 673
sio_lld_is_tx_ongoing
SIO Driver, 673
sio_lld_put
SIO Driver, 688
sio_lld_read
SIO Driver, 686
sio_lld_start
SIO Driver, 684
sio_lld_start_operation
SIO Driver, 685
sio_lld_stop
SIO Driver, 685
sio_lld_stop_operation
SIO Driver, 686
sio_lld_write
SIO Driver, 687
SIO_NO_ERROR
SIO Driver, 662
SIO_NOISE_ERROR
SIO Driver, 663
SIO_OVERRUN_ERROR
SIO Driver, 662
SIO_PARITY_ERROR
SIO Driver, 662
SIO_READY
SIO Driver, 675
SIO_STOP
SIO Driver, 675
SIO_UNINIT
SIO Driver, 675
SIO_USE_SYNCHRONIZATION
Configuration, 882
SIO Driver, 682
sioAsyncRead
SIO Driver, 680
sioAsyncReadI
SIO Driver, 666
sioAsyncWrite
SIO Driver, 681
sioAsyncWriteI
SIO Driver, 667
siocb_t
SIO Driver, 675
SIOConfig
SIO Driver, 674
sioControlX
SIO Driver, 667
SIOD1
SIO Driver, 689
SIODriver
SIO Driver, 674
sioflags_t
SIO Driver, 674
sioGetAndClearEvents
SIO Driver, 679
sioGetAndClearEventsI
SIO Driver, 665
sioGetFlagsX
SIO Driver, 663
sioGetX
SIO Driver, 665
sioInit
SIO Driver, 676
sioIsRXEmptyX
SIO Driver, 664
sioIsTXFullX
SIO Driver, 664
sioObjectInit
SIO Driver, 676
SIOOperation
SIO Driver, 674
sioPutX
SIO Driver, 666
sioStart
SIO Driver, 676
sioStartOperation
SIO Driver, 678
siostate_t
SIO Driver, 675
sioStop
SIO Driver, 677
sioStopOperation
SIO Driver, 679
sioSynchronizeRX
SIO Driver, 682
sioSynchronizeTX
SIO Driver, 683
sioSynchronizeTXEnd
SIO Driver, 683
size
flash_descriptor_t, 973
flash_sector_descriptor_t, 974
hal_i2s_config, 991
MACReceiveDescriptor, 1025
MACTransmitDescriptor, 1026
mfs_data_header_t, 1032
mfs_transaction_op_t, 1033
sleep_event
CANDriver, 956
SNOR_BUS_DRIVER
<table>
<thead>
<tr>
<th>Function</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI READY</td>
<td>704</td>
</tr>
<tr>
<td>SPI_SELECT_MODE</td>
<td>694</td>
</tr>
<tr>
<td>SPI_STOP</td>
<td>703</td>
</tr>
<tr>
<td>SPI_SUPPORTS_CIRCULAR</td>
<td>702</td>
</tr>
<tr>
<td>SPI_UNINIT</td>
<td>703</td>
</tr>
<tr>
<td>SPI_USE_CIRCULAR</td>
<td>694</td>
</tr>
<tr>
<td>SPI_USE_MUTUAL_EXCLUSION</td>
<td>693</td>
</tr>
<tr>
<td>spiAbort</td>
<td>713</td>
</tr>
<tr>
<td>spiAbortI</td>
<td>712</td>
</tr>
<tr>
<td>spiAcquireBus</td>
<td>718</td>
</tr>
<tr>
<td>spicallback_t</td>
<td>703</td>
</tr>
<tr>
<td>SPIConfig</td>
<td>703</td>
</tr>
<tr>
<td>SPID1</td>
<td>725</td>
</tr>
<tr>
<td>SPIDriver</td>
<td>703</td>
</tr>
<tr>
<td>spiExchange</td>
<td>715</td>
</tr>
<tr>
<td>spiIgnore</td>
<td>714</td>
</tr>
<tr>
<td>spiInit</td>
<td>704</td>
</tr>
<tr>
<td>spisNewBufferComplete</td>
<td>694</td>
</tr>
<tr>
<td>spiObjectInit</td>
<td>704</td>
</tr>
<tr>
<td>spip</td>
