7.2.4.3 adcIsBufferComplete .................................................. 33
7.2.4.4 _adc_reset_i ................................................................. 34
7.2.4.5 _adc_reset_s ................................................................. 34
7.2.4.6 _adc_wakeup_isr ........................................................... 35
7.2.4.7 _adc_timeout_isr .......................................................... 35
7.2.4.8 _adc_isr_half_code ....................................................... 36
7.2.4.9 _adc_isr_full_code ....................................................... 36
7.2.4.10 _adc_isr_error_code ..................................................... 37
7.2.4.11 PLATFORM_ADC_USE_ADC1 ................................................. 38
7.2.4.12 adc_lld_driver_fields ................................................... 38
7.2.4.13 adc_lld_config_fields .................................................. 38
7.2.4.14 adc_lld_configuration_group_fields ................................... 39

7.2.5 Typedef Documentation .................................................. 39
7.2.5.1 ADCDriver ................................................................. 39
7.2.5.2 ADCConfig ................................................................. 39
7.2.5.3 ADCConversionGroup ..................................................... 39
7.2.5.4 adccallback_t ............................................................... 39
7.2.5.5 adccallback_t ............................................................... 40
7.2.5.6 adcsample_t ............................................................... 40
7.2.5.7 adc_channels_num_t ...................................................... 40

7.2.6 Enumeration Type Documentation ....................................... 40
7.2.6.1 adcstate_t ................................................................. 40
7.2.6.2 adccallback_t ............................................................... 41

7.2.7 Function Documentation .................................................. 41
7.2.7.1 adcInit() ................................................................. 41
7.2.7.2 adcObjectInit() .......................................................... 42
7.2.7.3 adcStart() ................................................................. 42
7.2.7.4 adcStop() ................................................................. 43
7.2.7.5 adcStartConversion() .................................................... 44
7.2.7.6 adcStartConversionI() ................................................... 45
7.2.7.7 adcStopConversion() ..................................................... 46
7.2.7.8 adcStopConversionI() .................................................... 47
7.2.7.9 adcConvert() ............................................................... 47
7.2.7.10 adcAcquireBus() .......................................................... 48
7.2.7.11 adcReleaseBus() .......................................................... 49
7.2.7.12 adc_lld_init() ............................................................. 50
7.2.7.13 adc_lld_start() ............................................................ 50
7.2.7.14 adc_lld_stop() ............................................................. 50
7.2.7.15 adc_lld_start_conversion() ............................................. 51
7.2.7.16 adc_lld_stop_conversion() .............................................. 51

7.2.8 Variable Documentation .................................................. 51
7.2.8.1 ADCD1 ................................................................. 52
7.3 I/O Buffers Queues

7.3.1 Detailed Description

7.3.2 Macro Definition Documentation

7.3.2.1 BUFFERS_CHUNKS_SIZE

7.3.2.2 BQ_BUFFER_SIZE

7.3.2.3 bqSizeX

7.3.2.4 bqSpaceI

7.3.2.5 bqGetLinkX

7.3.2.6 bqSetLinkX

7.3.2.7 bqIsSuspendedX

7.3.2.8 bqSuspendI

7.3.2.9 bqResumeX

7.3.2.10 ibqIsEmptyI

7.3.2.11 ibqIsFullI

7.3.2.12 obqIsEmptyI

7.3.2.13 obqIsFullI

7.3.3 Typedef Documentation

7.3.3.1 io_buffers_queue_t

7.3.3.2 bqnotify_t

7.3.3.3 input_buffers_queue_t

7.3.3.4 output_buffers_queue_t

7.3.4 Function Documentation

7.3.4.1 ibqObjectInit()

7.3.4.2 ibqResetI()

7.3.4.3 ibqGetEmptyBufferI()

7.3.4.4 ibqPostFullBufferI()

7.3.4.5 ibqGetFullBufferTimeout() .

7.3.4.6 ibqGetFullBufferTimeoutS() .

7.3.4.7 ibqReleaseEmptyBuffer()

7.3.4.8 ibqReleaseEmptyBufferS()

7.3.4.9 ibqGetTimeout() .

7.3.4.10 ibqReadTimeout() .

7.3.4.11 obqObjectInit() .

7.3.4.12 obqResetI()

7.3.4.13 obqGetFullBufferI()

7.3.4.14 obqReleaseEmptyBufferI()

7.3.4.15 obqGetEmptyBufferTimeout() .

7.3.4.16 obqGetEmptyBufferTimeoutS() .

7.3.4.17 obqPostFullBuffer() .

7.3.4.18 obqPostFullBufferS() .

7.3.4.19 obqPutTimeout() .

7.3.4.20 obqWriteTimeout() .
<p>| 7.4 CAN Driver                                      | 80 |
| 7.4.1 Detailed Description                          | 80 |
| 7.4.2 Driver State Machine                         | 80 |
| 7.4.3 Macro Definition Documentation               | 83 |
| 7.4.3.1 CAN_LIMIT_WARNING                          | 83 |
| 7.4.3.2 CAN_LIMIT_ERROR                             | 83 |
| 7.4.3.3 CAN_BUS_OFF_ERROR                           | 84 |
| 7.4.3.4 CAN_FRAMING_ERROR                           | 84 |
| 7.4.3.5 CAN_OVERFLOW_ERROR                          | 84 |
| 7.4.3.6 CAN_ANY_MAILBOX                             | 84 |
| 7.4.3.7 CAN_USE_SLEEP_MODE                          | 84 |
| 7.4.3.8 CAN_ENFORCE_USE_CALLBACKS                   | 84 |
| 7.4.3.9 CAN_MAILBOX_TO_MASK                         | 85 |
| 7.4.3.10 canTransmit                                | 85 |
| 7.4.3.11 canReceive                                 | 85 |
| 7.4.3.12 _can_tx_empty_isr                          | 85 |
| 7.4.3.13 _can_rx_full_isr                           | 86 |
| 7.4.3.14 _can_wakeup_isr                            | 86 |
| 7.4.3.15 _can_error_isr                             | 86 |
| 7.4.3.16 CAN_TX_MAILBOXES                           | 86 |
| 7.4.3.17 CAN_RX_MAILBOXES                           | 87 |
| 7.4.3.18 PLATFORM_CAN_USE_CAN1                      | 87 |
| 7.4.4 Typedef Documentation                         | 87 |
| 7.4.4.1 CANDriver                                   | 87 |
| 7.4.4.2 canmbx_t                                    | 87 |
| 7.4.4.3 can_callback_t                              | 87 |
| 7.4.5 Enumeration Type Documentation                | 88 |
| 7.4.5.1 canstate_t                                  | 88 |
| 7.4.6 Function Documentation                        | 88 |
| 7.4.6.1 canInit()                                   | 88 |
| 7.4.6.2 canObjectInit()                             | 89 |
| 7.4.6.3 canStart()                                  | 90 |
| 7.4.6.4 canStop()                                   | 90 |
| 7.4.6.5 canTryTransmit()                            | 91 |
| 7.4.6.6 canTryReceiveI()                            | 92 |
| 7.4.6.7 canTryAbortX()                              | 93 |
| 7.4.6.8 canTransmitTimeout()                        | 94 |
| 7.4.6.9 canReceiveTimeout()                         | 95 |
| 7.4.6.10 canSleep()                                 | 96 |
| 7.4.6.11 canWakeup()                                | 97 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.6.12 can_lld_init()</td>
<td>98</td>
</tr>
<tr>
<td>7.4.6.13 can_lld_start()</td>
<td>99</td>
</tr>
<tr>
<td>7.4.6.14 can_lld_stop()</td>
<td>100</td>
</tr>
<tr>
<td>7.4.6.15 can_lld_is_tx_empty()</td>
<td>100</td>
</tr>
<tr>
<td>7.4.6.16 can_lld_transmit()</td>
<td>101</td>
</tr>
<tr>
<td>7.4.6.17 can_lld_is_rx_nonempty()</td>
<td>101</td>
</tr>
<tr>
<td>7.4.6.18 can_lld_receive()</td>
<td>102</td>
</tr>
<tr>
<td>7.4.6.19 can_lld_abort()</td>
<td>102</td>
</tr>
<tr>
<td>7.4.6.20 can_lld_sleep()</td>
<td>102</td>
</tr>
<tr>
<td>7.4.6.21 can_lld_wakeup()</td>
<td>103</td>
</tr>
<tr>
<td>7.4.7 Variable Documentation</td>
<td>103</td>
</tr>
<tr>
<td>7.4.7.1 CAND1</td>
<td>103</td>
</tr>
<tr>
<td>7.5 Crypto Driver</td>
<td>104</td>
</tr>
<tr>
<td>7.5.1 Detailed Description</td>
<td>104</td>
</tr>
<tr>
<td>7.5.2 Macro Definition Documentation</td>
<td>109</td>
</tr>
<tr>
<td>7.5.2.1 HAL_CRY_USE_FALLBACK [1/2]</td>
<td>109</td>
</tr>
<tr>
<td>7.5.2.2 HAL_CRY_USE_FALLBACK [2/2]</td>
<td>110</td>
</tr>
<tr>
<td>7.5.2.3 HAL_CRY_ENFORCE_FALLBACK</td>
<td>110</td>
</tr>
<tr>
<td>7.5.2.4 PLATFORM_CRY_USE_CRY1</td>
<td>110</td>
</tr>
<tr>
<td>7.5.3 Typedef Documentation</td>
<td>110</td>
</tr>
<tr>
<td>7.5.3.1 bitsize_t</td>
<td>110</td>
</tr>
<tr>
<td>7.5.3.2 crykey_t</td>
<td>111</td>
</tr>
<tr>
<td>7.5.3.3 CRYDriver</td>
<td>111</td>
</tr>
<tr>
<td>7.5.4 Enumeration Type Documentation</td>
<td>111</td>
</tr>
<tr>
<td>7.5.4.1 crystate_t</td>
<td>111</td>
</tr>
<tr>
<td>7.5.4.2 cryerror_t</td>
<td>111</td>
</tr>
<tr>
<td>7.5.4.3 cryalgorithm_t</td>
<td>112</td>
</tr>
<tr>
<td>7.5.5 Function Documentation</td>
<td>112</td>
</tr>
<tr>
<td>7.5.5.1 cryInit()</td>
<td>112</td>
</tr>
<tr>
<td>7.5.5.2 cryObjectInit()</td>
<td>113</td>
</tr>
<tr>
<td>7.5.5.3 cryStart()</td>
<td>113</td>
</tr>
<tr>
<td>7.5.5.4 cryStop()</td>
<td>114</td>
</tr>
<tr>
<td>7.5.5.5 cryLoadAESTransientKey()</td>
<td>115</td>
</tr>
<tr>
<td>7.5.5.6 cryEncryptAES()</td>
<td>116</td>
</tr>
<tr>
<td>7.5.5.7 cryDecryptAES()</td>
<td>117</td>
</tr>
<tr>
<td>7.5.5.8 cryEncryptAES_ECB()</td>
<td>118</td>
</tr>
<tr>
<td>7.5.5.9 cryDecryptAES_ECB()</td>
<td>119</td>
</tr>
<tr>
<td>7.5.5.10 cryEncryptAES_CBC()</td>
<td>120</td>
</tr>
<tr>
<td>7.5.5.11 cryDecryptAES_CBC()</td>
<td>121</td>
</tr>
<tr>
<td>7.5.5.12 cryEncryptAES_CFB()</td>
<td>122</td>
</tr>
<tr>
<td>7.5.5.13 cryDecryptAES_CFB()</td>
<td>123</td>
</tr>
<tr>
<td>7.5.5.14 cryEncryptAES_CTR()</td>
<td>124</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>cryDecryptAES_CTR()</td>
<td>125</td>
</tr>
<tr>
<td>cryEncryptAES_GCM()</td>
<td>126</td>
</tr>
<tr>
<td>cryDecryptAES_GCM()</td>
<td>127</td>
</tr>
<tr>
<td>cryLoadDESTransientKey()</td>
<td>129</td>
</tr>
<tr>
<td>cryEncryptDES()</td>
<td>130</td>
</tr>
<tr>
<td>cryDecryptDES()</td>
<td>131</td>
</tr>
<tr>
<td>cryEncryptDES_ECB()</td>
<td>132</td>
</tr>
<tr>
<td>cryDecryptDES_ECB()</td>
<td>133</td>
</tr>
<tr>
<td>cryEncryptDES_CBC()</td>
<td>134</td>
</tr>
<tr>
<td>cryDecryptDES_CBC()</td>
<td>135</td>
</tr>
<tr>
<td>crySHA1Init()</td>
<td>136</td>
</tr>
<tr>
<td>crySHA1Update()</td>
<td>137</td>
</tr>
<tr>
<td>crySHA1Final()</td>
<td>138</td>
</tr>
<tr>
<td>crySHA256Init()</td>
<td>139</td>
</tr>
<tr>
<td>crySHA256Update()</td>
<td>140</td>
</tr>
<tr>
<td>crySHA256Final()</td>
<td>141</td>
</tr>
<tr>
<td>crySHA512Init()</td>
<td>142</td>
</tr>
<tr>
<td>crySHA512Update()</td>
<td>143</td>
</tr>
<tr>
<td>crySHA512Final()</td>
<td>144</td>
</tr>
<tr>
<td>cryLoadHMACTransientKey()</td>
<td>145</td>
</tr>
<tr>
<td>cryHMACSHA256Init()</td>
<td>146</td>
</tr>
<tr>
<td>cryHMACSHA256Update()</td>
<td>147</td>
</tr>
<tr>
<td>cryHMACSHA256Final()</td>
<td>148</td>
</tr>
<tr>
<td>cryHMACSHA512Init()</td>
<td>149</td>
</tr>
<tr>
<td>cryHMACSHA512Update()</td>
<td>150</td>
</tr>
<tr>
<td>cryHMACSHA512Final()</td>
<td>151</td>
</tr>
<tr>
<td>cry_lld_init()</td>
<td>152</td>
</tr>
<tr>
<td>cry_lld_start()</td>
<td>153</td>
</tr>
<tr>
<td>cry_lld_stop()</td>
<td>154</td>
</tr>
<tr>
<td>cry_lld_aes_loadkey()</td>
<td>155</td>
</tr>
<tr>
<td>cry_lld_encrypt_AES()</td>
<td>156</td>
</tr>
<tr>
<td>cry_lld_decrypt_AES()</td>
<td>157</td>
</tr>
<tr>
<td>cry_lld_encrypt_AES_ECB()</td>
<td>158</td>
</tr>
<tr>
<td>cry_lld_decrypt_AES_ECB()</td>
<td>159</td>
</tr>
<tr>
<td>cry_lld_encrypt_AES_CBC()</td>
<td>160</td>
</tr>
<tr>
<td>cry_lld_decrypt_AES_CBC()</td>
<td>161</td>
</tr>
<tr>
<td>cry_lld_encrypt_AES_CFB()</td>
<td>162</td>
</tr>
<tr>
<td>cry_lld_decrypt_AES_CFB()</td>
<td>163</td>
</tr>
<tr>
<td>cry_lld_encrypt_AES_GCM()</td>
<td>164</td>
</tr>
<tr>
<td>cry_lld_decrypt_AES_GCM()</td>
<td>165</td>
</tr>
</tbody>
</table>
7.5.5.57 cry_lld_des_loadkey() .......................................................... 164
7.5.5.58 cry_lld_encrypt_DES() ......................................................... 164
7.5.5.59 cry_lld_decrypt_DES() .......................................................... 165
7.5.5.60 cry_lld_encrypt_DES_ECB() ..................................................... 166
7.5.5.61 cry_lld_decrypt_DES_ECB() ..................................................... 167
7.5.5.62 cry_lld_encrypt_DES_CBC() ..................................................... 168
7.5.5.63 cry_lld_decrypt_DES_CBC() ..................................................... 169
7.5.5.64 cry_lld_SHA1_init() .............................................................. 169
7.5.5.65 cry_lld_SHA1_update() ........................................................... 170
7.5.5.66 cry_lld_SHA1_final() ............................................................. 171
7.5.5.67 cry_lld_SHA256_init() ........................................................... 172
7.5.5.68 cry_lld_SHA256_update() ........................................................ 172
7.5.5.69 cry_lld_SHA256_final() ......................................................... 173
7.5.5.70 cry_lld_SHA512_init() ............................................................ 173
7.5.5.71 cry_lld_SHA512_update() ....................................................... 174
7.5.5.72 cry_lld_SHA512_final() .......................................................... 175
7.5.5.73 cry_lld_hmac_loadkey() .......................................................... 175
7.5.5.74 cry_lld_HMACSHA256_init() .................................................... 176
7.5.5.75 cry_lld_HMACSHA256_update() ................................................ 176
7.5.5.76 cry_lld_HMACSHA256_final() .................................................. 177
7.5.5.77 cry_lld_HMACSHA512_init() .................................................... 178
7.5.5.78 cry_lld_HMACSHA512_update() ................................................ 178
7.5.5.79 cry_lld_HMACSHA512_final() .................................................. 179
7.5.6 Variable Documentation ............................................................. 180
7.5.6.1 CRYD1 ................................................................................. 180
7.6 DAC Driver .................................................................................... 181
7.6.1 Detailed Description ..................................................................... 181
7.6.2 Macro Definition Documentation .................................................. 183
7.6.2.1 DAC_USE_WAIT .................................................................... 183
7.6.2.2 DAC_USE_MUTUAL_EXCLUSION .............................................. 184
7.6.2.3 dacIsBufferComplete .............................................................. 184
7.6.2.4 _dac_wait_s ........................................................................ 184
7.6.2.5 _dac_reset_i ........................................................................ 185
7.6.2.6 _dac_reset_s ........................................................................ 185
7.6.2.7 _dac_wakeup_isr .................................................................. 186
7.6.2.8 _dac_timeout_isr .................................................................. 186
7.6.2.9 _dac_isr_half_code ............................................................... 187
7.6.2.10 _dac_isr_full_code ............................................................... 187
7.6.2.11 _dac_isr_error_code ............................................................. 188
7.6.2.12 DAC_MAX_CHANNELS ............................................................. 189
7.6.2.13 PLATFORM_DAC_USE_DAC1 ................................................. 189
7.6.2.14 dac_lld_driver_fields ............................................................... 189
7.6.2.15 dac_lld_config_fields ........................................... 189
7.6.2.16 dac_lld_conversion_group_fields ............................... 189

7.6.3 Typedef Documentation ........................................... 190
7.6.3.1 DACDriver ....................................................... 190
7.6.3.2 DACConfig ...................................................... 190
7.6.3.3 DACConversionGroup ........................................... 190
7.6.3.4 daccallback_t .................................................. 190
7.6.3.5 dacerrorcallback_t ............................................. 190
7.6.3.6 dacchannel_t ................................................... 191
7.6.3.7 dacsample_t .................................................... 191

7.6.4 Enumeration Type Documentation ................................. 191
7.6.4.1 dacstate_t ...................................................... 191
7.6.4.2 dacerror_t ..................................................... 192

7.6.5 Function Documentation ......................................... 192
7.6.5.1 dacInit() ....................................................... 192
7.6.5.2 dacObjectInit() ................................................ 192
7.6.5.3 dacStart() ..................................................... 193
7.6.5.4 dacStop() ....................................................... 194
7.6.5.5 dacPutChannelX() ............................................. 195
7.6.5.6 dacStartConversion() .......................................... 196
7.6.5.7 dacStartConversionI() .......................................... 196
7.6.5.8 dacStopConversion() ........................................... 197
7.6.5.9 dacStopConversionI() .......................................... 198
7.6.5.10 dacConvert() .................................................. 199
7.6.5.11 dacAcquireBus() ............................................... 200
7.6.5.12 dacReleaseBus() ............................................... 201
7.6.5.13 dac_lld_init() ................................................ 202
7.6.5.14 dac_lld_start() .............................................. 202
7.6.5.15 dac_lld_stop() ................................................ 202
7.6.5.16 dac_lld_put_channel() ....................................... 203
7.6.5.17 dac_lld_start_conversion() ................................... 203
7.6.5.18 dac_lld_stop_conversion() ................................... 204

7.6.6 Variable Documentation .......................................... 204
7.6.6.1 DACD1 .......................................................... 204

7.7 EFL Driver .......................................................... 205
7.7.1 Detailed Description .............................................. 205

7.7.2 Macro Definition Documentation ................................. 206
7.7.2.1 _efl_flash_methods_alone ..................................... 206
7.7.2.2 _efl_flash_methods ........................................... 207
7.7.2.3 _efl_driver_data ............................................. 207
7.7.2.4 PLATFORM_EFL_USE_EFL .................................... 207
7.7.2.5 efl_lld_driver_fields ....................................... 207
7.8.4 Typedef Documentation .................................................. 225
  7.8.4.1 flash_offset_t .................................................. 226
  7.8.4.2 flash_sector_t .................................................. 226
7.8.5 Enumeration Type Documentation ...................................... 226
  7.8.5.1 flash_state_t .................................................. 226
  7.8.5.2 flash_error_t .................................................. 226
7.8.6 Function Documentation ................................................ 226
  7.8.6.1 flashWaitErase() ............................................... 226
  7.8.6.2 flashGetSectorOffset() ........................................ 227
  7.8.6.3 flashGetSectorSize() ........................................... 227
7.9 GPT Driver ................................................................. 228
  7.9.1 Detailed Description ............................................... 228
  7.9.2 Driver State Machine ................................................ 228
  7.9.3 GPT Operations .................................................... 229
  7.9.4 Macro Definition Documentation .................................... 231
    7.9.4.1 gptChangeInterval() ........................................... 231
    7.9.4.2 gptGetIntervalX() ............................................. 231
    7.9.4.3 gptGetCounterX() ............................................. 232
    7.9.4.4 _gpt_isr_invoke_cb .......................................... 233
    7.9.4.5 PLATFORM_GPT_USE_GPT1 .................................... 233
    7.9.4.6 gpt_lld_change_interval .................................... 233
  7.9.5 Typedef Documentation ............................................... 234
    7.9.5.1 GPTDriver .................................................. 234
    7.9.5.2 gptcallback_t ............................................... 234
    7.9.5.3 gptfreq_t .................................................. 235
    7.9.5.4 gptcnt_t .................................................. 235
  7.9.6 Enumeration Type Documentation .................................... 235
    7.9.6.1 gptstate_t .................................................. 235
  7.9.7 Function Documentation ............................................. 235
    7.9.7.1 gptInit() .................................................. 235
    7.9.7.2 gptObjectInit() ............................................. 236
    7.9.7.3 gptStart() .................................................. 236
    7.9.7.4 gptStop() .................................................. 237
    7.9.7.5 gptChangeInterval() ........................................ 238
    7.9.7.6 gptStartContinuous() ...................................... 239
    7.9.7.7 gptStartContinuousI() .................................... 240
    7.9.7.8 gptStartOneShot() .......................................... 240
    7.9.7.9 gptStartOneShotI() ......................................... 241
    7.9.7.10 gptStopTimer() ............................................ 242
    7.9.7.11 gptStopTimerI() ............................................ 242
    7.9.7.12 gptPolledDelay() .......................................... 243
    7.9.7.13 gpt_lld_init() ............................................. 244
7.9.7.14 gpt_lld_start() ......................................................... 244
7.9.7.15 gpt_lld_stop() ....................................................... 244
7.9.7.16 gpt_lld_start_timer() ............................................. 245
7.9.7.17 gpt_lld_stop_timer() ............................................. 245
7.9.7.18 gpt_lld_polled_delay() ........................................... 246
7.9.8 Variable Documentation ............................................. 246
7.9.8.1 GPTD1 ................................................................. 246
7.10 I2C Driver ............................................................... 247
7.10.1 Detailed Description ............................................... 247
7.10.2 Driver State Machine ............................................. 247
7.10.3 Macro Definition Documentation .................................. 250
7.10.3.1 I2C_NO_ERROR ...................................................... 250
7.10.3.2 I2C_BUS_ERROR ..................................................... 250
7.10.3.3 I2C_ARBITRATION_LOST ....................................... 250
7.10.3.4 I2C_ACK_FAILURE ................................................ 250
7.10.3.5 I2C_OVERRUN ...................................................... 251
7.10.3.6 I2C_PEC_ERROR ..................................................... 251
7.10.3.7 I2C_TIMEOUT ........................................................ 251
7.10.3.8 I2C_SMB_ALERT ..................................................... 251
7.10.3.9 I2C_USE_MUTUAL_EXCLUSION .................................. 251
7.10.3.10 _i2c_wakeup_isr .................................................. 251
7.10.3.11 _i2c_wakeup_error_isr ......................................... 252
7.10.3.12 i2cMasterTransmit ............................................. 252
7.10.3.13 i2cMasterReceive ............................................. 253
7.10.3.14 PLATFORM_I2C_USE_I2C1 ..................................... 253
7.10.3.15 i2c_lld_get_errors ............................................... 253
7.10.4 Typedef Documentation ............................................ 253
7.10.4.1 i2caddr_t .......................................................... 254
7.10.4.2 i2cflags_t .......................................................... 254
7.10.4.3 I2CDriver ........................................................... 254
7.10.5 Enumeration Type Documentation ................................ 254
7.10.5.1 i2cstate_t .......................................................... 254
7.10.6 Function Documentation .......................................... 255
7.10.6.1 i2cInit() ............................................................ 255
7.10.6.2 i2cObjectInit() .................................................... 255
7.10.6.3 i2cStart() ........................................................... 256
7.10.6.4 i2cStop() ............................................................ 257
7.10.6.5 i2cGetErrors() ..................................................... 257
7.10.6.6 i2cMasterTransmitTimeout() .................................. 258
7.10.6.7 i2cMasterReceiveTimeout() ................................... 259
7.10.6.8 i2cAcquireBus() ................................................... 259
7.10.6.9 i2cReleaseBus() ................................................... 260
7.12.3 ICU Operations. .......................................................... 277
7.12.4 Macro Definition Documentation ................................. 280
  7.12.4.1 icuStartCaptureI ............................................ 280
  7.12.4.2 icuStopCaptureI ........................................... 281
  7.12.4.3 icuEnableNotificationsI ................................... 281
  7.12.4.4 icuDisableNotificationsI .................................. 282
  7.12.4.5 icuAreNotificationsEnabledX ................................ 282
  7.12.4.6 icuGetWidthX .............................................. 283
  7.12.4.7 icuGetPeriodX .............................................. 283
  7.12.4.8 _icu_isr_invoke_width_cb ................................ 284
  7.12.4.9 _icu_isr_invoke_period_cb ................................ 284
  7.12.4.10 _icu_isr_invoke_overflow_cb ............................. 285
  7.12.4.11 PLATFORM_ICU_USE_ICU1 ................................ 285
  7.12.4.12 icu_lld_get_width ...................................... 286
  7.12.4.13 icu_lld_get_period ...................................... 286
  7.12.4.14 icu_lld_are_notifications_enabled ...................... 286
7.12.5 Typedef Documentation ........................................... 287
  7.12.5.1 ICUDriver .................................................. 287
  7.12.5.2 icucallback_t ............................................ 287
  7.12.5.3 icufreq_t ................................................ 287
  7.12.5.4 icucnt_t .................................................. 288
7.12.6 Enumeration Type Documentation ................................ 288
  7.12.6.1 icustate_t ................................................ 288
  7.12.6.2 icumode_t ................................................ 288
7.12.7 Function Documentation .......................................... 288
  7.12.7.1 icuInit() .................................................. 289
  7.12.7.2 icuObjectInit() ......................................... 289
  7.12.7.3 icuStart() ................................................ 289
  7.12.7.4 icuStop() ................................................ 291
  7.12.7.5 icuStartCapture() ...................................... 292
  7.12.7.6 icuWaitCapture() ...................................... 293
  7.12.7.7 icuStopCapture() ...................................... 294
  7.12.7.8 icuEnableNotifications() ................................ 295
  7.12.7.9 icuDisableNotifications() ................................ 295
  7.12.7.10 icu_lld_init() ......................................... 296
  7.12.7.11 icu_lld_start() ....................................... 296
  7.12.7.12 icu_lld_stop() ....................................... 297
  7.12.7.13 icu_lld_start_capture() ................................ 297
  7.12.7.14 icu_lld_wait_capture() ................................ 297
  7.12.7.15 icu_lld_stop_capture() ................................ 298
  7.12.7.16 icu_lld_enable_notifications() ....................... 298
  7.12.7.17 icu_lld_disable_notifications() ...................... 299
7.17.2 Macro Definition Documentation .................................................... 318
  7.17.2.1 FILE_OK ................................................................. 318
  7.17.2.2 FILE_ERROR ............................................................ 318
  7.17.2.3 FILE_EOF ............................................................... 318
  7.17.2.4 _file_stream_methods ............................................... 319
  7.17.2.5 _file_stream_data ................................................... 319
  7.17.2.6 fileStreamWrite ..................................................... 320
  7.17.2.7 fileStreamRead ...................................................... 320
  7.17.2.8 fileStreamPut ....................................................... 320
  7.17.2.9 fileStreamGet ....................................................... 321
  7.17.2.10 fileStreamClose .................................................... 321
  7.17.2.11 fileStreamGetPosition ........................................... 322
  7.17.2.12 fileStreamGetSize ................................................ 323
  7.17.2.13 fileStreamGetPosition ........................................... 323
  7.17.2.14 fileStreamGetPosition ........................................... 324
  7.17.2.15 fileStreamGetPosition ........................................... 324
  7.17.3 Typedef Documentation ....................................................... 324
  7.17.3.1 fileoffset_t ........................................................ 324

7.18 Abstract Persistent Storage Interface .............................................. 325
  7.18.1 Detailed Description ...................................................... 325
  7.18.2 Macro Definition Documentation ............................................ 326
    7.18.2.1 _base_pers_storage_methods_alone ................................ 326
    7.18.2.2 _base_pers_storage_methods ...................................... 326
    7.18.2.3 _base_persistent_storage_data ................................. 326
    7.18.2.4 getBasePersistentStorage ....................................... 326
    7.18.2.5 psGetStorageSize ................................................ 326
    7.18.2.6 psRead ............................................................ 327
    7.18.2.7 psWrite .......................................................... 328
  7.18.3 Typedef Documentation ....................................................... 328
    7.18.3.1 ps_offset_t ........................................................ 328
  7.18.4 Enumeration Type Documentation ............................................ 328
    7.18.4.1 ps_error_t ........................................................ 329

7.19 Memory Streams Class ............................................................... 330
  7.19.1 Detailed Description ...................................................... 330
  7.19.2 Macro Definition Documentation ............................................ 330
    7.19.2.1 _memory_stream_data ............................................ 330
  7.19.3 Function Documentation .................................................. 330
    7.19.3.1 msObjectInit() ................................................... 330

7.20 Null Streams Class ................................................................. 332
  7.20.1 Detailed Description ...................................................... 332
  7.20.2 Macro Definition Documentation ............................................ 332
    7.20.2.1 _null_stream_data .............................................. 332
7.23.4 Enumeration Type Documentation ................................. 355
  7.23.4.1 macstate_t ................................................. 355

7.23.5 Function Documentation ........................................... 355
  7.23.5.1 macInit() .................................................. 355
  7.23.5.2 macObjectInit() ............................................ 356
  7.23.5.3 macStart() .................................................. 357
  7.23.5.4 macStop() ................................................... 357
  7.23.5.5 macWaitTransmitDescriptor() .............................. 358
  7.23.5.6 macReleaseTransmitDescriptor() ........................... 359
  7.23.5.7 macWaitReceiveDescriptor() ................................ 360
  7.23.5.8 macReleaseReceiveDescriptor() ............................ 361
  7.23.5.9 macPollLinkStatus() ...................................... 361
  7.23.5.10 mac_lld_init() ............................................. 362
  7.23.5.11 mac_lld_start() .......................................... 362
  7.23.5.12 mac_lld_stop() ............................................ 363
  7.23.5.13 mac_lld_get_transmit_descriptor() ....................... 363
  7.23.5.14 mac_lld_release_transmit_descriptor() ................. 364
  7.23.5.15 mac_lld_get_receive_descriptor() ....................... 364
  7.23.5.16 mac_lld_release_receive_descriptor() .................... 365
  7.23.5.17 mac_lld_poll_link_status() .............................. 365
  7.23.5.18 mac_lld_write_transmit_descriptor() .................... 366
  7.23.5.19 mac_lld_read_receive_descriptor() ...................... 366
  7.23.5.20 mac_lld_get_next_transmit_buffer() ..................... 367
  7.23.5.21 mac_lld_get_next_receive_buffer() ...................... 367

7.23.6 Variable Documentation .......................................... 368
  7.23.6.1 ETHD1 ....................................................... 368

7.24 Managed Flash Storage Driver ..................................... 369
  7.24.1 Detailed Description ......................................... 369

7.24.2 Macro Definition Documentation ................................ 371
  7.24.2.1 ALIGNED_REC_SIZE ........................................ 372
  7.24.2.2 ALIGNED_DHDR_SIZE ...................................... 372
  7.24.2.3 ALIGNED_SIZEOF .......................................... 372
  7.24.2.4 PAIR ....................................................... 372
  7.24.2.5 RET_ON_ERROR ............................................. 372
  7.24.2.6 MFS_CFG_MAX_RECORDS .................................... 373
  7.24.2.7 MFS_CFG_MAX_REPAIR_ATTEMPTS ........................... 373
  7.24.2.8 MFS_CFG_WRITE_VERIFY ................................... 373
  7.24.2.9 MFS_CFG_STRONG_CHECKING ................................ 373
  7.24.2.10 MFS_CFG_BUFFER_SIZE .................................. 373
  7.24.2.11 MFS_CFG_MEMORY_ALIGNMENT .............................. 374
  7.24.2.12 MFS_CFG_TRANSACTION_MAX ................................ 374

7.24.3 Typedef Documentation ........................................... 374
7.25.2.10 MII_STAT1000 .................................................. 395
7.25.2.11 MII_ESTATUS ................................................. 395
7.25.2.12 MII_PHYSTS ................................................... 395
7.25.2.13 MII_MICR ...................................................... 395
7.25.2.14 MII_DCOUNT ............................................... 395
7.25.2.15 MII_FCSCOUNTER ......................................... 395
7.25.2.16 MII_NWAYTEST .............................................. 395
7.25.2.17 MII_RERRCOUNTER ...................................... 396
7.25.2.18 MII_SREVISION ............................................. 396
7.25.2.19 MII_RESV1 .................................................. 396
7.25.2.20 MII_LBRERROR ............................................. 396
7.25.2.21 MII_PHYADDR ................................................ 396
7.25.2.22 MII_RESV2 .................................................. 396
7.25.2.23 MII_TPISTATUS ............................................ 396
7.25.2.24 MII_NCONFIG ............................................. 397
7.25.2.25 BMCR_RESV ................................................. 397
7.25.2.26 BMCR_CTST ................................................ 397
7.25.2.27 BMCR_FULLDPLX .......................................... 397
7.25.2.28 BMCR_ANRESTART ....................................... 397
7.25.2.29 BMCR_ISOLATE ........................................... 397
7.25.2.30 BMCR_PDOWN ............................................. 397
7.25.2.31 BMCR_ANENABLE ......................................... 398
7.25.2.32 BMCR_SPEED100 ......................................... 398
7.25.2.33 BMCR_LOOPBACK ......................................... 398
7.25.2.34 BMCR_RESET .............................................. 398
7.25.2.35 BMSR_ERCAP ............................................... 398
7.25.2.36 BMSR_JCD .................................................. 398
7.25.2.37 BMSR_LSTATUS ........................................... 398
7.25.2.38 BMSR_ANEGCAPABLE .................................. 399
7.25.2.39 BMSR_RFAULT ............................................ 399
7.25.2.40 BMSR_ANEGCOMPLETE ................................. 399
7.25.2.41 BMSR_MFPRESUPPCAP ................................. 399
7.25.2.42 BMSR_RESV .............................................. 399
7.25.2.43 BMSR_10HALF ........................................... 399
7.25.2.44 BMSR_10FULL ........................................... 399
7.25.2.45 BMSR_100HALF .......................................... 400
7.25.2.46 BMSR_100FULL ........................................... 400
7.25.2.47 BMSR_100BASE4 ........................................ 400
7.25.2.48 ADVERTISE_SLCT .................................... 400
7.25.2.49 ADVERTISE_CSMA ...................................... 400
7.25.2.50 ADVERTISE_10HALF .................................. 400
7.25.2.51 ADVERTISE_10FULL .................................. 400
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.25.2.52 ADVERTISE_100HALF</td>
<td>401</td>
</tr>
<tr>
<td>7.25.2.53 ADVERTISE_100FULL</td>
<td>401</td>
</tr>
<tr>
<td>7.25.2.54 ADVERTISE_100BASE4</td>
<td>401</td>
</tr>
<tr>
<td>7.25.2.55 ADVERTISE_PAUSE_CAP</td>
<td>401</td>
</tr>
<tr>
<td>7.25.2.56 ADVERTISE_PAUSEASYM</td>
<td>401</td>
</tr>
<tr>
<td>7.25.2.57 ADVERTISE_RESV</td>
<td>401</td>
</tr>
<tr>
<td>7.25.2.58 ADVERTISE_RFAULT</td>
<td>401</td>
</tr>
<tr>
<td>7.25.2.59 ADVERTISE_LPACK</td>
<td>402</td>
</tr>
<tr>
<td>7.25.2.60 ADVERTISE_NPAGE</td>
<td>402</td>
</tr>
<tr>
<td>7.25.2.61 LPA_SLCT</td>
<td>402</td>
</tr>
<tr>
<td>7.25.2.62 LPA_10HALF</td>
<td>402</td>
</tr>
<tr>
<td>7.25.2.63 LPA_10FULL</td>
<td>402</td>
</tr>
<tr>
<td>7.25.2.64 LPA_100HALF</td>
<td>402</td>
</tr>
<tr>
<td>7.25.2.65 LPA_100FULL</td>
<td>402</td>
</tr>
<tr>
<td>7.25.2.66 LPA_100BASE4</td>
<td>403</td>
</tr>
<tr>
<td>7.25.2.67 LPA_PAUSE_CAP</td>
<td>403</td>
</tr>
<tr>
<td>7.25.2.68 LPA_PAUSEASYM</td>
<td>403</td>
</tr>
<tr>
<td>7.25.2.69 LPA_RESV</td>
<td>403</td>
</tr>
<tr>
<td>7.25.2.70 LPA_RFAULT</td>
<td>403</td>
</tr>
<tr>
<td>7.25.2.71 LPA_LPACK</td>
<td>403</td>
</tr>
<tr>
<td>7.25.2.72 LPA_NPAGE</td>
<td>403</td>
</tr>
<tr>
<td>7.25.2.73 EXPANSION_NWAY</td>
<td>404</td>
</tr>
<tr>
<td>7.25.2.74 EXPANSION_LCWP</td>
<td>404</td>
</tr>
<tr>
<td>7.25.2.75 EXPANSION_ENABLENPAGE</td>
<td>404</td>
</tr>
<tr>
<td>7.25.2.76 EXPANSION_NPCAPABLE</td>
<td>404</td>
</tr>
<tr>
<td>7.25.2.77 EXPANSION_MFAULTS</td>
<td>404</td>
</tr>
<tr>
<td>7.25.2.78 EXPANSION_RESV</td>
<td>404</td>
</tr>
<tr>
<td>7.25.2.79 NWAYTEST_RESV1</td>
<td>404</td>
</tr>
<tr>
<td>7.25.2.80 NWAYTEST_LOOPBACK</td>
<td>405</td>
</tr>
<tr>
<td>7.25.2.81 NWAYTEST_RESV2</td>
<td>405</td>
</tr>
<tr>
<td>7.26 MMC over SPI Driver</td>
<td>406</td>
</tr>
<tr>
<td>7.26.1 Detailed Description</td>
<td>406</td>
</tr>
<tr>
<td>7.26.2 Driver State Machine</td>
<td>406</td>
</tr>
<tr>
<td>7.26.3 Driver Operations</td>
<td>406</td>
</tr>
<tr>
<td>7.26.4 Macro Definition Documentation</td>
<td>408</td>
</tr>
<tr>
<td>7.26.4.1 MMC_NICE_WAITING</td>
<td>408</td>
</tr>
<tr>
<td>7.26.4.2 _mmc_driver_methods</td>
<td>408</td>
</tr>
<tr>
<td>7.26.4.3 mmclsCardInserted</td>
<td>408</td>
</tr>
<tr>
<td>7.26.4.4 mmclsWriteProtected</td>
<td>409</td>
</tr>
<tr>
<td>7.26.5 Function Documentation</td>
<td>409</td>
</tr>
<tr>
<td>7.26.5.1 crc7()</td>
<td>410</td>
</tr>
<tr>
<td>7.26.5.2 wait()</td>
<td>410</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
7.26.5.3 send_hdr() ............................................................... 411
7.26.5.4 recv1() ................................................................. 411
7.26.5.5 recv3() ................................................................. 412
7.26.5.6 send_command_R1() ............................................... 413
7.26.5.7 send_command_R3() ............................................... 414
7.26.5.8 read_CxD() ........................................................... 415
7.26.5.9 sync() ................................................................... 416
7.26.5.10 mmcInit() ............................................................. 417
7.26.5.11 mmcObjectInit() ..................................................... 417
7.26.5.12 mmcStart() ............................................................ 418
7.26.5.13 mmcStop() ............................................................. 418
7.26.5.14 mmcConnect() ........................................................ 419
7.26.5.15 mmcDisconnect() .................................................... 420
7.26.5.16 mmcStartSequentialRead() ...................................... 421
7.26.5.17 mmcSequentialRead() ............................................ 422
7.26.5.18 mmcStopSequentialRead() ....................................... 423
7.26.5.19 mmcStartSequentialWrite() ...................................... 423
7.26.5.20 mmcSequentialWrite() ............................................ 424
7.26.5.21 mmcStopSequentialWrite() ....................................... 425
7.26.5.22 mmcSync() ............................................................. 426
7.26.5.23 mmcGetInfo() ........................................................ 427
7.26.5.24 mmcErase() ........................................................... 428

7.26.6 Variable Documentation ............................................... 428
7.26.6.1 mmc_vmt ............................................................... 429
7.26.6.2 crc7_lookup_table ................................................... 429

7.27 MMC/SD Block Device ..................................................... 430
7.27.1 Detailed Description ................................................... 430
7.27.2 Macro Definition Documentation .................................. 434
7.27.2.1 MMCSD_BLOCK_SIZE ............................................. 434
7.27.2.2 MMCSD_R1_ERROR_MASK ........................................ 435
7.27.2.3 MMCSD_CMD8_PATTERN .......................................... 435
7.27.2.4 _mmcsd_block_device_methods .................................. 435
7.27.2.5 _mmcsd_block_device_data ....................................... 435
7.27.2.6 MMCSD_R1_ERROR .................................................... 435
7.27.2.7 MMCSD_R1_STS ....................................................... 436
7.27.2.8 MMCSD_R1_IS_CARD_LOCKED ................................... 436
7.27.2.9 mmcsdGetCardCapacity ........................................... 436

7.27.3 Function Documentation ............................................. 437
7.27.3.1 _mmcsd_get_slice() ................................................ 437
7.27.3.2 _mmcsd_get_capacity() .......................................... 437
7.27.3.3 _mmcsd_get_capacity_ext() ...................................... 438
7.27.3.4 _mmcsd_unpack_sdc_cid() ....................................... 439
7.27.3.5 _mmcsd_unpack_mmc_cid() ........................................ 439
7.27.3.6 _mmcsd_unpack_csd_mmc() ........................................ 440
7.27.3.7 _mmcsd_unpack_csd_v10() .......................................... 441
7.27.3.8 _mmcsd_unpack_csd_v20() .......................................... 441

7.28 Serial NOR Flash Driver .............................................. 443
7.28.1 Detailed Description ............................................... 443
7.28.2 Driver State Machine ............................................... 443
7.28.3 Macro Definition Documentation ................................. 445
  7.28.3.1 SNOR_BUS_DRIVER .............................................. 445
  7.28.3.2 SNOR_SHARED_BUS .............................................. 445
  7.28.3.3 _snor_flash_methods_alone .................................... 446
  7.28.3.4 _snor_flash_methods ........................................... 446
7.28.4 Function Documentation .......................................... 446
  7.28.4.1 snor_get_descriptor() ........................................ 446
  7.28.4.2 bus_acquire() ................................................ 446
  7.28.4.3 bus_release() ................................................ 447
  7.28.4.4 bus_stop() ................................................... 447
  7.28.4.5 bus_cmd() ..................................................... 448
  7.28.4.6 bus_cmd_send() ............................................... 448
  7.28.4.7 bus_cmd_receive() ............................................ 448
  7.28.4.8 bus_cmd_addr() ............................................... 449
  7.28.4.9 bus_cmd_addr_send() ........................................ 449
  7.28.4.10 bus_cmd_addr_receive() ................................... 450
  7.28.4.11 bus_cmd_dummy_receive() .................................. 450
  7.28.4.12 bus_cmd_addr_dummy_receive() .............................. 451
  7.28.4.13 snorObjectInit() ............................................ 451
  7.28.4.14 snorStart() ................................................ 452
  7.28.4.15 snorStop() .................................................. 452
  7.28.4.16 snorMemoryMap() ............................................ 452
  7.28.4.17 snorMemoryUnmap() ........................................ 453
7.28.5 Variable Documentation .......................................... 454
  7.28.5.1 snor_vmt ..................................................... 454

7.29 Base Object .......................................................... 455
7.29.1 Detailed Description ............................................. 455
7.29.2 Macro Definition Documentation ............................... 455
  7.29.2.1 _base_object_methods ....................................... 455
  7.29.2.2 _base_object_data .......................................... 456
  7.29.2.3 objGetInstance ............................................. 456

7.30 PAL Driver ............................................................ 457
7.30.1 Detailed Description ............................................ 457
7.30.2 Implementation Rules ............................................ 457
  7.30.2.1 Writing on input pads ....................................... 457
7.30.2.2 Reading from output pads ........................................ 457
7.30.2.3 Writing unused or unimplemented port bits ............... 458
7.30.2.4 Reading from unused or unimplemented port bits ........ 458
7.30.2.5 Reading or writing on pins associated to other functionalities ... 458

7.30.3 Macro Definition Documentation ............................... 463
7.30.3.1 PAL_MODE_RESET .................................................. 463
7.30.3.2 PAL_MODE_UNCONNECTED ...................................... 464
7.30.3.3 PAL_MODE_INPUT .................................................. 464
7.30.3.4 PAL_MODE_INPUT_PULLUP ..................................... 464
7.30.3.5 PAL_MODE_INPUT_PULLDOWN ................................ 464
7.30.3.6 PAL_MODE_INPUT_ANALOG ..................................... 464
7.30.3.7 PAL_MODE_OUTPUT_PUSHPULL ......................... 464
7.30.3.8 PAL_MODE_OUTPUT_OPENDRAIN ............................. 465
7.30.3.9 PAL_LOW ......................................................... 465
7.30.3.10 PAL_HIGH ....................................................... 465
7.30.3.11 PAL_EVENT_MODE_EDGES_MASK ......................... 465
7.30.3.12 PAL_EVENT_MODE_DISABLED .............................. 465
7.30.3.13 PAL_EVENT_MODE_RISING_EDGE ......................... 465
7.30.3.14 PAL_EVENT_MODE_FALLING_EDGE ....................... 466
7.30.3.15 PAL_EVENT_MODE_BOTH_EDGES ......................... 466
7.30.3.16 PAL_USE_CALLBACKS ........................................ 466
7.30.3.17 PAL_USE_WAIT ............................................... 466
7.30.3.18 PAL_PORT_BIT ............................................... 466
7.30.3.19 PAL_GROUP_MASK ............................................. 467
7.30.3.20 _IOBUS_DATA ............................................... 467
7.30.3.21 IOBUS_DECL ................................................... 468
7.30.3.22 _pal_init_event [1/3] ...................................... 468
7.30.3.23 _pal_init_event [2/3] ...................................... 468
7.30.3.24 _pal_init_event [3/3] ...................................... 469
7.30.3.25 _pal_clear_event [1/3] .................................... 469
7.30.3.26 _pal_clear_event [2/3] .................................... 470
7.30.3.27 _pal_clear_event [3/3] .................................... 470
7.30.3.28 _pal_isr_code [1/3] ...................................... 471
7.30.3.29 _pal_isr_code [2/3] ...................................... 471
7.30.3.30 _pal_isr_code [3/3] ...................................... 472
7.30.3.31 palInit ......................................................... 472
7.30.3.32 palReadPort .................................................. 473
7.30.3.33 palReadLatch ............................................... 473
7.30.3.34 palWritePort ................................................ 474
7.30.3.35 palSetPort .................................................. 474
7.30.3.36 palClearPort ............................................... 475
7.30.3.37 palTogglePort ............................................. 475
7.30.3.38 palReadGroup .................................................. 476
7.30.3.39 palWriteGroup .................................................. 476
7.30.3.40 palSetGroupMode ............................................. 477
7.30.3.41 palReadPad ................................................... 477
7.30.3.42 palWritePad ................................................... 478
7.30.3.43 palSetPad ...................................................... 479
7.30.3.44 palClearPad ................................................... 479
7.30.3.45 palTogglePad .................................................. 480
7.30.3.46 palSetPadMode ............................................... 480
7.30.3.47 palReadLine ................................................... 481
7.30.3.48 palWriteLine .................................................. 481
7.30.3.49 palSetLine .................................................... 482
7.30.3.50 palClearLine ................................................ 482
7.30.3.51 palToggleLine ................................................ 483
7.30.3.52 palSetLineMode .............................................. 483
7.30.3.53 palEnablePadEventI ......................................... 484
7.30.3.54 palDisablePadEventI ....................................... 484
7.30.3.55 palEnablePadEvent .......................................... 485
7.30.3.56 palDisablePadEvent ....................................... 485
7.30.3.57 palEnableLineEventI ....................................... 486
7.30.3.58 palDisableLineEventI ..................................... 486
7.30.3.59 palEnableLineEvent ........................................ 487
7.30.3.60 palDisableLineEvent ....................................... 487
7.30.3.61 palIsPadEventEnabledX ................................... 488
7.30.3.62 palIsLineEventEnabledX .................................. 488
7.30.3.63 palSetPadCallback ......................................... 489
7.30.3.64 palSetLineCallback ....................................... 489
7.30.3.65 PAL_IOPORTS_WIDTH ....................................... 490
7.30.3.66 PAL_WHOLE_PORT .......................................... 490
7.30.3.67 PAL_LINE ..................................................... 490
7.30.3.68 PAL_PORT ..................................................... 490
7.30.3.69 PAL_PAD ....................................................... 491
7.30.3.70 PAL_NOLINE ................................................ 491
7.30.3.71 IPORT1 .......................................................... 491
7.30.3.72 pal_lld_init .................................................. 491
7.30.3.73 pal_lld_readport ............................................ 491
7.30.3.74 pal_lld_readlatch .......................................... 492
7.30.3.75 pal_lld_writeport ......................................... 492
7.30.3.76 pal_lld_setport ............................................. 493
7.30.3.77 pal_lld_clearport ......................................... 493
7.30.3.78 pal_lld_toggleport ........................................ 494
7.30.3.79 pal_lld_readgroup .......................................... 494
7.30.3.80 pal_lld_writegroup ........................................ 495
7.30.3.81 pal_lld_setgroupmode .................................... 496
7.30.3.82 pal_lld_readpad ........................................... 496
7.30.3.83 pal_lld_writepad .......................................... 497
7.30.3.84 pal_lld_setpad ............................................ 498
7.30.3.85 pal_lld_clearpad .......................................... 498
7.30.3.86 pal_lld_togglepad ........................................ 499
7.30.3.87 pal_lld_setpadmode ...................................... 499
7.30.3.88 pal_lld_get_pad_event .................................. 500
7.30.3.89 pal_lld_get_line_event ................................ 500

7.30.4 Typedef Documentation ....................................... 501
7.30.4.1 palcallback_t ............................................. 501
7.30.4.2 ioportmask_t ............................................. 501
7.30.4.3 iomode_t .................................................. 501
7.30.4.4 ioline_t ................................................... 501
7.30.4.5 ioportid_t ................................................. 501
7.30.4.6 iopadid_t .................................................. 501

7.30.5 Function Documentation ..................................... 502
7.30.5.1 palReadBus() .............................................. 502
7.30.5.2 palWriteBus() ............................................. 502
7.30.5.3 palSetBusMode() ......................................... 503
7.30.5.4 palSetPadCallback() ...................................... 504
7.30.5.5 palSetLineCallback() ..................................... 504
7.30.5.6 palWaitPadTimeoutS() .................................... 504
7.30.5.7 palWaitPadTimeout() ..................................... 505
7.30.5.8 palWaitLineTimeoutS() ................................... 506
7.30.5.9 palWaitLineTimeout() .................................... 507
7.30.5.10 _pal_lld_init() ........................................ 508
7.30.5.11 _pal_lld_setgroupmode() ................................. 508

7.31 Peripheral Interfaces .......................................... 510
7.31.1 Detailed Description ....................................... 510

7.32 PWM Driver .......................................................... 511
7.32.1 Detailed Description ....................................... 511
7.32.2 Driver State Machine ...................................... 511
7.32.3 PWM Operations .............................................. 511
7.32.4 Macro Definition Documentation ......................... 514
7.32.4.1 PWM_OUTPUT_MASK ..................................... 514
7.32.4.2 PWM_OUTPUT_DISABLED .................................. 515
7.32.4.3 PWM_OUTPUT_ACTIVE_HIGH ......................... 515
7.32.4.4 PWM_OUTPUT_ACTIVE_LOW ......................... 515
7.32.4.5 PWM_FRACTION_TO_WIDTH ......................... 515
7.32.4.6 PWM_DEGREES_TO_WIDTH ............................. 516
7.32.7.1 PWM_PERCENTAGE_TO_WIDTH .............................. 516
7.32.7.2 pwmChangePeriodI ..................................... 517
7.32.7.3 pwmEnableChannelI .................................... 518
7.32.7.4 pwmDisableChannelI .................................... 518
7.32.7.5 pwmIsChannelEnabledI .................................. 519
7.32.7.6 pwmEnablePeriodicNotificationI ....................... 520
7.32.7.7 pwmDisablePeriodicNotificationI ..................... 520
7.32.7.8 pwmEnableChannelNotificationI ....................... 521
7.32.7.9 pwmDisableChannelNotificationI ..................... 521
7.32.7.10 PWM_CHANNELS ........................................ 522
7.32.7.11 PLATFORM_PWM_USE_PWM1 .............................. 522
7.32.7.12 pwm_lld_change_period ................................. 523

7.32.5 Typedef Documentation ................................... 523
7.32.5.1 PWMDriver ............................................... 523
7.32.5.2 pwmcallback_t ......................................... 523
7.32.5.3 pwmmode_t ............................................... 524
7.32.5.4 pwmchannel_t .......................................... 524
7.32.5.5 pwmchnmsk_t ........................................... 524
7.32.5.6 pwmcnt_t ................................................ 524

7.32.6 Enumeration Type Documentation .......................... 524
7.32.6.1 pwmstate_t .............................................. 524

7.32.7 Function Documentation ................................... 525
7.32.7.1 pwmInit() ............................................... 525
7.32.7.2 pwmObjectInit() ....................................... 525
7.32.7.3 pwmStart() ............................................. 526
7.32.7.4 pwmStop() ............................................... 527
7.32.7.5 pwmChangePeriod() ..................................... 527
7.32.7.6 pwmEnableChannel() .................................... 528
7.32.7.7 pwmDisableChannel() ................................... 529
7.32.7.8 pwmEnablePeriodicNotification() ...................... 530
7.32.7.9 pwmDisablePeriodicNotification() ..................... 531
7.32.7.10 pwmEnableChannelNotification() ...................... 532
7.32.7.11 pwmDisableChannelNotification() ..................... 533
7.32.7.12 pwm_lld_init() ........................................ 534
7.32.7.13 pwm_lld_start() ....................................... 534
7.32.7.14 pwm_lld_stop() ........................................ 535
7.32.7.15 pwm_lld_enable_channel() .............................. 535
7.32.7.16 pwm_lld_disable_channel() ............................. 536
7.32.7.17 pwm_lld_enable_periodic_notification() .............. 536
7.32.7.18 pwm_lld_disable_periodic_notification() ............. 537
7.32.7.19 pwm_lld_enable_channel_notification() ............... 537
7.32.7.20 pwm_lld_disable_channel_notification() ............... 538
7.33.4.13 oqPutTimeout() .......................................................... 560
7.33.4.14 oqGetI() ................................................................. 562
7.33.4.15 oqWriteI() .............................................................. 562
7.33.4.16 oqWriteTimeout() ...................................................... 563

7.34 RTC Driver ................................................................. 565
7.34.1 Detailed Description .................................................... 565
7.34.2 Macro Definition Documentation ..................................... 567
  7.34.2.1 RTC_BASE_YEAR ....................................................... 567
  7.34.2.2 _rtc_driver_methods ............................................... 567
  7.34.2.3 RTC_SUPPORTS_CALLBACKS ........................................ 567
  7.34.2.4 RTC_ALARMS .......................................................... 567
  7.34.2.5 RTC_HAS_STORAGE ................................................... 568
  7.34.2.6 PLATFORM_RTC_USE_RTC1 ......................................... 568
  7.34.2.7 rtc_lld_driver_fields ............................................. 568
7.34.3 Typedef Documentation ............................................... 568
  7.34.3.1 RTCDriver ............................................................ 568
  7.34.3.2 rtcalarm_t .......................................................... 568
  7.34.3.3 rtccb_t ................................................................ 569
7.34.4 Enumeration Type Documentation ...................................... 569
  7.34.4.1 rtcevent_t ............................................................. 569
7.34.5 Function Documentation ............................................... 569
  7.34.5.1 rtcInit() ................................................................ 569
  7.34.5.2 rtcObjectInit() ....................................................... 569
  7.34.5.3 rtcSetTime() .......................................................... 570
  7.34.5.4 rtcGetTime() .......................................................... 571
  7.34.5.5 rtcSetAlarm() ........................................................ 571
  7.34.5.6 rtcGetAlarm() ......................................................... 572
  7.34.5.7 rtcSetCallback() ..................................................... 573
  7.34.5.8 rtcConvertDateTimeToStructTm() .................................. 573
  7.34.5.9 rtcConvertStructTmToDateTm() .................................... 574
  7.34.5.10 rtcConvertDateTimeToFAT() ...................................... 574
  7.34.5.11 rtc_lld_init() ....................................................... 575
  7.34.5.12 rtc_lld_set_time() ................................................ 575
  7.34.5.13 rtc_lld_get_time() ............................................... 576
  7.34.5.14 rtc_lld_set_alarm() ............................................. 576
  7.34.5.15 rtc_lld_get_alarm() ............................................. 577

7.35 SDC Driver ................................................................. 578
7.35.1 Detailed Description .................................................... 578
7.35.2 Driver State Machine ................................................... 578
7.35.3 Driver Operations ....................................................... 578
7.35.4 Macro Definition Documentation ..................................... 582
  7.35.4.1 SDC_INIT_RETRY ....................................................... 582
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.37.6</td>
<td>sduObjectInit()</td>
<td>637</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduStart()</td>
<td>637</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduStop()</td>
<td>638</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduSuspendHookI()</td>
<td>638</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduWakeupHook()</td>
<td>639</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduConfigureHookI()</td>
<td>639</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduRequestsHook()</td>
<td>640</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduSOFHookI()</td>
<td>641</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduDataTransmitted()</td>
<td>641</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduDataReceived()</td>
<td>642</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduInterruptTransmitted()</td>
<td>643</td>
</tr>
<tr>
<td>7.37.6</td>
<td>sduControl()</td>
<td>643</td>
</tr>
<tr>
<td>7.38</td>
<td>SIO Driver</td>
<td>644</td>
</tr>
<tr>
<td>7.38.1</td>
<td>Detailed Description</td>
<td>644</td>
</tr>
<tr>
<td>7.38.2</td>
<td>Driver State Machine</td>
<td>644</td>
</tr>
<tr>
<td>7.38.3</td>
<td>Macro Definition Documentation</td>
<td>647</td>
</tr>
<tr>
<td>7.38.3.1</td>
<td>SIO_NO_ERROR</td>
<td>647</td>
</tr>
<tr>
<td>7.38.3.2</td>
<td>SIO_PARITY_ERROR</td>
<td>647</td>
</tr>
<tr>
<td>7.38.3.3</td>
<td>SIO_FRAMING_ERROR</td>
<td>647</td>
</tr>
<tr>
<td>7.38.3.4</td>
<td>SIO_OVERRUN_ERROR</td>
<td>647</td>
</tr>
<tr>
<td>7.38.3.5</td>
<td>SIO_NOISE_ERROR</td>
<td>647</td>
</tr>
<tr>
<td>7.38.3.6</td>
<td>SIO_BREAK_DETECTDED</td>
<td>648</td>
</tr>
<tr>
<td>7.38.3.7</td>
<td>sioGetFlagsX</td>
<td>648</td>
</tr>
<tr>
<td>7.38.3.8</td>
<td>sioRXIsEmptyX</td>
<td>648</td>
</tr>
<tr>
<td>7.38.3.9</td>
<td>sioTXIsFullX</td>
<td>648</td>
</tr>
<tr>
<td>7.38.3.10</td>
<td>sioRXGetX</td>
<td>649</td>
</tr>
<tr>
<td>7.38.3.11</td>
<td>sioTXPutX</td>
<td>649</td>
</tr>
<tr>
<td>7.38.3.12</td>
<td>sioReadX</td>
<td>650</td>
</tr>
<tr>
<td>7.38.3.13</td>
<td>sioWriteX</td>
<td>651</td>
</tr>
<tr>
<td>7.38.3.14</td>
<td>sioControlX</td>
<td>651</td>
</tr>
<tr>
<td>7.38.3.15</td>
<td>PLATFORM_SIO_USE_SIO1</td>
<td>652</td>
</tr>
<tr>
<td>7.38.3.16</td>
<td>sio_lld_rx_is_empty</td>
<td>652</td>
</tr>
<tr>
<td>7.38.3.17</td>
<td>sio_lld_tx_is_full</td>
<td>653</td>
</tr>
<tr>
<td>7.38.3.18</td>
<td>sio_lld_rx_get</td>
<td>653</td>
</tr>
<tr>
<td>7.38.3.19</td>
<td>sio_lld_tx_put</td>
<td>654</td>
</tr>
<tr>
<td>7.38.4</td>
<td>Typedef Documentation</td>
<td>654</td>
</tr>
<tr>
<td>7.38.4.1</td>
<td>SIODriver</td>
<td>654</td>
</tr>
<tr>
<td>7.38.4.2</td>
<td>SIOConfig</td>
<td>654</td>
</tr>
<tr>
<td>7.38.4.3</td>
<td>sioflags_t</td>
<td>655</td>
</tr>
<tr>
<td>7.38.4.4</td>
<td>siocb_t</td>
<td>655</td>
</tr>
<tr>
<td>7.38.4.5</td>
<td>sioecb_t</td>
<td>655</td>
</tr>
<tr>
<td>7.38.5</td>
<td>Enumeration Type Documentation</td>
<td>655</td>
</tr>
</tbody>
</table>
7.38.5.1 siostate_t ......................................................... 655
7.38.6 Function Documentation ........................................... 656
  7.38.6.1 sioInit() ......................................................... 656
  7.38.6.2 sioObjectInit() ............................................... 656
  7.38.6.3 sioStart() ....................................................... 657
  7.38.6.4 sioStop() ......................................................... 657
  7.38.6.5 sio_lld_init() .................................................. 658
  7.38.6.6 sio_lld_start() ................................................. 658
  7.38.6.7 sio_lld_stop() .................................................. 659
  7.38.6.8 sio_lld_control() .............................................. 659
7.38.7 Variable Documentation ............................................ 660
  7.38.7.1 SIOD1 ........................................................... 660
7.39 SPI Driver .............................................................. 661
  7.39.1 Detailed Description ............................................ 661
  7.39.2 Driver State Machine ............................................ 661
  7.39.3 Macro Definition Documentation .................................. 664
    7.39.3.1 SPI_USE_WAIT ............................................... 665
    7.39.3.2 SPI_USE_CIRCULAR ........................................... 665
    7.39.3.3 SPI_USE_MUTUAL_EXCLUSION ................................ 665
    7.39.3.4 SPI_SELECT_MODE ........................................... 665
    7.39.3.5 spilsBufferComplete ........................................ 665
    7.39.3.6 spiSelectl .................................................... 666
    7.39.3.7 spiUnselectl .................................................. 666
    7.39.3.8 spiStartIgnorel .............................................. 667
    7.39.3.9 spiStartExchangel ........................................... 667
    7.39.3.10 spiStartSendl .............................................. 668
    7.39.3.11 spiStartReceivel ........................................... 669
    7.39.3.12 spiPolledExchange .......................................... 670
    7.39.3.13 _spi_wakeup_isr ........................................... 670
    7.39.3.14 _spi_isr_code ............................................... 671
    7.39.3.15 _spi_isr_half_code ......................................... 672
    7.39.3.16 _spi_isr_full_code ......................................... 672
    7.39.3.17 SPI_SUPPORTS_CIRCULAR .................................... 673
    7.39.3.18 PLATFORM_SPI_USE_SPI1 .................................... 673
    7.39.3.19 spi_lld_driver_fields ...................................... 673
    7.39.3.20 spi_lld_config_fields ..................................... 673
  7.39.4 Typedef Documentation .......................................... 674
    7.39.4.1 SPIDriver ..................................................... 674
    7.39.4.2 SPIConfig .................................................... 674
    7.39.4.3 spicallback_t ................................................ 674
  7.39.5 Enumeration Type Documentation .................................. 674
    7.39.5.1 spistate_t .................................................... 674
7.39.6 Function Documentation

7.39.6.1 spiInit() .................................................. 675
7.39.6.2 spiObjectInit() ......................................... 675
7.39.6.3 spiStart() ................................................. 676
7.39.6.4 spiStop() ................................................ 677
7.39.6.5 spiSelect() .............................................. 678
7.39.6.6 spiUnselect() ........................................... 679
7.39.6.7 spiStartIgnore() ....................................... 679
7.39.6.8 spiStartExchange() .................................... 680
7.39.6.9 spiStartSend() ......................................... 681
7.39.6.10 spiStartReceive() ................................... 682
7.39.6.11 spiAbort() ............................................. 683
7.39.6.12 spiAbort() ............................................. 684
7.39.6.13 spiIgnore() ............................................ 685
7.39.6.14 spiExchange() ......................................... 686
7.39.6.15 spiSend() .............................................. 687
7.39.6.16 spiReceive() .......................................... 688
7.39.6.17 spiAcquireBus() ....................................... 689
7.39.6.18 spiReleaseBus() ...................................... 690
7.39.6.19 spi_lld_init() ......................................... 691
7.39.6.20 spi_lld_start() ....................................... 691
7.39.6.21 spi_lld_stop() ......................................... 691
7.39.6.22 spi_lld_select() ...................................... 692
7.39.6.23 spi_lld_unselect() ................................... 692
7.39.6.24 spi_lld_ignore() ...................................... 693
7.39.6.25 spi_lld_exchange() ................................... 693
7.39.6.26 spi_lld_send() ......................................... 694
7.39.6.27 spi_lld_receive() ..................................... 694
7.39.6.28 spi_lld_abort() ....................................... 695
7.39.6.29 spi_lld_polled_exchange() ......................... 695

7.39.7 Variable Documentation ........................................ 696
7.39.7.1 SPID1 .................................................. 696

7.40 ST Driver ........................................................... 697
7.40.1 Detailed Description ........................................... 697

7.40.2 Function Documentation ....................................... 698
7.40.2.1 stInit() ................................................ 698
7.40.2.2 stStartAlarm() ....................................... 698
7.40.2.3 stStopAlarm() ......................................... 699
7.40.2.4 stSetAlarm() ........................................... 699
7.40.2.5 stGetAlarm() .......................................... 700
7.40.2.6 stGetCounter() ....................................... 701
7.40.2.7 stIsAlarmActive() .................................... 701
7.40.2.8 stlsAlarmActiveN() .......................... 702
7.40.2.9 stStartAlarmN() ............................ 702
7.40.2.10 stStopAlarmN() ............................. 703
7.40.2.11 stSetAlarmN() ............................... 703
7.40.2.12 stGetAlarmN() ............................... 704
7.40.2.13 st_lld_init() ............................... 704
7.40.2.14 st_lld_get_counter() ....................... 705
7.40.2.15 st_lld_start_alarm() ....................... 705
7.40.2.16 st_lld_stop_alarm() ....................... 705
7.40.2.17 st_lld_set_alarm() ......................... 706
7.40.2.18 st_lld_get_alarm() ......................... 706
7.40.2.19 st_lld_is_alarm_active() .................. 706

7.41 TRNG Driver ........................................ 708
  7.41.1 Detailed Description .......................... 708
  7.41.2 Macro Definition Documentation ............... 709
      7.41.2.1 PLATFORM_TRNG_USE_TRNG1 .............. 709
      7.41.2.2 trng_lld_driver_fields ................. 710
      7.41.2.3 trng_lld_config_fields ................. 710
  7.41.3 Typedef Documentation ......................... 710
      7.41.3.1 TRNGDriver ............................. 710
      7.41.3.2 TRNGConfig ............................. 710
  7.41.4 Enumeration Type Documentation ............... 710
      7.41.4.1 trngstate_t ............................ 710
  7.41.5 Function Documentation ....................... 711
      7.41.5.1 trngInit() ............................. 711
      7.41.5.2 trngObjectInit() ....................... 711
      7.41.5.3 trngStart() ............................. 712
      7.41.5.4 trngStop() ............................... 712
      7.41.5.5 trngGenerate() ........................... 713
      7.41.5.6 trng_lld_init() ......................... 714
      7.41.5.7 trng_lld_start() ........................ 714
      7.41.5.8 trng_lld_stop() ......................... 715
      7.41.5.9 trng_lld_generate() ..................... 715
  7.41.6 Variable Documentation ....................... 716
      7.41.6.1 TRNGD1 ................................. 716

7.42 UART Driver ...................................... 717
  7.42.1 Detailed Description ........................ 717
  7.42.2 Driver State Machine ......................... 717
      7.42.2.1 Transmitter sub State Machine .......... 718
      7.42.2.2 Receiver sub State Machine .......... 718
  7.42.3 Macro Definition Documentation ............... 721
      7.42.3.1 UART_NO_ERROR ......................... 722
7.42.6.11 uartStopReceive() ........................................ 742
7.42.6.12 uartStopReceiveI() ..................................... 743
7.42.6.13 uartSendTimeout() ....................................... 744
7.42.6.14 uartSendFullTimeout() .................................. 745
7.42.6.15 uartReceiveTimeout() .................................... 746
7.42.6.16 uartAcquireBus() ......................................... 747
7.42.6.17 uartReleaseBus() ......................................... 748
7.42.6.18 uart_lld_init() ........................................... 749
7.42.6.19 uart_lld_start() .......................................... 749
7.42.6.20 uart_lld_stop() .......................................... 749
7.42.6.21 uart_lld_start_send() .................................... 750
7.42.6.22 uart_lld_stop_send() .................................... 750
7.42.6.23 uart_lld_start_receive() ................................. 751
7.42.6.24 uart_lld_stop_receive() ................................. 751

7.42.7 Variable Documentation ...................................... 752
7.42.7.1 UARTD1 .................................................... 752

7.43 USB Driver ...................................................... 753
7.43.1 Detailed Description ......................................... 753
7.43.2 Driver State Machine ....................................... 753
7.43.3 USB Operations .............................................. 753
  7.43.3.1 USB Implementation ................................. 754
  7.43.3.2 USB Endpoints ...................................... 754
  7.43.3.3 USB Callbacks ...................................... 755
7.43.4 Macro Definition Documentation .......................... 760
  7.43.4.1 USB_DESC_INDEX ..................................... 760
  7.43.4.2 USB_DESC_BYTE ..................................... 760
  7.43.4.3 USB_DESC_WORD ..................................... 760
  7.43.4.4 USB_DESC_BCD ..................................... 761
  7.43.4.5 USB_DESC_DEVICE .................................. 761
  7.43.4.6 USB_DESC_CONFIGURATION_SIZE .................. 761
  7.43.4.7 USB_DESC_CONFIGURATION ......................... 762
  7.43.4.8 USB_DESC_INTERFACE_SIZE ....................... 762
  7.43.4.9 USB_DESC_INTERFACE ............................... 762
  7.43.4.10 USB_DESC_INTERFACE_ASSOCIATION_SIZE .... 763
  7.43.4.11 USB_DESC_INTERFACE_ASSOCIATION .......... 763
  7.43.4.12 USB_DESC_ENDPOINT_SIZE ..................... 763
  7.43.4.13 USB_DESC_ENDPOINT .............................. 763
  7.43.4.14 USB_EP_MODE_TYPE ................................ 764
  7.43.4.15 USB_EP_MODE_TYPE_CTRL ......................... 764
  7.43.4.16 USB_EP_MODE_TYPE_ISOC ......................... 764
  7.43.4.17 USB_EP_MODE_TYPE_BULK ......................... 764
7.43.4.19 USB_USE_WAIT .................................. 764
7.43.4.20 usbGetDriverStateI ............................... 764
7.43.4.21 usbConnectBus ................................ 765
7.43.4.22 usbDisconnectBus ............................... 765
7.43.4.23 usbGetFrameNumberX ........................... 766
7.43.4.24 usbGetTransmitStatusI ....................... 766
7.43.4.25 usbGetReceiveStatusI ......................... 767
7.43.4.26 usbGetReceiveTransactionSizeX ............... 767
7.43.4.27 usbSetupTransfer ................................ 768
7.43.4.28 usbReadSetup ................................ 768
7.43.4.29 _usb_isr_invoke_event_cb ..................... 769
7.43.4.30 _usb_isr_invoke_sof_cb ....................... 769
7.43.4.31 _usb_isr_invoke_setup_cb ..................... 770
7.43.4.32 _usb_isr_invoke_in_cb ......................... 770
7.43.4.33 _usb_isr_invoke_out_cb ....................... 771
7.43.4.34 USB_MAX_ENDPOINTS ............................. 771
7.43.4.35 USB_EP0_STATUS_STAGE ....................... 772
7.43.4.36 USB_SET_ADDRESS_MODE ......................... 772
7.43.4.37 USB_SET_ADDRESS_ACK_HANDLING ............... 772
7.43.4.38 PLATFORM_USB_USE_USB1 ...................... 772
7.43.4.39 usb_lld_get_frame_number .................... 772
7.43.4.40 usb_lld_get_transaction_size ................ 773
7.43.4.41 usb_lld_connect_bus ......................... 773
7.43.4.42 usb_lld_disconnect_bus ....................... 774
7.43.4.43 usb_lld_wakeup_host .......................... 774

7.43.5 Typedef Documentation ............................. 774
7.43.5.1 USBDriver ....................................... 774
7.43.5.2 usbecp_t ....................................... 774
7.43.5.3 usbcallback_t .................................. 774
7.43.5.4 usbecallback_t .................................. 775
7.43.5.5 usbeventcb_t .................................... 775
7.43.5.6 usbreqhandler_t ................................ 775

7.43.6 Enumeration Type Documentation .................. 776
7.43.6.1 usbstate_t ...................................... 776
7.43.6.2 usbecpstatus_t .................................. 776
7.43.6.3 usbep0state_t .................................. 777
7.43.6.4 usbevent_t ...................................... 777