<td>695</td>
</tr>
<tr>
<td>MMCCConfig</td>
<td>1041</td>
</tr>
<tr>
<td>spiPolledExchange</td>
<td>699</td>
</tr>
<tr>
<td>spiReceive</td>
<td>717</td>
</tr>
<tr>
<td>spiReleaseBus</td>
<td>719</td>
</tr>
<tr>
<td>spiSelect</td>
<td>707</td>
</tr>
<tr>
<td>spiSelectI</td>
<td>695</td>
</tr>
<tr>
<td>spiStart</td>
<td>716</td>
</tr>
<tr>
<td>spiStartExchange</td>
<td>705</td>
</tr>
<tr>
<td>spiStartExchangeI</td>
<td>709</td>
</tr>
<tr>
<td>spiStartExchangeI</td>
<td>696</td>
</tr>
<tr>
<td>spiStartIgnore</td>
<td>708</td>
</tr>
<tr>
<td>spiStartIgnoreI</td>
<td>696</td>
</tr>
<tr>
<td>spiStartReceive</td>
<td>711</td>
</tr>
<tr>
<td>spiStartReceiveI</td>
<td>698</td>
</tr>
<tr>
<td>spiStartSend</td>
<td>710</td>
</tr>
<tr>
<td>spiStartSendI</td>
<td>697</td>
</tr>
<tr>
<td>spiStop</td>
<td>703</td>
</tr>
<tr>
<td>spiUnselect</td>
<td>706</td>
</tr>
<tr>
<td>spiUnselectI</td>
<td>707</td>
</tr>
<tr>
<td>Ssline</td>
<td>1000</td>
</tr>
<tr>
<td>ssline</td>
<td>1000</td>
</tr>
<tr>
<td>ssline</td>
<td>1000</td>
</tr>
<tr>
<td>ssline</td>
<td>1000</td>
</tr>
<tr>
<td>ssline</td>
<td>1000</td>
</tr>
<tr>
<td>ssmask</td>
<td>1000</td>
</tr>
<tr>
<td>ssmask</td>
<td>1000</td>
</tr>
<tr>
<td>ssport</td>
<td>1000</td>
</tr>
<tr>
<td>ssport</td>
<td>1000</td>
</tr>
<tr>
<td>spisBufferComplete</td>
<td>694</td>
</tr>
<tr>
<td>ST Driver</td>
<td>726</td>
</tr>
<tr>
<td>spistate_t</td>
<td>703</td>
</tr>
<tr>
<td>st_callbacks</td>
<td>737</td>
</tr>
<tr>
<td>st_cal callbacks</td>
<td>737</td>
</tr>
<tr>
<td>st ld get alarm</td>
<td>736</td>
</tr>
<tr>
<td>st ld get counter</td>
<td>735</td>
</tr>
<tr>
<td>st ld init</td>
<td>735</td>
</tr>
<tr>
<td>st ld is alarm active</td>
<td>736</td>
</tr>
<tr>
<td>st ld set alarm</td>
<td>736</td>
</tr>
<tr>
<td>st ld start alarm</td>
<td>735</td>
</tr>
<tr>
<td>st ld stop alarm</td>
<td>736</td>
</tr>
<tr>
<td>stBind</td>
<td>727</td>
</tr>
<tr>
<td>stBindAlarmN</td>
<td>732</td>
</tr>
<tr>
<td>stGetAlarm</td>
<td>730</td>
</tr>
<tr>
<td>stGetAlarmN</td>
<td>733</td>
</tr>
<tr>
<td>stGetCounter</td>
<td>727</td>
</tr>
<tr>
<td>stinit</td>
<td>727</td>
</tr>
<tr>
<td>stisAlarmActive</td>
<td>730</td>
</tr>
<tr>
<td>stisAlarmActiveN</td>
<td>734</td>
</tr>
<tr>
<td>stSetAlarm</td>
<td>729</td>
</tr>
<tr>
<td>stSetAlarmN</td>
<td>733</td>
</tr>
<tr>
<td>stSetCallback</td>
<td>731</td>
</tr>
<tr>
<td>stStartAlarm</td>
<td>728</td>
</tr>
<tr>
<td>stStartAlarmN</td>
<td>732</td>
</tr>
<tr>
<td>stStopAlarm</td>
<td>729</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
stStopAlarmN, 733

streamRead
Abstract Streams Interface, 308
Streams, 305
streamUnget
Abstract Streams Interface, 310
streamWrite
Abstract Streams Interface, 307

stSetAlarm
ST Driver, 729
stSetAlarmN
ST Driver, 733
stSetCallback
ST Driver, 731
stStartAlarm
ST Driver, 728
stStartAlarmN
ST Driver, 732
stStopAlarm
ST Driver, 729
stStopAlarmN
ST Driver, 733
Support Code, 891

sync
MMC over SPI Driver, 429
sync_rx
hal_sio_driver, 995
sync_tx
hal_sio_driver, 996
sync_txend
hal_sio_driver, 996

sysinterval_t
OSAL, 908
syssts_t
OSAL, 908
systime_t
OSAL, 908

tdqueue
MACDriver, 1024

thread
hal_adc_driver, 982
hal_dac_driver, 986
hal_spi_driver, 1002
hal_wspi_driver, 1006
USBInEndpointState, 1108
USBOutEndpointState, 1109
thread_reference_t
OSAL, 909

threadrx
UARTDriver, 1092

treads
palevent_t, 1052
threads_queue_t, 1087

threadtx
UARTDriver, 1092

TIME
CANRxFrame, 958
time_conv_t
  OSAL, 908
top
  io_buffers_queue, 1017
tr_limit_offset
  MFSDriver, 1039
tr_next_offset
  MFSDriver, 1038
tr_nops
  MFSDriver, 1039
tr_ops
  MFSDriver, 1039
transmitting
  USBDriver, 1100
TRNG Driver, 738
  PLATFORM_TRNG_USE_TRNG1, 739
  trng_ld_config_fields, 740
  trng_ld_driver_fields, 739
  trng_ld_generate, 745
  trng_ld_init, 744
  trng_ld_start, 744
  trng_ld_stop, 745
  TRNG_READY, 741
  TRNG_RUNNING, 741
  TRNG_STOP, 741
  TRNG_UNINIT, 741
  TRNGConfig, 740
  TRNGD1, 746
  TRNGDriver, 740
  trngGenerate, 743
  trngInit, 741
  trngObjectInit, 741
  trngStart, 742
  trngstate_t, 740
  trngStop, 742
  trng_ld_config_fields
    TRNG Driver, 740
  trng_ld_driver_fields
    TRNG Driver, 739
  trng ld_generate
    TRNG Driver, 745
  trng ld_init
    TRNG Driver, 744
  trng ld_start
    TRNG Driver, 744
  trng ld_stop
    TRNG Driver, 745
  TRNG_READY
    TRNG Driver, 741
  TRNG_RUNNING
    TRNG Driver, 741
  TRNG_STOP
    TRNG Driver, 741
  TRNG_UNINIT
    TRNG Driver, 741
  TRNGConfig
    TRNG Driver, 740
  TRNGD1
  TRNGDriver
    TRNG Driver, 746
  trngGenerate
    TRNG Driver, 740
trngInit
    TRNG Driver, 743
trngObjectInit
    TRNG Driver, 741
trngStart
    TRNG Driver, 742
trngstate_t
    TRNG Driver, 740
  tx_buffer
    hal_i2s_config, 990
  tx_cb
    hal_sio_operation, 997
  tx_end_cb
    hal_sio_operation, 998
  txbuf
    USBDriver, 1100
  txcnt
    USBDriver, 1107
txeEmpty_event
    CANDriver, 956
  txend1_cb
    UARTConfig, 1089
  txend2_cb
    UARTConfig, 1089
txqueue
    CANDriver, 955
txsize
    USBDriver, 1107
  txstate
    UARTDriver, 1091
  UART Driver, 747
    _uart_rx_char_match_isr_code, 760
    _uart_rx_complete_isr_code, 757
    _uart_rx_error_isr_code, 758
    _uart_rx_idle_code, 758
    _uart_timeout_isr_code, 759
    _uart_tx1_isr_code, 756
    _uart_tx2_isr_code, 756
    _uart_wakeup_rx_cm_isr, 755
    _uart_wakeup_rx_complete_isr, 754
    _uart_wakeup_rx_error_isr, 754
    _uart_wakeup_rx_timeout_isr, 755
    _uart_wakeup_txi_isr, 753
    _uart_wakeup_txs_isr, 753
    PLATFORM_UART_USE_UART1, 760
  UART_BREAK_DETECTED, 752
  UART_FRAMING_ERROR, 752
  uart ld_init, 778
  uart ld_start, 779
  uart ld_start_receive, 781
  uart ld_start_send, 780