7.43.7 Function Documentation ............................ 778
7.43.7.1 set_address() .................................. 778
7.43.7.2 default_handler() ............................... 778
7.43.7.3 usbInit() ........................................ 779
7.43.7.4 usbObjectInit() ................................ 779
7.43.7.5 usbStart() .......................................................... 780
7.43.7.6 usbStop() .......................................................... 780
7.43.7.7 usbInitEndpoint() ............................................. 781
7.43.7.8 usbDisableEndpoints() ...................................... 782
7.43.7.9 usbStartReceive() ............................................ 783
7.43.7.10 usbStartTransmit() ......................................... 783
7.43.7.11 usbReceive() .................................................. 784
7.43.7.12 usbTransmit() .................................................. 785
7.43.7.13 usbStallReceive() ............................................. 786
7.43.7.14 usbStallTransmit() ......................................... 787
7.43.7.15 usbWakeUpHost() ............................................. 788
7.43.7.16 _usb_reset() ................................................... 788
7.43.7.17 _usb_suspend() .............................................. 789
7.43.7.18 _usb_wakeup() ................................................ 790
7.43.7.19 _usb_ep0setup() .............................................. 790
7.43.7.20 _usb_ep0in() ................................................... 791
7.43.7.21 _usb_ep0out() .................................................. 791
7.43.7.22 usb_lld_init() ............................................... 792
7.43.7.23 usb_lld_start() .............................................. 792
7.43.7.24 usb_lld_stop() ............................................... 793
7.43.7.25 usb_lld_reset() .............................................. 793
7.43.7.26 usb_lld_set_address() ................................... 794
7.43.7.27 usb_lld_init_endpoint() .................................. 794
7.43.7.28 usb_lld_disable_endpoints() ............................ 795
7.43.7.29 usb_lld_get_status_out() ................................. 795
7.43.7.30 usb_lld_get_status_in() ................................. 796
7.43.7.31 usb_lld_read_setup() .................................... 796
7.43.7.32 usb_lld_prepare_receive() ............................... 797
7.43.7.33 usb_lld_prepare_transmit() ............................. 797
7.43.7.34 usb_lld_start_out() ....................................... 798
7.43.7.35 usb_lld_start_in() ......................................... 798
7.43.7.36 usb_lld_stall_out() ........................................ 798
7.43.7.37 usb_lld_stall_in() ......................................... 799
7.43.7.38 usb_lld_clear_out() ...................................... 799
7.43.7.39 usb_lld_clear_in() ....................................... 799
7.43.8 Variable Documentation ........................................ 800
7.43.8.1 usbgetdescriptor_t .......................................... 800
7.43.8.2 USBD1 .......................................................... 800
7.43.8.3 ep0_state ....................................................... 800
7.43.8.4 in .............................................................. 800
7.43.8.5 out .............................................................. 801
7.43.8.6 ep0config ....................................................... 801
7.44 USB CDC Header .......................................................... 802
  7.44.1 Detailed Description ............................................. 802
7.45 WDG Driver ............................................................. 804
  7.45.1 Detailed Description ............................................. 804
  7.45.2 Macro Definition Documentation .............................. 805
    7.45.2.1 wdgResetI .................................................. 805
    7.45.2.2 PLATFORM_WDG_USE_WDG1 ............................... 805
  7.45.3 Typedef Documentation ........................................ 805
    7.45.3.1 WDGDriver ................................................ 805
  7.45.4 Enumeration Type Documentation ............................. 806
    7.45.4.1 wdgstate_t .............................................. 806
  7.45.5 Function Documentation ...................................... 806
    7.45.5.1 wdgInit() ................................................ 806
    7.45.5.2 wdgStart() .............................................. 807
    7.45.5.3 wdgStop() ................................................ 807
    7.45.5.4 wdgReset() .............................................. 808
    7.45.5.5 wdg_lld_init() ......................................... 809
    7.45.5.6 wdg_lld_start() ....................................... 809
    7.45.5.7 wdg_lld_stop() ....................................... 809
    7.45.5.8 wdg_lld_reset() ....................................... 810
7.46 WSPI Driver .......................................................... 811
  7.46.1 Detailed Description ........................................... 811
  7.46.2 Macro Definition Documentation ............................. 814
    7.46.2.1 WSPI_USE_WAIT .......................................... 815
    7.46.2.2 WSPI_USE_MUTUAL_EXCLUSION ........................... 815
    7.46.2.3 wspiStartCommandI .................................... 815
    7.46.2.4 wspiStartSendI ....................................... 816
    7.46.2.5 wspiStartReceiveI .................................... 816
    7.46.2.6 wspiMapFlashI ......................................... 817
    7.46.2.7 wspiUnmapFlashI ..................................... 818
    7.46.2.8 _wspi_wakeup_isr ..................................... 818
    7.46.2.9 _wspi_isr_code ....................................... 819
    7.46.2.10 _wspi_error_code .................................... 819
    7.46.2.11 PLATFORM_WSPI_USE_WSPI1 ............................ 820
    7.46.2.12 wspi_lld_driver_fields ............................... 820
    7.46.2.13 wspi_lld_config_fields ............................... 821
  7.46.3 Typedef Documentation ...................................... 821
    7.46.3.1 WSPIDriver ............................................. 821
    7.46.3.2 WSPIConfig ............................................. 821
    7.46.3.3 wspicallback_t ...................................... 821
  7.46.4 Enumeration Type Documentation ............................ 821
    7.46.4.1 wspistate_t ........................................... 821
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.48.2.7</td>
<td>HAL_USE_GPT</td>
<td>844</td>
</tr>
<tr>
<td>7.48.2.8</td>
<td>HAL_USE_I2C</td>
<td>844</td>
</tr>
<tr>
<td>7.48.2.9</td>
<td>HAL_USE_I2S</td>
<td>845</td>
</tr>
<tr>
<td>7.48.2.10</td>
<td>HAL_USE_ICU</td>
<td>845</td>
</tr>
<tr>
<td>7.48.2.11</td>
<td>HAL_USE_MAC</td>
<td>845</td>
</tr>
<tr>
<td>7.48.2.12</td>
<td>HAL_USE_MMC_SPI</td>
<td>845</td>
</tr>
<tr>
<td>7.48.2.13</td>
<td>HAL_USE_PWM</td>
<td>845</td>
</tr>
<tr>
<td>7.48.2.14</td>
<td>HAL_USE_RTC</td>
<td>845</td>
</tr>
<tr>
<td>7.48.2.15</td>
<td>HAL_USE_SDC</td>
<td>846</td>
</tr>
<tr>
<td>7.48.2.16</td>
<td>HAL_USE_SERIAL</td>
<td>846</td>
</tr>
<tr>
<td>7.48.2.17</td>
<td>HAL_USE_SERIAL_USB</td>
<td>846</td>
</tr>
<tr>
<td>7.48.2.18</td>
<td>HAL_USE_SIO</td>
<td>846</td>
</tr>
<tr>
<td>7.48.2.19</td>
<td>HAL_USE_SPI</td>
<td>846</td>
</tr>
<tr>
<td>7.48.2.20</td>
<td>HAL_USE_TRNG</td>
<td>846</td>
</tr>
<tr>
<td>7.48.2.21</td>
<td>HAL_USE_UART</td>
<td>847</td>
</tr>
<tr>
<td>7.48.2.22</td>
<td>HAL_USE_USB</td>
<td>847</td>
</tr>
<tr>
<td>7.48.2.23</td>
<td>HAL_USE_WDG</td>
<td>847</td>
</tr>
<tr>
<td>7.48.2.24</td>
<td>HAL_USE_WSPI</td>
<td>847</td>
</tr>
<tr>
<td>7.48.2.25</td>
<td>PAL_USE_CALLBACKS</td>
<td>847</td>
</tr>
<tr>
<td>7.48.2.26</td>
<td>PAL_USE_WAIT</td>
<td>848</td>
</tr>
<tr>
<td>7.48.2.27</td>
<td>ADC_USE_WAIT</td>
<td>848</td>
</tr>
<tr>
<td>7.48.2.28</td>
<td>ADC_USE_MUTUAL_EXCLUSION</td>
<td>848</td>
</tr>
<tr>
<td>7.48.2.29</td>
<td>CAN_USE_SLEEP_MODE</td>
<td>848</td>
</tr>
<tr>
<td>7.48.2.30</td>
<td>CAN_ENFORCE_USE_CALLBACKS</td>
<td>848</td>
</tr>
<tr>
<td>7.48.2.31</td>
<td>HAL_CRY_USE_FALLBACK</td>
<td>849</td>
</tr>
<tr>
<td>7.48.2.32</td>
<td>HAL_CRY_ENFORCE_FALLBACK</td>
<td>849</td>
</tr>
<tr>
<td>7.48.2.33</td>
<td>DAC_USE_WAIT</td>
<td>849</td>
</tr>
<tr>
<td>7.48.2.34</td>
<td>DAC_USE_MUTUAL_EXCLUSION</td>
<td>849</td>
</tr>
<tr>
<td>7.48.2.35</td>
<td>I2C_USE_MUTUAL_EXCLUSION</td>
<td>850</td>
</tr>
<tr>
<td>7.48.2.36</td>
<td>MAC_USE_ZERO_COPY</td>
<td>850</td>
</tr>
<tr>
<td>7.48.2.37</td>
<td>MAC_USE_EVENTS</td>
<td>850</td>
</tr>
<tr>
<td>7.48.2.38</td>
<td>MMC_NICE_WAITING</td>
<td>850</td>
</tr>
<tr>
<td>7.48.2.39</td>
<td>SDC_INIT_RETRY</td>
<td>850</td>
</tr>
<tr>
<td>7.48.2.40</td>
<td>SDC_MMC_SUPPORT</td>
<td>851</td>
</tr>
<tr>
<td>7.48.2.41</td>
<td>SDC_NICE_WAITING</td>
<td>851</td>
</tr>
<tr>
<td>7.48.2.42</td>
<td>SDC_INIT_OCR_V20</td>
<td>851</td>
</tr>
<tr>
<td>7.48.2.43</td>
<td>SDC_INIT_OCR</td>
<td>851</td>
</tr>
<tr>
<td>7.48.2.44</td>
<td>SERIAL_DEFAULT_BITRATE</td>
<td>851</td>
</tr>
<tr>
<td>7.48.2.45</td>
<td>SERIAL_BUFFERS_SIZE</td>
<td>852</td>
</tr>
<tr>
<td>7.48.2.46</td>
<td>SERIAL_USB_BUFFERS_SIZE</td>
<td>852</td>
</tr>
<tr>
<td>7.48.2.47</td>
<td>SERIAL_USB_BUFFERS_NUMBER</td>
<td>852</td>
</tr>
<tr>
<td>7.48.2.48</td>
<td>SPI_USE_WAIT</td>
<td>852</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.48.2.49</td>
<td>SPI_USE_CIRCULAR</td>
<td>853</td>
</tr>
<tr>
<td>7.48.2.50</td>
<td>SPI_USE_MUTUAL_EXCLUSION</td>
<td>853</td>
</tr>
<tr>
<td>7.48.2.51</td>
<td>SPI_SELECT_MODE</td>
<td>853</td>
</tr>
<tr>
<td>7.48.2.52</td>
<td>UART_USE_WAIT</td>
<td>853</td>
</tr>
<tr>
<td>7.48.2.53</td>
<td>UART_USE_MUTUAL_EXCLUSION</td>
<td>854</td>
</tr>
<tr>
<td>7.48.2.54</td>
<td>USB_USE_WAIT</td>
<td>854</td>
</tr>
<tr>
<td>7.48.2.55</td>
<td>WSPI_USE_WAIT</td>
<td>854</td>
</tr>
<tr>
<td>7.48.2.56</td>
<td>WSPI_USE_MUTUAL_EXCLUSION</td>
<td>854</td>
</tr>
<tr>
<td>7.49</td>
<td>Normal Drivers</td>
<td>855</td>
</tr>
<tr>
<td>7.49.1</td>
<td>Detailed Description</td>
<td>855</td>
</tr>
<tr>
<td>7.50</td>
<td>Complex Drivers</td>
<td>857</td>
</tr>
<tr>
<td>7.50.1</td>
<td>Detailed Description</td>
<td>857</td>
</tr>
<tr>
<td>7.51</td>
<td>Interfaces and Classes</td>
<td>858</td>
</tr>
<tr>
<td>7.51.1</td>
<td>Detailed Description</td>
<td>858</td>
</tr>
<tr>
<td>7.52</td>
<td>Inner Code</td>
<td>859</td>
</tr>
<tr>
<td>7.52.1</td>
<td>Detailed Description</td>
<td>859</td>
</tr>
<tr>
<td>7.53</td>
<td>Support Code</td>
<td>860</td>
</tr>
<tr>
<td>7.53.1</td>
<td>Detailed Description</td>
<td>860</td>
</tr>
<tr>
<td>7.54</td>
<td>OSAL</td>
<td>861</td>
</tr>
<tr>
<td>7.54.1</td>
<td>Detailed Description</td>
<td>861</td>
</tr>
<tr>
<td>7.54.2</td>
<td>Macro Definition Documentation</td>
<td>866</td>
</tr>
<tr>
<td>7.54.2.1</td>
<td>OSAL_ST_RESOLUTION</td>
<td>866</td>
</tr>
<tr>
<td>7.54.2.2</td>
<td>OSAL_ST_FREQUENCY</td>
<td>866</td>
</tr>
<tr>
<td>7.54.2.3</td>
<td>OSAL_ST_MODE</td>
<td>866</td>
</tr>
<tr>
<td>7.54.2.4</td>
<td>OSAL_IRQ_PRIORITY_LEVELS</td>
<td>867</td>
</tr>
<tr>
<td>7.54.2.5</td>
<td>OSAL_IRQ_MAXIMUM_PRIORITY</td>
<td>867</td>
</tr>
<tr>
<td>7.54.2.6</td>
<td>OSAL_DBG_ENABLE_ASSERTS</td>
<td>867</td>
</tr>
<tr>
<td>7.54.2.7</td>
<td>OSAL_DBG_ENABLE_CHECKS</td>
<td>867</td>
</tr>
<tr>
<td>7.54.2.8</td>
<td>osal_dbgAssert</td>
<td>867</td>
</tr>
<tr>
<td>7.54.2.9</td>
<td>osal_dbgCheck</td>
<td>868</td>
</tr>
<tr>
<td>7.54.2.10</td>
<td>osal_dbgCheckClassI</td>
<td>868</td>
</tr>
<tr>
<td>7.54.2.11</td>
<td>osal_dbgCheckClassS</td>
<td>869</td>
</tr>
<tr>
<td>7.54.2.12</td>
<td>OSAL_IRQ_IS_VALID_PRIORITY</td>
<td>869</td>
</tr>
<tr>
<td>7.54.2.13</td>
<td>OSAL_IRQ_PROLOGUE</td>
<td>869</td>
</tr>
<tr>
<td>7.54.2.14</td>
<td>OSAL_IRQ_EPILOGUE</td>
<td>869</td>
</tr>
<tr>
<td>7.54.2.15</td>
<td>OSAL_IRQ_HANDLER</td>
<td>869</td>
</tr>
<tr>
<td>7.54.2.16</td>
<td>OSAL_S2I</td>
<td>870</td>
</tr>
<tr>
<td>7.54.2.17</td>
<td>OSAL_MS2I</td>
<td>870</td>
</tr>
<tr>
<td>7.54.2.18</td>
<td>OSAL_US2I</td>
<td>871</td>
</tr>
<tr>
<td>7.54.2.19</td>
<td>OSAL_S2RTC</td>
<td>871</td>
</tr>
<tr>
<td>7.54.2.20</td>
<td>OSAL_MS2RTC</td>
<td>872</td>
</tr>
<tr>
<td>7.54.2.21</td>
<td>OSAL_US2RTC</td>
<td>873</td>
</tr>
</tbody>
</table>
7.54.2.22 osalThreadSleepSeconds ........................................ 873
7.54.2.23 osalThreadSleepMilliseconds .................................. 874
7.54.2.24 osalThreadSleepMicroseconds ................................. 874

7.54.3 Typedef Documentation ........................................... 875
  7.54.3.1 syssts_t .................................................. 875
  7.54.3.2 msg_t .................................................... 875
  7.54.3.3 systime_t ................................................. 875
  7.54.3.4 sysinterval_t ............................................. 875
  7.54.3.5 rtcnt_t .................................................. 875
  7.54.3.6 thread_reference_t ........................................ 876
  7.54.3.7 eventflags_t ............................................. 876
  7.54.3.8 event_source_t ........................................... 876
  7.54.3.9 eventcallback_t ........................................... 876
  7.54.3.10 mutex_t ................................................ 876

7.54.4 Function Documentation .......................................... 877
  7.54.4.1 osalInit() ............................................... 877
  7.54.4.2 osalSysHalt() ........................................... 877
  7.54.4.3 osalSysPolledDelayX() .................................. 878
  7.54.4.4 osalOsTimerHandlerI() .................................. 878
  7.54.4.5 osalOsRescheduleS() ..................................... 878
  7.54.4.6 osalOsGetSystemTimeX() ................................ 879
  7.54.4.7 osalThreadSleepS() ....................................... 879
  7.54.4.8 osalThreadSleep() ........................................ 879
  7.54.4.9 osalThreadSuspendS() ..................................... 881
  7.54.4.10 osalThreadSuspendTimeoutS() .............................. 881
  7.54.4.11 osalThreadResumeI() .................................... 882
  7.54.4.12 osalThreadResumeS() .................................... 883
  7.54.4.13 osalThreadEnqueueTimeoutS() ............................. 883
  7.54.4.14 osalThreadDequeueNextI() ................................ 884
  7.54.4.15 osalThreadDequeueAllI() ................................ 884
  7.54.4.16 osalEventBroadcastFlagsI() .............................. 885
  7.54.4.17 osalEventBroadcastFlags() ................................ 885
  7.54.4.18 osalEventSetCallback() .................................. 886
  7.54.4.19 osalMutexLock() ......................................... 887
  7.54.4.20 osalMutexUnlock() ....................................... 887
  7.54.4.21 osalSysDisable() ........................................ 887
  7.54.4.22 osalSysEnable() .......................................... 888
  7.54.4.23 osalSysLock() ............................................ 888
  7.54.4.24 osalSysUnlock() .......................................... 888
  7.54.4.25 osalSysLockFromISR() ................................... 889
  7.54.4.26 osalSysUnlockFromISR() .................................. 889
  7.54.4.27 osalSysGetStatusAndLockX() .............................. 889
8 Data Structure Documentation

8.1 BaseAsynchronousChannel Struct Reference

8.1.1 Detailed Description .............................................. 896
8.1.2 Field Documentation ............................................. 896
8.1.2.1 vmt .......................................................... 896

8.2 BaseAsynchronousChannelVMT Struct Reference

8.2.1 Detailed Description .............................................. 898

8.3 BaseBlockDevice Struct Reference

8.3.1 Detailed Description .............................................. 900
8.3.2 Field Documentation ............................................. 900
8.3.2.1 vmt .......................................................... 900

8.4 BaseBlockDeviceVMT Struct Reference

8.4.1 Detailed Description .............................................. 901

8.5 BaseChannel Struct Reference

8.5.1 Detailed Description .............................................. 903
8.5.2 Field Documentation ............................................. 903
8.5.2.1 vmt .......................................................... 903

8.6 BaseChannelVMT Struct Reference

8.6.1 Detailed Description .............................................. 905

8.7 BaseFlash Struct Reference

8.7.1 Detailed Description .............................................. 907
8.7.2 Field Documentation ............................................. 907
8.7.2.1 vmt .......................................................... 907

8.8 BaseFlashVMT Struct Reference

8.8.1 Detailed Description .............................................. 909

8.9 BaseObject Struct Reference

8.9.1 Detailed Description .............................................. 910
8.9.2 Field Documentation ............................................. 910
8.9.2.1 vmt .......................................................... 910

8.10 BaseObjectVMT Struct Reference

8.10.1 Detailed Description ............................................. 910
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.27.2.4</td>
<td>sectors</td>
<td>936</td>
</tr>
<tr>
<td>8.27.2.5</td>
<td>sectors_size</td>
<td>936</td>
</tr>
<tr>
<td>8.27.2.6</td>
<td>address</td>
<td>937</td>
</tr>
<tr>
<td>8.27.2.7</td>
<td>size</td>
<td>937</td>
</tr>
<tr>
<td>8.28</td>
<td>flash_sector_descriptor_t Struct Reference</td>
<td>937</td>
</tr>
<tr>
<td>8.28.1</td>
<td>Detailed Description</td>
<td>937</td>
</tr>
<tr>
<td>8.28.2</td>
<td>Field Documentation</td>
<td>938</td>
</tr>
<tr>
<td>8.28.2.1</td>
<td>offset</td>
<td>938</td>
</tr>
<tr>
<td>8.28.2.2</td>
<td>size</td>
<td>938</td>
</tr>
<tr>
<td>8.29</td>
<td>GPTConfig Struct Reference</td>
<td>938</td>
</tr>
<tr>
<td>8.29.1</td>
<td>Detailed Description</td>
<td>939</td>
</tr>
<tr>
<td>8.29.2</td>
<td>Field Documentation</td>
<td>939</td>
</tr>
<tr>
<td>8.29.2.1</td>
<td>frequency</td>
<td>939</td>
</tr>
<tr>
<td>8.29.2.2</td>
<td>callback</td>
<td>939</td>
</tr>
<tr>
<td>8.30</td>
<td>GPTDriver Struct Reference</td>
<td>940</td>
</tr>
<tr>
<td>8.30.1</td>
<td>Detailed Description</td>
<td>940</td>
</tr>
<tr>
<td>8.30.2</td>
<td>Field Documentation</td>
<td>940</td>
</tr>
<tr>
<td>8.30.2.1</td>
<td>state</td>
<td>941</td>
</tr>
<tr>
<td>8.30.2.2</td>
<td>config</td>
<td>941</td>
</tr>
<tr>
<td>8.31</td>
<td>hal_adc_config Struct Reference</td>
<td>941</td>
</tr>
<tr>
<td>8.31.1</td>
<td>Detailed Description</td>
<td>941</td>
</tr>
<tr>
<td>8.32</td>
<td>hal_adc_configuration_group Struct Reference</td>
<td>941</td>
</tr>
<tr>
<td>8.32.1</td>
<td>Detailed Description</td>
<td>942</td>
</tr>
<tr>
<td>8.32.2</td>
<td>Field Documentation</td>
<td>943</td>
</tr>
<tr>
<td>8.32.2.1</td>
<td>circular</td>
<td>943</td>
</tr>
<tr>
<td>8.32.2.2</td>
<td>num_channels</td>
<td>943</td>
</tr>
<tr>
<td>8.32.2.3</td>
<td>end_cb</td>
<td>943</td>
</tr>
<tr>
<td>8.32.2.4</td>
<td>error_cb</td>
<td>943</td>
</tr>
<tr>
<td>8.33</td>
<td>hal_adc_driver Struct Reference</td>
<td>944</td>
</tr>
<tr>
<td>8.33.1</td>
<td>Detailed Description</td>
<td>945</td>
</tr>
<tr>
<td>8.33.2</td>
<td>Field Documentation</td>
<td>945</td>
</tr>
<tr>
<td>8.33.2.1</td>
<td>state</td>
<td>945</td>
</tr>
<tr>
<td>8.33.2.2</td>
<td>config</td>
<td>945</td>
</tr>
<tr>
<td>8.33.2.3</td>
<td>samples</td>
<td>945</td>
</tr>
<tr>
<td>8.33.2.4</td>
<td>depth</td>
<td>945</td>
</tr>
<tr>
<td>8.33.2.5</td>
<td>grp</td>
<td>946</td>
</tr>
<tr>
<td>8.33.2.6</td>
<td>thread</td>
<td>946</td>
</tr>
<tr>
<td>8.33.2.7</td>
<td>mutex</td>
<td>946</td>
</tr>
<tr>
<td>8.34</td>
<td>hal_dac_config Struct Reference</td>
<td>946</td>
</tr>
<tr>
<td>8.34.1</td>
<td>Detailed Description</td>
<td>946</td>
</tr>
<tr>
<td>8.35</td>
<td>hal_dac_conversion_group Struct Reference</td>
<td>947</td>
</tr>
<tr>
<td>8.35.1</td>
<td>Detailed Description</td>
<td>947</td>
</tr>
</tbody>
</table>
8.42.2.1 state .................................................. 960
8.42.2.2 config ............................................. 960
8.43 hal_spi_config Struct Reference ......................... 960
8.43.1 Detailed Description ................................... 961
8.43.2 Field Documentation ................................... 961
  8.43.2.1 circular ........................................... 961
  8.43.2.2 end_cb ............................................ 961
  8.43.2.3 ssline ............................................ 961
  8.43.2.4 ssport ............................................ 962
  8.43.2.5 ssmask ............................................ 962
  8.43.2.6 sspad ............................................. 962
8.44 hal_spi_driver Struct Reference .......................... 962
8.44.1 Detailed Description .................................. 963
8.44.2 Field Documentation .................................. 963
  8.44.2.1 state ............................................. 963
  8.44.2.2 config ............................................ 963
  8.44.2.3 thread ............................................ 963
  8.44.2.4 mutex ............................................. 963
8.45 hal_trng_config Struct Reference ......................... 964
8.45.1 Detailed Description .................................. 964
8.46 hal_trng_driver Struct Reference ......................... 964
8.46.1 Detailed Description .................................. 965
8.46.2 Field Documentation .................................. 965
  8.46.2.1 state ............................................. 965
  8.46.2.2 config ............................................ 965
8.47 hal_wspi_config Struct Reference .......................... 965
8.47.1 Detailed Description .................................. 966
8.47.2 Field Documentation .................................. 966
  8.47.2.1 end_cb ............................................ 966
  8.47.2.2 error_cb ........................................... 967
8.48 hal_wspi_driver Struct Reference ......................... 967
8.48.1 Detailed Description .................................. 967
8.48.2 Field Documentation .................................. 968
  8.48.2.1 state ............................................. 968
  8.48.2.2 config ............................................ 968
  8.48.2.3 thread ............................................ 968
  8.48.2.4 mutex ............................................. 968
8.49 HMACSHA256Context Struct Reference ..................... 968
8.49.1 Detailed Description .................................. 969
8.50 HMACSHA512Context Struct Reference ..................... 969
8.50.1 Detailed Description .................................. 969
8.51 I2CConfig Struct Reference .............................. 969
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.51.1</td>
<td>Detailed Description</td>
<td>970</td>
</tr>
<tr>
<td>8.52</td>
<td>I2CDriver Struct Reference</td>
<td>970</td>
</tr>
<tr>
<td>8.52.1</td>
<td>Detailed Description</td>
<td>970</td>
</tr>
<tr>
<td>8.52.2</td>
<td>Field Documentation</td>
<td>971</td>
</tr>
<tr>
<td></td>
<td>8.52.2.1 state</td>
<td>971</td>
</tr>
<tr>
<td></td>
<td>8.52.2.2 config</td>
<td>971</td>
</tr>
<tr>
<td></td>
<td>8.52.2.3 errors</td>
<td>971</td>
</tr>
<tr>
<td>8.53</td>
<td>ICUConfig Struct Reference</td>
<td>971</td>
</tr>
<tr>
<td>8.53.1</td>
<td>Detailed Description</td>
<td>972</td>
</tr>
<tr>
<td>8.53.2</td>
<td>Field Documentation</td>
<td>972</td>
</tr>
<tr>
<td></td>
<td>8.53.2.1 mode</td>
<td>972</td>
</tr>
<tr>
<td></td>
<td>8.53.2.2 frequency</td>
<td>972</td>
</tr>
<tr>
<td></td>
<td>8.53.2.3 width cb</td>
<td>973</td>
</tr>
<tr>
<td></td>
<td>8.53.2.4 period cb</td>
<td>973</td>
</tr>
<tr>
<td></td>
<td>8.53.2.5 overflow cb</td>
<td>973</td>
</tr>
<tr>
<td>8.54</td>
<td>ICUDriver Struct Reference</td>
<td>973</td>
</tr>
<tr>
<td>8.54.1</td>
<td>Detailed Description</td>
<td>974</td>
</tr>
<tr>
<td>8.54.2</td>
<td>Field Documentation</td>
<td>974</td>
</tr>
<tr>
<td></td>
<td>8.54.2.1 state</td>
<td>974</td>
</tr>
<tr>
<td></td>
<td>8.54.2.2 config</td>
<td>974</td>
</tr>
<tr>
<td>8.55</td>
<td>io_buffers_queue Struct Reference</td>
<td>974</td>
</tr>
<tr>
<td>8.55.1</td>
<td>Detailed Description</td>
<td>976</td>
</tr>
<tr>
<td>8.55.2</td>
<td>Field Documentation</td>
<td>976</td>
</tr>
<tr>
<td></td>
<td>8.55.2.1 waiting</td>
<td>976</td>
</tr>
<tr>
<td></td>
<td>8.55.2.2 suspended</td>
<td>976</td>
</tr>
<tr>
<td></td>
<td>8.55.2.3 bcounter</td>
<td>976</td>
</tr>
<tr>
<td></td>
<td>8.55.2.4 bwrptr</td>
<td>976</td>
</tr>
<tr>
<td></td>
<td>8.55.2.5 brdptr</td>
<td>977</td>
</tr>
<tr>
<td></td>
<td>8.55.2.6 btop</td>
<td>977</td>
</tr>
<tr>
<td></td>
<td>8.55.2.7 bsize</td>
<td>977</td>
</tr>
<tr>
<td></td>
<td>8.55.2.8 bn</td>
<td>977</td>
</tr>
<tr>
<td></td>
<td>8.55.2.9 buffers</td>
<td>977</td>
</tr>
<tr>
<td></td>
<td>8.55.2.10 ptr</td>
<td>978</td>
</tr>
<tr>
<td></td>
<td>8.55.2.11 top</td>
<td>978</td>
</tr>
<tr>
<td></td>
<td>8.55.2.12 notify</td>
<td>978</td>
</tr>
<tr>
<td></td>
<td>8.55.2.13 link</td>
<td>978</td>
</tr>
<tr>
<td>8.56</td>
<td>io_queue Struct Reference</td>
<td>979</td>
</tr>
<tr>
<td>8.56.1</td>
<td>Detailed Description</td>
<td>980</td>
</tr>
<tr>
<td>8.56.2</td>
<td>Field Documentation</td>
<td>980</td>
</tr>
<tr>
<td></td>
<td>8.56.2.1 q_waiting</td>
<td>980</td>
</tr>
<tr>
<td></td>
<td>8.56.2.2 q_counter</td>
<td>980</td>
</tr>
<tr>
<td></td>
<td>8.56.2.3 q_buffer</td>
<td>980</td>
</tr>
</tbody>
</table>
8.109.2.1 ud_size .............................................. 1058
8.109.2.2 ud_string .......................................... 1058
8.110 USBDriver Struct Reference .............................................. 1059
  8.110.1 Detailed Description .............................................. 1060
  8.110.2 Field Documentation .............................................. 1060
    8.110.2.1 state .............................................. 1060
    8.110.2.2 config .............................................. 1060
    8.110.2.3 transmitting ...................................... 1060
    8.110.2.4 receiving ....................................... 1061
    8.110.2.5 epc .............................................. 1061
    8.110.2.6 in_params ....................................... 1061
    8.110.2.7 out_params ..................................... 1061
    8.110.2.8 ep0state ....................................... 1061
    8.110.2.9 ep0next ....................................... 1062
    8.110.2.10 ep0n .......................................... 1062
    8.110.2.11 ep0endcb ..................................... 1062
    8.110.2.12 setup .......................................... 1062
    8.110.2.13 status ......................................... 1062
    8.110.2.14 address ....................................... 1062
    8.110.2.15 configuration ................................ 1063
    8.110.2.16 saved_state ................................... 1063
8.111 USBEndpointConfig Struct Reference .............................................. 1063
  8.111.1 Detailed Description .............................................. 1065
  8.111.2 Field Documentation .............................................. 1065
    8.111.2.1 ep_mode ......................................... 1065
    8.111.2.2 setup_cb ....................................... 1065
    8.111.2.3 in_cb ........................................... 1066
    8.111.2.4 out_cb ........................................... 1066
    8.111.2.5 in_maxsize ...................................... 1066
    8.111.2.6 out_maxsize .................................... 1066
    8.111.2.7 in_state ....................................... 1066
    8.111.2.8 out_state ...................................... 1066
8.112 USBInEndpointState Struct Reference .............................................. 1067
  8.112.1 Detailed Description .............................................. 1067
  8.112.2 Field Documentation .............................................. 1067
    8.112.2.1 txsize .......................................... 1067
    8.112.2.2 txcnt ........................................... 1068
    8.112.2.3 txbuf ........................................... 1068
    8.112.2.4 thread ......................................... 1068
8.113 USBOutEndpointState Struct Reference .............................................. 1068
  8.113.1 Detailed Description .............................................. 1069
  8.113.2 Field Documentation .............................................. 1069
8.113.2.1 rxsize ............................................. 1069
8.113.2.2 rxcnt ............................................. 1069
8.113.2.3 rxbuf ............................................. 1069
8.113.2.4 thread ............................................. 1069

8.114 WDGConfig Struct Reference .................................. 1070
8.114.1 Detailed Description ........................................ 1070

8.115 WGDDriver Struct Reference .................................... 1070
8.115.1 Detailed Description ........................................ 1071
8.115.2 Field Documentation .................................... 1071
  8.115.2.1 state ............................................. 1071
  8.115.2.2 config ............................................. 1071

8.116 wspi_command_t Struct Reference ................................ 1071
8.116.1 Detailed Description ........................................ 1072
8.116.2 Field Documentation .................................... 1072
  8.116.2.1 cfg ............................................. 1072
  8.116.2.2 cmd ............................................. 1072
  8.116.2.3 addr ............................................. 1072
  8.116.2.4 alt ............................................. 1072
  8.116.2.5 dummy ............................................. 1072