ChibiOS/HAL
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>usb_lld_get_frame_number, 802</td>
<td></td>
</tr>
<tr>
<td>usb_lld_get_status_in, 826</td>
<td></td>
</tr>
<tr>
<td>usb_lld_get_status_out, 825</td>
<td></td>
</tr>
<tr>
<td>usb_lld_get_transaction_size, 803</td>
<td></td>
</tr>
<tr>
<td>usb_lld_init, 822</td>
<td></td>
</tr>
<tr>
<td>usb_lld_init_endpoint, 824</td>
<td></td>
</tr>
<tr>
<td>usb_lld_prepare_receive, 827</td>
<td></td>
</tr>
<tr>
<td>usb_lld_prepare_transmit, 827</td>
<td></td>
</tr>
<tr>
<td>usb_lld_read_setup, 826</td>
<td></td>
</tr>
<tr>
<td>usb_lld_reset, 823</td>
<td></td>
</tr>
<tr>
<td>usb_lld_set_address, 824</td>
<td></td>
</tr>
<tr>
<td>usb_lld_start, 822</td>
<td></td>
</tr>
<tr>
<td>usb_lld_start_in, 828</td>
<td></td>
</tr>
<tr>
<td>usb_lld_start_out, 827</td>
<td></td>
</tr>
<tr>
<td>usb_lld_stop, 823</td>
<td></td>
</tr>
<tr>
<td>usb_lld_wakeup_host, 804</td>
<td></td>
</tr>
<tr>
<td>USB_MAX_ENDPOINTS, 801</td>
<td></td>
</tr>
<tr>
<td>USB_READY, 806</td>
<td></td>
</tr>
<tr>
<td>USB_SELECTED, 806</td>
<td></td>
</tr>
<tr>
<td>USB_SET_ADDRESS_ACK_HANDLING, 802</td>
<td></td>
</tr>
<tr>
<td>USB_SET_ADDRESS_MODE, 802</td>
<td></td>
</tr>
<tr>
<td>USB_STOP, 806</td>
<td></td>
</tr>
<tr>
<td>USB_SUSPENDED, 806</td>
<td></td>
</tr>
<tr>
<td>USB_UNINIT, 806</td>
<td></td>
</tr>
<tr>
<td>USB_USE_WAIT, 794</td>
<td></td>
</tr>
<tr>
<td>usbcallback_t, 804</td>
<td></td>
</tr>
<tr>
<td>usbConnectBus, 795</td>
<td></td>
</tr>
<tr>
<td>USBD1, 830</td>
<td></td>
</tr>
<tr>
<td>usbDisableEndpointsI, 812</td>
<td></td>
</tr>
<tr>
<td>usbDisconnectBus, 795</td>
<td></td>
</tr>
<tr>
<td>USBDriver, 804</td>
<td></td>
</tr>
<tr>
<td>usbep0state_t, 807</td>
<td></td>
</tr>
<tr>
<td>usbep_t, 804</td>
<td></td>
</tr>
<tr>
<td>usbepcallback_t, 805</td>
<td></td>
</tr>
<tr>
<td>usbepstatus_t, 806</td>
<td></td>
</tr>
<tr>
<td>usbevent_t, 807</td>
<td></td>
</tr>
<tr>
<td>usbeventcb_t, 805</td>
<td></td>
</tr>
<tr>
<td>usbgetdescriptor_t, 830</td>
<td></td>
</tr>
<tr>
<td>usbGetDriverStateI, 794</td>
<td></td>
</tr>
<tr>
<td>usbGetFrameNumberX, 795</td>
<td></td>
</tr>
<tr>
<td>usbGetReceiveStatusI, 796</td>
<td></td>
</tr>
<tr>
<td>usbGetReceiveTransactionSizeX, 797</td>
<td></td>
</tr>
<tr>
<td>usbGetTransmitStatusI, 796</td>
<td></td>
</tr>
<tr>
<td>usbInit, 809</td>
<td></td>
</tr>
<tr>
<td>usbInitEndpointI, 811</td>
<td></td>
</tr>
<tr>
<td>usbObjectInit, 809</td>
<td></td>
</tr>
<tr>
<td>usbReadSetup, 798</td>
<td></td>
</tr>
<tr>
<td>usbReceive, 814</td>
<td></td>
</tr>
<tr>
<td>usbreqhandler_t, 805</td>
<td></td>
</tr>
<tr>
<td>usbSetupTransfer, 798</td>
<td></td>
</tr>
<tr>
<td>usbStallTransmit, 817</td>
<td></td>
</tr>
<tr>
<td>usbStart, 810</td>
<td></td>
</tr>
<tr>
<td>usbStartReceive, 813</td>
<td></td>
</tr>
<tr>
<td>usbStartTransmit, 813</td>
<td></td>
</tr>
<tr>
<td>usbstate_t, 806</td>
<td></td>
</tr>
<tr>
<td>usbStop, 810</td>
<td></td>
</tr>
<tr>
<td>usbTransmit, 815</td>
<td></td>
</tr>
<tr>
<td>usbWakeupHost, 818</td>
<td></td>
</tr>
<tr>
<td>USB_ACTIVE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 806</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_BCD</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 790</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_BYTE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 790</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_CONFIGURATION</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 791</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_CONFIGURATION_SIZE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 791</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_DEVICE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 791</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_ENDPOINT</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 793</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_ENDPOINT_SIZE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 793</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_INTERFACE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 792</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_INTERFACE_ASSOCIATION</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 793</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_INTERFACE_ASSOCIATION_SIZE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 792</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_INTERFACE_SIZE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 792</td>
<td></td>
</tr>
<tr>
<td>USB_DESC_WORD</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 790</td>
<td></td>
</tr>
<tr>
<td>USB_EP0_ERROR</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 807</td>
<td></td>
</tr>
<tr>
<td>USB_EP0_IN_SENDING_STS</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 807</td>
<td></td>
</tr>
<tr>
<td>USB_EP0_IN_TX</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 807</td>
<td></td>
</tr>
<tr>
<td>USB_EP0_IN_WAITING_TX0</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 807</td>
<td></td>
</tr>
<tr>
<td>USB_EP0_OUT_RX</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 807</td>
<td></td>
</tr>
<tr>
<td>USB_EP0_OUT_WAITING_STS</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 807</td>
<td></td>
</tr>
<tr>
<td>USB_EP0_STATUS_STAGE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 801</td>
<td></td>
</tr>
<tr>
<td>USB_EP0_STP_WAITING</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 807</td>
<td></td>
</tr>
<tr>
<td>USB_EP_MODE_TYPE</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 793</td>
<td></td>
</tr>
<tr>
<td>USB_EP_MODE_TYPE_BULK</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 794</td>
<td></td>
</tr>
<tr>
<td>USB_EP_MODE_TYPE_CTRL</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 794</td>
<td></td>
</tr>
<tr>
<td>USB_EP_MODE_TYPE_INTR</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 794</td>
<td></td>
</tr>
<tr>
<td>USB_EP_MODE_TYPE_ISOC</td>
<td></td>
</tr>
<tr>
<td>USB Driver, 794</td>
<td></td>
</tr>
<tr>
<td>USB_EVENT_ADDRESS</td>
<td></td>
</tr>
</tbody>
</table>
USB Driver, 807
USB_EVENT_CONFIGURED
USB Driver, 807
USB_EVENT_RESET
USB Driver, 807
USB_EVENT_STALLED
USB Driver, 807
USB_EVENT_SUSPEND
USB Driver, 807
USB_EVENT_UNCONFIGURED
USB Driver, 807
USB_EVENT_WAKEUP
USB Driver, 807
usb_lld_clear_in
USB Driver, 829
usb_lld_clear_out
USB Driver, 829
usb_lld_connect_bus
USB Driver, 803
usb_lld_disable_endpoints
USB Driver, 825
usb_lld_disconnect_bus
USB Driver, 803
usb_lld_get_frame_number
USB Driver, 802
usb_lld_get_status_in
USB Driver, 826
usb_lld_get_status_out
USB Driver, 825
usb_lld_get_transaction_size
USB Driver, 803
usb_lld_init
USB Driver, 822
usb_lld_init_endpoint
USB Driver, 824
usb_lld_prepare_receive
USB Driver, 827
usb_lld_prepare_transmit
USB Driver, 827
usb_lld_read_setup
USB Driver, 826
usb_lld_reset
USB Driver, 823
usb_lld_set_address
USB Driver, 824
usb_lld_stall_in
USB Driver, 829
usb_lld_stall_out
USB Driver, 828
usb_lld_start
USB Driver, 822
usb_lld_start_in
USB Driver, 828
usb_lld_start_out
USB Driver, 827
usb_lld_stop
USB Driver, 823
usb_lld_wakeup_host
USB Driver, 804
USB_MAX_ENDPOINTS
USB Driver, 801
USB_READY
USB Driver, 806
USB_SELECTED
USB Driver, 806
USB_SET_ADDRESS_ACK_HANDLING
USB Driver, 802
USB_SET_ADDRESS_MODE
USB Driver, 802
USB_STOP
USB Driver, 806
USB_SUSPENDED
USB Driver, 806
USB_UNINIT
USB Driver, 806
USB_USE_WAIT
USB Driver, 794
usbcallback_t
USB Driver, 804
USBConfig, 1095
event_cb, 1097
get_descriptor_cb, 1097
requests_hook_cb, 1097
sof_cb, 1097
usbConnectBus
USB Driver, 795
USBD1
USB Driver, 830
USBDescriptor, 1098
ud_size, 1098
ud_string, 1098
usbDisableEndpointsI
USB Driver, 812
usbDisconnectBus
USB Driver, 795
USBDriver, 1099
address, 1102
config, 1100
configuration, 1102
ep0endcb, 1102
ep0n, 1102
ep0next, 1101
ep0state, 1101
epc, 1101
in_params, 1101
out_params, 1101
receiving, 1100
saved_state, 1103
setup, 1102
state, 1100
status, 1102
transmitting, 1100
USB Driver, 804
USBEndpointConfig, 1103
ep_mode, 1105
in_cb, 1105
in_maxsize, 1106
in_state, 1106
out_cb, 1106
out_maxsize, 1106
out_state, 1106
setup_cb, 1105