9 File Documentation ............................................. 1073
  9.1 chprintf.c File Reference ....................................... 1073
    9.1.1 Detailed Description ..................................... 1073
  9.2 chprintf.h File Reference ....................................... 1073
    9.2.1 Detailed Description ..................................... 1074
  9.3 hal.c File Reference ............................................ 1074
    9.3.1 Detailed Description ..................................... 1074
  9.4 hal.h File Reference ............................................ 1075
    9.4.1 Detailed Description ..................................... 1076
  9.5 hal_adc.c File Reference ........................................ 1076
    9.5.1 Detailed Description ..................................... 1077
  9.6 hal_adc.h File Reference ........................................ 1077
    9.6.1 Detailed Description ..................................... 1079
  9.7 hal_adc_lld.c File Reference .................................... 1079
    9.7.1 Detailed Description ..................................... 1079
  9.8 hal_adc_lld.h File Reference .................................... 1080
    9.8.1 Detailed Description ..................................... 1080
  9.9 hal_buffers.c File Reference .................................... 1081
    9.9.1 Detailed Description ..................................... 1081
  9.10 hal_buffers.h File Reference .................................. 1082
    9.10.1 Detailed Description ..................................... 1084
  9.11 hal_can.c File Reference ...................................... 1084
<table>
<thead>
<tr>
<th>Section</th>
<th>File Reference</th>
<th>Detailed Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.11.1</td>
<td>Detailed Description</td>
<td>1085</td>
</tr>
<tr>
<td>9.12</td>
<td>hal_can.h</td>
<td>1085</td>
</tr>
<tr>
<td>9.12.1</td>
<td>Detailed Description</td>
<td>1086</td>
</tr>
<tr>
<td>9.13</td>
<td>hal_can_lld.c</td>
<td>1086</td>
</tr>
<tr>
<td>9.13.1</td>
<td>Detailed Description</td>
<td>1087</td>
</tr>
<tr>
<td>9.14</td>
<td>hal_can_lld.h</td>
<td>1087</td>
</tr>
<tr>
<td>9.14.1</td>
<td>Detailed Description</td>
<td>1088</td>
</tr>
<tr>
<td>9.15</td>
<td>hal_channels.h</td>
<td>1088</td>
</tr>
<tr>
<td>9.15.1</td>
<td>Detailed Description</td>
<td>1089</td>
</tr>
<tr>
<td>9.16</td>
<td>hal_crypto.c</td>
<td>1090</td>
</tr>
<tr>
<td>9.16.1</td>
<td>Detailed Description</td>
<td>1090</td>
</tr>
<tr>
<td>9.17</td>
<td>hal_crypto.h</td>
<td>1090</td>
</tr>
<tr>
<td>9.17.1</td>
<td>Detailed Description</td>
<td>1092</td>
</tr>
<tr>
<td>9.18</td>
<td>hal_crypto_lld.c</td>
<td>1095</td>
</tr>
<tr>
<td>9.18.1</td>
<td>Detailed Description</td>
<td>1096</td>
</tr>
<tr>
<td>9.19</td>
<td>hal_crypto_lld.h</td>
<td>1098</td>
</tr>
<tr>
<td>9.19.1</td>
<td>Detailed Description</td>
<td>1101</td>
</tr>
<tr>
<td>9.20</td>
<td>hal_dac.c</td>
<td>1101</td>
</tr>
<tr>
<td>9.20.1</td>
<td>Detailed Description</td>
<td>1102</td>
</tr>
<tr>
<td>9.21</td>
<td>hal_dac.h</td>
<td>1102</td>
</tr>
<tr>
<td>9.21.1</td>
<td>Detailed Description</td>
<td>1104</td>
</tr>
<tr>
<td>9.22</td>
<td>hal_dac_lld.c</td>
<td>1104</td>
</tr>
<tr>
<td>9.22.1</td>
<td>Detailed Description</td>
<td>1105</td>
</tr>
<tr>
<td>9.23</td>
<td>hal_dac_lld.h</td>
<td>1105</td>
</tr>
<tr>
<td>9.23.1</td>
<td>Detailed Description</td>
<td>1106</td>
</tr>
<tr>
<td>9.24</td>
<td>hal_efl.c</td>
<td>1106</td>
</tr>
<tr>
<td>9.24.1</td>
<td>Detailed Description</td>
<td>1106</td>
</tr>
<tr>
<td>9.25</td>
<td>hal_efl.h</td>
<td>1106</td>
</tr>
<tr>
<td>9.25.1</td>
<td>Detailed Description</td>
<td>1107</td>
</tr>
<tr>
<td>9.26</td>
<td>hal_efl_lld.c</td>
<td>1107</td>
</tr>
<tr>
<td>9.26.1</td>
<td>Detailed Description</td>
<td>1108</td>
</tr>
<tr>
<td>9.27</td>
<td>hal_efl_lld.h</td>
<td>1108</td>
</tr>
<tr>
<td>9.27.1</td>
<td>Detailed Description</td>
<td>1109</td>
</tr>
<tr>
<td>9.28</td>
<td>hal_files.h</td>
<td>1109</td>
</tr>
<tr>
<td>9.28.1</td>
<td>Detailed Description</td>
<td>1110</td>
</tr>
<tr>
<td>9.29</td>
<td>hal_flash.c</td>
<td>1110</td>
</tr>
<tr>
<td>9.29.1</td>
<td>Detailed Description</td>
<td>1111</td>
</tr>
<tr>
<td>9.30</td>
<td>hal_flash.h</td>
<td>1111</td>
</tr>
<tr>
<td>9.30.1</td>
<td>Detailed Description</td>
<td>1112</td>
</tr>
<tr>
<td>9.31</td>
<td>hal_gpt.c</td>
<td>1113</td>
</tr>
<tr>
<td>9.31.1</td>
<td>Detailed Description</td>
<td>1113</td>
</tr>
<tr>
<td>9.32</td>
<td>hal_gpt.h</td>
<td>1113</td>
</tr>
<tr>
<td>Section</td>
<td>File Reference</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>9.95.1</td>
<td>hal_spi_lld.h</td>
<td>1191</td>
</tr>
<tr>
<td>9.96.1</td>
<td>hal_spi_lld.h</td>
<td>1191</td>
</tr>
<tr>
<td>9.97.1</td>
<td>hal_st.c</td>
<td>1192</td>
</tr>
<tr>
<td>9.98.1</td>
<td>hal_st.c</td>
<td>1193</td>
</tr>
<tr>
<td>9.99.1</td>
<td>hal_st_lld.c</td>
<td>1194</td>
</tr>
<tr>
<td>9.100.1</td>
<td>hal_st_lld.h</td>
<td>1194</td>
</tr>
<tr>
<td>9.101.1</td>
<td>hal_streams.h</td>
<td>1195</td>
</tr>
<tr>
<td>9.102.1</td>
<td>hal_trng.c</td>
<td>1195</td>
</tr>
<tr>
<td>9.103.1</td>
<td>hal_trng.h</td>
<td>1196</td>
</tr>
<tr>
<td>9.104.1</td>
<td>hal_trng_lld.c</td>
<td>1197</td>
</tr>
<tr>
<td>9.105.1</td>
<td>hal_trng_lld.h</td>
<td>1197</td>
</tr>
<tr>
<td>9.106.1</td>
<td>hal_uart.c</td>
<td>1198</td>
</tr>
<tr>
<td>9.107.1</td>
<td>hal_uart.h</td>
<td>1199</td>
</tr>
<tr>
<td>9.108.1</td>
<td>hal_uart_lld.c</td>
<td>1200</td>
</tr>
<tr>
<td>9.109.1</td>
<td>hal_uart_lld.h</td>
<td>1200</td>
</tr>
<tr>
<td>9.110.1</td>
<td>hal_usb.c</td>
<td>1203</td>
</tr>
<tr>
<td>9.111.1</td>
<td>hal_usb.h</td>
<td>1205</td>
</tr>
<tr>
<td>9.112.1</td>
<td>hal_usb_cdc.h</td>
<td>1208</td>
</tr>
<tr>
<td>9.113.1</td>
<td>hal_usb_lld.c</td>
<td>1209</td>
</tr>
<tr>
<td>9.113.2</td>
<td>Variable</td>
<td>1211</td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
<td>1211</td>
</tr>
<tr>
<td></td>
<td>in</td>
<td>1211</td>
</tr>
<tr>
<td></td>
<td>out</td>
<td>1211</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>9.115</td>
<td>hal_wdg.c File Reference</td>
<td>1213</td>
</tr>
<tr>
<td>9.115.1</td>
<td>Detailed Description</td>
<td>1213</td>
</tr>
<tr>
<td>9.116</td>
<td>hal_wdg.h File Reference</td>
<td>1213</td>
</tr>
<tr>
<td>9.116.1</td>
<td>Detailed Description</td>
<td>1214</td>
</tr>
<tr>
<td>9.117</td>
<td>hal_wdg_lld.c File Reference</td>
<td>1214</td>
</tr>
<tr>
<td>9.117.1</td>
<td>Detailed Description</td>
<td>1214</td>
</tr>
<tr>
<td>9.118</td>
<td>hal_wdg_lld.h File Reference</td>
<td>1215</td>
</tr>
<tr>
<td>9.118.1</td>
<td>Detailed Description</td>
<td>1215</td>
</tr>
<tr>
<td>9.119</td>
<td>hal_wspi.c File Reference</td>
<td>1215</td>
</tr>
<tr>
<td>9.119.1</td>
<td>Detailed Description</td>
<td>1216</td>
</tr>
<tr>
<td>9.120</td>
<td>hal_wspi.h File Reference</td>
<td>1216</td>
</tr>
<tr>
<td>9.120.1</td>
<td>Detailed Description</td>
<td>1219</td>
</tr>
<tr>
<td>9.121</td>
<td>hal_wspi_lld.c File Reference</td>
<td>1219</td>
</tr>
<tr>
<td>9.121.1</td>
<td>Detailed Description</td>
<td>1220</td>
</tr>
<tr>
<td>9.122</td>
<td>hal_wspi_lld.h File Reference</td>
<td>1220</td>
</tr>
<tr>
<td>9.122.1</td>
<td>Detailed Description</td>
<td>1221</td>
</tr>
<tr>
<td>9.123</td>
<td>halconf.h File Reference</td>
<td>1221</td>
</tr>
<tr>
<td>9.123.1</td>
<td>Detailed Description</td>
<td>1223</td>
</tr>
<tr>
<td>9.124</td>
<td>memstreams.c File Reference</td>
<td>1223</td>
</tr>
<tr>
<td>9.124.1</td>
<td>Detailed Description</td>
<td>1224</td>
</tr>
<tr>
<td>9.125</td>
<td>memstreams.h File Reference</td>
<td>1224</td>
</tr>
<tr>
<td>9.125.1</td>
<td>Detailed Description</td>
<td>1224</td>
</tr>
<tr>
<td>9.126</td>
<td>nullstreams.c File Reference</td>
<td>1224</td>
</tr>
<tr>
<td>9.126.1</td>
<td>Detailed Description</td>
<td>1225</td>
</tr>
<tr>
<td>9.127</td>
<td>nullstreams.h File Reference</td>
<td>1225</td>
</tr>
<tr>
<td>9.127.1</td>
<td>Detailed Description</td>
<td>1225</td>
</tr>
<tr>
<td>9.128</td>
<td>osal.c File Reference</td>
<td>1225</td>
</tr>
<tr>
<td>9.128.1</td>
<td>Detailed Description</td>
<td>1227</td>
</tr>
<tr>
<td>9.129</td>
<td>osal.h File Reference</td>
<td>1227</td>
</tr>
<tr>
<td>9.129.1</td>
<td>Detailed Description</td>
<td>1230</td>
</tr>
</tbody>
</table>
Chapter 1

ChibiOS/HAL

1.1 Copyright

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

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

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

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

1.2 Introduction

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

1.3 Related Documents

- ChibiOS/HAL General Architecture
Chapter 2

Deprecated List

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

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

Global `sdGetWouldBlock` (SerialDriver *sdp)

Global `sdPutWouldBlock` (SerialDriver *sdp)
Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

HAL .......................................................... 838
 Peripheral Interfaces ....................................... 510
 Generic Flash Interface ................................... 217
 Configuration ............................................... 841
 Normal Drivers ............................................. 855
 HAL Driver .................................................. 23
 ADC Driver .................................................. 29
 CAN Driver .................................................. 80
 Crypto Driver ............................................... 104
 DAC Driver .................................................. 181
 EFL Driver ................................................... 205
 GPT Driver ................................................... 228
 I2C Driver ................................................... 247
 I2S Driver ................................................... 264
 ICU Driver ................................................... 277
 MAC Driver .................................................. 349
 PAL Driver ................................................... 457
 PWM Driver .................................................. 511
 RTC Driver ................................................... 565
 SDC Driver ................................................... 578
 Serial Driver ............................................... 612
 SIO Driver ................................................... 644
 SPI Driver ................................................... 661
 ST Driver ................................................... 697
 TRNG Driver ............................................... 708
 UART Driver ............................................... 717
 USB Driver ................................................... 753
 WDG Driver ................................................... 804
 WSPI Driver ................................................... 811
 Complex Drivers .............................................. 857
 Managed Flash Storage Driver ......................... 369
 MMC over SPI Driver .................................... 406
 Serial NOR Flash Driver ................................ 443
 Serial over USB Driver ................................... 632
 Interfaces and Classes .................................... 858
Module Index

Blocks ................................................................. 301
Abstract I/O Block Device ........................................ 340
Streams ................................................................. 302
  Abstract Streams Interface ...................................... 303
  Abstract I/O Channel Interface ................................. 307
  Abstract Files Interface .......................................... 317
  Abstract Persistent Storage Interface ......................... 325
  Memory Streams Class ............................................. 330
  Null Streams Class ................................................. 332
  Output Formatter Utility ......................................... 334
Base Object ........................................................ 455
Inner Code .......................................................... 859
  I/O Buffers Queues ............................................... 53
  MMC/SD Block Device ............................................ 430
  I/O Bytes Queues ................................................ 540
Support Code ....................................................... 860
  MII/RMII Header .................................................. 391
  USB CDC Header .................................................. 802
OSAL ............................................................... 861
Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

BaseBlockDeviceVMT .................................................. 901
  MMCSDBlockDeviceVMT ........................................... 1009
  MMCDriverVMT ..................................................... 1005
  SDCDriverVMT ..................................................... 1029
BaseFlashVMT ...................................................... 908
  EFlashDriverVMT ................................................ 929
  SNORDriverVMT ................................................... 1046
BaseObject .......................................................... 908
  BaseBlockDevice .................................................. 898
    MMCSDBlockDevice ............................................. 1006
    MMCDriver ..................................................... 1002
  BaseFlash ........................................................ 905
    hal_efl_driver ................................................. 952
    SNORDriver ................................................... 1043
  BasePersistentStorage ........................................ 910
  BaseSequentialStream ......................................... 913
  BaseChannel ..................................................... 902
    BaseAsynchronousChannel ..................................... 895
    SerialDriver ................................................ 1031
    SerialUSBDriver ............................................. 1038
  FileStream ...................................................... 931
    RTCDriverVMT ................................................ 1024
    MemoryStream ................................................ 987
    NullStream ................................................... 1010
BaseObjectVMT ..................................................... 910
BasePersistentStorageVMT ..................................... 912
BaseSequentialStreamVMT ...................................... 915
  BaseChannelVMT ................................................ 904
    BaseAsynchronousChannelVMT ................................ 897
    SerialDriverVMT ............................................. 1034
    SerialUSBDriverVMT ......................................... 1040
  FileStreamVMT .................................................. 934
  BlockDeviceInfo ............................................... 916
<table>
<thead>
<tr>
<th>Class/Hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDCConfig</td>
</tr>
<tr>
<td>SDCDriver</td>
</tr>
<tr>
<td>SerialConfig</td>
</tr>
<tr>
<td>SerialUSBConfig</td>
</tr>
<tr>
<td>SHA1Context</td>
</tr>
<tr>
<td>SHA256Context</td>
</tr>
<tr>
<td>SHA512Context</td>
</tr>
<tr>
<td>SNORConfig</td>
</tr>
<tr>
<td>threads_queue_t</td>
</tr>
<tr>
<td>UARTConfig</td>
</tr>
<tr>
<td>UARTDriver</td>
</tr>
<tr>
<td>unpacked_mmc_cid_1_t</td>
</tr>
<tr>
<td>unpacked_mmc_csd_1_t</td>
</tr>
<tr>
<td>unpacked_sdc_cid_1_t</td>
</tr>
<tr>
<td>unpacked_sdc_csd_10_t</td>
</tr>
<tr>
<td>unpacked_sdc_csd_20_t</td>
</tr>
<tr>
<td>USBConfig</td>
</tr>
<tr>
<td>USBDescriptor</td>
</tr>
<tr>
<td>USBDriver</td>
</tr>
<tr>
<td>USBEndpointConfig</td>
</tr>
<tr>
<td>USBInEndpointState</td>
</tr>
<tr>
<td>USBOutEndpointState</td>
</tr>
<tr>
<td>WDGConfig</td>
</tr>
<tr>
<td>WDGDriver</td>
</tr>
<tr>
<td>wspi_command_t</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

- BaseAsynchronousChannel
  - Base asynchronous channel class ........................................................................ 895
- BaseAsynchronousChannelVMT
  - BaseAsynchronousChannel virtual methods table .............................................. 897
- BaseBlockDevice
  - Base block device class ...................................................................................... 898
- BaseBlockDeviceVMT
  - BaseBlockDevice virtual methods table ............................................................. 901
- BaseChannel
  - Base channel class ............................................................................................ 902
- BaseChannelVMT
  - BaseChannel virtual methods table ................................................................... 904
- BaseFlash
  - Base flash class ................................................................................................ 905
- BaseFlashVMT
  - BaseFlash virtual methods table ....................................................................... 908
- BaseObject
  - Base stream class ............................................................................................ 908
- BaseObjectVMT
  - BaseObject virtual methods table ..................................................................... 910
- BasePersistentStorage
  - Base persistent storage class .......................................................................... 910
- BasePersistentStorageVMT
  - BasePersistentStorage virtual methods table ...................................................... 912
- BaseSequentialStream
  - Base stream class ............................................................................................ 913
- BaseSequentialStreamVMT
  - BaseSequentialStream virtual methods table ..................................................... 915
- BlockDeviceInfo
  - Block device info ................................................................................................ 916
- CANConfig
  - Driver configuration structure ........................................................................... 917
- CANDriver
  - Structure representing an CAN driver .............................................................. 917
- CANRxFrame
  - CAN received frame .......................................................................................... 921
<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANTxFrame</td>
<td>CAN transmission frame</td>
<td>924</td>
</tr>
<tr>
<td>cdc_linecoding_t</td>
<td>Type of Line Coding structure</td>
<td>927</td>
</tr>
<tr>
<td>CRYConfig</td>
<td>Driver configuration structure</td>
<td>927</td>
</tr>
<tr>
<td>CRYDriver</td>
<td>Structure representing an CRY driver</td>
<td>928</td>
</tr>
<tr>
<td>EFlashDriverVMT</td>
<td>EFlash virtual methods table</td>
<td>929</td>
</tr>
<tr>
<td>event_source</td>
<td>Events source object</td>
<td>930</td>
</tr>
<tr>
<td>FileStream</td>
<td>Base file stream class</td>
<td>931</td>
</tr>
<tr>
<td>FileStreamVMT</td>
<td>FileStream virtual methods table</td>
<td>934</td>
</tr>
<tr>
<td>flash_descriptor_t</td>
<td>Type of a flash device descriptor</td>
<td>935</td>
</tr>
<tr>
<td>flash_sector_descriptor_t</td>
<td>Flash sector descriptor</td>
<td>937</td>
</tr>
<tr>
<td>GPTConfig</td>
<td>Driver configuration structure</td>
<td>938</td>
</tr>
<tr>
<td>GPTDriver</td>
<td>Structure representing a GPT driver</td>
<td>940</td>
</tr>
<tr>
<td>hal_adc_config</td>
<td>Driver configuration structure</td>
<td>941</td>
</tr>
<tr>
<td>hal_adc_configuration_group</td>
<td>Conversion group configuration structure</td>
<td>941</td>
</tr>
<tr>
<td>hal_adc_driver</td>
<td>Structure representing an ADC driver</td>
<td>944</td>
</tr>
<tr>
<td>hal_dac_config</td>
<td>Driver configuration structure</td>
<td>946</td>
</tr>
<tr>
<td>hal_dac_conversion_group</td>
<td>DAC Conversion group structure</td>
<td>947</td>
</tr>
<tr>
<td>hal_dac_driver</td>
<td>Structure representing a DAC driver</td>
<td>948</td>
</tr>
<tr>
<td>hal_efl_config</td>
<td>Type of a structure representing a flash driver configuration</td>
<td>951</td>
</tr>
<tr>
<td>hal_efl_driver</td>
<td>Structure representing an embedded flash driver</td>
<td>952</td>
</tr>
<tr>
<td>hal_i2s_config</td>
<td>Driver configuration structure</td>
<td>954</td>
</tr>
<tr>
<td>hal_i2s_driver</td>
<td>Structure representing an I2S driver</td>
<td>956</td>
</tr>
<tr>
<td>hal_sio_config</td>
<td>Driver configuration structure</td>
<td>957</td>
</tr>
<tr>
<td>hal_sio_driver</td>
<td>Structure representing a SIO driver</td>
<td>959</td>
</tr>
<tr>
<td>hal_spi_config</td>
<td>Driver configuration structure</td>
<td>960</td>
</tr>
<tr>
<td>hal_spi_driver</td>
<td>Structure representing an SPI driver</td>
<td>962</td>
</tr>
<tr>
<td>hal_trng_config</td>
<td>Driver configuration structure</td>
<td>964</td>
</tr>
<tr>
<td>hal_trng_driver</td>
<td>Structure representing a TRNG driver</td>
<td>964</td>
</tr>
<tr>
<td>hal_wspi_config</td>
<td>Driver configuration structure</td>
<td>965</td>
</tr>
</tbody>
</table>
### 5.1 Data Structures

<table>
<thead>
<tr>
<th>Structure/Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>hal_wspi_driver</td>
<td>967</td>
</tr>
<tr>
<td>Structure representing an WSPI driver</td>
<td></td>
</tr>
<tr>
<td>HMACSHA256Context</td>
<td>968</td>
</tr>
<tr>
<td>Type of a HMAC_SHA256 context</td>
<td></td>
</tr>
<tr>
<td>HMACSHA512Context</td>
<td>969</td>
</tr>
<tr>
<td>Type of a HMAC_SHA512 context</td>
<td></td>
</tr>
<tr>
<td>I2CConfig</td>
<td>969</td>
</tr>
<tr>
<td>Type of I2C driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>I2CDriver</td>
<td>970</td>
</tr>
<tr>
<td>Structure representing an I2C driver</td>
<td></td>
</tr>
<tr>
<td>ICUConfig</td>
<td>971</td>
</tr>
<tr>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>ICUDriver</td>
<td>973</td>
</tr>
<tr>
<td>Structure representing an ICU driver</td>
<td></td>
</tr>
<tr>
<td>io_buffers_queue</td>
<td>974</td>
</tr>
<tr>
<td>Structure of a generic buffers queue</td>
<td></td>
</tr>
<tr>
<td>io_queue</td>
<td>979</td>
</tr>
<tr>
<td>Generic I/O queue structure</td>
<td></td>
</tr>
<tr>
<td>IOBus</td>
<td>981</td>
</tr>
<tr>
<td>I/O bus descriptor</td>
<td></td>
</tr>
<tr>
<td>MACConfig</td>
<td>983</td>
</tr>
<tr>
<td>Driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>MACDriver</td>
<td>984</td>
</tr>
<tr>
<td>Structure representing a MAC driver</td>
<td></td>
</tr>
<tr>
<td>MACReceiveDescriptor</td>
<td>985</td>
</tr>
<tr>
<td>Structure representing a receive descriptor</td>
<td></td>
</tr>
<tr>
<td>MACTransmitDescriptor</td>
<td>986</td>
</tr>
<tr>
<td>Structure representing a transmit descriptor</td>
<td></td>
</tr>
<tr>
<td>MemoryStream</td>
<td>987</td>
</tr>
<tr>
<td>Memory stream object</td>
<td></td>
</tr>
<tr>
<td>MemStreamVMT</td>
<td>990</td>
</tr>
<tr>
<td>MemStream virtual methods table</td>
<td></td>
</tr>
<tr>
<td>mfs_bank_header_t</td>
<td>990</td>
</tr>
<tr>
<td>Type of a bank header</td>
<td></td>
</tr>
<tr>
<td>mfs_data_header_t</td>
<td>992</td>
</tr>
<tr>
<td>Type of a data block header</td>
<td></td>
</tr>
<tr>
<td>mfs_transaction_op_t</td>
<td>993</td>
</tr>
<tr>
<td>Type of a buffered write/erase operation within a transaction</td>
<td></td>
</tr>
<tr>
<td>MFSConfig</td>
<td>995</td>
</tr>
<tr>
<td>Type of a MFS configuration structure</td>
<td></td>
</tr>
<tr>
<td>MFSDriver</td>
<td>997</td>
</tr>
<tr>
<td>Type of an MFS instance</td>
<td></td>
</tr>
<tr>
<td>MMCCConfig</td>
<td>1001</td>
</tr>
<tr>
<td>MMC/SD over SPI driver configuration structure</td>
<td></td>
</tr>
<tr>
<td>MMCDriver</td>
<td>1002</td>
</tr>
<tr>
<td>Structure representing a MMC/SD over SPI driver</td>
<td></td>
</tr>
<tr>
<td>MMCDriverVMT</td>
<td>1005</td>
</tr>
<tr>
<td>MMCDriver virtual methods table</td>
<td></td>
</tr>
<tr>
<td>MMCSDBlockDevice</td>
<td>1006</td>
</tr>
<tr>
<td>MCC/SD block device class</td>
<td></td>
</tr>
<tr>
<td>MMCSDBlockDeviceVMT</td>
<td>1009</td>
</tr>
<tr>
<td>MMCSDBlockDevice virtual methods table</td>
<td></td>
</tr>
<tr>
<td>NullStream</td>
<td>1010</td>
</tr>
<tr>
<td>Null stream object</td>
<td></td>
</tr>
<tr>
<td>NullStreamVMT</td>
<td>1012</td>
</tr>
<tr>
<td>NullStream virtual methods table</td>
<td></td>
</tr>
<tr>
<td>PALConfig</td>
<td>1012</td>
</tr>
<tr>
<td>Generic I/O ports static initializer</td>
<td></td>
</tr>
<tr>
<td>Data Structure</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>palevent_t</strong></td>
<td>Type of a PAL event record</td>
</tr>
<tr>
<td><strong>PWMChannelConfig</strong></td>
<td>Type of a PWM driver channel configuration structure</td>
</tr>
<tr>
<td><strong>PWMConfig</strong></td>
<td>Type of a PWM driver configuration structure</td>
</tr>
<tr>
<td><strong>PWMDriver</strong></td>
<td>Structure representing a PWM driver</td>
</tr>
<tr>
<td><strong>RTCAlarm</strong></td>
<td>Type of a structure representing an RTC alarm time stamp</td>
</tr>
<tr>
<td><strong>RTCDateTime</strong></td>
<td>Type of a structure representing an RTC date/time stamp</td>
</tr>
<tr>
<td><strong>RTCDev</strong></td>
<td>Structure representing an RTC driver</td>
</tr>
<tr>
<td><strong>RTCDevVMT</strong></td>
<td><strong>RTCDev</strong> virtual methods table</td>
</tr>
<tr>
<td><strong>SDCConfig</strong></td>
<td>Driver configuration structure</td>
</tr>
<tr>
<td><strong>SDCDriver</strong></td>
<td>Structure representing an SDC driver</td>
</tr>
<tr>
<td><strong>SDCDriverVMT</strong></td>
<td><strong>SDCDriver</strong> virtual methods table</td>
</tr>
<tr>
<td><strong>SerialConfig</strong></td>
<td>PLATFORM Serial Driver configuration structure</td>
</tr>
<tr>
<td><strong>SerialDriver</strong></td>
<td>Full duplex serial driver class</td>
</tr>
<tr>
<td><strong>SerialDriverVMT</strong></td>
<td><strong>SerialDriver</strong> virtual methods table</td>
</tr>
<tr>
<td><strong>SerialUSBConfig</strong></td>
<td>Serial over USB Driver configuration structure</td>
</tr>
<tr>
<td><strong>SerialUSBDev</strong></td>
<td>Full duplex serial driver class</td>
</tr>
<tr>
<td><strong>SerialUSBDevVMT</strong></td>
<td><strong>SerialUSBDev</strong> virtual methods table</td>
</tr>
<tr>
<td><strong>SHA1Context</strong></td>
<td>Type of a SHA1 context</td>
</tr>
<tr>
<td><strong>SHA256Context</strong></td>
<td>Type of a SHA256 context</td>
</tr>
<tr>
<td><strong>SHA512Context</strong></td>
<td>Type of a SHA512 context</td>
</tr>
<tr>
<td><strong>SNORConfig</strong></td>
<td>Type of a SNOR configuration structure</td>
</tr>
<tr>
<td><strong>SNORDriver</strong></td>
<td>Type of SNOR flash class</td>
</tr>
<tr>
<td><strong>SNORDriverVMT</strong></td>
<td>SNOR virtual methods table</td>
</tr>
<tr>
<td><strong>threads_queue_t</strong></td>
<td>Type of a thread queue</td>
</tr>
<tr>
<td><strong>UARTConfig</strong></td>
<td>Driver configuration structure</td>
</tr>
<tr>
<td><strong>UARTDriver</strong></td>
<td>Structure representing an UART driver</td>
</tr>
<tr>
<td><strong>unpacked_mmc_cid_t</strong></td>
<td>Unpacked CID register from MMC</td>
</tr>
<tr>
<td><strong>unpacked_mmc_csd_t</strong></td>
<td>Unpacked CSD register from MMC</td>
</tr>
<tr>
<td><strong>unpacked_sdc_cid_t</strong></td>
<td>Unpacked CID register from SDC</td>
</tr>
</tbody>
</table>
5.1 Data Structures

unpacked_sdc_csd_10_t
   Unpacked CSD v1.0 register from SDC ........................................ 1054
unpacked_sdc_csd_20_t
   Unpacked CSD v2.0 register from SDC ........................................ 1055
USBConfig
   Type of an USB driver configuration structure .............................. 1055
USBDescriptor
   Type of an USB descriptor .................................................. 1058
USBDriver
   Structure representing an USB driver ...................................... 1059
USBEndpointConfig
   Type of an USB endpoint configuration structure ......................... 1063
USBInEndpointState
   Type of an IN endpoint state structure ................................... 1067
USBOutEndpointState
   Type of an OUT endpoint state structure .................................. 1068
WDGConfig
   Driver configuration structure ............................................. 1070
WDGDriver
   Structure representing a WDG driver ....................................... 1070
wspi_command_t
   Type of a WSPI command descriptor ...................................... 1071
Chapter 6

File Index

6.1 File List

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

- **chprintf.c**: Mini printf-like functionality .................................................. 1073
- **chprintf.h**: Mini printf-like functionality .................................................. 1073
- **hal.c**: HAL subsystem code ................................................................. 1074
- **hal.h**: HAL subsystem header ............................................................... 1075
- **hal_adc.c**: ADC Driver code ................................................................. 1076
- **hal_adc.h**: ADC Driver macros and structures ........................................ 1077
- **hal_adc_lld.c**: PLATFORM ADC subsystem low level driver source .......... 1079
- **hal_adc_lld.h**: PLATFORM ADC subsystem low level driver header .......... 1080
- **hal_buffers.c**: I/O Buffers code ............................................................. 1081
- **hal_buffers.h**: I/O Buffers macros and structures .................................... 1082
- **hal_can.c**: CAN Driver code ................................................................. 1084
- **hal_can.h**: CAN Driver macros and structures ......................................... 1085
- **hal_can_lld.c**: PLATFORM CAN subsystem low level driver source .......... 1086
- **hal_can_lld.h**: PLATFORM CAN subsystem low level driver header .......... 1087
- **hal_channels.h**: I/O channels access ..................................................... 1089
- **hal_crypto.c**: Cryptographic Driver code ............................................... 1090
- **hal_crypto.h**: Cryptographic Driver macros and structures ...................... 1092
- **hal_crypto_lld.c**: PLATFORM cryptographic subsystem low level driver source 1096
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hal_crypto_lld.h</code></td>
<td>PLATFORM cryptographic subsystem low level driver header</td>
<td>1098</td>
</tr>
<tr>
<td><code>hal_dac.c</code></td>
<td>DAC Driver code</td>
<td>1101</td>
</tr>
<tr>
<td><code>hal_dac.h</code></td>
<td>DAC Driver macros and structures</td>
<td>1102</td>
</tr>
<tr>
<td><code>hal_dac_lld.c</code></td>
<td>PLATFORM DAC subsystem low level driver source</td>
<td>1104</td>
</tr>
<tr>
<td><code>hal_dac_lld.h</code></td>
<td>PLATFORM DAC subsystem low level driver header</td>
<td>1105</td>
</tr>
<tr>
<td><code>hal_efl.c</code></td>
<td>Embedded Flash Driver code</td>
<td>1106</td>
</tr>
<tr>
<td><code>hal_efl.h</code></td>
<td>Embedded Flash Driver macros and structures</td>
<td>1106</td>
</tr>
<tr>
<td><code>hal_efl_lld.c</code></td>
<td>PLATFORM Embedded Flash subsystem low level driver source</td>
<td>1107</td>
</tr>
<tr>
<td><code>hal_efl_lld.h</code></td>
<td>PLATFORM Embedded Flash subsystem low level driver header</td>
<td>1108</td>
</tr>
<tr>
<td><code>hal_files.h</code></td>
<td>Data files</td>
<td>1109</td>
</tr>
<tr>
<td><code>hal_flash.c</code></td>
<td>Generic flash driver class code</td>
<td>1110</td>
</tr>
<tr>
<td><code>hal_flash.h</code></td>
<td>Generic flash driver class header</td>
<td>1111</td>
</tr>
<tr>
<td><code>hal_gpt.c</code></td>
<td>GPT Driver code</td>
<td>1113</td>
</tr>
<tr>
<td><code>hal_gpt.h</code></td>
<td>GPT Driver macros and structures</td>
<td>1113</td>
</tr>
<tr>
<td><code>hal_gpt_lld.c</code></td>
<td>PLATFORM GPT subsystem low level driver source</td>
<td>1115</td>
</tr>
<tr>
<td><code>hal_gpt_lld.h</code></td>
<td>PLATFORM GPT subsystem low level driver header</td>
<td>1115</td>
</tr>
<tr>
<td><code>hal_i2c.c</code></td>
<td>I2C Driver code</td>
<td>1116</td>
</tr>
<tr>
<td><code>hal_i2c.h</code></td>
<td>I2C Driver macros and structures</td>
<td>1117</td>
</tr>
<tr>
<td><code>hal_i2c_lld.c</code></td>
<td>PLATFORM I2C subsystem low level driver source</td>
<td>1119</td>
</tr>
<tr>
<td><code>hal_i2c_lld.h</code></td>
<td>PLATFORM I2C subsystem low level driver header</td>
<td>1119</td>
</tr>
<tr>
<td><code>hal_i2s.c</code></td>
<td>I2S Driver code</td>
<td>1120</td>
</tr>
<tr>
<td><code>hal_i2s.h</code></td>
<td>I2S Driver macros and structures</td>
<td>1121</td>
</tr>
<tr>
<td><code>hal_i2s_lld.c</code></td>
<td>PLATFORM I2S subsystem low level driver source</td>
<td>1122</td>
</tr>
<tr>
<td><code>hal_i2s_lld.h</code></td>
<td>PLATFORM I2S subsystem low level driver header</td>
<td>1123</td>
</tr>
<tr>
<td><code>hal_icu.c</code></td>
<td>ICU Driver code</td>
<td>1124</td>
</tr>
<tr>
<td><code>hal_icu.h</code></td>
<td>ICU Driver macros and structures</td>
<td>1124</td>
</tr>
<tr>
<td><code>hal_icu_lld.c</code></td>
<td>PLATFORM ADC subsystem low level driver source</td>
<td>1126</td>
</tr>
<tr>
<td><code>hal_icu_lld.h</code></td>
<td>PLATFORM ADC subsystem low level driver header</td>
<td>1127</td>
</tr>
<tr>
<td><code>hal_ioblock.h</code></td>
<td>I/O block devices access</td>
<td>1128</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>hal_lld.c</td>
<td>PLATFORM HAL subsystem low level driver source</td>
<td>1129</td>
</tr>
<tr>
<td>hal_lld.h</td>
<td>PLATFORM HAL subsystem low level driver header</td>
<td>1130</td>
</tr>
<tr>
<td>hal_mac.c</td>
<td>MAC Driver code</td>
<td>1130</td>
</tr>
<tr>
<td>hal_mac.h</td>
<td>MAC Driver macros and structures</td>
<td>1131</td>
</tr>
<tr>
<td>hal_mac_lld.c</td>
<td>PLATFORM MAC subsystem low level driver source</td>
<td>1132</td>
</tr>
<tr>
<td>hal_mac_lld.h</td>
<td>PLATFORM MAC subsystem low level driver header</td>
<td>1133</td>
</tr>
<tr>
<td>hal_mfs.c</td>
<td>Managed Flash Storage module code</td>
<td>1134</td>
</tr>
<tr>
<td>hal_mfs.h</td>
<td>Managed Flash Storage module header</td>
<td>1136</td>
</tr>
<tr>
<td>hal_mii.h</td>
<td>MII macros and structures</td>
<td>1137</td>
</tr>
<tr>
<td>hal_mmc_spi.c</td>
<td>MMC over SPI driver code</td>
<td>1140</td>
</tr>
<tr>
<td>hal_mmc_spi.h</td>
<td>MMC over SPI driver header</td>
<td>1141</td>
</tr>
<tr>
<td>hal_mmcsd.c</td>
<td>MMC/SD cards common code</td>
<td>1142</td>
</tr>
<tr>
<td>hal_mmcsd.h</td>
<td>MMC/SD cards common header</td>
<td>1143</td>
</tr>
<tr>
<td>hal_objects.h</td>
<td>Base object</td>
<td>1147</td>
</tr>
<tr>
<td>hal_pal.c</td>
<td>I/O Ports Abstraction Layer code</td>
<td>1148</td>
</tr>
<tr>
<td>hal_pal.h</td>
<td>I/O Ports Abstraction Layer macros, types and structures</td>
<td>1149</td>
</tr>
<tr>
<td>hal_pal_lld.c</td>
<td>PLATFORM PAL subsystem low level driver source</td>
<td>1152</td>
</tr>
<tr>
<td>hal_pal_lld.h</td>
<td>PLATFORM PAL subsystem low level driver header</td>
<td>1153</td>
</tr>
<tr>
<td>hal_persistent.h</td>
<td>Generic persistent storage class header</td>
<td>1155</td>
</tr>
<tr>
<td>hal_pwm.c</td>
<td>PWM Driver code</td>
<td>1156</td>
</tr>
<tr>
<td>hal_pwm.h</td>
<td>PWM Driver macros and structures</td>
<td>1156</td>
</tr>
<tr>
<td>hal_pwm_lld.c</td>
<td>PLATFORM PWM subsystem low level driver source</td>
<td>1158</td>
</tr>
<tr>
<td>hal_pwm_lld.h</td>
<td>PLATFORM PWM subsystem low level driver header</td>
<td>1159</td>
</tr>
<tr>
<td>hal_queues.c</td>
<td>I/O Queues code</td>
<td>1161</td>
</tr>
<tr>
<td>hal_queues.h</td>
<td>I/O Queues macros and structures</td>
<td>1162</td>
</tr>
<tr>
<td>hal_rtc.c</td>
<td>RTC Driver code</td>
<td>1164</td>
</tr>
<tr>
<td>hal_rtc.h</td>
<td>RTC Driver macros and structures</td>
<td>1164</td>
</tr>
<tr>
<td>hal_rtc_lld.c</td>
<td>PLATFORM RTC subsystem low level driver source</td>
<td>1166</td>
</tr>
<tr>
<td>hal_rtc_lld.h</td>
<td>PLATFORM RTC subsystem low level driver header</td>
<td>1166</td>
</tr>
<tr>
<td>File Name</td>
<td>Description</td>
<td>Page No.</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>hal_sdc.c</td>
<td>SDC Driver code</td>
<td>1168</td>
</tr>
<tr>
<td>hal_sdc.h</td>
<td>SDC Driver macros and structures</td>
<td>1169</td>
</tr>
<tr>
<td>hal_sdc_lld.c</td>
<td>PLATFORM SDC subsystem low level driver source</td>
<td>1171</td>
</tr>
<tr>
<td>hal_sdc_lld.h</td>
<td>PLATFORM SDC subsystem low level driver header</td>
<td>1172</td>
</tr>
<tr>
<td>hal_serial.c</td>
<td>Serial Driver code</td>
<td>1174</td>
</tr>
<tr>
<td>hal_serial.h</td>
<td>Serial Driver macros and structures</td>
<td>1175</td>
</tr>
<tr>
<td>hal_serial_lld.c</td>
<td>PLATFORM serial subsystem low level driver source</td>
<td>1177</td>
</tr>
<tr>
<td>hal_serial_lld.h</td>
<td>PLATFORM serial subsystem low level driver header</td>
<td>1177</td>
</tr>
<tr>
<td>hal_serial_nor.c</td>
<td>Serial NOR serial flash driver code</td>
<td>1178</td>
</tr>
<tr>
<td>hal_serial_nor.h</td>
<td>Serial NOR driver header</td>
<td>1179</td>
</tr>
<tr>
<td>hal_serial_usb.c</td>
<td>Serial over USB Driver code</td>
<td>1181</td>
</tr>
<tr>
<td>hal_serial_usb.h</td>
<td>Serial over USB Driver macros and structures</td>
<td>1182</td>
</tr>
<tr>
<td>hal_sio.c</td>
<td>SIO Driver code</td>
<td>1183</td>
</tr>
<tr>
<td>hal_sio.h</td>
<td>SIO Driver macros and structures</td>
<td>1184</td>
</tr>
<tr>
<td>hal_sio_lld.c</td>
<td>PLATFORM SIO subsystem low level driver source</td>
<td>1185</td>
</tr>
<tr>
<td>hal_sio_lld.h</td>
<td>PLATFORM SIO subsystem low level driver header</td>
<td>1186</td>
</tr>
<tr>
<td>hal_spi.c</td>
<td>SPI Driver code</td>
<td>1187</td>
</tr>
<tr>
<td>hal_spi.h</td>
<td>SPI Driver macros and structures</td>
<td>1188</td>
</tr>
<tr>
<td>hal_spi_lld.c</td>
<td>PLATFORM SPI subsystem low level driver source</td>
<td>1190</td>
</tr>
<tr>
<td>hal_spi_lld.h</td>
<td>PLATFORM SPI subsystem low level driver header</td>
<td>1191</td>
</tr>
<tr>
<td>hal_st.c</td>
<td>ST Driver code</td>
<td>1192</td>
</tr>
<tr>
<td>hal_st.h</td>
<td>ST Driver macros and structures</td>
<td>1193</td>
</tr>
<tr>
<td>hal_st_lld.c</td>
<td>PLATFORM ST subsystem low level driver source</td>
<td>1194</td>
</tr>
<tr>
<td>hal_st_lld.h</td>
<td>PLATFORM ST subsystem low level driver header</td>
<td>1194</td>
</tr>
<tr>
<td>hal_streams.h</td>
<td>Data streams</td>
<td>1195</td>
</tr>
<tr>
<td>hal_trng.c</td>
<td>TRNG Driver code</td>
<td>1195</td>
</tr>
<tr>
<td>hal_trng.h</td>
<td>TRNG Driver macros and structures</td>
<td>1196</td>
</tr>
<tr>
<td>hal_trng_lld.c</td>
<td>PLATFORM TRNG subsystem low level driver source</td>
<td>1197</td>
</tr>
<tr>
<td>hal_trng_lld.h</td>
<td>PLATFORM TRNG subsystem low level driver header</td>
<td>1198</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>hal_uart.c</td>
<td>UART Driver code</td>
<td>1198</td>
</tr>
<tr>
<td>hal_uart.h</td>
<td>UART Driver macros and structures</td>
<td>1199</td>
</tr>
<tr>
<td>hal_uart_lld.c</td>
<td>PLATFORM UART subsystem low level driver source</td>
<td>1202</td>
</tr>
<tr>
<td>hal_uart_lld.h</td>
<td>PLATFORM UART subsystem low level driver header</td>
<td>1202</td>
</tr>
<tr>
<td>hal_usb.c</td>
<td>USB Driver code</td>
<td>1203</td>
</tr>
<tr>
<td>hal_usb.h</td>
<td>USB Driver macros and structures</td>
<td>1205</td>
</tr>
<tr>
<td>hal_usb_cdc.h</td>
<td>USB CDC macros and structures</td>
<td>1208</td>
</tr>
<tr>
<td>hal_usb_lld.c</td>
<td>PLATFORM USB subsystem low level driver source</td>
<td>1209</td>
</tr>
<tr>
<td>hal_usb_lld.h</td>
<td>PLATFORM USB subsystem low level driver header</td>
<td>1211</td>
</tr>
<tr>
<td>hal_wdg.c</td>
<td>WDG Driver code</td>
<td>1213</td>
</tr>
<tr>
<td>hal_wdg.h</td>
<td>WDG Driver macros and structures</td>
<td>1213</td>
</tr>
<tr>
<td>hal_wdg_lld.c</td>
<td>WDG Driver subsystem low level driver source template</td>
<td>1214</td>
</tr>
<tr>
<td>hal_wdg_lld.h</td>
<td>WDG Driver subsystem low level driver header template</td>
<td>1215</td>
</tr>
<tr>
<td>hal_wspi.c</td>
<td>WSPI Driver code</td>
<td>1215</td>
</tr>
<tr>
<td>hal_wspi.h</td>
<td>WSPI Driver macros and structures</td>
<td>1216</td>
</tr>
<tr>
<td>hal_wspi_lld.c</td>
<td>PLATFORM WSPI subsystem low level driver source</td>
<td>1219</td>
</tr>
<tr>
<td>hal_wspi_lld.h</td>
<td>PLATFORM WSPI subsystem low level driver header</td>
<td>1220</td>
</tr>
<tr>
<td>halconf.h</td>
<td>HAL configuration header</td>
<td>1221</td>
</tr>
<tr>
<td>memstreams.c</td>
<td>Memory streams code</td>
<td>1223</td>
</tr>
<tr>
<td>memstreams.h</td>
<td>Memory streams structures and macros</td>
<td>1224</td>
</tr>
<tr>
<td>nullstreams.c</td>
<td>Null streams code</td>
<td>1224</td>
</tr>
<tr>
<td>nullstreams.h</td>
<td>Null streams structures and macros</td>
<td>1225</td>
</tr>
<tr>
<td>osal.c</td>
<td>OSAL module code</td>
<td>1225</td>
</tr>
<tr>
<td>osal.h</td>
<td>OSAL module header</td>
<td>1227</td>
</tr>
</tbody>
</table>
Chapter 7

Module Documentation

7.1 HAL Driver

Hardware Abstraction Layer.

7.1.1 Detailed Description

Hardware Abstraction Layer.

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

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

Macros

- `#define _CHIBIOS_HAL_`
  ChibiOS/HAL identification macro.
- `#define CH_HAL_STABLE 1`
  Stable release flag.

ChibiOS/HAL version identification

- `#define HAL_VERSION "7.1.5"`
  HAL version string.
- `#define CH_HAL_MAJOR 7`
  HAL version major number.
- `#define CH_HAL_MINOR 1`
  HAL version minor number.
- `#define CH_HAL_PATCH 5`
  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"

Functions

- void halInit (void)
  HAL initialization.
- void hal_lld_init (void)
  Low level HAL driver initialization.

7.1.2 Macro Definition Documentation

7.1.2.1 CHIBIOS_HAL_

#define CHIBIOS_HAL_

ChibiOS/HAL identification macro.

7.1.2.2 CH_HAL_STABLE

#define CH_HAL_STABLE 1

Stable release flag.

7.1.2.3 HAL_VERSION

#define HAL_VERSION "7.1.5"

HAL version string.
7.1 HAL Driver

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 1

HAL version minor number.

7.1.2.6 CH_HAL_PATCH

#define CH_HAL_PATCH 5

HAL version patch number.

7.1.2.7 HAL_SUCCESS

#define HAL_SUCCESS false

HAL operation success.

7.1.2.8 HAL_FAILED

#define HAL_FAILED true

HAL operation failed.

7.1.3 Function Documentation
7.1.3.1 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.
7.1.3.2  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.
7.2 ADC Driver

Generic ADC Driver.

7.2.1 Detailed Description

Generic ADC Driver.

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

Precondition

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

7.2.2 Driver State Machine

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

7.2.3 ADC Operations

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

The ADCConversionGroup is the 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 in 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.

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.
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.
7.2 ADC Driver

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.

- `enum adcerr_t { ADC_ERR_DMAFAILURE = 0, ADC_ERR_OVERFLOW = 1, ADC_ERR_AWD = 2 }
` Possible ADC failure causes.

Variables

- `ADCDriver ADCD1`
  `ADC1 driver identifier.`

7.2.4 Macro Definition Documentation

7.2.4.1 ADC_USE_WAIT

```c
#define ADC_USE_WAIT TRUE
```
Enables synchronous APIs.

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

7.2.4.2 ADC_USE_MUTUAL_EXCLUSION

```c
#define ADC_USE_MUTUAL_EXCLUSION TRUE
```
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

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

7.2.4.3 adcIsBufferComplete

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

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

| in | adcp | pointer to the ADCDriver 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.

7.2.4.4  _adc_reset_i

#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

#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 |
7.2 ADC Driver

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

7.2.4.6 _adc_wakeup_isr

#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

#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

#define _adc_isr_half_code(
   adcp )

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

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

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

Parameters
in 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(
   adcp )

Value:
{   
   if ((adcp)->grpp->circular) {
      /* Callback handling.*/
      if ((adcp)->grpp->end_cb != NULL) {
         (adcp)->state = ADC_COMPLETE;
         (adcp)->grpp->end_cb(adcp);
      }
      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 {
      /* 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->end_cb(adcp);
         }
      }
   }
}

ChibiOS/HAL
7.2 ADC Driver

Common ISR code, full buffer event.

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  adcp  pointer to the ADCDriver object
```

Function Class:

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

7.2.4.10 _adc_isr_error_code

```c
#define _adc_isr_error_code(
    adcp,
    err )
```

Value:

```c
adc_lld_stop_conversion(adcp);
if ((adcp)->grpp->error_cb != NULL) {
    (adcp)->state = ADC_ERROR;
    (adcp)->grpp->error_cb(adcp, err);
    if ((adcp)->state == ADC_ERROR) {
        (adcp)->state = ADC_READY;
        (adcp)->grpp = NULL;
    }
} else {
    (adcp)->state = ADC_READY;
    (adcp)->grpp = NULL;
}_adc_timeout_isr(adcp);
```

Common ISR code, error event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread timeout signaling, if any.
- Driver state transitions.
Note
This macro is meant to be used in the low level drivers implementation only.

Parameters

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

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

7.2.4.11 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.

7.2.4.12 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.

7.2.4.13 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.
7.2.4.14   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 adcerrrorcallback_t

typedef void(* adcerrrorcallback_t)(ADCDriver *adcp, adcerrror_t err)

Type of an ADC error callback.

Parameters

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

### 7.2.6 Enumeration Type Documentation

### 7.2.6.1 adcstate_t

enum adcstate_t

Driver state machine possible states.
### 7.2 ADC Driver

#### 7.2.6 ADC States

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

#### 7.2.6.2 adccerror_t

```c
enum adccerror_t
```

Possible ADC failure causes.

**Note**

Error codes are architecture dependent and should not relied upon.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_ERR_DMAFAILURE</td>
<td>DMA operations failure.</td>
</tr>
<tr>
<td>ADC_ERR_OVERFLOW</td>
<td>ADC overflow condition.</td>
</tr>
<tr>
<td>ADC_ERR_AWD</td>
<td>Analog watchdog triggered.</td>
</tr>
</tbody>
</table>

#### 7.2.7 Function Documentation

##### 7.2.7.1 adcInit()

```c
void adcInit ( void )
```

ADC Driver initialization.

**Note**

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

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

Here is the call graph for this function:

```
adcInit -> adc_lld_init -> adcObjectInit
```

### 7.2.7.2 adcObjectInit()

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

Initializes the standard part of a ADCDriver structure.

**Parameters**
- **out adcp**: pointer to the ADCDriver object

Function Class:

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

Here is the call graph for this function:

```
adcObjectInit -> osalMutexObjectInit
```

### 7.2.7.3 adcStart()

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

Configures and activates the ADC peripheral.
7.2 ADC Driver

Parameters

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

```
adcStart
   ↓
   adc_lld_start
   ↓
   osalSysLock
   ↓
   osalSysUnlock
```

7.2.7.4 adcStop()

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

Deactivates the ADC peripheral.

Parameters

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

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

Here is the call graph for this function:

```
adcStop
  adc_lld_stop
    osalSysLock
    osalSysUnlock
```

### 7.2.7.5 adcStartConversion()

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

Starts an ADC conversion.

Starts an asynchronous conversion operation.

**Note**

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>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 for adcStartConversionI()]

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

**Function Class:**

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

```
adcStartConversionI  adc_lld_start_conversion
```

### 7.2.7.7 adcStopConversion()

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

Stops an ongoing conversion.

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

**Parameters**

- **in adcp** pointer to the ADCDriver object

**Function Class:**

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

Here is the call graph for this function:

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

```c
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:

![Call Graph]

### 7.2.7.9 adcConvert()

```c
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 $M \times N$ 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>adcp</th>
<th>pointer to the ADCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>grpp</td>
<td>pointer to a ADCConversionGroup object</td>
</tr>
<tr>
<td>out</td>
<td>samples</td>
<td>pointer to the samples buffer</td>
</tr>
<tr>
<td>in</td>
<td>depth</td>
<td>buffer depth (matrix rows number). The buffer depth must be one or an even number.</td>
</tr>
</tbody>
</table>

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 acdStopConversion() or acdStopConversion1(), 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:

```
adcConvert
    ↘
    ↘
adcStartConversionI
osalSysLock
osalThreadSuspendS
osalSysUnlock
adc_lld_start_conversion
```

### 7.2.7.10 adcAcquireBus()

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

Gains exclusive access to the ADC peripheral.

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

Precondition

In order to use this function the option ADC_USE_MUTUAL_EXCLUSION must be enabled.
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()

```c
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:

```
adc_lld_init → adcObjectInit → osalMutexObjectInit
```

7.2.7.13 adc_lld_start()

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

Configures and activates the ADC peripheral.

Parameters
- `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()

```c
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()

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()

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
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

- **BUFFERS_CHUNKS_SIZE 64**
  
  Maximum size of blocks copied in critical sections.

- **BQ_BUFFER_SIZE(n, size)**
  
  Computes the size of a buffers queue buffer size.

### Macro Functions

- **bqSizeX(bqp)**
  
  Returns the queue's number of buffers.

- **bqSpaceX(bqp)**
  
  Returns the ready buffers number.

- **bqGetLinkX(bqp)**
  
  Returns the queue application-defined link.

- **bqSetLinkX(bqp, lk)**
  
  Sets the queue application-defined link.

- **bqIsSuspendedX(bqp)**
  
  Returns the suspended state of the queue.

- **bqSuspendI(bqp)**
  
  Puts the queue in suspended state.

- **bqResumeX(bqp)**
  
  Resumes normal queue operations.

- **ibqIsEmptyI(ibqp)**
  
  Evaluates to true if the specified input buffers queue is empty.

- **ibqIsFullI(ibqp)**
  
  Evaluates to true if the specified input buffers queue is full.

- **obqIsEmptyI(obqp)**
  
  Evaluates to true if the specified output buffers queue is empty.

- **obqIsFullI(obqp)**
  
  Evaluates to true if the specified output buffers queue is full.
Typedefs

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

Data Structures

- struct io_buffers_queue
  Structure of a generic buffers queue.

Functions

- void ibqObjectInit (inputBuffersQueue_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.
7.3 I/O Buffers Queues

- `msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t *obqp, sysinterval_t timeout)`
  Gets the next empty buffer from the queue.

- `void obqPostFullBuffer (output_buffers_queue_t *obqp, size_t size)`
  Posts a new filled buffer to the queue.

- `void obqPostFullBufferS (output_buffers_queue_t *obqp, size_t size)`
  Posts a new filled buffer to the queue.

- `msg_t obqPutTimeout (output_buffers_queue_t *obqp, uint8_t b, sysinterval_t timeout)`
  Output queue write with timeout.

- `size_t obqWriteTimeout (output_buffers_queue_t *obqp, const uint8_t *bp, size_t n, sysinterval_t timeout)`
  Output queue write with timeout.

- `bool obqTryFlushI (output_buffers_queue_t *obqp)`
  Flushes the current, partially filled, buffer to the queue.

- `void obqFlush (output_buffers_queue_t *obqp)`
  Flushes the current, partially filled, buffer to the queue.

---

7.3.2 Macro Definition Documentation

7.3.2.1 BUFFERS_CHUNKS_SIZE

```c
#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

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

Computes the size of a buffers queue buffer size.

**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

#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

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

Return the ready buffers number.

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

Parameters

  in  bqp  pointer to an io_buffers_queue_t structure

Returns

  The number of ready buffers.

Function Class:

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

7.3.2.5 bqGetLinkX

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

Returns the queue application-defined link.
7.3 I/O Buffers Queues

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

#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

#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 )

Value:

```
{  
  (bqp)->suspended = true; 
  osalThreadDequeueAllI(&(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 )

Value:

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

Resumes normal queue operations.
Parameters

\[
\begin{array}{|c|c|c|}
\hline
\text{in} & bqp & \text{pointer to an io\_buffers\_queue\_t structure} \\
\hline
\end{array}
\]

Function Class:

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

### 7.3.2.10 ibqIsEmptyI

\[
\text{#define ibqIsEmptyI}( \\
\quad \text{ibqp} ) \ (\text{bool}) \ (\text{bqSpaceI}(\text{ibqp}) \ == \ 0U) \\
\]

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

Parameters

\[
\begin{array}{|c|c|c|}
\hline
\text{in} & \text{ibqp} & \text{pointer to an input\_buffers\_queue\_t structure} \\
\hline
\end{array}
\]

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 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

\[
\text{#define ibqIsFullI}( \\
\quad \text{ibqp} ) \\
\]

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.
Parameters

| in | ibqp | pointer to an input_buffers_queue_t structure |

Returns

The queue status.

Return values

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

Function Class:

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

7.3.2.12 obqIsEmpty

#define obqIsEmpty(
    obqp)

Value:

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

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

Parameters

| in | obqp | pointer to an output_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 I/O Buffers Queues

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 output_buffers_queue_t structure

Returns
- The queue status.

Return values
- false if the queue is not full.
- true if the queue is full.

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

7.3.3 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) (io_buffers_queue_t *bqp)

Double buffer notification callback type.
Module Documentation

7.3.3.3 input_buffers_queue_t

typedef io_buffers_queue_t input_buffers_queue_t

Type of an input buffers queue.

7.3.3.4 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ibqp pointer to the input_buffers_queue_t object</td>
</tr>
<tr>
<td>in</td>
<td>suspended initial state of the queue</td>
</tr>
<tr>
<td>in</td>
<td>bp pointer to a memory area allocated for buffers</td>
</tr>
<tr>
<td>in</td>
<td>size buffers size</td>
</tr>
<tr>
<td>in</td>
<td>n number of buffers</td>
</tr>
<tr>
<td>in</td>
<td>infy callback called when a buffer is returned to the queue</td>
</tr>
<tr>
<td>in</td>
<td>link 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:

```
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**

| in | ibqp | pointer to the input_buffers_queue_t object |

**Returns**

A pointer to the next buffer to be filled.

**Return values**

- `NULL` if the queue is full.

**Function Class:**

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

### 7.3.4.4 ibqPostFullBufferI()

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

Posts a new filled buffer to the queue.

**Parameters**

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

**Function Class:**

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

![Call Graph]

### 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 object</code></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:

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

Here is the call graph for this function:

```
ibqGetFullBufferTimeout
ibqGetFullBufferTimeoutS
osalSysLock
osalSysUnlock
```

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

### 7.3.4.7 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:

```
ibqReleaseEmptyBuffer
    ibqReleaseEmptyBufferS
        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**

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

### 7.3.4.9 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**

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

**Returns**

A byte value from the queue.
Return values

<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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>ibqp</code></td>
</tr>
<tr>
<td>out</td>
<td><code>bp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>n</code></td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Returns

The 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:

```
ibqReadTimeout
ibqGetFullBufferTimeoutS
osalSysLock
```

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>out</td>
<td>obqp</td>
</tr>
<tr>
<td>in</td>
<td>suspended</td>
</tr>
<tr>
<td>in</td>
<td>bp</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
<tr>
<td>in</td>
<td>onfy</td>
</tr>
<tr>
<td>in</td>
<td>link</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:

![Call Graph](obqObjectInit => osalThreadQueueObjectInit)

### 7.3.4.12 obqResetI()

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

Resets an output buffers queue.

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

**Note**

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

**Parameters**

- **in obqp** pointer to the `output_buffers_queue_t` object

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](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 output_buffers_queue_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sizep</td>
<td>pointer to the filled buffer size</td>
</tr>
</tbody>
</table>

**Returns**

A pointer to the filled buffer.

**Return values**

- **NULL** if the queue is empty.

**Function Class:**

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

7.3.4.14 obqReleaseEmptyBufferI()

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

Releases the next filled buffer back in the queue.

**Parameters**

| in  | obqp | pointer to the output_buffers_queue_t object |

**Function Class:**

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

Here is the call graph for this function:

```
obqReleaseEmptyBufferI
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
| 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>in</th>
<th>obqp</th>
<th>pointer to the <code>output_buffers_queue_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>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></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_OK</code></td>
<td>if a buffer has been acquired.</td>
</tr>
<tr>
<td><code>MSG_TIMEOUT</code></td>
<td>if the specified time expired.</td>
</tr>
<tr>
<td><code>MSG_RESET</code></td>
<td>if the queue has been reset or has been put in suspended state.</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

7.3.4.16 obqGetEmptyBufferTimeoutS()

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></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>• 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

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

Function Class:

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 ( 
    output_buffers_queue_t * obqp, 
    size_t size )
```

Posts a new filled buffer to the queue.

Note

The object callback is called after releasing the buffer.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>obqp pointer to the output_buffers_queue_t object</td>
</tr>
<tr>
<td>in</td>
<td>size 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 Diagram]

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

**Function Class:**

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

Returns

A byte value from the queue.

Return values

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

Function Class:

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

Here is the call graph for this function:

```
    obqPutTimeout
       ^                       ^
       |                       |
    obqGetEmptyBufferTimeoutS  osalSysLock
```

7.3.4.20  `obqWriteTimeout()`

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

Output queue write with timeout.

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

| in | obqp | pointer to the output_buffers_queue_t object |
| in | bp   | pointer to the data buffer |
| in | n    | the maximum amount of data to be transferred, the value 0 is reserved |
| 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 number of bytes effectively transferred.

### Return values

| 0  | if a timeout occurred. |

### Function Class:

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

Here is the call graph for this function:

![Call Graph](obqTryFlushI)  

#### 7.3.4.21 obqTryFlushI()

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

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

### Note

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

| in | obqp | pointer to the output_buffers_queue_t object |

Returns

The operation status.

Return values

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

Function Class:

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

7.3.4.22 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.
7.4 CAN Driver

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)
  Forbids 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 CAND1`
  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

```c
#define CAN_BUS_OFF_ERROR 4U
```

Bus off condition reached.

### 7.4.3.4 CAN_FRAMING_ERROR

```c
#define CAN_FRAMING_ERROR 8U
```

Framing error of some kind on the CAN bus.

### 7.4.3.5 CAN_OVERFLOW_ERROR

```c
#define CAN_OVERFLOW_ERROR 16U
```

Overflow in receive queue.

### 7.4.3.6 CAN_ANY_MAILBOX

```c
#define CAN_ANY_MAILBOX 0U
```

Special mailbox identifier.

### 7.4.3.7 CAN_USE_SLEEP_MODE

```c
#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

```c
#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();
osalThreadDequeueAllI(&(canp)->txqueue, MSG_OK);
osalEventBroadcastFlagsI(&(canp)->txempty_event, flags);
osalSysUnlockFromISR();

TX mailbox empty event.
7.4.3.13 _can_rx_full_isr

#define _can_rx_full_isr(
  canp,
  flags )

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

RX mailbox empty full event.

7.4.3.14 _can_wakeup_isr

#define _can_wakeup_isr(
  canp )

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

Wakeup event.

7.4.3.15 _can_error_isr

#define _can_error_isr(
  canp,
  flags )

Value:
  {
    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.
Parameters

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

### 7.4.5 Enumeration Type Documentation

#### 7.4.5.1 canstate_t

```c
enum canstate_t
```

Driver state machine possible states.

**Enumerator**

<table>
<thead>
<tr>
<th>ENUM</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>CAN_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>CAN_STARTING</td>
<td>Starting.</td>
</tr>
<tr>
<td>CAN_STOPPING</td>
<td>Stopping.</td>
</tr>
<tr>
<td>CAN_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>CAN_SLEEP</td>
<td>Sleep state.</td>
</tr>
</tbody>
</table>

### 7.4.6 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:

7.4.6.2 canObjectInit()

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

Initializes the standard part of a CANDriver structure.

Parameters

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

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

7.4.6.4 canStop()

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

Deactivates the CAN peripheral.
### 7.4 CAN Driver

**Parameters**

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

![Call Graph](call_graph.png)

#### 7.4.6.5 `canTryTransmitI()`

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

Can frame transmission attempt.

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

**Parameters**

- **in** `canp` pointer to the `CANDriver` object
- **in** `mailbox` mailbox number, `CAN_ANY_MAILBOX` for any mailbox
- **in** `ctfp` pointer to the CAN frame to be transmitted

**Returns**

The operation result.
Return values

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

Function Class:

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

Here is the call graph for this function:

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>
7.4 CAN Driver

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

    canTryReceiveI
       ^
        
    can_lld_receive
```

7.4.6.7 canTryAbortX()

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

Tries to abort an ongoing transmission.

Parameters

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

Function Class:

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

Here is the call graph for this function:

```
    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

| MSG_OK     | the frame has been queued for transmission. |
| MSG_TIMEOUT| The operation has timed out. |
| MSG_RESET  | The driver has been stopped while waiting. |
7.4 CAN 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]

7.4.6.9 canReceiveTimeout()

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

Can frame receive.

The function waits until a frame is received.

Note

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>canp</code> pointer to the CANDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>mailbox</code> mailbox number, CAN_ANY_MAILBOX for any mailbox</td>
</tr>
<tr>
<td>out</td>
<td><code>crfp</code> pointer to the buffer where the CAN frame is copied</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_IMMEDIATE</code> immediate timeout (useful in an event driven scenario where a thread never blocks for I/O).</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

ChibiOS/HAL
Returns

The operation result.

Return values

| MSG_OK     | a frame has been received and placed in the buffer. |
| MSG_TIMEOUT| The operation has timed out.                        |
| MSG_RESET  | The driver has been stopped while waiting.         |

Function Class:

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

Here is the call graph for this function:

```
  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

<table>
<thead>
<tr>
<th>in</th>
<th>canp</th>
<th>pointer to the <code>CANDriver</code> object</th>
</tr>
</thead>
</table>
7.4 CAN 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]

7.4.6.11 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.
Parameters

| in   | canp | pointer to the CANDriver object |

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

| in   | canp | pointer to the CANDriver object |

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

| in   | canp | pointer to the CANDriver object |
| in   | mailbox | mailbox number, CAN_ANY_MAILBOX for any mailbox |

Returns
The queue space availability.

Return values

| false | no space in the transmit queue. |
| true  | transmit slot available. |
Function Class:
Not an API, this function is for internal use only.

7.4.6.16 can_lld_transmit()

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

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

7.4.6.17 can_lld_is_rx_nonempty()

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

Returns
The queue space availability.

Return values

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

\[
\text{in } \text{canp} \text{ pointer to the CANDriver object}
\]

Function Class:

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

### 7.4.6.21 can_lld_wakeup()

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

Enforces leaving the sleep mode.

Parameters

\[
\text{in } \text{canp} \text{ pointer to the CANDriver object}
\]

Function Class:

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

### 7.4.7 Variable Documentation

#### 7.4.7.1 CAND1

```c
CANDriver CAND1
```

CAN1 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)
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)

Initializes the DES transient key.
• cryerror_t cryLoadDESTransientKey (CRYDriver *cryp, size_t size, const uint8_t *keyp)

Encryption of a single block using (T)DES.
• cryerror_t cryEncryptDES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)

Decryption of a single block using (T)DES.
• cryerror_t cryDecryptDES (CRYDriver *cryp, crykey_t key_id, const uint8_t *in, uint8_t *out)

Encryption operation using (T)DES-ECB.
• cryerror_t cryEncryptDES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)

Decryption operation using (T)DES-ECB.
• cryerror_t cryDecryptDES_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 cryEncryptDES_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 cryDecryptDES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)

Hash initialization using SHA1.
• cryerror_t crySHA1Init (CRYDriver *cryp, SHA1Context *sha1ctxp)

Hash update using SHA1.
• cryerror_t crySHA1Update (CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)

Hash finalization using SHA1.
• cryerror_t crySHA1Final (CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)

Hash initialization using SHA256.
• cryerror_t crySHA256Init (CRYDriver *cryp, SHA256Context *sha256ctxp)

Hash update 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.

• 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.
Decryption operation using AES-CFB.

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

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

- `cryerror_t cry_lld_encrypt_AES_GCM (CRYDriver *cryp, crykey_t key_id, size_t auth_size, const uint8_t *auth_in, size_t text_size, const uint8_t *text_in, uint8_t *text_out, const uint8_t *iv, size_t tag_size, uint8_t *tag_out)`
  
  Encryption operation using AES-GCM.

- `cryerror_t cry_lld_decrypt_AES_GCM (CRYDriver *cryp, crykey_t key_id, size_t auth_size, const uint8_t *auth_in, size_t text_size, const uint8_t *text_in, uint8_t *text_out, const uint8_t *iv, size_t tag_size, const uint8_t *tag_in)`
  
  Decryption operation using AES-GCM.

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

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

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

- `cryerror_t cry_lld_encrypt DES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`
  
  Encryption operation using (T)DES-ECB.

- `cryerror_t cry_lld_decrypt DES_ECB (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out)`
  
  Decryption operation using (T)DES-ECB.

- `cryerror_t cry_lld_encrypt DES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  
  Encryption operation using (T)DES-CBC.

- `cryerror_t cry_lld_decrypt DES_CBC (CRYDriver *cryp, crykey_t key_id, size_t size, const uint8_t *in, uint8_t *out, const uint8_t *iv)`
  
  Decryption operation using (T)DES-CBC.

- `cryerror_t cry_lld_SHA1_init (CRYDriver *cryp, SHA1Context *sha1ctxp)`
  
  Hash initialization using SHA1.

- `cryerror_t cry_lld_SHA1_update (CRYDriver *cryp, SHA1Context *sha1ctxp, size_t size, const uint8_t *in)`
  
  Hash update using SHA1.

- `cryerror_t cry_lld_SHA1_final (CRYDriver *cryp, SHA1Context *sha1ctxp, uint8_t *out)`
  
  Hash finalization using SHA1.

- `cryerror_t cry_lld_SHA256_init (CRYDriver *cryp, SHA256Context *sha256ctxp)`
  
  Hash initialization using SHA256.

- `cryerror_t cry_lld_SHA256_update (CRYDriver *cryp, SHA256Context *sha256ctxp, size_t size, const uint8_t *in)`
  
  Hash update using SHA256.

- `cryerror_t cry_lld_SHA256_final (CRYDriver *cryp, SHA256Context *sha256ctxp, uint8_t *out)`
  
  Hash finalization using SHA256.

- `cryerror_t cry_lld_SHA512_init (CRYDriver *cryp, SHA512Context *sha512ctxp)`
  
  Hash initialization using SHA512.

- `cryerror_t cry_lld_SHA512_update (CRYDriver *cryp, SHA512Context *sha512ctxp, size_t size, const uint8_t *in)`
  
  Hash update using SHA512.

- `cryerror_t cry_lld_SHA512_final (CRYDriver *cryp, SHA512Context *sha512ctxp, uint8_t *out)`
  
  Hash finalization using SHA512.
7.5 Crypto Driver

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

#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 a 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 cipher/mode.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>Invalid key size.</td>
</tr>
</tbody>
</table>
### 7.5.4.3 cryalgorithm_t

**enum 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>

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

### 7.5.5 Function Documentation

#### 7.5.5.1 cryInit()

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

Cryptographic Driver initialization.

**Note**

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

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

Here is the call graph for this function:

```
cryInit  ➔  cry_lld_init
```

### 7.5.5.2 cryObjectInit()

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

Initializes the standard part of a CRYDriver structure.

**Parameters**

| out cryp | pointer to the CRYDriver object |

Function Class:

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

### 7.5.5.3 cryStart()

```c
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. |
**Function Class:**

Normal API, 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)

### 7.5.5.4 cryStop()

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

Deactivates the cryptographic peripheral.

**Parameters**

- **in cryp** pointer to the CRYDriver object

Function Class:

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

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

Encryption of a single block using AES.

Note

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

Parameters

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

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>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>
7.5 Crypto Driver

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>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:

```
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**

- `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>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>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 Diagram](image)

7.5.5.10 cryEncryptAES_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-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 input vector</td>
</tr>
</tbody>
</table>
7.5 Crypto Driver

Returns

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

![Call Graph]

7.5.5.11 cryDecryptAES_CBC()

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

Decryption operation using AES-CBC.

Note

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

Parameters

| 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 ciphertext |
| out out buffer for the output plaintext |
| in iv 128 bits input vector |
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:

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

Here is the call graph for this function:

![Call Graph](cryDecryptAES_CBC cry_lld_decrypt_AES_CBC)

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

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

```
cryEncryptAES_CFB  cry_lld_encrypt_AES_CFB
```

### 7.5.5.13 cryDecryptAES_CFB()

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

Decryption operation using AES-CFB.

Note

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><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 ciphertext</td>
</tr>
<tr>
<td><code>out</code></td>
<td>buffer for the output plaintext</td>
</tr>
<tr>
<td><code>iv</code></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](cryDecryptAES_CFB | cry_lld_decrypt_AES_CFB)

7.5.5.14 cryEncryptAES_CTR()

crypterror_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 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 input vector + counter, it contains a 96 bits IV and a 32 bits counter</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

```
    cryEncryptAES_CTR
       ┌──cry_lld_encrypt_AES_CTR
```

7.5.5.15 cryDecryptAES_CTR()

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

Decryption operation using AES-CTR.

Note

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

Parameters

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

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

```
    cryDecryptAES_CTR
      ┌──────────────────┐
      │
      │ cry_lld_decrypt_AES_CTR │
      │
      └──────────────────┘
```

7.5.5.16 cryEncryptAES_GCM()

crypterror_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.
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>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 (cry
    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>
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:

```
    cryDecryptAES_GCM  
    └─ cry_lld_decrypt_AES_GCM
```

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:

```
    cryLoadDESTransientKey  cry_lld_des_loadkey
```

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

#### Returns
The operation status.

#### Return values

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

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

Here is the call graph for this function:

```
cryDecryptDES cry_lld_decrypt_DES
```

7.5.5.21 cryEncryptDES_ECB()

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

Encryption operation using (T)DES-ECB.

Note

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the 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

<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>
</tbody>
</table>
7.5 Crypto Driver

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRYPT_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
<tr>
<td>CRYPT_INV_KEY_ID</td>
<td>if the specified key identifier is invalid or refers to an empty key slot.</td>
</tr>
<tr>
<td>CRYPT_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_ECB               cry_lld_encryptDES_ECB
```

7.5.5.22 cryDecryptDES_ECB()

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

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

![Call Graph]

7.5.5.23 cryEncryptDES_CBC()

cyerror_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>in size</td>
<td>size of both buffers, this number must be a multiple of 8</td>
</tr>
<tr>
<td>in in</td>
<td>buffer containing the input 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_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_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()

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

Decryption operation using (T)DES-CBC.

Note

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</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>size</td>
<td>size of both buffers, this number must be a multiple of 8</td>
</tr>
<tr>
<td>in</td>
<td>in buffer containing the input ciphertext</td>
</tr>
<tr>
<td>out</td>
<td>out buffer for the output plaintext</td>
</tr>
<tr>
<td>in</td>
<td>iv 64 bits input vector</td>
</tr>
</tbody>
</table>
Returns
The operation status.

Return values

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

Function Class:
Normal API, 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
```

7.5.5.25 crySHA1Init()

cryerror_t crySHA1Init (  
    CRYDriver * cryp,  
    SHA1Context * sha1ctxp )

Hash initialization using SHA1.

Note
Use of this algorithm is not recommended because proven weak.

Parameters

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

```
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>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></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_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 ( 
    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>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>out</td>
<td>out</td>
<td>160 bits output buffer</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.
7.5 Crypto Driver

Return values

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

Function Class:

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

Here is the call graph for this function:

```
  crySHA1Final       cry_lld_SHA1_final
```

7.5.5.28 crySHA256Init()

cryerror_t crySHA256Init ( 
    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>

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](call_graph.png)

7.5.5.29  crySHA256Update()

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

Hash update using SHA256.

Parameters

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

Returns

The operation status.

Return values

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

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

Here is the call graph for this function:

```
crySHA256Update    cry_lld_SHA256_update
```

7.5.5.30 crySHA256Final()

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

Hash finalization using SHA256.

Parameters

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

Returns

The operation status.

Return values

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

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

Here is the call graph for this function:

![Call Graph]

### 7.5.5.31 crySHA512Init()

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

| 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. |
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:

```
crySHA512Init  
cry_lld_SHA512_init
```

7.5.5.32 crySHA512Update()

```c
cryerror_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></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:

```
crySHA512Update -> cry_lld_SHA512_update
```

### 7.5.5.33 crySHA512Final()

```c
cryerror_t crySHA512Final (  
    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></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>
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:

```
crySHA512Final  cry_lld_SHA512_final
```

7.5.5.34 cryLoadHMACTransientKey()

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

Initializes the HMAC transient key.

Note

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

Parameters

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

Returns

The operation status.

Return values

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

```
| cryLoadHMACTransientKey | cry_lld_hmac_loadkey |
```

7.5.5.35 cryHMACSHA256Init()

cryerror_t cryHMACSHA256Init (  
  CRYDriver * cryp,  
  HMACSHA256Context * hmacSHA256ctxp )

Hash initialization using HMAC_SHA256.

**Parameters**

| in | cryp | pointer to the CRYDriver object |
| out | hmacSHA256ctxp | pointer to a HMAC_SHA256 context to be initialized |

**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

| 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:

cryHMACSHA256Final -> cry_lld_HMACSHA256_final

7.5.5.38 cryHMACSHA512Init()

cryerror_t cryHMACSHA512Init (  
     CRYDriver * cryp,  
     HMACSHA512Context * hmacsha512ctxp )

Hash initialization using HMAC_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>hmacsha512ctxp</td>
<td>pointer to a HMAC_SHA512 context to be initialized</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

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

Function Class:

Normal API, 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()

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

Hash update using HMAC.

Parameters

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

ChibiOS/HAL
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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
</tr>
<tr>
<td>in</td>
<td>hmacsha512ctxp</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>pointer to a HMAC_SHA512 context</td>
</tr>
<tr>
<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:

```
cryHMACSHA512Final  cry_lld_HMACSHA512
        _final
```

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

<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>
</tbody>
</table>
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
- `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
- `cryp` pointer to the `CRYDriver` object
- `size` key size in bytes
- `keyp` pointer to the key data

Returns
The operation status.

Return values
- `CRY_NOERROR` if the operation succeeded.
Return values

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

Function Class:

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

### 7.5.5.45 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**

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

**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.
7.5.5.46 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

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

<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>
<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.

7.5.5.48 cry_lld_decrypt_AES_ECB()

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

Decryption operation using AES-ECB.

Note

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

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

Returns

The operation status.

Return values

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

Function Class:

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

7.5.5.49 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>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>Type</th>
<th>Parameter</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, this number must be a multiple of 16</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input plaintext</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td>in</td>
<td>iv</td>
<td>128 bits initial vector</td>
</tr>
</tbody>
</table>

Returns

The operation status.
Return values

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

Function Class:

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

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

Returns
The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the operation is unsupported on this device instance.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_TYPE</td>
<td>the selected key is invalid for this operation.</td>
</tr>
</tbody>
</table>
7.5 Crypto Driver

Return values

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

7.5.53 cry_lld_encrypt_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-CTR.

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

Parameters

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

Returns
The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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.
7.5.5.54 cry_lld_decrypt_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.

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

| 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.

7.5.5.55 cry_lld_encrypt_AES_GCM()

cryerror_t cry_lld_encrypt_AES_GCM {  
    CRYDriver * cryp,  
    crykey_t key_id,  
    ChibiOS/HAL
Encryption operation using AES-GCM.

Note

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
</tr>
<tr>
<td>in</td>
<td>key_id</td>
</tr>
<tr>
<td>in</td>
<td>auth_size</td>
</tr>
<tr>
<td>in</td>
<td>auth_in</td>
</tr>
<tr>
<td>in</td>
<td>text_size</td>
</tr>
<tr>
<td>in</td>
<td>text_in</td>
</tr>
<tr>
<td>out</td>
<td>text_out</td>
</tr>
<tr>
<td>in</td>
<td>iv</td>
</tr>
<tr>
<td>in</td>
<td>tag_size</td>
</tr>
<tr>
<td>out</td>
<td>tag_out</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

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

7.5.5.56 cry_lld_decrypt_AES_GCM()
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

<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:

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.
7.5.5 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.
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_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:

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

Returns

The operation status.

Return values

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

<table>
<thead>
<tr>
<th>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>Type</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 the plaintext buffer, this number must be a multiple of 8</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>
</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.

### 7.5.5.62 cry_lld_encrypt_DES_CBC()

```c
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>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 plaintext</td>
</tr>
<tr>
<td><code>out</code></td>
<td>buffer for the output ciphertext</td>
</tr>
<tr>
<td><code>iv</code></td>
<td>64 bits input vector</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

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

Function Class:

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>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 the plaintext buffer, this number must be a multiple of 8</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>64 bits input vector</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>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>cyp</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 * cyp,  
    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>cyp</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

| CRY_NOERROR | if the operation succeeded. |
Return values

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

Function Class:

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

7.5.5.66 cry_lld_SHA1_final()

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

Returns

The operation status.