usbep0state_t  
USB Driver, 807
usbep_t  
USB Driver, 804
usbepcallback_t  
USB Driver, 805
usbepstatus_t  
USB Driver, 806
usbevent_t  
USB Driver, 807
usbeventcb_t  
USB Driver, 805
usbgetdescriptor_t  
USB Driver, 830

usbGetDriverStateI  
USB Driver, 809
usbGetFrameNumberX  
USB Driver, 795
usbGetReceiveStatusI  
USB Driver, 796
usbGetReceiveTransactionSizeX  
USB Driver, 797
usbGetTransmitStatusI  
USB Driver, 796
USBInEndpointState, 1107
txbuf, 1108
txcnt, 1107
txsize, 1107
usbInit  
USB Driver, 809
usbInitEndpointI  
USB Driver, 811
usbObjectInit  
USB Driver, 809
USBOutEndpointState, 1108
rxbuf, 1109
rxcnt, 1109
rxsize, 1109
thread, 1109
usbReplace  
SerialUSBDriver, 1075
usbReadSetup  
USB Driver, 798
usbReceive  
USB Driver, 814
usbreqhandler_t  
USB Driver, 805
usbSetupTransfer  
USB Driver, 798
usbStallReceiveI  
USB Driver, 817
usbStartI  
USB Driver, 810
usbStartReceiveI  
USB Driver, 813
usbStartTransmitI  
USB Driver, 813
usbstate_t  
USB Driver, 806
usbStop  
USB Driver, 810
usbTransmit  
USB Driver, 815
usbWakeupHost  
USB Driver, 818
used_space  
MFSDriver, 1038

vmt  
BaseAsynchronousChannel, 928
BaseBlockDevice, 932
BaseBufferedStream, 935
BaseChannel, 938
BaseFlash, 942
BaseObject, 944
BasePersistentStorage, 946
BaseSequentialStream, 949
BufferedStreamAdapter, 952
FileStream, 969
hal_elf_driver, 989
hal_sio_driver, 995
MemoryStream, 1028
MMCDevice, 1043
MMCSDBlockDevice, 1047
NullStream, 1050
RTCDriver, 1062
SDCDriver, 1067
SerialDriver, 1072
SerialUSBDriver, 1078
SNORDriver, 1085

wait  
MMC over SPI Driver, 423
waiting  
io_buffers_queue, 1015
wakeup_event  
CANDriver, 956
WDG Driver, 834

plat_WDG_DEFAULT, 835
plat_WDG_DEFAULT, 835
wdw_lld_init, 838
wdw_lld_reset, 839
wdw_lld_start, 839
wdw_lld_stop, 839
WDG_READY, 836
WDG_STOP, 836
WDG_UNINIT, 836
WDGDriver, 835