Return values

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

Function Class:

Not an API, this function is for internal use only.
7.5.5.67 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

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

Function Class:

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

7.5.5.68 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

| CRY_NOERROR | if the operation succeeded. |
### 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**

- **in cryp** pointer to the CRYDriver object
- **in sha256ctxp** pointer to a SHA256 context
- **out out** 256 bits output buffer

**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_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.

**Parameters**

- **in cryp** pointer to the CRYDriver object
- **in sha512ctxp** pointer to a SHA512 context

**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_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.
Module Documentation

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

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

7.5.5.71 cry_lld_SHA512_update()

ccryerror_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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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.
7.5.5.72 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>sha512ctxp</td>
<td>pointer to a SHA512 context</td>
</tr>
<tr>
<td>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>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.

7.5.5.73 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryp</td>
<td>pointer to the CRYDriver object</td>
</tr>
<tr>
<td>size</td>
<td>key size in bytes</td>
</tr>
<tr>
<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>CRY_NOERROR</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>CRY_ERR_INV_ALGO</td>
<td>if the algorithm is unsupported.</td>
</tr>
<tr>
<td>CRY_ERR_INV_KEY_SIZE</td>
<td>if the specified key size is invalid for the specified algorithm.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.5.5.74 cry_lld_HMACSHA256_init()

```c
cryerror_t cry_lld_HMACSHA256_init (
    CRYDriver * cryp,
    HMACSHA256Context * hmacsha256ctxp
)
```

Hash initialization using HMAC_SHA256.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cryp</td>
</tr>
<tr>
<td>out</td>
<td>hmacsha256ctxp</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.

### 7.5.5.75 cry_lld_HMACSHA256_update()

```c
cryerror_t cry_lld_HMACSHA256_update (  
    CRYDriver * cryp,
```
Hash update using HMAC.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>hmacsha256ctxp</td>
<td>pointer to a HMAC_SHA256 context</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of input buffer</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>buffer containing the input text</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

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

Function Class:

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

7.5.5.76 cry_lld_HMACSHA256_final()

cryerror_t cry_lld_HMACSHA256_final (  
    CRYDriver * cryp,  
    HMACSHA256Context * hmacsha256ctxp,  
    uint8_t * out )

Hash finalization using HMAC.

Parameters

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

Returns

The operation status.
Return values

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

Function Class:

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

7.5.5.77 cry_lld_HMACSHA512_init()

cryerror_t cry_lld_HMACSHA512_init {
    CRYDriver * cryp,
    HMACSHA512Context * hmacsha512ctxp
}

Hash initialization using HMAC_SHA512.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cryp</th>
<th>pointer to the CRYDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>hmacsha512ctxp</td>
<td>pointer to a HMAC_SHA512 context to be initialized</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

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

Function Class:

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

7.5.5.78 cry_lld_HMACSHA512_update()

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

Hash update using HMAC.
7.5 Crypto Driver

Parameters

| in | cryp | pointer to the CRYDriver object |
| in | hmacsha512ctxp | pointer to a HMAC_SHA512 context |
| in | size | size of input buffer |
| in | in | buffer containing the input text |

Returns

The operation status.

Return values

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

Function Class:

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

7.5.5.79 cry_lld_HMACSHA512_final()

cryerror_t cry_lld_HMACSHA512_final (  
  CRYDriver * cryp,  
  HMACSHA512Context * hmacsha512ctxp,  
  uint8_t * out )

Hash finalization using HMAC.

Parameters

| in | cryp | pointer to the CRYDriver object |
| in | hmacsha512ctxp | pointer to a HMAC_SHA512 context |
| out | out | 512 bits output buffer |

Returns

The operation status.

Return values

| CRY_NOERROR | if the operation succeeded. |
| CRY_ERR_INV_ALGO | if the operation is unsupported on this device instance. |
| CRY_ERR_OP_FAILURE | if the operation failed, implementation dependent. |
Function Class:
   Not an API, this function is for internal use only.

7.5.6 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_l(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)`
  
  Resumes a thread waiting for a conversion completion.

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

- `#define _dac_wakeup_isr(dacp)`
  
  Wakes up the waiting thread.

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

- `#define _dac_isr_half_code(dacp)`
  
  Common ISR code, half buffer event.

- `#define _dac_isr_full_code(dacp)`
  
  Common ISR code, full buffer event.

- `#define _dac_isr_error_code(dacp, err)`
  
  Common ISR code, error event.
Configuration options

- `#define PLATFORM_DAC_USE_DAC1 FALSE`

  DAC1 CH1 driver enable switch.

Typedefs

- `typedef struct hal_dac_driver DACDriver`
  Type of a structure representing an DAC driver.
- `typedef struct hal_dac_config DACConfig`
  Type of a structure representing an DAC driver configuration.
- `typedef struct hal_dac_conversion_group DACConversionGroup`
  Type of a DAC conversion group.
- `typedef void(*daccallback_t)(DACDriver *dacp)`
  DAC notification callback type.
- `typedef void(*dacerrorcallback_t)(DACDriver *dacp, dacerror_t err)`
  DAC error callback type.
- `typedef uint32_t dacchannel_t`
  Type of a DAC channel index.
- `typedef uint16_t dacsample_t`
  Type representing a DAC sample.

Data Structures

- `struct hal_dac_conversion_group`
  DAC Conversion group structure.
- `struct hal_dac_config`
  Driver configuration structure.
- `struct hal_dac_driver`
  Structure representing a DAC driver.

Functions

- `void dacInit (void)`
  DAC Driver initialization.
- `void dacObjectInit (DACDriver *dacp)`
  Initializes the standard part of a DACDriver structure.
- `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, dacsamples_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, dacsamples_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

| in | dacp | pointer to the DACDriver object |

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

| in | dacp | pointer to the DACDriver object |

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

| in | dacp | pointer to the DACDriver object |
Function Class:

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

7.6.2.7 _dac_wakeup_isr

#define _dac_wakeup_isr(
    dacp )

Value:

{  
    osalSysLockFromISR();  
    osalThreadResumeI(&(dacp)->thread, MSG_OK);  
    osalSysUnlockFromISR();  
}

Wakes up the waiting thread.

Parameters

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

7.6.2.8 _dac_timeout_isr

#define _dac_timeout_isr(
    dacp )

Value:

{  
    osalSysLockFromISR();  
    osalThreadResumeI(&(dacp)->thread, MSG_TIMEOUT);  
    osalSysUnlockFromISR();  
}

Wakes up the waiting thread with a timeout message.

Parameters

<table>
<thead>
<tr>
<th>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 DAC Driver

7.6.2.9  _dac_isr_half_code

#define _dac_isr_half_code( 
   dacp )

Value:
{
   if ((dacp)->grpp->end_cb != NULL) {
      (dacp)->grpp->end_cb(dacp);
   }
}

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

• Callback invocation.

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

Parameters
in dacp pointer to the DACDriver object

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

7.6.2.10  _dac_isr_full_code

#define _dac_isr_full_code( 
   dacp )

Value:
{
   if ((dacp)->grpp->end_cb) {
      (dacp)->state = DAC_COMPLETE;
      (dacp)->grpp->end_cb(dacp);
      if ( (dacp)->state == DAC_COMPLETE) 
         (dacp)->state = DAC_ACTIVE;
   }
}

Common ISR code, full buffer event.

This code handles the portable part of the ISR code:

• Callback invocation.
• Driver state transitions.

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

#define _dac_isr_error_code(
  dacp,
  err
)

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.
7.6.2.12 DAC_MAX_CHANNELS

```c
#define DAC_MAX_CHANNELS 2
```

Maximum number of DAC channels per unit.

7.6.2.13 PLATFORM_DAC_USE_DAC1

```c
#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

```c
#define dac_lld_driver_fields
```

Value:

```c
/* 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

```c
#define dac_lld_config_fields
```

Value:

```c
/* 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

```c
#define dac_lld_conversion_group_fields
```

Value:

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

Low level fields of the DAC group configuration structure.
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 daccp**: 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>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_UNINIT</td>
</tr>
<tr>
<td>DAC_STOP</td>
</tr>
<tr>
<td>DAC_READY</td>
</tr>
<tr>
<td>DAC_ACTIVE</td>
</tr>
<tr>
<td>DAC_COMPLETE</td>
</tr>
<tr>
<td>DAC_ERROR</td>
</tr>
</tbody>
</table>
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>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_ERR_DMAFAILURE</td>
<td>DMA operations failure.</td>
</tr>
<tr>
<td>DAC_ERR_UNDERFLOW</td>
<td>DAC overflow condition.</td>
</tr>
</tbody>
</table>

7.6.5 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 * dacp )

Initializes the standard part of a DACDriver structure.
### 7.6 DAC Driver

**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:

```
dacObjectInit → osalMutexObjectInit
```

### 7.6.5.3 dacStart()

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

Configures and activates the DAC peripheral.

**Parameters**

| in | dacp | pointer to the DACDriver object |
| in | config | pointer to the DACConfig object, it can be NULL if the low level driver implementation supports a default configuration |
Function Class:

Normal API, 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
     \   /  
    osalSysLock
     |   |
     |   |
    osalSysUnlock
    |
    \-> dac_lld_start
```

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 |
7.6 DAC Driver

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

Here is the call graph for this function:

7.6.5.5 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dacp pointer to the DACDriver object</td>
</tr>
<tr>
<td>in</td>
<td>channel DAC channel number</td>
</tr>
<tr>
<td>in</td>
<td>sample 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:
# 7.6.5.6 dacStartConversion()

```c
void dacStartConversion (  
    DACDriver * dacp,  
    const DACConversionGroup * grpp,  
    dacsample_t * samples,  
    size_t depth )
```

Starts a DAC conversion.

Starts an asynchronous conversion operation.

**Note**

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>dacp</th>
<th>pointer to the DACDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>grpp</td>
<td>pointer to a DACConversionGroup object</td>
</tr>
<tr>
<td>in</td>
<td>samples</td>
<td>pointer to the samples buffer</td>
</tr>
<tr>
<td>in</td>
<td>depth</td>
<td>buffer depth (matrix rows number). The buffer depth must be one or an even number.</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

![Call Graph Diagram](image-url)

# 7.6.5.7 dacStartConversionI()

```c
void dacStartConversionI (  
    DACDriver * dacp,
```

...
7.6 DAC Driver

```c
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 $M \times N$ 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

- *dacp* pointer to the DACDriver object
- *grpp* pointer to a DACConversionGroup object
- *samples* pointer to the samples buffer
- *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:

```
    dacStartConversionI -> dac_lld_start_conversion
```

### 7.6.5.8 dacStopConversion()

```c
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.
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:

```
    dacStopConversion
    ↓
    dac_lld_stop_conversion
    ↓
    osalSysLock
    ↓
    osalSysUnlock
```

### 7.6.5.9 dacStopConversionI()

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

Stops an ongoing conversion.

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

Parameters

| 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.

Here is the call graph for this function:

![Call Graph]

7.6.5.10  **dacConvert()**

```c
msg_t dacConvert ( 
    DACDriver ∗ dacp, 
    const DACConversionGroup ∗ grpp, 
    dacsample_t ∗ samples, 
    size_t depth 
)
```

Performs a DAC conversion.

Performs a synchronous conversion operation.

**Note**

The buffer is organized as a matrix of $M \times N$ 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dacp</td>
<td>pointer to the <strong>DACDriver</strong> object</td>
</tr>
<tr>
<td>in</td>
<td>grpp</td>
<td>pointer to a <strong>DACConversionGroup</strong> 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>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSG_OK</strong></td>
<td>Conversion finished.</td>
</tr>
<tr>
<td><strong>MSG_RESET</strong></td>
<td>The conversion has been stopped using <strong>acdStopConversion()</strong> or <strong>acdStopConversionI()</strong>, the result buffer may contain incorrect data.</td>
</tr>
<tr>
<td><strong>MSG_TIMEOUT</strong></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:

7.6.5.12 dacReleaseBus()

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

Releases exclusive access to the DAC bus.

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:
7.6.5.13 dac_lld_init()

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()

void dac_lld_start (  
    DACDriver * dacp  )

Configures and activates the DAC peripheral.

Parameters  
in 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()

void dac_lld_stop (  
    DACDriver * dacp  )

Deactivates the DAC peripheral.
7.6 DAC Driver

Parameters

| in | dacp | pointer to the DACDriver object |

Function Class:

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

7.6.5.16 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

| in | dacp | pointer to the DACDriver object |
| in | channel | DAC channel number |
| in | sample | value to be output |

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.
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()

```c
void dac_lld_stop_conversion(
    DACDriver * dACP
)
```

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

```c
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 an sector erase operation.*

- **flash_error_t efl_lld_query_erase (void *instance, uint32_t *msec)**

  *Queries the driver for erase operation progress.*

- **flash_error_t efl_lld_verify_erase (void *instance, flash_sector_t sector)**

  *Returns the erase state of a sector.*

Variables

- **EFlashDriver EFLD1**

  *EFL1 driver identifier.*

7.7.2  Macro Definition Documentation

7.7.2.1  _efl_flash_methods_alone

#define _efl_flash_methods_alone

EFlashDriver specific methods.
7.7 EFL Driver

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.
7.7.2.6  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 ( )

  void )

Embedded Flash Driver initialization.

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

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

Here is the call graph for this function:
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:

```
eflStart  ->  osalSysLock
```
7.7.4.4  eflStop()

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

Stops the driver.

**Parameters**

| 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 *eflp)
```

Configures and activates the Embedded Flash peripheral.

**Parameters**

| in | eflp | pointer to a EFlashDriver structure |

**Function Class:**

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

### 7.7.4.7  efl_lld_stop()

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

Deactivates the Embedded Flash peripheral.

**Parameters**

| in | eflp | pointer to a EFlashDriver structure |

**Function Class:**

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

### 7.7.4.8  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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>instance</code></td>
<td>pointer to a EFlashDriver instance</td>
</tr>
<tr>
<td><code>offset</code></td>
<td>flash offset</td>
</tr>
<tr>
<td><code>n</code></td>
<td>number of bytes to be read</td>
</tr>
<tr>
<td><code>rp</code></td>
<td>pointer to the data buffer</td>
</tr>
</tbody>
</table>

Returns
An error code.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_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_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:

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_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:
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_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:
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>Flash 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()

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in instance</td>
<td>pointer to a EFlashDriver instance</td>
</tr>
<tr>
<td>in sector</td>
<td>sector to be verified</td>
</tr>
</tbody>
</table>

Returns
An error code.

Return values

<table>
<thead>
<tr>
<th>Flash 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 a 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`.
7.8.3.4  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 *pp);
/* Erase whole flash device.*/
flash_error_t (*start_erase_all)(void *instance);
/* Erase single sector.*/
flash_error_t (*start_erase_sector)(void *instance,
flash_sector_t sector);
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

```c
#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

```c
#define _base_flash_data

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

_BaseFlash specific data.

### 7.8.3.10 getBaseFlash

```c
#define getBaseFlash(ip) ((BaseFlash ∗)(ip)->vmt)
```

Instance getter.

This special method is used to get the instance of this class object from a derived class.

### 7.8.3.11 flashGetDescriptor

```c
#define flashGetDescriptor(ip) (ip)->vmt->get_descriptor(ip)
```

Gets the flash descriptor structure.

**Parameters**

```
in ip pointer to a BaseFlash or derived class
```

**Returns**

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

7.8.3.12 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>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:
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>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_PROGRAM</td>
<td>if the program operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.8.3.14 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>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>
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>

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>

Function Class:

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

Return values

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

### 7.8.3.17 flashVerifyErase

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

Returns the erase state of a sector.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ip</td>
<td>pointer to a <code>BaseFlash</code> 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>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if the sector is erased.</td>
</tr>
<tr>
<td>FLASH_BUSY_ERASING</td>
<td>if there is an erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_ERROR_VERIFY</td>
<td>if the verify operation failed.</td>
</tr>
<tr>
<td>FLASH_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

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

### 7.8.4 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()

```c
flash_error_t flashWaitErase ( BaseFlash * devp )
```

Waits until the current erase operation is finished.

**Parameters**

- **in** devp: pointer to a `BaseFlash` object
Returns

An error code.

Return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH_NO_ERROR</td>
<td>if there is no erase operation in progress.</td>
</tr>
<tr>
<td>FLASH_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 devp</th>
<th>pointer to a BaseFlash object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in 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 devp</th>
<th>pointer to a BaseFlash object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in 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 GPT Driver

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 gptChangeIntervalX(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
)
```

**Value:**

```c
\{ 
    gpt_lld_change_interval(gptp, interval);
\}
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

**Precondition**

The GPT unit must be running in continuous mode.

**Postcondition**

The GPT unit interval is changed to the new value.

**Parameters**

<table>
<thead>
<tr>
<th><code>in</code></th>
<th><code>gptp</code></th>
<th>pointer to a <code>GPTDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>interval</code></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
)
```

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 GPT Driver

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpt</code></td>
<td>pointer to a <code>GPTDriver</code> object</td>
</tr>
<tr>
<td><code>interval</code></td>
<td>new cycle time in timer ticks</td>
</tr>
</tbody>
</table>

Function Class:

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

7.9.5 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gpt</code></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.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>GPT_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>GPT_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>GPT_CONTINUOUS</td>
<td>Active in continuous mode.</td>
</tr>
<tr>
<td>GPT_ONESHOT</td>
<td>Active in one shot mode.</td>
</tr>
</tbody>
</table>

7.9.7  Function Documentation

7.9.7.1  gptInit()

void gptInit (  
    void  )

GPT Driver initialization.
Note

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

Function Class:

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

Here is the call graph for this function:

```
gptInit → gpt_lld_init → gptObjectInit
```

### 7.9.7.2 gptObjectInit()

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

Initializes the standard part of a `GPTDriver` structure.

**Parameters**

- **out** `gptp` pointer to the `GPTDriver` object

**Function Class:**

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

### 7.9.7.3 gptStart()

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

Configures and activates the GPT peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>gptp</th>
<th>pointer to the <code>GPTDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the <code>GPTConfig</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:

7.9.7.4 gptStop()

```c
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:

```
gptStop
  \|-- gpt_lld_stop
    \|-- osalSysLock
    \|-- osalSysUnlock
```

7.9.7.5  gptChangeInterval()

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

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

Precondition

The GPT unit must be running in continuous mode.

Postcondition

The GPT unit interval is changed to the new value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in gptp</td>
<td>pointer to a GPTDriver object</td>
</tr>
<tr>
<td>in interval</td>
<td>new cycle time in timer ticks</td>
</tr>
</tbody>
</table>
7.9 GPT Driver

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.9.7.6 gptStartContinuous()

```c
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:

![Call Graph](image)
### 7.9.7.7 `gptStartContinuousI()`

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

Starts the timer in continuous mode.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>gptp</code></td>
<td>pointer to the <code>GPTDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>interval</code></td>
<td>period in ticks</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

![Call Graph]

### 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>gptp</code></td>
<td>pointer to the <code>GPTDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>interval</code></td>
<td>time interval in ticks</td>
</tr>
</tbody>
</table>
7.9 GPT Driver

Function Class:

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

Here is the call graph for this function:

![Call Graph]

7.9.7.9  gptStartOneShotI()

void gptStartOneShotI (  
  GPTDriver ∗ gptp,  
  gptcnt_t interval )

Starts the timer in one shot mode.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>gptp</th>
<th>pointer to the GPTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>interval</td>
<td>time interval in ticks</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

![Call Graph for gptStartOneShotI()]
### 7.9.7.10 gptStopTimer()

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

Stops the timer.

**Parameters**

- `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](image)

### 7.9.7.11 gptStopTimerI()

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

Stops the timer.

**Parameters**

- `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](image)

### 7.9.7.12 gptPolledDelay()

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

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

**Note**

The configured callback is not invoked when using this function.

**Parameters**

<table>
<thead>
<tr>
<th>In</th>
<th>gptp</th>
<th>pointer to the GPTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>interval</td>
<td>time interval in ticks</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)
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:

```
gpt_lld_init
   gptObjectInit
```

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.
7.9 GPT Driver

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()

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()

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()

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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>gptp</td>
<td>pointer to the GPTDriver object</td>
</tr>
<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

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).

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.
7.10 I2C Driver

**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.
- void i2cAcquireBus (I2CDriver *i2cp)
  Gains exclusive access to the I2C bus.
- void i2cReleaseBus (I2CDriver *i2cp)
  Releases exclusive access to the I2C bus.
- void i2c_lld_init (void)
  Low level I2C driver initialization.
- void i2c_lld_start (I2CDriver *i2cp)
  Configures and activates the I2C peripheral.
- void i2c_lld_stop (I2CDriver *i2cp)
  Deactivates the I2C peripheral.
- msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  Receives data via the I2C bus as master.
- msg_t i2c_lld_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, sysinterval_t timeout)
  Transmits data via the I2C bus as master.
Enumerations

- enum i2cstate_t {
  I2C_UNINIT = 0, I2C_STOP = 1, I2C_READY = 2, I2C_ACTIVE_TX = 3,
  I2C_ACTIVE_RX = 4, I2C_LOCKED = 5
}

Driver state machine possible states.

Variables

- I2CDriver I2CD1
  I2C1 driver identifier.

7.10.3 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:

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

Wakes up the waiting thread notifying no errors.
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

```c
#define __i2c_wakeup_error_isr(i2cp)
```

**Value:**
```c
do {
  osalSysLockFromISR();
  osalThreadResume1((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

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

**Value:**
```c
(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

```c
#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

```c
#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

```c
#define i2c_lld_get_errors(
    i2cp
) ((i2cp)->errors)
```

Get errors from I2C driver.

Parameters
- **i2cp**: pointer to the I2CDriver object

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

d enum i2cstate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>I2C_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>I2C_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>I2C_ACTIVE_TX</td>
<td>Transmitting.</td>
</tr>
<tr>
<td>I2C_ACTIVE_RX</td>
<td>Receiving.</td>
</tr>
<tr>
<td>I2C_LOCKED</td>
<td>Bus locked.</td>
</tr>
</tbody>
</table>
7.10.6 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]

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 |

---

ChibiOS/HAL
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:

```
+----------------+------------------+
|                |                  |
| i2cObjectInit  | osalMutexObjectInit |
|                |                  |
```

### 7.10.6.3 i2cStart()

```c
void 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:

```
+----------------+------------------+
|                |                  |
| i2c_lld_start  | osalSysLock      |
|                |                  |
+----------------+------------------+
|                |                  |
| i2cStart       | osalSysUnlock    |
```
### 7.10.6.4 i2cStop()

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

Deactivates the I2C peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.10.6.5 i2cGetErrors()

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

Returns the errors mask associated to the previous operation.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
</tbody>
</table>

**Returns**

The errors mask.
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>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>i2cp</code></td>
<td><code>I2CDriver ∗</code></td>
<td>pointer to the <code>I2CDriver</code> object</td>
</tr>
<tr>
<td><code>addr</code></td>
<td><code>i2caddr_t</code></td>
<td>slave device address (7 bits) without R/W bit</td>
</tr>
<tr>
<td><code>txbuf</code></td>
<td><code>const uint8_t ∗</code></td>
<td>pointer to transmit buffer</td>
</tr>
<tr>
<td><code>txbytes</code></td>
<td><code>size_t</code></td>
<td>number of bytes to be transmitted</td>
</tr>
<tr>
<td><code>rxbuf</code></td>
<td><code>uint8_t ∗</code></td>
<td>pointer to receive buffer</td>
</tr>
<tr>
<td><code>rxbytes</code></td>
<td><code>size_t</code></td>
<td>number of bytes to be received, set it to 0 if you want transmit only</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td><code>sysinterval_t</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return 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.                                 |

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 rxbytes,
    sysinterval_t timeout)
```

Receives data from the I2C bus.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
<tr>
<td>addr</td>
<td>slave device address (7 bits) without R/W bit</td>
</tr>
<tr>
<td>rxbuf</td>
<td>pointer to receive buffer</td>
</tr>
<tr>
<td>rxbytes</td>
<td>number of bytes to be received</td>
</tr>
<tr>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **MSG_OK** if the function succeeded.
- **MSG_RESET** if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.
- **MSG_TIMEOUT** if a timeout occurred before operation end.

**Function Class:**

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

7.10.6.8 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.

ChibiOS/HAL
Parameters

\[
\text{in } \text{i2cp} \quad \text{pointer to the I2CDriver object}
\]

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:

```
\text{i2cAcquireBus} \quad \text{osaMutexLock}
```

### 7.10.6.9 \text{i2cReleaseBus()}

```c
void \text{i2cReleaseBus} ( \\
\quad \text{I2CDriver \ast \text{i2cp}})
```

Releases exclusive access to the I2C bus.

Precondition

In order to use this function the option \text{I2C\_USE\_MUTUAL\_EXCLUSION} must be enabled.

Parameters

\[
\text{in } \text{i2cp} \quad \text{pointer to the I2CDriver object}
\]

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:

```
\text{i2cReleaseBus} \quad \text{osaMutexUnlock}
```
7.10.6.10  i2c_lld_init()

void i2c_lld_init ( void )

Low level I2C driver initialization.

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

Here is the call graph for this function:

```
    i2c_lld_init  i2cObjectInit  osalMutexObjectInit
```

7.10.6.11  i2c_lld_start()

void i2c_lld_start ( I2CDriver ∗ i2cp )

Configures and activates the I2C peripheral.

Parameters

| in  | i2cp | pointer to the I2CDriver object |

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

7.10.6.12  i2c_lld_stop()

void i2c_lld_stop ( I2CDriver ∗ i2cp )

Deactivates the I2C peripheral.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
</tbody>
</table>

Function Class:

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
<tr>
<td>in</td>
<td>addr</td>
<td>slave device address</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
<td>pointer to the receive buffer</td>
</tr>
<tr>
<td>in</td>
<td>rxbytes</td>
<td>number of bytes to be received</td>
</tr>
</tbody>
</table>
| 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if the function succeeded.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if one or more I2C errors occurred, the errors can be retrieved using i2cGetErrors().</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if a timeout occurred before operation end. After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.10 I2C Driver

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2cp</td>
<td>pointer to the I2CDriver object</td>
</tr>
<tr>
<td>in</td>
<td>addr</td>
<td>slave device address</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>pointer to the transmit buffer</td>
</tr>
<tr>
<td>in</td>
<td>txbytes</td>
<td>number of bytes to be transmitted</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
<td>pointer to the receive buffer</td>
</tr>
<tr>
<td>in</td>
<td>rxbytes</td>
<td>number of bytes to be received</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_INFINITE 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>MSG_OK</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 I2C1

**I2CDriver I2C1**

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 HAL_USE_I2S option must be enabled in halconf.h.

7.11.2 Driver State Machine

Macros

• #define i2s_lld_driver_fields
  Low level fields of the I2S driver structure.

• #define i2s_lld_config_fields
  Low level fields of the I2S configuration structure.

I2S modes

• #define I2S_MODE_SLAVE 0
• #define I2S_MODE_MASTER 1

Macro Functions

• #define i2sIsBufferComplete(i2sp) ((bool)((i2sp)->state == I2S_COMPLETE))
  Buffer state.

• #define i2sStartExchange(I2S)(i2sp)
  Starts a I2S data exchange.

• #define i2sStopExchange(I2S)(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.
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**

- **in i2sp** pointer to the I2SDriver object

**Returns**

- The buffer state.

**Return values**

- **false** if the driver filled/sent the first half of the buffer.
- **true** if the driver filled/sent the second half of the buffer.

**Function Class:**

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

#### 7.11.3.2 i2sStartExchangeI

```c
#define i2sStartExchangeI(i2sp)

Value:

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

Starts a I2S data exchange.
7.11 I2S Driver

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

\[
\text{in } \text{\texttt{i2sp} pointer to the \texttt{I2CDriver} object}
\]

Function Class:

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

7.11.3.5 \texttt{i2s_isr_full_code}

\[
\texttt{\#define \_i2s\_isr\_full\_code(} \\
\texttt{\texttt{i2sp})}
\]

Value:

\[
\text{if ((i2sp)->config->end\_cb)} \{ \\
\text{(i2sp)->state = \texttt{I2S\_COMPLETE};} \\
\text{(i2sp)->config->end\_cb(i2sp);} \\
\text{if ((i2sp)->state == \texttt{I2S\_COMPLETE}) \{} \\
\text{(i2sp)->state = \texttt{I2S\_ACTIVE};} \\
\text{\}} \\
\}
\]

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

\[
\text{in } \text{\texttt{i2sp} pointer to the \texttt{I2CDriver} object}
\]

Function Class:

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

7.11.3.6 \texttt{PLATFORM\_I2S\_USE\_I2S1}

\[
\texttt{\#define PLATFORM\_I2S\_USE\_I2S1 FALSE}
\]

I2SD1 driver enable switch.

If set to \texttt{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

```c
enum i2sstate_t
```

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2S_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>I2S_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>I2S_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>I2S_ACTIVE</td>
<td>Active.</td>
</tr>
<tr>
<td>I2S_COMPLETE</td>
<td>Transmission complete.</td>
</tr>
</tbody>
</table>

7.11.6 Function Documentation

7.11.6.1 i2sInit()

```c
void i2sInit ( 
    void )
```

I2S Driver initialization.

Note

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

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

Here is the call graph for this function:

```
    i2sInit  i2s_lld_init  i2sObjectInit
```

### 7.11.6.2 i2sObjectInit()

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

Initializes the standard part of a I2SDriver structure.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>i2sp</td>
<td>pointer to the I2SDriver object</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.11.6.3 i2sStart()

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

Configures and activates the I2S peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i2sp</td>
<td>pointer to the I2SDriver object</td>
</tr>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the I2SConfig object</td>
</tr>
</tbody>
</table>
7.11.6.4  i2sStop()

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:

![Call Graph](image)

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:

![Call Graph](image)
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**

|   | i2sp | pointer to the I2SDriver object |

**Function Class:**

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

Here is the call graph for this function:

![Call Graph for i2sStopExchange](image)

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:

![Call Graph for i2s_lld_init](image)
7.11.6.8 i2s_lld_start()

```c
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()

```c
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()

```c
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).

7.12.3 ICU Operations.

This driver abstracts a generic Input Capture Unit composed of:

- A clock prescaler.
- A main up counter.
• Two capture registers triggered by the rising and falling edges on the sampled input.

The ICU unit can be programmed to synchronize on the rising or falling edge of the sample input:

• **ICU_INPUT_ACTIVE_HIGH**, a rising edge is the start signal.
• **ICU_INPUT_ACTIVE_LOW**, a falling edge is the start signal.

callbacks are optionally invoked when:

• On the PWM de-activation edge.
• On the PWM activation edge, measurements for the previous cycle are available from this callback and can be retrieved using `icuGetPeriodX()` and `icuGetWidthX()`.

**Macros**

• `#define icu_lld_get_width(icup) 0`
  
  Returns the width of the latest pulse.
• `#define icu_lld_get_period(icup) 0`
  
  Returns the width of the latest cycle.
• `#define icu_lld_are_notifications_enabled(icup) false`
  
  Check on notifications status.

**Macro Functions**

• `#define icuStartCapture(icup)`
  
  Starts the input capture.
• `#define icuStopCapture(icup)`
  
  Stops the input capture.
• `#define icuEnableNotifications(icup) icu_lld_enable_notifications(icup)`
  
  Enables notifications.
• `#define icuDisableNotifications(icup) icu_lld_disable_notifications(icup)`
  
  Disables notifications.
• `#define icuAreNotificationsEnabled(icup) icu_lld_are_notifications_enabled(icup)`
  
  Check on notifications status.
• `#define 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.
Module Documentation

- **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

#define icuStartCaptureI( 
  icup )

Value:

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

Starts the input capture.

Parameters

- **icup** pointer to the ICUDriver object
Function Class:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.4.2 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 ICUObject 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 ICUObject 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 \texttt{icuStart()} and the capture started using \texttt{icuStartCapture()}.

Note
   If the notification is already disabled then the call has no effect.

Parameters
   \begin{verbatim}
   in  icup  pointer to the ICUDriver object
   \end{verbatim}

Function Class:
   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
   \begin{verbatim}
   in  icup  pointer to the ICUDriver object
   \end{verbatim}

Returns
   The notifications status.
Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if notifications are not enabled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if notifications are enabled.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.12.4.6 icuGetWidthX

```c
#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

```c
#define icuGetPeriodX(icup) icu_lld_get_period(icup)
```

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

**Note**

This function is meant to be invoked from the width capture callback.
Parameters

| in | icup | pointer to the **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.
### 7.12 ICU Driver

**Parameters**

| in | icup | pointer to the **ICUDriver** object |

**Function Class:**

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

#### 7.12.4.10 _icu_isr_invoke_overflow_cb

```c
#define _icu_isr_invoke_overflow_cb(icup)

Value:

```c
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 **ICUDriver** object |

**Function Class:**

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

#### 7.12.4.11 PLATFORM_ICU_USE_ICU1

```c
#define PLATFORM_ICU_USE_ICU1 FALSE
```

**ICUD1 driver enable switch.**

If set to **TRUE** the support for ICUD1 is included.

**Note**

The default is **FALSE**.
7.12.4.12 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 ICU_DRIVER 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 ICU_DRIVER 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.
7.12 ICU Driver

Parameters

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

Returns

The notifications status.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if notifications are not enabled.</td>
</tr>
<tr>
<td>true</td>
<td>if notifications are enabled.</td>
</tr>
</tbody>
</table>

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

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

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 ICU Driver

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](image)

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.
Function Class:

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

Here is the call graph for this function:

![Call Graph]

7.12.7.4 icuStop()

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

Deactivates the ICU peripheral.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in icup</td>
<td>pointer to the ICU Driver object</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:

7.12.7.5 icuStartCapture()

```c
void icuStartCapture (ICUDriver *icu);
```

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()

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

Waits for a completed capture.

**Note**

The operation could be performed in polled mode depending on.
In order to use this function notifications must be disabled.

**Precondition**

The driver must be in ICU_WAITING or ICU_ACTIVE states.

**Postcondition**

After the capture is available the driver is in ICU_ACTIVE state. If a capture fails then the driver is in ICU-_WAITING state.

**Parameters**

- **in icup** pointer to the ICUDriver object

**Returns**

The capture status.

**Return values**

<table>
<thead>
<tr>
<th>false</th>
<th>if the capture is successful.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if a timer overflow 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:

```
  icuWaitCapture
     icu_lld_wait_capture
       osalSysLock
     osalSysUnlock
```

### 7.12.7.7 icuStopCapture()

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

Stops the input capture.

**Parameters**

- **in icup** pointer to the ICUDriver object

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

Here is the call graph for this function:

```
  icuStopCapture
     osalSysLock
     osalSysUnlock
```
7.12 ICU Driver

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.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in icup</code></td>
<td>pointer to the <code>ICUDriver</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:

```
7.12.7.10 icu_lld_init()

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:

```
7.12.7.11 icu_lld_start()

void icu_lld_start (  
    ICUDriver * icup )
Configures and activates the ICU peripheral.
Parameters

| in | icup | pointer to the ICUDriver object |

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

| in | icup | pointer to the ICUDriver object |

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

| in | icup | pointer to the ICUDriver object |

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*icup</td>
<td>pointer to ICUDriver object</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*icup</td>
<td>pointer to ICUDriver object</td>
<td></td>
</tr>
</tbody>
</table>

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

| `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

| `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
- Output Formatter Utility
7.15 Abstract Streams Interface

7.15.1 Detailed Description

This module defines 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 the 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.

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.

Data Structures

- struct BaseSequentialStreamVMT
  BaseSequentialStream virtual methods table.
- struct BaseSequentialStream
  Base 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>in</th>
<th>ip</th>
<th>pointer to a BaseSequentialStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</tbody>
</table>

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.
7.15 Abstract Streams Interface

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

#define streamRead(
ip, 
bp, 
n) ((ip)->vmt->read(ip, bp, n))

Sequential Stream read.
The function reads data from a stream into a buffer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseSequentialStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</tbody>
</table>

Returns
The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

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

7.15.2.5 streamPut

#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

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseChannel or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>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>STM_OK</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>STM_RESET</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.

### 7.15.2.6 streamGet

```c
#define streamGet(ip) ((ip)->vmt->get(ip))
```

Sequential Stream blocking byte read.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

Parameters

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

Returns

A byte value from the queue.