ChibiOS/HAL
<table>
<thead>
<tr>
<th>Index entry</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>wdgInit</td>
<td>836</td>
</tr>
<tr>
<td>wdgReset</td>
<td>838</td>
</tr>
<tr>
<td>wdgResetI</td>
<td>835</td>
</tr>
<tr>
<td>wdgStart</td>
<td>836</td>
</tr>
<tr>
<td>wdgstate_t</td>
<td>836</td>
</tr>
<tr>
<td>wdgStop</td>
<td>837</td>
</tr>
<tr>
<td>wdg_lld_init</td>
<td>838</td>
</tr>
<tr>
<td>wdg_lld_reset</td>
<td>839</td>
</tr>
<tr>
<td>wdg_lld_start</td>
<td>839</td>
</tr>
<tr>
<td>wdg_lld_stop</td>
<td>839</td>
</tr>
<tr>
<td>WDG READY</td>
<td>836</td>
</tr>
<tr>
<td>WDG_STOP</td>
<td>836</td>
</tr>
<tr>
<td>WDG_UNINIT</td>
<td>836</td>
</tr>
<tr>
<td>WDGDriver</td>
<td>1110</td>
</tr>
<tr>
<td>config</td>
<td>1111</td>
</tr>
<tr>
<td>state</td>
<td>1111</td>
</tr>
<tr>
<td>WDG Driver</td>
<td>835</td>
</tr>
<tr>
<td>wdgInit</td>
<td>836</td>
</tr>
<tr>
<td>wdgReset</td>
<td>838</td>
</tr>
<tr>
<td>wdgResetI</td>
<td>835</td>
</tr>
<tr>
<td>wdgStart</td>
<td>836</td>
</tr>
<tr>
<td>wdgstate_t</td>
<td>836</td>
</tr>
<tr>
<td>wdgStop</td>
<td>837</td>
</tr>
<tr>
<td>width_cb</td>
<td>1111</td>
</tr>
<tr>
<td>ICUConfig</td>
<td>1011</td>
</tr>
<tr>
<td>WSPI Driver</td>
<td>841</td>
</tr>
<tr>
<td>_wspi_error_code</td>
<td>849</td>
</tr>
<tr>
<td>_wspi_isr_code</td>
<td>849</td>
</tr>
<tr>
<td>_wspi_wakeup_isr</td>
<td>848</td>
</tr>
<tr>
<td>PLATFORM_WSPI_USE_WSPI1</td>
<td>850</td>
</tr>
<tr>
<td>WSPI COMPLETE</td>
<td>852</td>
</tr>
<tr>
<td>wspi_lld_command</td>
<td>865</td>
</tr>
<tr>
<td>wspi_lld_config_fields</td>
<td>850</td>
</tr>
<tr>
<td>wspi_lld_driver_fields</td>
<td>850</td>
</tr>
<tr>
<td>wspi_lld_init</td>
<td>864</td>
</tr>
<tr>
<td>wspi_lld_map_flash</td>
<td>866</td>
</tr>
<tr>
<td>wspi_lld_receive</td>
<td>866</td>
</tr>
<tr>
<td>wspi_lld_send</td>
<td>865</td>
</tr>
<tr>
<td>wspi_lld_start</td>
<td>864</td>
</tr>
<tr>
<td>wspi_lld_stop</td>
<td>864</td>
</tr>
<tr>
<td>wspi_lld_unmap_flash</td>
<td>867</td>
</tr>
<tr>
<td>WSPI_MEMMAP</td>
<td>852</td>
</tr>
<tr>
<td>WSPI_READY</td>
<td>852</td>
</tr>
<tr>
<td>WSPI_RECEIVE</td>
<td>852</td>
</tr>
<tr>
<td>WSPI_SEND</td>
<td>852</td>
</tr>
<tr>
<td>WSPI_STOP</td>
<td>852</td>
</tr>
<tr>
<td>WSPI_UNINIT</td>
<td>852</td>
</tr>
<tr>
<td>WSPI_USE_MUTUAL_EXCLUSION</td>
<td>845</td>
</tr>
<tr>
<td>wspiAcquireBus</td>
<td>862</td>
</tr>
<tr>
<td>wspiCallbackBus</td>
<td>851</td>
</tr>
<tr>
<td>wspiCommand</td>
<td>859</td>
</tr>
<tr>
<td>WSPICfg</td>
<td>851</td>
</tr>
<tr>
<td>WSPID1</td>
<td>867</td>
</tr>
<tr>
<td>WSPIDriver</td>
<td>851</td>
</tr>
<tr>
<td>wspiInit</td>
<td>852</td>
</tr>
<tr>
<td>wspiMapFlash</td>
<td>861</td>
</tr>
<tr>
<td>wspiMapFlashI</td>
<td>847</td>
</tr>
<tr>
<td>wspiObjectInit</td>
<td>852</td>
</tr>