Return values

| STM_RESET | if an end-of-file condition has been met. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.16 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 \ _base_asynchronous_channel_data`  
  `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 chnAddFlagsI(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 Abstract I/O Channel Interface

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

msg_t (*putt)(void *instance, uint8_t b, sysinterval_t time);
/* Channel put method with timeout specification.*/
msg_t (*putt)(void *instance, sysinterval_t time);
/* Channel get method with timeout specification.*/
size_t (*writet)(void *instance, const uint8_t *bp, size_t n, sysinterval_t time);
/* Channel write method with timeout specification.*/
size_t (*readt)(void *instance, uint8_t *bp, size_t n, sysinterval_t time);
/* Channel read method with timeout specification.*/
msg_t (*ctl)(void *instance, unsigned int operation, void *arg);
/* Channel control method.*/

_base_channel_methods

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

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseChannel or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written to the channel</td>
</tr>
<tr>
<td>in</td>
<td>time</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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

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

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseChannel or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

A byte value from the queue.
7.16 Abstract I/O Channel Interface

Return values

<table>
<thead>
<tr>
<th>STM_TIMEOUT</th>
<th>if the specified time expired.</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM_RESET</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>in</th>
<th>ip</th>
<th>pointer to a BaseChannel or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
</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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>pointer to a BaseChannel or derived class</td>
</tr>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred</td>
</tr>
<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

```c
#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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>pointer to a BaseChannel or derived class</td>
</tr>
<tr>
<td>in</td>
<td>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 Abstract I/O Channel Interface

7.16.2.11 chnReadTimeout

#define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->read(ip, bp, n, time))

Channel blocking read with timeout.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.

Parameters

| in | ip     | pointer to a BaseChannel or derived class |
| in | bp     | pointer to the data buffer                 |
| in | n      | the maximum amount of data to be transferred |
| in | time   | the number of ticks before the operation timeouts; the following special values are allowed: |
|    |        | • TIME_IMMEDIATE immediate timeout.        |
|    |        | • TIME_INFINITE no timeout.                |

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

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

Control operation on a channel.

Parameters

| in  | ip     | pointer to a BaseChannel or derived class |
| in  | operation | control operation code                  |
| in,out | arg     | operation argument                      |
Returns

The control operation status.

Return values

| MSG_OK      | in case of success. |
| MSG_TIMEOUT | in case of operation timeout. |
| MSG_RESET   | in case of operation reset. |

Function Class:

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 Abstract I/O Channel Interface

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

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BaseAsynchronousChannel or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>flags</td>
<td>condition flags to be added to the listener 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.
7.17 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 fileStreamGetError(ip) ((ip)->vmt->geterror(ip))`  
  Returns an implementation dependent error code.
- `#define fileStreamGetSize(ip, offset) ((ip)->vmt->getsize(ip, offset))`  
  Returns the current file size.
- `#define fileStreamGetPosition(ip, offset) ((ip)->vmt->getposition(ip, offset))`  
  Returns the current file pointer position.
- `#define fileStreamSetPosition(ip, offset) ((ip)->vmt->setposition(ip, offset))`  
  Moves the file current pointer to an absolute position.
**Typedefs**

- typedef uint32_t fileoffset_t
  
  *File offset type.*

**Data Structures**

- struct FileStreamVMT
  
  *FileStream virtual methods table.*

- struct FileStream
  
  *Base file stream class.*

### 7.17.2 Macro Definition Documentation

#### 7.17.2.1 FILE_OK

```c
#define FILE_OK STM_OK
```

*No error return code.*

#### 7.17.2.2 FILE_ERROR

```c
#define FILE_ERROR STM_TIMEOUT
```

*Error code from the file stream methods.*

#### 7.17.2.3 FILE_EOF

```c
#define FILE_EOF STM_RESET
```

*End-of-file condition for file get/put methods.*
7.17 Abstract Files Interface

7.17.2.4  _file_stream_methods

#define _file_stream_methods

Value:

__base_sequential_stream_methods

/* File close method. */
msg_t (*close)(void *instance);
/* Get last error code method. */
msg_t (*geterror)(void *instance);
/* File get size method. */
msg_t (*getsize)(void *instance, fileoffset_t *offset);
/* File get current position method. */
msg_t (*getposition)(void *instance, fileoffset_t *offset);
/* File set current position method. */
msg_t (*setposition)(void *instance, fileoffset_t offset);

FileStream specific methods.

7.17.2.5  _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)

File stream 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>ip</td>
<td>pointer to a FileStream 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. 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.

### 7.17.2.7 fileStreamRead

```c
#define fileStreamRead(
    ip,
    bp,
    n ) streamRead(ip, bp, n)
```

File stream read.

The function reads data from a file stream into a buffer.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a <code>FileStream</code> 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

```c
#define fileStreamPut(
    ip,
    b ) streamPut(ip, b)
```

File stream blocking byte write.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written to the channel</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_OK</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>FILE_ERROR</td>
<td>operation failed.</td>
</tr>
<tr>
<td>FILE_EOF</td>
<td>if an end-of-file condition has been met.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.17.2.9 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>FILE_ERROR</td>
<td>operation failed.</td>
</tr>
<tr>
<td>FILE_EOF</td>
<td>if an end-of-file condition has been met.</td>
</tr>
</tbody>
</table>

Function Class:

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

#define fileStreamClose(ip) ((ip)->vmt->close(ip))

File Stream close.

The function closes a file stream.

Parameters

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

Returns

The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_OK</td>
<td>no error.</td>
</tr>
<tr>
<td>FILE_ERROR</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.17.2.11 fileStreamGetError

#define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))

Returns an implementation dependent error code.

Precondition

The previously called function must have returned FILE_ERROR.

Parameters

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

Returns

Implementation dependent error code.
7.17 Abstract Files Interface

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

#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

#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>

ChibiOS/HAL
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

#define FileStreamSetPosition(
    ip,
    offset ) ((ip)->vmt->setPosition(ip, offset))

Moves the file current pointer to an absolute position.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a FileStream or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>offset</td>
<td>new absolute position</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>FILE_OK</th>
<th>no error.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_ERROR</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.17.3 Typedef Documentation

7.17.3.1 fileoffset_t

typedef uint32_t fileoffset_t

File offset type.
7.18 Abstract Persistent Storage Interface

7.18.1 Detailed Description

This module defines 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

#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

| PS_NO_ERROR | if there is no erase operation in progress. |
| PS_ERROR_READ | if the read 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.
7.18.2.7 psWrite

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

Write operation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to a BasePersistentStorage or derived class</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>offset</td>
<td>persistent storage 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

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_WRITE</td>
<td>if the write operation failed.</td>
</tr>
<tr>
<td>PS_ERROR_HW_FAILURE</td>
<td>if access to the memory failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.18.3 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 Abstract Persistent Storage Interface

7.18.4.1 ps_error_t

```c
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_sequential_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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>msp</code></td>
<td>pointer to the <code>MemoryStream</code> object to be initialized</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>buffer</code></td>
<td>pointer to the memory buffer for the memory stream</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>size</code></td>
<td>total size of the memory stream buffer</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>eos</code></td>
<td>initial End Of Stream offset. Normally you need to put this to zero for RAM buffers or equal to <code>size</code> for ROM streams.</td>
</tr>
</tbody>
</table>
7.20 Null Streams Class

7.20.1 Detailed Description

A null streams.

Macros

- #define _null_stream_data _base_sequential_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_sequential_stream_data

NullStream specific data.

7.20.3 Function Documentation

7.20.3.1 nullObjectInit()

void nullObjectInit ( 
  
  NullStream * nsp )

Null stream object initialization.
### Parameters

| out | nsp | pointer to the NullStream object to be initialized |
7.21 Output Formatter Utility

7.21.1 Detailed Description

Mini printf-like functionality.

Macros

- \#define CHPRINTF_USE_FLOAT FALSE
  
  Float type support.

Functions

- int chvprintf (BaseSequentialStream *chp, const char *fmt, va_list ap)
  
  System formatted output function.
- int chprintf (BaseSequentialStream *chp, const char *fmt, ...)
  
  System formatted output function.
- int chsnprintf (char *str, size_t size, const char *fmt, ...)
  
  System formatted output function.
- int chvsnprintf (char *str, size_t size, const char *fmt, va_list ap)
  
  System formatted output function.

7.21.2 Macro Definition Documentation

7.21.2.1 CHPRINTF_USE_FLOAT

#define CHPRINTF_USE_FLOAT FALSE

Float type support.

7.21.3 Function Documentation
7.21 Output Formatter Utility

7.21.3.1 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>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chp</td>
<td>pointer to a BaseSequentialStream implementing object</td>
</tr>
<tr>
<td>fmt</td>
<td>formatting string</td>
</tr>
<tr>
<td>ap</td>
<td>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.

7.21.3.2 chprintf()
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

<table>
<thead>
<tr>
<th>in</th>
<th>chp</th>
<th>pointer to a BaseSequentialStream implementing object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fmt</td>
<td>formatting string</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.

Here is the call graph for this function:
7.21.3.3 chsnprintf()

int chsnprintf(
    char * str,
    size_t size,
    const char * fmt,
    ...
)

System formatted output function.

This function implements a minimal snprintf()-like functionality. The general parameters format is: %[-][width][.precision][l][ll][L][l]p. The following parameter types (p) are supported:

- x hexadecimal integer.
- X hexadecimal long.
- o octal integer.
- O octal long.
- d decimal signed integer.
- D decimal signed long.
- u decimal unsigned integer.
- U decimal unsigned long.
- c character.
- s string.

Postcondition

str is NUL-terminated, unless size is 0.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>str</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
</tr>
<tr>
<td>in</td>
<td>fmt</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:

```
chsnprintf chvsnprintf
chvprintf
msObjectInit
```

### 7.21.3.4 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:

```
chvsprintf
  chvprintf
    msObjectInit
```

ChibiOS/HAL
### 7.22 Abstract I/O Block Device

#### 7.22.1 Detailed Description

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).

![State Machine Diagram]

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.
### 7.22 Abstract I/O Block Device

- `#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.

### 7.22.3 Macro Definition Documentation

#### 7.22.3.1 _base_block_device_methods

```c
#define _base_block_device_methods

_value:  
  _base_object_methods  
  /* Removable media detection.*/  
  bool (*is_inserted)(void *instance);  
  /* Removable write protection detection.*/  
  bool (*is_protected)(void *instance);  
  /* Connection to the block device.*/  
  bool (*connect)(void *instance);  
  /* Disconnection from the block device.*/  
  bool (*disconnect)(void *instance);  
  /* Reads one or more blocks.*/  
  bool (*read)(void *instance, uint32_t startblk,  
               uint8_t *buffer, uint32_t n);  
  /* Writes one or more blocks.*/  
  bool (*write)(void *instance, uint32_t startblk,  
                const uint8_t *buffer, uint32_t n);  
  /* Write operations synchronization.*/  
  bool (*sync)(void *instance);  
  /* Obtains info about the media.*/  
  bool (*get_info)(void *instance, BlockDeviceInfo *bdip);  

BaseBlockDevice specific methods.
```
7.22.3.2 _base_block_device_data

#define _base_block_device_data

Value:

_value.data

/* Driver state. */
blkstate_t state;

BaseBlockDevice specific data.

7.22.3.3 blkGetDriverState

#define blkGetDriverState(ip) ((ip)->state)

Returns the driver state.

Note
Can be called in ISR context.

Parameters

in ip pointer to a BaseBlockDevice or derived class

Returns
The driver state.

Function Class:
Special function, this function has special requirements see the notes.

7.22.3.4 blkIsTransferring

#define blkIsTransferring(ip)

Value:

(((ip)->state) == BLK_CONNECTING) ||
(((ip)->state) == BLK_DISCONNECTING) ||
(((ip)->state) == BLK_READING) ||
(((ip)->state) == BLK_WRITING)

Determines if the device is transferring data.

Note
Can be called in ISR context.
Abstract I/O Block Device

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.

7.22.3.5 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.
7.22.3.6  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

<table>
<thead>
<tr>
<th>false</th>
<th>writable media.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>non writable media.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.22.3.7  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

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>
7.22 Abstract I/O Block Device

Function Class:

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

### 7.22.3.8 blkDisconnect

```c
#define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))
```

Terminates operations on the block device.

This operation safely terminates operations on the block device.

**Parameters**

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

**Returns**

The operation status.

**Return values**

- **HAL_SUCCESS**: operation succeeded.
- **HAL_FAILED**: operation failed.

Function Class:

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

### 7.22.3.9 blkRead

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

Reads one or more blocks.

**Parameters**

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

ChibiOS/HAL
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.22.3.10 blkWrite

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

Writes one or more blocks.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>pointer to a BaseBlockDevice or derived class</td>
</tr>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to write</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the write buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to write</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

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

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.22 Abstract I/O Block Device

### 7.22.3.11 blkSync

```c
#define blkSync(ip) {{ip)->vmt->sync(ip)}
```

Ensures write synchronization.

**Parameters**

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

**Returns**

The operation status.

**Return values**

- `HAL_SUCCESS` operation succeeded.
- `HAL_FAILED` operation failed.

**Function Class:**

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

### 7.22.3.12 blkGetInfo

```c
#define blkGetInfo(ip, bdip) {{ip)->vmt->get_info(ip, bdip)}
```

Returns a media information structure.

**Parameters**

| in | ip | pointer to a `BaseBlockDevice` or derived class |
| out | bdip | pointer to a `BlockDeviceInfo` structure |

**Returns**

The operation status.

**Return values**

- `HAL_SUCCESS` operation succeeded.
- `HAL_FAILED` operation failed.
Function Class:

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

7.22.4 Enumeration Type Documentation

7.22.4.1 blkstate_t

definitions 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>
7.23 MAC Driver

Generic MAC Driver.

7.23.1 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, 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.

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)
7.23 MAC Driver

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.

7.23.2 Macro Definition Documentation

7.23.2.1 MAC_USE_ZERO_COPY

#define MAC_USE_ZERO_COPY FALSE

Enables an event sources for incoming packets.

7.23.2.2 MAC_USE_EVENTS

#define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.23.2.3 macGetReceiveEventSource

#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.

7.23.2.4 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.

7.23.2.5 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdp</td>
<td>pointer to a MACReceiveDescriptor structure</td>
</tr>
<tr>
<td>buf</td>
<td>pointer to the buffer that will receive the read data</td>
</tr>
<tr>
<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.

7.23.2.6 macGetNextTransmitBuffer

```c
#define macGetNextTransmitBuffer(tdp, size, sizep) mac_lld_get_next_transmit_buffer(tdp, size, sizep)
```

Returns a pointer to the next transmit buffer in the descriptor chain.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tdp</td>
<td>pointer to a MACTransmitDescriptor structure</td>
</tr>
<tr>
<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>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.
7.23.2.7 *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**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>rdp</td>
<td>pointer to a MACReceiveDescriptor structure</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>sizep</td>
<td>pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.</td>
</tr>
</tbody>
</table>

**Returns**

Pointer to the returned buffer.

**Return values**

| NULL | if the buffer chain has been entirely scanned. |

**Function Class:**

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

7.23.2.8 *MAC_SUPPORTS_ZERO_COPY*

```c
#define MAC_SUPPORTS_ZERO_COPY TRUE
```

This implementation supports the zero-copy mode API.

7.23.2.9 *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*. 

---

ChibiOS/HAL
7.23.3 Typedef Documentation

7.23.3.1 MACDriver

typedef struct MACDriver MACDriver

Type of a structure representing a MAC driver.

7.23.4 Enumeration Type Documentation

7.23.4.1 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>

7.23.5 Function Documentation

7.23.5.1 macInit()

void macInit ()

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:

macInit \rightarrow \text{mac\_ld\_init} \rightarrow \text{macObjectInit}

7.23.5.2 macObjectInit()

void macObjectInit (MACDriver *macp)

Initialize the standard part of a MACDriver structure.

Parameters

\begin{tabular}{|c|c|}
\hline
\textbf{out} & \textit{macp} pointer to the MACDriver object \\
\hline
\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:

macObjectInit \rightarrow \text{osalEventObjectInit} \rightarrow \text{osalThreadQueueObjectInit}
7.23.5.3  macStart()

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

Configures and activates the MAC peripheral.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>macp</th>
<th>pointer to the MACDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>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:

```
    macStart
       /    
      /     
    osalSysLock
      |     |
    osalSysUnlock
    /      
    |      
mac_lld_start
```

7.23.5.4  macStop()

void macStop (  
    MACDriver * macp )

Deactivates the MAC peripheral.

Parameters

| in  | macp | pointer to the MACDriver object |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.23.5.5 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>macp</code></td>
<td>pointer to the MACDriver object</td>
</tr>
<tr>
<td><code>tdp</code></td>
<td>pointer to a MACTransmitDescriptor structure</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>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• TIME_INFINITE 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>the descriptor was obtained.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>the operation timed out, descriptor not initialized.</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.23.5.6 macReleaseTransmitDescriptor()

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:
7.23.5.7  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>Mode</th>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>macp</code></td>
<td>pointer to the MACDriver object</td>
</tr>
<tr>
<td>out</td>
<td><code>rdp</code></td>
<td>pointer to a MACReceiveDescriptor 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</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_OK</code></td>
<td>the descriptor was obtained.</td>
</tr>
<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:
7.23 MAC Driver

7.23.5.8 macReleaseReceiveDescriptor()

```c
void macReleaseReceiveDescriptor ( 
    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:

Normal API, 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)

7.23.5.9 macPollLinkStatus()

```c
bool macPollLinkStatus ( 
    MACDriver * macp )
```

Updates and returns the link status.

Parameters

| in  | macp | pointer to the MACDriver object |

Returns

The link status.

Return values

| true | if the link is active. |
| false | if the link is down. |
Function Class:

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

Here is the call graph for this function:

```
macPollLinkStatus  \--\ mac_lld_poll_link_status
```

### 7.23.5.10 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
```

### 7.23.5.11 mac_lld_start()

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

Configures and activates the MAC peripheral.
7.23 MAC Driver

Parameters

| In | macp | Pointer to the MACDriver object |

Function Class:

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

7.23.5.12 mac_lld_stop()

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

Deactivates the MAC peripheral.

Parameters

| In | macp | Pointer to the MACDriver object |

Function Class:

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

7.23.5.13 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

| MSG_OK  | the descriptor has been obtained. |
| MSG_TIMEOUT | descriptor not available. |

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

### 7.23.5.14 `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.

### 7.23.5.15 `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

| MSG_OK  | the descriptor has been obtained. |
| MSG_TIMEOUT | descriptor not available. |
Function Class:
Not an API, this function is for internal use only.

### 7.23.5.16 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.

### 7.23.5.17 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
- `true` if the link is active.
- `false` if the link is down.

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

```c
size_t mac_lld_write_transmit_descriptor ( 
    MACTransmitDescriptor * tdp, 
    uint8_t * buf, 
    size_t size )
```

Writes to a transmit descriptor's stream.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tdp</code></td>
<td>pointer to a <code>MACTransmitDescriptor</code> structure</td>
</tr>
<tr>
<td><code>buf</code></td>
<td>pointer to the buffer containing the data to be written</td>
</tr>
<tr>
<td><code>size</code></td>
<td>number of bytes to be written</td>
</tr>
</tbody>
</table>

Returns

The number of bytes written into the descriptor's stream, this value can be less than the amount specified in the parameter `size` if the maximum frame size is reached.

Function Class:

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

---

7.23.5.19  mac_lld_read_receive_descriptor()

```c
size_t mac_lld_read_receive_descriptor ( 
    MACReceiveDescriptor * rdp, 
    uint8_t * buf, 
    size_t size )
```

Reads from a receive descriptor's stream.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rdp</code></td>
<td>pointer to a <code>MACReceiveDescriptor</code> structure</td>
</tr>
<tr>
<td><code>buf</code></td>
<td>pointer to the buffer that will receive the read data</td>
</tr>
<tr>
<td><code>size</code></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:

Not an API, this function is for internal use only.
7.23.5.20  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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tdp</td>
<td>pointer to a MACTransmitDescriptor structure</td>
</tr>
<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 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.

7.23.5.21  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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>rdp</td>
<td>pointer to a MACReceiveDescriptor structure</td>
</tr>
<tr>
<td><strong>out</strong></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.

7.23.6 Variable Documentation

7.23.6.1 ETHD1

MACDriver ETHD1

MAC1 driver identifier.
7.24 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 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.

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 *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.

7.24.2 Macro Definition Documentation
7.24.2.1 ALIGNED_REC_SIZE

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

Data record size aligned.

7.24.2.2 ALIGNED_DHDR_SIZE

#define ALIGNED_DHDR_SIZE ALIGNED_REC_SIZE(0)

Data record header size aligned.

7.24.2.3 ALIGNED_SIZEOF

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

Aligned size of a type.

7.24.2.4 PAIR

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

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

7.24.2.5 RET_ON_ERROR

#define RET_ON_ERROR(err)

Value:

do {
  mfs_error_t e = (err);
  if (e != MFS_NO_ERROR) {
    return e;
  } else { 
    while (false)
  }
} while (false)

Error check helper.
7.24.2.6  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.

7.24.2.7  MFS_CFG_MAX_REPAIR_ATTEMPTS

#define MFS_CFG_MAX_REPAIR_ATTEMPTS 3

Maximum number of repair attempts on partition mount.

7.24.2.8  MFS_CFG_WRITE_VERIFY

#define MFS_CFG_WRITE_VERIFY TRUE

Verify written data.

7.24.2.9  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.

7.24.2.10  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.
7.24.2.11  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.

7.24.2.12  MFS_CFG_TRANSACTION_MAX

#define MFS_CFG_TRANSACTION_MAX 16

Maximum number of objects writable in a single transaction.

7.24.3  Typedef Documentation

7.24.3.1  mfs_id_t

typedef uint32_t mfs_id_t

Type of a record identifier.

7.24.4  Enumeration Type Documentation

7.24.4.1  mfs_bank_t

enum mfs_bank_t

Type of a flash bank.

7.24.4.2  mfs_state_t

enum mfs_state_t

Type of driver state machine states.
7.24.3 mfs_error_t

```c
type enum mfs_error_t
```

Type of an MFS error code.

**Note**

Errors are negative integers, informative warnings are positive integers.

7.24.4 mfs_bank_state_t

```c
type enum mfs_bank_state_t
```

Type of a bank state assessment.

7.24.5 Function Documentation

7.24.5.1 mfs_flash_read()

```c
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.
7.24.5.2 mfs_flash_write()

```c
static mfs_error_t mfs_flash_write ( 
    MFSDriver * mfsp, 
    flash_offset_t offset, 
    size_t n, 
    const uint8_t * wp ) [static]
```

Flash write.

Note

If the option MFS_CFG_WRITE_VERIFY is enabled then the flash is also read back for verification.

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.

7.24.5.3 mfs_flash_copy()

```c
static mfs_error_t mfs_flash_copy ( 
    MFSDriver * mfsp, 
    flash_offset_t doffset, 
    flash_offset_t soffset, 
    uint32_t n ) [static]
```

Flash copy.

Note

If the option MFS_CFG_WRITE_VERIFY is enabled then the flash is also read back for verification.

Parameters

<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:

7.24.5.4 mfs_bank_erase()

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>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 erased</td>
</tr>
</tbody>
</table>

Returns
The operation status.

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

static mfs_error_t mfs_bank_verify_erase (  
    MFSDriver * mfsp,  
    mfs_bank_t bank ) [static]

Erases and verifies all sectors belonging to a bank.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bank</td>
<td>bank to be verified</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

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

7.24.5.6 mfs_bank_write_header()

static mfs_error_t mfs_bank_write_header (  
    MFSDriver * mfsp,  
    mfs_bank_t bank,  
    uint32_t cnt ) [static]

Writes the validation header in a bank.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bank</td>
<td>bank to be validated</td>
</tr>
<tr>
<td>in</td>
<td>cnt</td>
<td>value for the flash usage counter</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Function Class:

Not an API, this function is for internal use only.
7.24.5.7 mfs_bank_check_header()

```c
static mfs_bank_state_t mfs_bank_check_header (MFSDriver * mfsp) [static]
```

Checks integrity of the header in the shared buffer.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mfsp</td>
<td>pointer to the <strong>MFSDriver</strong> object</td>
</tr>
</tbody>
</table>

**Returns**

The header state.

**Function Class:**

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

7.24.5.8 mfs_bank_scan_records()

```c
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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mfsp</td>
<td>pointer to the <strong>MFSDriver</strong> object</td>
</tr>
<tr>
<td>in</td>
<td>bank</td>
<td>the bank identifier</td>
</tr>
<tr>
<td>out</td>
<td>wflagp</td>
<td>warning flag on anomalies</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Function Class:**

Not an API, this function is for internal use only.
7.24.5.9 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](attachment://call_graph.png)

7.24.5.10 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

<table>
<thead>
<tr>
<th>Type</th>
<th>mfsp</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td></td>
</tr>
</tbody>
</table>

pointer to the MFSDriver object

Returns

The operation status.

Function Class:

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

7.24.5.11  mfs_try_mount()

```
static mfs_error_t mfs_try_mount ( 
        MFSDriver * mfsp ) [static]
```

Performs a flash partition mount attempt.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>mfsp</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
</tr>
</tbody>
</table>

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.

7.24.5.12  mfs_mount()

```
mfs_error_t mfs_mount ( 
        MFSDriver * mfsp )
```

Configures and activates a MFS driver.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>mfsp</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
</tr>
</tbody>
</table>

pointer to the MFSDriver object
Returns

The operation status.

Return values

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

7.24.5.13 mfsObjectInit()

```c
void mfsObjectInit ( 
    MFSDriver ∗mfsp )
```

Initializes an instance.

Parameters

- **out mfsp**: pointer to the MFSDriver object

Function Class:

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

7.24.5.14 mfsStart()

```c
mfs_error_t mfsStart ( 
    MFSDriver ∗mfsp, 
    const MFSConfig ∗config )
```

Configures and activates a MFS driver.

Parameters

- **in mfsp**: pointer to the MFSDriver object
- **in config**: pointer to the configuration
Returns

The operation status.

Return values

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

7.24.5.15 mfsStop()

void mfsStop (MFSDriver * mfsp)

Deactivates a MFS driver.

Parameters

| in   | mfsp  | 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.

7.24.5.16 mfsErase()

mfs_error_t mfsErase (MFSDriver * mfsp)

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

Parameters

| in   | mfsp  | pointer to the MFSDriver object |

ChibiOS/HAL
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return Code</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_NO_ERROR</td>
<td>if the operation has been successfully completed.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter</td>
</tr>
<tr>
<td></td>
<td>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.

7.24.5.17 mfsReadRecord()

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>in</td>
<td>mfsp pointer to the MFSDriver object</td>
</tr>
<tr>
<td>in</td>
<td>id record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS</td>
</tr>
<tr>
<td>in, out</td>
<td>np on input is the maximum buffer size, on return it is the size of the data copied into the buffer</td>
</tr>
<tr>
<td>out</td>
<td>buffer pointer to a buffer for record data</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>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_INV_SIZE</td>
<td>if the passed buffer is not large enough to contain the record data.</td>
</tr>
<tr>
<td>MFS_ERR_NOT_FOUND</td>
<td>if the specified id does not exists.</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.

### 7.24.5.18 mfsWriteRecord()

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

Creates or updates a data record.

**Parameters**

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

**Returns**

The operation status.

**Return values**

| MFS_NO_ERROR | if the operation has been successfully completed. |
| MFS_WARN_GC   | if the operation triggered a garbage collection. |
| MFS_ERR_INV_STATE | if the driver is in not in MFS_READY state. |
| MFS_ERR_OUT_OF_MEM | if there is not enough flash space for the operation. |
| MFS_ERR_TRANSACTION_NUM | if the transaction operations buffer space has been exceeded. |
| MFS_ERR_TRANSACTION_SIZE | if the transaction allocated space has been exceeded. |
| MFS_ERR_FLASH_FAILURE | if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state. |
| MFS_ERR_INTERNAL | if an internal logic failure is detected. |

Function Class:

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

### 7.24.5.19 mfsEraseRecord()

```c
mfs_error_t mfsEraseRecord ( 
    MFSDriver * mfsp, 
    mfs_id_t id )
```
Erases a data record.
### Parameters

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mfsp</td>
<td>pointer to the MFSDriver object</td>
</tr>
<tr>
<td>in</td>
<td>id</td>
<td>record numeric identifier, the valid range is between 1 and MFS_CFG_MAX_RECORDS</td>
</tr>
</tbody>
</table>

### 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_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.

### 7.24.5.20 mfsPerformGarbageCollection()

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

Enforces a garbage collection operation.

Garbage collection involves: integrity check, optionally repairs, obsolete data removal, data compaction and a flash bank swap.

### Parameters

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mfsp</td>
<td>pointer to the MFSDriver object</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>MFS_NO_ERROR</td>
<td>if the operation has been successfully completed.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>MFS_ERR_INV_STATE</th>
<th>if the driver is not in MFS_READY state.</th>
</tr>
</thead>
<tbody>
<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.

7.24.5.21 mfsStartTransaction()

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>in</th>
<th>mfsp</th>
<th>pointer to the MFSDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>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>MFS_NO_ERROR</th>
<th>if the operation has been successfully completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>if the driver is 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.

### 7.24.5.22 mfsCommitTransaction()

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

A transaction is committed and finalized atomically.

**Parameters**

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

### 7.24.5.23 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**

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

ChibiOS/HAL
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MFS_NO_ERROR</th>
<th>if the operation has been successfully completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFS_ERR_INV_STATE</td>
<td>if the driver is in not in MFS_TRANSACTION state.</td>
</tr>
<tr>
<td>MFS_ERR_FLASH_FAILURE</td>
<td>if the flash memory is unusable because HW failures. Makes the driver enter the MFS_ERROR state.</td>
</tr>
<tr>
<td>MFS_ERR_INTERNAL</td>
<td>if an internal logic failure is detected.</td>
</tr>
</tbody>
</table>

Function Class:

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

MII/RMII Support Header.

7.25.1 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_HANNPTR 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_ISOLOAD 0x0400
- #define BMCR_PDOW 0x0800
- #define BMCR_ANDABLE 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

7.25.2 Macro Definition Documentation

7.25.2.1 MII_BMCR

\#define MII_BMCR 0x00
Basic mode control register.

7.25.2.2 MII_BMSR

\#define MII_BMSR 0x01
Basic mode status register.
7.25.2.3 MII_PHYSID1

#define MII_PHYSID1 0x02
PHYS ID 1.

7.25.2.4 MII_PHYSID2

#define MII_PHYSID2 0x03
PHYS ID 2.

7.25.2.5 MII_ADVERTISE

#define MII_ADVERTISE 0x04
Advertisement control reg.

7.25.2.6 MII_LPA

#define MII_LPA 0x05
Link partner ability reg.

7.25.2.7 MII_EXPANSION

#define MII_EXPANSION 0x06
Expansion register.

7.25.2.8 MII_ANNPTR

#define MII_ANNPTR 0x07
1000BASE-T control.

7.25.2.9 MII_CTRL1000

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

#define MII_STAT1000 0x0a
1000BASE-T status.

7.25.2.11 MII_ESTATUS

#define MII_ESTATUS 0x0f
Extended Status.

7.25.2.12 MII_PHYSTS

#define MII_PHYSTS 0x10
PHY Status register.

7.25.2.13 MII_MICR

#define MII_MICR 0x11
MII Interrupt ctrl register.

7.25.2.14 MII_DCOUNTER

#define MII_DCOUNTER 0x12
Disconnect counter.

7.25.2.15 MII_FCSCOUNTER

#define MII_FCSCOUNTER 0x13
False carrier counter.

7.25.2.16 MII_NWAYTEST

#define MII_NWAYTEST 0x14
N-way auto-neg test reg.
7.25.2.17  MII_RERRCOUNTER

#define MII_RERRCOUNTER 0x15

Receive error counter.

7.25.2.18  MII_SREVISION

#define MII_SREVISION 0x16

Silicon revision.

7.25.2.19  MII_RESV1

#define MII_RESV1 0x17

Reserved.

7.25.2.20  MII_LBRERROR

#define MII_LBRERROR 0x18

Lpback, rx, bypass error.

7.25.2.21  MII_PHYADDR

#define MII_PHYADDR 0x19

PHY address.

7.25.2.22  MII_RESV2

#define MII_RESV2 0x1a

Reserved.

7.25.2.23  MII_TPISTATUS

#define MII_TPISTATUS 0x1b

TPI status for 10Mbps.
7.25.2.24 MII\_NCONFIG

```
#define MII\_NCONFIG 0x1c
```

Network interface config.

7.25.2.25 BMCR\_RESV

```
#define BMCR\_RESV 0x007f
```

Unused.

7.25.2.26 BMCR\_CTST

```
#define BMCR\_CTST 0x0080
```

Collision test.

7.25.2.27 BMCR\_FULLDPLX

```
#define BMCR\_FULLDPLX 0x0100
```

Full duplex.

7.25.2.28 BMCR\_ANRESTART

```
#define BMCR\_ANRESTART 0x0200
```

Auto negotiation restart.

7.25.2.29 BMCR\_ISOLATE

```
#define BMCR\_ISOLATE 0x0400
```

Disconnect DP83840 from MII.

7.25.2.30 BMCR\_PDOWN

```
#define BMCR\_PDOWN 0x0800
```

Powerdown.
7.25.2.31  BMCR_ANENABLE

#define BMCR_ANENABLE 0x1000
Enable auto negotiation.

7.25.2.32  BMCR_SPEED100

#define BMCR_SPEED100 0x2000
Select 100Mbps.

7.25.2.33  BMCR_LOOPBACK

#define BMCR_LOOPBACK 0x4000
TXD loopback bit.

7.25.2.34  BMCR_RESET

#define BMCR_RESET 0x8000
Reset.

7.25.2.35  BMSR_ERCAP

#define BMSR_ERCAP 0x0001
Ext-reg capability.

7.25.2.36  BMSR_JCD

#define BMSR_JCD 0x0002
Jabber detected.

7.25.2.37  BMSR_LSTATUS

#define BMSR_LSTATUS 0x0004
Link status.
7.25.2.38 BMSR_ANEGCAPABLE

#define BMSR_ANEGCAPABLE 0x0008

Able to do auto-negotiation.

7.25.2.39 BMSR_RFAULT

#define BMSR_RFAULT 0x0010

Remote fault detected.

7.25.2.40 BMSR_ANEGCOMPLETE

#define BMSR_ANEGCOMPLETE 0x0020

Auto-negotiation complete.

7.25.2.41 BMSR_MFPRESUPPCAP

#define BMSR_MFPRESUPPCAP 0x0040

Able to suppress preamble.

7.25.2.42 BMSR_RESV

#define BMSR_RESV 0x0780

Unused.

7.25.2.43 BMSR_10HALF

#define BMSR_10HALF 0x0800

Can do 10mbps, half-duplex.

7.25.2.44 BMSR_10FULL

#define BMSR_10FULL 0x1000

Can do 10mbps, full-duplex.
7.25.2.45 BMSR_100HALF

#define BMSR_100HALF 0x2000
Can do 100mbps, half-duplex.

7.25.2.46 BMSR_100FULL

#define BMSR_100FULL 0x4000
Can do 100mbps, full-duplex.

7.25.2.47 BMSR_100BASE4

#define BMSR_100BASE4 0x8000
Can do 100mbps, 4k packets.

7.25.2.48 ADVERTISE_SLCT

#define ADVERTISE_SLCT 0x001f
Selector bits.

7.25.2.49 ADVERTISE_CSMA

#define ADVERTISE_CSMA 0x0001
Only selector supported.

7.25.2.50 ADVERTISE_10HALF

#define ADVERTISE_10HALF 0x0020
Try for 10mbps half-duplex.

7.25.2.51 ADVERTISE_10FULL

#define ADVERTISE_10FULL 0x0040
Try for 10mbps full-duplex.
7.25.2.52  ADVERTISE_100HALF

#if define ADVERTISE_100HALF 0x0080
Try for 100mbps half-duplex.
#endif

7.25.2.53  ADVERTISE_100FULL

#if define ADVERTISE_100FULL 0x0100
Try for 100mbps full-duplex.
#endif

7.25.2.54  ADVERTISE_100BASE4

#if define ADVERTISE_100BASE4 0x0200
Try for 100mbps 4k packets.
#endif

7.25.2.55  ADVERTISE_PAUSE_CAP

#if define ADVERTISE_PAUSE_CAP 0x0400
Try for pause.
#endif

7.25.2.56  ADVERTISE_PAUSE_ASYM

#if define ADVERTISE_PAUSE_ASYM 0x0800
Try for asymetric pause.
#endif

7.25.2.57  ADVERTISE_RESV

#if define ADVERTISE_RESV 0x1000
Unused.
#endif

7.25.2.58  ADVERTISE_RFAULT

#if define ADVERTISE_RFAULT 0x2000
Say we can detect faults.
#endif
7.25.2.59  ADVERTISE_LPACK

#define ADVERTISE_LPACK 0x4000

Ack link partners response.

7.25.2.60  ADVERTISE_NPAGE

#define ADVERTISE_NPAGE 0x8000

Next page bit.

7.25.2.61  LPA_SLCT

#define LPA_SLCT 0x001f

Same as advertise selector.

7.25.2.62  LPA_10HALF

#define LPA_10HALF 0x0020

Can do 10mbps half-duplex.

7.25.2.63  LPA_10FULL

#define LPA_10FULL 0x0040

Can do 10mbps full-duplex.

7.25.2.64  LPA_100HALF

#define LPA_100HALF 0x0080

Can do 100mbps half-duplex.

7.25.2.65  LPA_100FULL

#define LPA_100FULL 0x0100

Can do 100mbps full-duplex.
7.25.2.66  LPA_100BASE4

#define LPA_100BASE4 0x0200

Can do 100mbps 4k packets.

7.25.2.67  LPA_PAUSE_CAP

#define LPA_PAUSE_CAP 0x0400

Can pause.

7.25.2.68  LPA_PAUSE_ASYM

#define LPA_PAUSE_ASYM 0x0800

Can pause asymmetrically.

7.25.2.69  LPA_RESV

#define LPA_RESV 0x1000

Unused.

7.25.2.70  LPA_RFAULT

#define LPA_RFAULT 0x2000

Link partner faulted.

7.25.2.71  LPA_LPACK

#define LPA_LPACK 0x4000

Link partner acked us.

7.25.2.72  LPA_NPAGE

#define LPA_NPAGE 0x8000

Next page bit.
7.25.2.73  EXPANSION_NWAY

#define EXPANSION_NWAY 0x0001

Can do N-way auto-nego.

7.25.2.74  EXPANSION_LCWP

#define EXPANSION_LCWP 0x0002

Got new RX page code word.

7.25.2.75  EXPANSION_ENABLENPAGE

#define EXPANSION_ENABLENPAGE 0x0004

This enables npage words.