<tr>
<td>wspiReceive</td>
<td>860</td>
</tr>
<tr>
<td>wspiReleaseBus</td>
<td>863</td>
</tr>
<tr>
<td>wspiSend</td>
<td>860</td>
</tr>
<tr>
<td>wspiStart</td>
<td>854</td>
</tr>
<tr>
<td>wspiStartCommand</td>
<td>856</td>
</tr>
<tr>
<td>wspiStartCommandI</td>
<td>845</td>
</tr>
<tr>
<td>wspiStartReceive</td>
<td>858</td>
</tr>
<tr>
<td>wspiStartReceivevel</td>
<td>846</td>
</tr>
<tr>
<td>wspiStartSend</td>
<td>857</td>
</tr>
<tr>
<td>wspiStartSendI</td>
<td>846</td>
</tr>
<tr>
<td>wspistate_t</td>
<td>851</td>
</tr>
<tr>
<td>wspiStop</td>
<td>855</td>
</tr>
<tr>
<td>wspiUnmapFlash</td>
<td>862</td>
</tr>
<tr>
<td>wspiUnmapFlashI</td>
<td>848</td>
</tr>
<tr>
<td>wspi_command_t</td>
<td>1111</td>
</tr>
<tr>
<td>addr</td>
<td>1112</td>
</tr>
<tr>
<td>alt</td>
<td>1112</td>
</tr>
<tr>
<td>cfg</td>
<td>1112</td>
</tr>
<tr>
<td>cmd</td>
<td>1112</td>
</tr>
<tr>
<td>dummy</td>
<td>1112</td>
</tr>
<tr>
<td>WSPI COMPLETE</td>
<td>852</td>
</tr>
<tr>
<td>WSPI Driver</td>
<td>852</td>
</tr>
<tr>
<td>wspi_lld_command</td>
<td>865</td>
</tr>
<tr>
<td>wspi_lld_config_fields</td>
<td>850</td>
</tr>
<tr>
<td>wspi_lld_driver_fields</td>
<td>850</td>
</tr>
<tr>
<td>wspi_lld_init</td>
<td>864</td>
</tr>
<tr>
<td>wspi_lld_map_flash</td>
<td>866</td>
</tr>
<tr>
<td>WSPI Driver</td>
<td>866</td>
</tr>
<tr>
<td>wspi_lld_receive</td>
<td>866</td>
</tr>
<tr>
<td>wspi_lld_send</td>
<td>865</td>
</tr>
<tr>
<td>WSPI Driver</td>
<td>865</td>
</tr>
<tr>
<td>wspi_lld_start</td>
<td>864</td>
</tr>
<tr>
<td>wspi_lld_stop</td>
<td>864</td>
</tr>
<tr>
<td>wspi_lld_unmap_flash</td>
<td>867</td>
</tr>
<tr>
<td>WSPI Driver</td>
<td>867</td>
</tr>
<tr>
<td>WSPI_MEMMAP</td>
<td>852</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
WSPI Driver, 852
WSPI_READY
WSPI Driver, 852
WSPI_RECEIVE
WSPI Driver, 852
WSPI_SEND
WSPI Driver, 852
WSPI_STOP
WSPI Driver, 852
WSPI_UNINIT
WSPI Driver, 852
WSPI_USE_MUTUAL_EXCLUSION
Configuration, 885
WSPI Driver, 845
WSPI_USE_WAIT
Configuration, 884
WSPI Driver, 844
wspiAcquireBus
WSPI Driver, 862
wspicableback_t
WSPI Driver, 851
wspiCommand
WSPI Driver, 859
WSPIConfig
WSPI Driver, 851
WSPID1
WSPI Driver, 867
WSPIDriver
WSPI Driver, 851
wspiInit
WSPI Driver, 852
wspiMapFlash
WSPI Driver, 861
wspiMapFlashI
WSPI Driver, 847
wspiObjectInit
WSPI Driver, 852
wspiReceive
WSPI Driver, 860
wspiReleaseBus
WSPI Driver, 863
wspiSend
WSPI Driver, 860
wspiStart
WSPI Driver, 854
wspiStartCommand
WSPI Driver, 856
wspiStartCommandI
WSPI Driver, 845
wspiStartReceive
WSPI Driver, 858
wspiStartReceiveI
WSPI Driver, 846
wspiStartSend
WSPI Driver, 857
wspiStartSendI
WSPI Driver, 846
wspistate_t

WSPI Driver, 851
wspiStop
WSPI Driver, 855
wspiUnmapFlash
WSPI Driver, 862
wspiUnmapFlashI
WSPI Driver, 848

year
RTCDateTime, 1060