7.25.2.76  EXPANSION_NPCAPABLE

#define EXPANSION_NPCAPABLE 0x0008

Link partner supports npage.

7.25.2.77  EXPANSION_MFAULTS

#define EXPANSION_MFAULTS 0x0010

Multiple faults detected.

7.25.2.78  EXPANSION_RESV

#define EXPANSION_RESV 0xffe0

Unused.

7.25.2.79  NWAYTEST_RESV1

#define NWAYTEST_RESV1 0x00ff

Unused.
7.25.2.80 NWAYTEST_LOOPBACK

#define NWAYTEST_LOOPBACK 0x0100

Enable loopback for N-way.

7.25.2.81 NWAYTEST_RESV2

#define NWAYTEST_RESV2 0xfe00

Unused.
7.26  MMC over SPI Driver

Generic MMC driver.

7.26.1  Detailed Description

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`.

7.26.2  Driver State Machine

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

7.26.3  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 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 **MMCCfg**
  
  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 MMCCfg *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.
408 Module Documentation

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 \( x^7 + x^3 + 1 \)).

7.26.4 Macro Definition Documentation

7.26.4.1 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.

7.26.4.2 _mmc_driver_methods

#define _mmc_driver_methods _mmcsd_block_device_methods

MMCDriver specific methods.

7.26.4.3 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

| 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.

### 7.26.4.4 mmcIsWriteProtected

```c
#define mmcIsWriteProtected(
    mmcp ) mmc_lld_is_write_protected(mmcp)
```

Returns the write protect status.

Parameters

| in  | mmcp | pointer to the MMCDriver object |

Returns

The card state.

Return values

| false | card not inserted. |
| true  | card inserted. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.26.5.1 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

7.26.5.2 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
```
7.26.5.3 send_hdr()

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

Sends a command header.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>the command id</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>the command argument</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.26.5.4 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
```

### 7.26.5.5 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 <code>MMCDriver</code> 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

0xFF timed out.

Function Class:

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

Here is the call graph for this function:

```
recvr3
recvr1
spiReceive
```

### 7.26.5.6 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmd</td>
<td>the command id</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>the command argument</td>
</tr>
</tbody>
</table>

**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:

![Call Graph](image)

### 7.26.5.7 send_command_R3()

```c
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
```

### 7.26.5.8 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**

- **HAL_SUCCESS** the operation succeeded.
- **HAL_FAILED** the operation failed.
Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

![Call Graph]

### 7.26.5.9 sync()

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

Waits that the card reaches an idle state.

**Parameters**

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

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

```
sync
spiReceive
spiSelect
spiUnselect
osalSysUnlock
osalSysLock
osalThreadSuspendS
```

### 7.26.5.10 mmcInit()

```c
void mmcInit (void )
```

MMC over SPI driver initialization.

**Note**

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

**Function Class:**

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

### 7.26.5.11 mmcObjectInit()

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

Initializes an instance.

**Parameters**

- `out mmcp` pointer to the `MMCDriver` object
Function Class:

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

7.26.5.12 mmcStart()

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

Configures and activates the MMC peripheral.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
<td>pointer to the MMCDriver object</td>
</tr>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the MMCConfig 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.

7.26.5.13 mmcStop()

void mmcStop (  
    MMCDriver * mmcp )

Disables the MMC peripheral.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mmcp</td>
<td>pointer to the MMCDriver 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)

### 7.26.5.14 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**

- `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_READY* 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:

```
        spiSelect
         /  \
        /    \
     spiUnselect    osalSysLock
      /  \
     /    \
   spiIgnore    recvr1
   /  \
  /    \
spiStart    osalSysUnlock
 /  \
(send_hdr
 |
)
mmcConnect
```

### 7.26.5.15 mmcDisconnect()

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

Brings the driver in a state safe for card removal.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mmcp</code></td>
<td>pointer to the <code>MMCDriver</code> object</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>HAL_SUCCESS</code></td>
<td>the operation succeeded and the driver is now in the <code>MMC_INSERTED</code> state.</td>
</tr>
<tr>
<td><code>HAL_FAILED</code></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:

```
   mmcDisconnect
       |       |
       v       v
  osalSysLock  osalSysUnlock
       |       |
       v       v
spiStart  spi_lld_start
       |       |
       v       v
  sync  spiSelect
       |       |
       v       v
spiReceive  spiUnselect
       |       |
       v       v
spi_lld_start  spi_lld_stop
```

7.26.5.16 mmcStartSequentialRead()

```c
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

<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:

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

Here is the call graph for this function:

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

7.26.5.17 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.
7.26.5.18 mmcStopSequentialRead()

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

Stops a sequential read gracefully.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
</table>

**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.

7.26.5.19 mmcStartSequentialWrite()

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

Starts a sequential write.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to write</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **HAL_SUCCESS** 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](call_graph.png)

7.26.5.20 mmcSequentialWrite()

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:

```
void mmcSequentialWrite()
```

### 7.26.5.21 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]

7.26.5.22 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

- `HAL_SUCCESS` the operation succeeded.
- `HAL_FAILED` the operation failed.
7.26 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:

```
    +----------------+  +----------------+  +----------------+
    | mmcSync        |  | sync            |  | osalSysUnlock   |
    +----------------+  +----------------+  +----------------+
            ^                      |                      |
            |  +----------------+  +----------------+  +----------------+
            |  | spiReceive      |  | osalSysLock     |  | spiUnselect     |
            +----------------+  +----------------+  +----------------+
                   ^                      |                      |
                   |  +----------------+  +----------------+  +----------------+
                   |  | osalSysUnlock   |  | spiUnselect     |  | osalSysLock     |
                   +----------------+  +----------------+  +----------------+
                               ^                                  |
                               |                                  |                      |
                               |                                  |  +----------------+  +----------------+  +----------------+
                               |                                  |  | spiReceive      |  | osalSysLock     |  | spiUnselect     |
                               +----------------+  +----------------+  +----------------+
                                           ^                      |
                                           |  +----------------+  +----------------+  +----------------+
                                           |  | osalSysUnlock   |  | spiUnselect     |  | osalSysLock     |
                                           +----------------+  +----------------+  +----------------+
                                                                 |
                                                                 |  +----------------+  +----------------+  +----------------+
                                                                 |  | osalSysUnlock   |  | spiUnselect     |  | osalSysLock     |
                                                                 +----------------+  +----------------+  +----------------+
```

7.26.5.23 mmcGetInfo()

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

Returns the media info.

Parameters

| in    | mmcp | pointer to the MMCDriver object |
| out   | bdip | pointer to a BlockDeviceInfo structure |

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.
7.26.5.24 mmcErase()

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

Erases blocks.

Parameters

<table>
<thead>
<tr>
<th>in mmcp</th>
<th>pointer to the MMCDriver object</th>
</tr>
</thead>
<tbody>
<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>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:

![Call Graph]

7.26.6 Variable Documentation
7.26 MMC over SPI Driver

7.26.6.1 mmc_vmt

const struct MMCDevDriverVMT mmc_vmt [static]

Initial value:
= {
  (size_t)0,
  (bool (*)(void *))mmc_lld_is_card_inserted,
  (bool (*)(void *))mmc_lld_is_write_protected,
  (bool (*)(void *))mmcConnect,
  (bool (*)(void *))mmcDisconnect,
  mmc_read,
  mmc_write,
  (bool (*)(void *))mmcSync,
  (bool (*)(void *, BlockDeviceInfo *))mmcGetInfo
}

Virtual methods table.

7.26.6.2 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, 0x77, 0x7a, 0x81, 0x80, 0x8f, 0x82, 0x95, 0x98, 0x85, 0x82,
  0x9a, 0x93, 0x9c, 0x9f, 0xa8, 0xa1, 0xa2, 0xa5, 0xb4, 0xb3, 0xb2, 0xb1,
  0xc6, 0xc5, 0xc2, 0xc1, 0xd3, 0xd0, 0xd7, 0xd8, 0xe9, 0xe2, 0xe5, 0xe8,
  0xf6, 0xf5, 0xf2, 0xf1, 0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f,
  0x48, 0x41, 0x5a, 0x53, 0x6c, 0x65, 0x77, 0x7a, 0x81, 0x80, 0x8f, 0x82,
  0x95, 0x98, 0x85, 0x82, 0x9a, 0x93, 0x9c, 0x9f, 0xa8, 0xa1, 0xa2, 0xa5,
  0xb4, 0xb3, 0xb2, 0xb1, 0xc6, 0xc5, 0xc2, 0xc1, 0xd3, 0xd0, 0xd7, 0xd8,
  0xe9, 0xe2, 0xe5, 0xe8, 0xf6, 0xf5, 0xf2, 0xf1,
}

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

7.27.1 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 MMCSDBlockDevice_CMD_SET_BUS_WIDTH`
- `#define MMCSD_CMD_SEND_IF_COND 8U`
- `#define MMCSD_CMD_SEND_EXT_CSD MMCSDBlockDevice_CMD_SEND_IF_COND`
- `#define MMCSD_CMD_SEND_CSD 9U`
- `#define MMCSD_CMD_SEND_CID 10U`
- `#define MMCSD_CMD_SEND_CSD 11U`
- `#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`
7.27 MMC/SD Block Device

- #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_FILE_FORMAT_SLICE 11U, 10U
- #define MMCSD_CSD_MMC_COPY_SLICE 14U, 14U
- #define MMCSD_CSD_MMC_PERM_WRITE_PROTECT_SLICE 13U, 13U
- #define MMCSD_CSD_MMC_TMP_WRITE_PROTECT_SLICE 12U, 12U
- #define MMCSD_CSD_MMC_FILE_FORMAT_SLICE 11U, 10U
- #define MMCSD_CSD_MMC_CRC_SLICE 7U, 1U
- #define MMCSD_CSD_20_CRC_SLICE 7U, 1U
- #define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 15U, 15U
- #define MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE 21U, 21U
- #define MMCSD_CSD_20_WRITE_BL_LEN_SLICE 25U, 12U
• `#define MMCSD_CSD_20_R2W_FACTOR_SLICE` 28U, 26U
• `#define MMCSD_CSD_20_WP_GRP_ENABLE_SLICE` 31U, 31U
• `#define MMCSD_CSD_20_WP_GRP_SIZE_SLICE` 38U, 32U
• `#define MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE` 45U, 39U
• `#define MMCSD_CSD_20_ERASE_BLK_EN_SLICE` 46U, 46U
• `#define MMCSD_CSD_20_C_SIZE_SLICE` 69U, 48U
• `#define MMCSD_CSD_20_DSR_IMP_SLICE` 76U, 76U
• `#define MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE` 77U, 77U
• `#define MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE` 78U, 78U
• `#define MMCSD_CSD_20_READ_BL_PARTIAL_SLICE` 79U, 79U
• `#define MMCSD_CSD_20_READ_BL_LEN_SLICE` 83U, 80U
• `#define MMCSD_CSD_20_CCC_SLICE` 95U, 84U
• `#define MMCSD_CSD_20_TRANS_SPEED_SLICE` 103U, 96U
• `#define MMCSD_CSD_20_NSAC_SLICE` 111U, 104U
• `#define MMCSD_CSD_20_TAAC_SLICE` 119U, 112U
• `#define MMCSD_CSD_20_CSD_STRUCTURE_SLICE` 127U, 126U
• `#define MMCSD_CSD_10_CRC_SLICE` MMCSD_CSD_20_CRC_SLICE
• `#define MMCSD_CSD_10_FILE_FORMAT_SLICE` MMCSD_CSD_20_FILE_FORMAT_SLICE
• `#define MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE` MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE
• `#define MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE` MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE
• `#define MMCSD_CSD_10_COPY_SLICE` MMCSD_CSD_20_COPY_SLICE
• `#define MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE` MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE
• `#define MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE` MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE
• `#define MMCSD_CSD_10_WRITE_BL_LEN_SLICE` MMCSD_CSD_20_WRITE_BL_LEN_SLICE
• `#define MMCSD_CSD_10_R2W_FACTOR_SLICE` MMCSD_CSD_20_R2W_FACTOR_SLICE
• `#define MMCSD_CSD_10_WP_GRP_ENABLE_SLICE` MMCSD_CSD_20_WP_GRP_ENABLE_SLICE
• `#define MMCSD_CSD_10_WP_GRP_SIZE_SLICE` MMCSD_CSD_20_WP_GRP_SIZE_SLICE
• `#define MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE` MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE
• `#define MMCSD_CSD_10_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_BLK_LEN_SLICE` MMCSD_CSD_20_READ_BLK_LEN_SLICE
• `#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_OID_SLICE 87U, 80U`
- `#define MMCSD_CID_SDC_PNM3_SLICE 95U, 88U`
- `#define MMCSD_CID_SDC_PNM4_SLICE 103U, 96U`
- `#define MMCSD_CID_MMC_CRC_SLICE 7U, 1U`
- `#define MMCSD_CID_MMC_MDT_M_SLICE 15U, 12U`
- `#define MMCSD_CID_MMC_MDT_Y_SLICE 11U, 8U`
- `#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 mmcGetCardCapacity(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.

7.27.2 Macro Definition Documentation

7.27.2.1 MMCSD_BLOCK_SIZE

#define MMCSD_BLOCK_SIZE 512U

Fixed block size for MMC/SD block devices.
7.27 MMC/SD Block Device

7.27.2.2 MMCSD_R1_ERROR_MASK

#define MMCSD_R1_ERROR_MASK 0xFDFFE008U

Mask of error bits in R1 responses.

7.27.2.3 MMCSD_CMD8_PATTERN

#define MMCSD_CMD8_PATTERN 0x000001AAU

Fixed pattern for CMD8.

7.27.2.4 _mmcsd_block_device_methods

#define _mmcsd_block_device_methods _base_block_device_methods

MMCSDBlockDevice specific methods.

7.27.2.5 _mmcsd_block_device_data

#define _mmcsd_block_device_data

Value:

_base_block_device_data
/+ Card CID.+/
uint32_t cid[4];
/+ Card CSD.+/
uint32_t csd[4];
/+ Total number of blocks in card.+/
uint32_t capacity;

MMCSDBlockDevice specific data.

Note
It is empty because MMCSDBlockDevice is only an interface without implementation.

7.27.2.6 MMCSD_R1_ERROR

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

Evaluates to true if the R1 response contains error flags.
7.27.2.7 MMCSD_R1_STS

#define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)

Returns the status field of an R1 response.

Parameters

- **in** `r1` the r1 response

7.27.2.8 MMCSD_R1_IS_CARD_LOCKED

#define MMCSD_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

7.27.2.9 mmcsdGetCardCapacity

#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.

7.27.3 Function Documentation

7.27.3.1 _mmcsd_get_slice()

```c
uint32_t _mmcsd_get_slice ( const uint32_t * data, uint32_t end, uint32_t start )
```

Gets a bit field from a words array.

**Note**

The bit zero is the LSb of the first word.

**Parameters**

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

7.27.3.2 _mmcsd_get_capacity()

```c
uint32_t _mmcsd_get_capacity ( const uint32_t * csd )
```

Extract card capacity from a CSD.

The capacity is returned as number of available blocks.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>csd</strong> 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:

```
_ mmcsd_get_capacity
  _ mmcsd_get_slice
```

7.27.3.3  _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

| in | ext_csd | the extended CSD record |

Returns

The card capacity.

Function Class:

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

void _mmcsd_unpack_sdc_cid (  
   const MMCSDBlockDevice * sdp,  
   unpacked_sdc_cid_t * cidsdc )

Unpacks SDC CID array in structure.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to the MMCSDBlockDevice object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>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:

7.27.3.5 _mmcsd_unpack_mmc_cid()

void _mmcsd_unpack_mmc_cid (  
   const MMCSDBlockDevice * sdp,  
   unpacked_mmc_cid_t * cidmmc )

Unpacks MMC CID array in structure.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to the MMCSDBlockDevice object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>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
```

7.27.3.6 _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
```
### 7.27.3.7  

**_mmcsd_unpack_csd_v10()**

```c
void _mmcsd_unpack_csd_v10 (  
    const MMCSDBlockDevice * sdcp,  
    unpacked_sdc_csd_10_t * csd10 )
```

Unpacks SDC CSD v1.0 array in structure.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the MMCSDBlockDevice object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>csd10</td>
<td>pointer to the unpacked_sdc_csd_10_t object</td>
</tr>
</tbody>
</table>

#### Function Class:

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

#### Here is the call graph for this function:

![Call Graph](image)

### 7.27.3.8  

**_mmcsd_unpack_csd_v20()**

```c
void _mmcsd_unpack_csd_v20 (  
    const MMCSDBlockDevice * sdcp,  
    unpacked_sdc_csd_20_t * csd20 )
```

Unpacks SDC CSD v2.0 array in structure.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdc20</th>
<th>pointer to the unpacked_sdc_csd_20_t 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:

![Call Graph Diagram]
7.28 Serial NOR Flash Driver

Serial NOR Flash driver.

7.28.1 Detailed Description

Serial NOR Flash driver.

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

7.28.2 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).

```
<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLS_STOP</td>
<td>Low Power</td>
</tr>
<tr>
<td>FLS_READY</td>
<td>Clock Enabled</td>
</tr>
<tr>
<td>FLS_UNINIT</td>
<td></td>
</tr>
<tr>
<td>FLS_READ</td>
<td>Reading</td>
</tr>
<tr>
<td>FLS_PGM</td>
<td>Programming</td>
</tr>
<tr>
<td>FLS_ERASE</td>
<td>Erasing</td>
</tr>
</tbody>
</table>
```

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.

7.28.3 Macro Definition Documentation

7.28.3.1 SNOR_BUS_DRIVER

#define SNOR_BUS_DRIVER SNOR_BUS_DRIVER_WSPI

Physical transport interface.

7.28.3.2 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.
7.28.3.3 _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.

7.28.3.4 _snor_flash_methods

#define _snor_flash_methods

Value:
_base_flash_methods
_snor_flash_methods_alone

SNORDriver specific methods with inherited ones.

7.28.4 Function Documentation

7.28.4.1 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.

7.28.4.2 bus_acquire()

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

Bus acquisition and lock.
### 7.28.4.3 bus_release()

```c
void bus_release (  
    BUSDriver * busp  )
```

Bus release.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BUSDriver</td>
<td>pointer to the bus driver</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.28.4.4 bus_stop()

```c
void bus_stop (  
    BUSDriver * busp  )
```

Stops the underlying bus driver.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BUSDriver</td>
<td>pointer to the bus driver</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
7.28.4.5 bus_cmd()

void bus_cmd ( 
    BUSDriver * busp, 
    uint32_t cmd )

Sends a naked command.

Parameters

| in | busp | pointer to the bus driver |
| in | cmd  | instruction code          |

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

7.28.4.6 bus_cmd_send()

void bus_cmd_send ( 
    BUSDriver * busp, 
    uint32_t cmd, 
    size_t n, 
    const uint8_t * p )

Sends a command followed by a data transmit phase.

Parameters

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

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

7.28.4.7 bus_cmd_receive()

void bus_cmd_receive ( 
    BUSDriver * busp, 
    uint32_t cmd, 
    size_t n, 
    uint8_t * p )

Sends a command followed by a data receive phase.
7.28 Serial NOR Flash Driver

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.

7.28.4.8  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.

7.28.4.9  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>p</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.

### 7.28.4.10 bus_cmd_addr_receive()

```c
void bus_cmd_addr_receive ( 
    BUSDriver * busp, 
    uint32_t cmd, 
    flash_offset_t offset, 
    size_t n, 
    uint8_t * p )
```

Sends a command followed by a flash address and a data receive phase.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>busp</td>
<td>pointer to the bus driver</td>
</tr>
<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>out</td>
<td>p</td>
<td>data buffer</td>
</tr>
</tbody>
</table>

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

### 7.28.4.11 bus_cmd_dummy_receive()

```c
void bus_cmd_dummy_receive ( 
    BUSDriver * busp, 
    uint32_t cmd, 
    uint32_t dummy, 
    size_t n, 
    uint8_t * p )
```

Sends a command followed by dummy cycles and a data receive phase.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>busp</td>
<td>pointer to the bus driver</td>
</tr>
<tr>
<td>in</td>
<td>cmd</td>
<td>instruction code</td>
</tr>
<tr>
<td>in</td>
<td>dummy</td>
<td>number of dummy cycles</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.

### 7.28.4.12 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.

**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>dummy</td>
<td>number of dummy cycles</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.

### 7.28.4.13 snorObjectInit()

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

Initializes an instance.

**Parameters**

| out  | devp   | pointer to the SNORDriver object |

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

void snorStart (  
    SNORDriver * devp,  
    const SNORConfig * config )

Configures and activates SNOR driver.

Parameters

| in  | devp          | pointer to the SNORDriver object |
| in  | config        | pointer to the configuration     |

Function Class:

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

7.28.4.15 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.

7.28.4.16 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:

```
    snorMemoryMap
       ^
       |   bus_acquire
       |   bus_release
       |   wspiMapFlash
```

7.28.4.17 snorMemoryUnmap()

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

Leaves the memory Mapping mode.

Parameters

| in | devp | pointer to the SNORDriver object |

Function Class:

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

```
454 Module Documentation

7.28.5 Variable Documentation

7.28.5.1 snor_vmt

const struct SNORDriverVMT snor_vmt [static]

Initial value:

```c
{ (size_t)0,
  snor_get_descriptor, snor_read, snor_program,
  snor_start_erase_all, snor_start_erase_sector,
  snor_query_erase, snor_verify_erase,
  snor_read_sfdp
}
```

Virtual methods table.

ChibiOS/HAL
7.29 Base Object

7.29.1 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.

7.29.2 Macro Definition Documentation

7.29.2.1 _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.
7.29.2.2 _base_object_data

#define _base_object_data

BaseObject specific data.

Note
This object defines no data.

7.29.2.3 objGetInstance

#define objGetInstance(
    type,
    ip ) (type)(((size_t)(ip)) - (ip)->vmt->instance_offset)

Returns the instance pointer starting from an interface pointer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>type</th>
<th>the type of the instance pointer, it is used for casting</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ip</td>
<td>the interface pointer</td>
</tr>
</tbody>
</table>

Returns
A pointer to the object implementing the interface
7.30 PAL Driver

I/O Ports Abstraction Layer.

7.30.1 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.

7.30.2 Implementation Rules

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

7.30.2.1 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.

7.30.2.2 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.
7.30.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

7.30.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

7.30.2.5 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)
7.30 PAL Driver

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_PUSH Pull 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) 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.
- `#define palEnablePadEventI(port, pad, mode)
Pad event enable.
• #define palDisablePadEvent(port, pad)

Pad event disable.
• #define palEnableLineEventI(line, mode) palEnablePadEventI(PAL_PORT(line), PAL_PAD(line), mode)

Line event enable.
• #define palDisableLineEventI(line) palDisablePadEventI(PAL_PORT(line), PAL_PAD(line))

Line event disable.
• #define palEnableLineEvent(line, mode)

Line event enable.
• #define palDisableLineEvent(line)

Line event disable.
• #define palIsPadEventEnabledX(port, pad) false

Pad event enable check.
• #define palIsLineEventEnabledX(line) pal_lld_ispadeventenabled(PAL_PORT(line), PAL_PAD(line))

Line event enable check.
• #define palSetPadCallback(port, pad, cb, arg)
Associates a callback to a pad.
• #define palSetLineCallback(line, cb, arg)
Associates a callback to a line.

Port related definitions
• #define PAL_IOPORTS_WIDTH 16U
Width, in bits, of an I/O port.
• #define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)
Whole port mask.

Line handling macros
• #define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad)))
Forms a line identifier.
• #define PAL_PORT(line) ((stm32_gpio_t ∗)(((uint32_t)(line)) & 0xFFFFFFF0U))
Decodes a port identifier from a line identifier.
• #define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))
Decodes a pad identifier from a line identifier.
• #define PAL_NOLINE 0U
Value identifying an invalid line.

Typedefs
• typedef 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 iopadid_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.

7.30.3 Macro Definition Documentation

7.30.3.1 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.
7.30.3.2 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.

7.30.3.3 PAL_MODE_INPUT

#define PAL_MODE_INPUT 2U

Regular input high-Z pad.

7.30.3.4 PAL_MODE_INPUT_PULLUP

#define PAL_MODE_INPUT_PULLUP 3U

Input pad with weak pull up resistor.

7.30.3.5 PAL_MODE_INPUT_PULLDOWN

#define PAL_MODE_INPUT_PULLDOWN 4U

Input pad with weak pull down resistor.

7.30.3.6 PAL_MODE_INPUT_ANALOG

#define PAL_MODE_INPUT_ANALOG 5U

Analog input mode.

7.30.3.7 PAL_MODE_OUTPUT_PUSHPULL

#define PAL_MODE_OUTPUT_PUSHPULL 6U

Push-pull output pad.
7.30.3.8 PAL_MODE_OUTPUT_OPENDRAIN

#define PAL_MODE_OUTPUT_OPENDRAIN 7U

Open-drain output pad.

7.30.3.9 PAL_LOW

#define PAL_LOW 0U

Logical low state.

7.30.3.10 PAL_HIGH

#define PAL_HIGH 1U

Logical high state.

7.30.3.11 PAL_EVENT_MODE_EDGES_MASK

#define PAL_EVENT_MODE_EDGES_MASK 3U

Mask of edges field.

7.30.3.12 PAL_EVENT_MODE_DISABLED

#define PAL_EVENT_MODE_DISABLED 0U

Channel disabled.

7.30.3.13 PAL_EVENT_MODE_RISING_EDGE

#define PAL_EVENT_MODE_RISING_EDGE 1U

Rising edge callback.
7.30.3.14 PAL_EVENT_MODE_FALLING_EDGE

#define PAL_EVENT_MODE_FALLING_EDGE 2U

Falling edge callback.

7.30.3.15 PAL_EVENT_MODE_BOTH_EDGES

#define PAL_EVENT_MODE_BOTH_EDGES 3U

Both edges callback.

7.30.3.16 PAL_USE_CALLBACKS

#define PAL_USE_CALLBACKS TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.30.3.17 PAL_USE_WAIT

#define PAL_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.30.3.18 PAL_PORT_BIT

#define PAL_PORT_BIT(n) ((ioportmask_t)(1U << (n)))

Port bit helper macro.
This macro calculates the mask of a bit within a port.
Parameters

| in  | n   | bit position within the port |

Returns

The bit mask.

### 7.30.3.19 PAL_GROUP_MASK

```c
#define PAL_GROUP_MASK(width) ((ioportmask_t)(1U << (width)) - 1U)
```

Bits group mask helper.

This macro calculates the mask of a bits group.

Parameters

| in  | width | group width |

Returns

The group mask.

### 7.30.3.20 _IOBUS_DATA

```c
#define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}
```

Data part of a static I/O bus initializer.

This macro should be used when statically initializing an I/O bus that is part of a bigger structure.

Parameters

| in  | name | name of the IObus variable |
| in  | port | I/O port descriptor |
| in  | width | bus width in bits |
| in  | offset | bus bit offset within the port |
7.30.3.21 IOBUS_DECL

#define IOBUS_DECL(
    name,
    port,
    width,
    offset ) IOBus name = _IOBUS_DATA(name, port, width, offset)

Static I/O bus initializer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>name of the IOBus variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>port</td>
<td>I/O port descriptor</td>
</tr>
<tr>
<td>in</td>
<td>width</td>
<td>bus width in bits</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>bus bit offset within the port</td>
</tr>
</tbody>
</table>

7.30.3.22 _pal_init_event [1/3]

#define _pal_init_event(
    e )

Value:

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

Initializes a PAL event object.

Parameters

| in  | e    | event index |

Function Class:

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

7.30.3.23 _pal_init_event [2/3]

#define _pal_init_event(
    e )

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

Initializes a PAL event object.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>event index</th>
</tr>
</thead>
</table>

Function Class:

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

**7.30.3.24  _pal_init_event [3/3]**

#define _pal_init_event{
  e }

Value:

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

Initializes a PAL event object.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>event index</th>
</tr>
</thead>
</table>

Function Class:

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

**7.30.3.25  _pal_clear_event [1/3]**

#define _pal_clear_event{
  e }

Value:

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

Clears a PAL event object.
Parameters

| in | e | event index |

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

7.30.3.26 _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.

7.30.3.27 _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.

7.30.3.28 _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();
    osalThreadDequeueAllI(&_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.

7.30.3.29 _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.
7.30.3.30 _pal_isr_code [3/3]

#define _pal_isr_code(e)

Value:

do {
  \osalSysLockFromISR();
  osalThreadDequeueAll(&_pal_events[e].threads, MSG_OK);
  \osalSysUnlockFromISR();
} while (false)

Common ISR code.

Note

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

Parameters

in  e  event index

Function Class:

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

7.30.3.31 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.
7.30.3.32 palReadPort

```c
#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.

7.30.3.33 palReadLatch

```c
#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.
7.30.3.34  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></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.

7.30.3.35  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></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.
7.30.3.36 palClearPort

#define palClearPort(
    port,
    bits ) palWritePort(port, palReadLatch(port) & ~{bits})

Clears a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().

The function can be called from any context.

Parameters

| in | port | port identifier |
| in | bits | bits to be cleared on the specified port |

Function Class:

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

7.30.3.37 palTogglePort

#define palTogglePort(
    port,
    bits ) palWritePort(port, palReadLatch(port) ^ {bits})

Toggles a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().

The function can be called from any context.

Parameters

| in | port | port identifier |
| in | bits | bits to be XORed on the specified port |

Function Class:

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

#define palReadGroup(
    port,
    mask,
    offset ) ((palReadPort(port) >> (offset)) & (mask))

Reads a group of bits.

Note

The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port identifier</td>
</tr>
<tr>
<td>mask</td>
<td>group mask, a logic AND is performed on the input data</td>
</tr>
<tr>
<td>offset</td>
<td>group bit offset within the port</td>
</tr>
</tbody>
</table>

Returns

The group logic states.

Function Class:

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

7.30.3.39  palWriteGroup

#define palWriteGroup(
    port,
    mask,
    offset,
    bits )

Value:

\[
\begin{align*}
palWritePort &\{\text{port}, \ (palReadLatch(\text{port}) \ & \ -(\text{mask} \ \& \ (\text{offset})) \ | \\
&\ (\text{bits} \ \& \ (\text{mask}) \ \& \ (\text{offset})) \} \
\end{align*}
\]

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.

7.30.3.40 palSetGroupMode

#define palSetGroupMode(
    port,
    mask,
    offset,
    mode )

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

Parameters

<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.

7.30.3.41 palReadPad

#define palReadPad(
    port,
    pad ) ((palReadPort(port) >> (pad)) & 1U)
Module Documentation

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

| PAL_LOW | low logic state. |
| PAL_HIGH | high logic state. |

Function Class:

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

7.30.3.42 palWritePad

#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>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>bit</td>
<td>logic value, the value must be PAL_LOW or PAL_HIGH</td>
</tr>
</tbody>
</table>
Function Class:
Special function, this function has special requirements see the notes.

7.30.3.43  palSetPad

#define palSetPad(
    port,
    pad ) palSetPort(port, PAL_PORT_BIT(pad))

Sets a pad logic state to PAL_HIGH.

Note
The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock().
The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:
Special function, this function has special requirements see the notes.

7.30.3.44  palClearPad

#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.

### 7.30.3.45 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.

### 7.30.3.46 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.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>pad mode</td>
</tr>
</tbody>
</table>

Function Class:
Special function, this function has special requirements see the notes.

7.30.3.47 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PAL_LOW</td>
<td>low logic state.</td>
</tr>
<tr>
<td>PAL_HIGH</td>
<td>high logic state.</td>
</tr>
</tbody>
</table>

Function Class:
Special function, this function has special requirements see the notes.

7.30.3.48 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></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.

7.30.3.49  **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

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

Function Class:

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

7.30.3.50  **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.
7.30 PAL Driver

Parameters

| in | line | line identifier |

Function Class:

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

7.30.3.51  palToggleLine

#define palToggleLine(
    line ) palTogglePad(PAL_PORT(line), PAL_PAD(line))

Toggles a line logic state.

Note
The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock(). The function can be called from any context.

Parameters

| in | line | line identifier |

Function Class:

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

7.30.3.52  palSetLineMode

#define palSetLineMode(
    line, mode ) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)

Line mode setup.

Note
The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between osalSysLock() and osalSysUnlock(). The function can be called from any context.
Parameters

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

7.30.3.53 palEnablePadEventI

#define palEnablePadEventI(
    port,
    pad,
    mode )

Pad event enable.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>pad event mode</td>
</tr>
</tbody>
</table>

Function Class:

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

7.30.3.54 palDisablePadEventI

#define palDisablePadEventI(
    port,
    pad )

Pad event disable.

This function also disables previously programmed event callbacks.
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:

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

### 7.30.3.55 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></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>

|   | mode | pad event mode |

Function Class:

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

### 7.30.3.56 palDisablePadEvent

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

**Value:**

ChibiOS/HAL
do {
    osalSysLock();
    palDisablePadEventI(port, pad);
    osalSysUnlock();
} while (false)

Pad event disable.

This function also disables previously programmed event callbacks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Function Class:

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

7.30.3.57 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

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>line event mode</td>
</tr>
</tbody>
</table>

Function Class:

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

7.30.3.58 palDisableLineEventI

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

Line event disable.

This function also disables previously programmed event callbacks.
Parameters

\textbf{in} \hspace{0.5em} \textit{line} \hspace{0.5em} line identifier

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.

\subsection*{7.30.3.59 palEnableLineEvent}

\begin{verbatim}
#define palEnableLineEvent(
    line,
    mode)

Value:

\begin{verbatim}
do {
    osalSysLock();
    palEnableLineEventI(line, mode);
    osalSysUnlock();
} while (false)
\end{verbatim}

Line event enable.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

\begin{verbatim}
in \hspace{0.5em} \textit{line} \hspace{0.5em} line identifier
\hspace{0.5em} in \hspace{0.5em} \textit{mode} \hspace{0.5em} line event mode
\end{verbatim}

Function Class:

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

\subsection*{7.30.3.60 palDisableLineEvent}

\begin{verbatim}
#define palDisableLineEvent(
    line)

Value:

\begin{verbatim}
do {
    osalSysLock();
    palDisableLineEventI(line);
    osalSysUnlock();
} while (false)
\end{verbatim}

Line event disable.

This function also disables previously programmed event callbacks.

ChibiOS/HAL
Parameters

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

Function Class:

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

### 7.30.3.61 palIsPadEventEnabledX

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

Pad event enable check.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
</tbody>
</table>

Returns

Pad event status.

Return values

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

Function Class:

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

### 7.30.3.62 palIsLineEventEnabledX

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

Line event enable check.

Parameters

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

### 7.30.3.63 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**

| in | port | port identifier |
| in | pad  | pad number within the port |
| in | cb   | event callback function |
| in | arg  | callback argument |

Function Class:

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

### 7.30.3.64 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>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:

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

7.30.3.65  PAL_IOPORTS_WIDTH

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

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

7.30.3.66  PAL_WHOLE_PORT

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

Whole port mask.

This macro specifies all the valid bits into a port.

7.30.3.67  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.

7.30.3.68  PAL_PORT

```c
#define PAL_PORT( 
    line ) ((stm32_gpio_t *)((uint32_t)(line)) & 0xFFFFFFF0U)
```

Decodes a port identifier from a line identifier.
#define PAL_PAD( line ) ((uint32_t)((uint32_t)(line) & 0x0000000FU))

Decodes a pad identifier from a line identifier.

#define PAL_NOLINE 0U

Value identifying an invalid line.

#define IOPORT1 0

First I/O port identifier.

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

#define pal_lld_init( ) _pal_lld_init()

Low level PAL subsystem initialization.

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

#define pal_lld_readport( port ) 0U

Reads the physical I/O port states.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
</table>
Returns
The port bits.

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

7.30.3.74  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

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
</table>

Returns
The latched logical states.

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

7.30.3.75  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

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
</table>

| in | bits  | bits to be written on the specified port |
Function Class:
Not an API, this function is for internal use only.

### 7.30.3.76  pal_lld_setport

```c
#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**

<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>bits</strong></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.

### 7.30.3.77  pal_lld_clearport

```c
#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.

### 7.30.3.78 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.

### 7.30.3.79 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.

7.30.3.80  pal_lld_writegroup

#define pal_lld_writegroup(
    port,
    mask,
    offset,
    bits )

Value:

void { 
    (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.
7.30.3.81  pal_lld_setgroupmode

```c
#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.

7.30.3.82  pal_lld_readpad

```c
#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.

7.30.3.83 pal_lld_writepad

#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.
### 7.30.3.84  `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.

### 7.30.3.85  `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>
Function Class:
Not an API, this function is for internal use only.

### 7.30.3.86 pal_lld_togglepad

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

**Value:**
```c
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.

### 7.30.3.87 pal_lld_setpadmode

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

**Value:**
```c
do {
    (void)port;
    (void)pad;
    (void)mode;
} while (false)
```

Pad mode setup.

This function programs a pad with the specified mode.

**Note**
The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Programming an unknown or unsupported mode is silently ignored.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>port</th>
<th>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>pad mode</td>
</tr>
</tbody>
</table>

Function Class:

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

7.30.3.88 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.

7.30.3.89 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

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
</table>
Function Class:
Not an API, this function is for internal use only.

7.30.4 Typedef Documentation

7.30.4.1 palcallback_t

typedef void (*)(palcallback_t) (void *arg)

Type of a PAL event callback.

7.30.4.2 iomask_t

typedef uint32_t iomask_t

Digital I/O port sized unsigned type.

7.30.4.3 iomode_t

typedef uint32_t iomode_t

Digital I/O modes.

7.30.4.4 ioline_t

typedef uint32_t ioline_t

Type of an I/O line.

7.30.4.5 ioid_t

typedef uint32_t ioid_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.
7.30.4.6 iopadid_t

typedef uint32_t iopadid_t

Type of an pad identifier.

7.30.5 Function Documentation

7.30.5.1 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.

7.30.5.2 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.

### 7.30.5.3 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.
7.30.5.4 palSetPadCallbackI()

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

Associates a callback to a port/pad.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> port</td>
<td>port identifier</td>
</tr>
<tr>
<td><strong>in</strong> pad</td>
<td>pad number within the port</td>
</tr>
<tr>
<td><strong>in</strong> cb</td>
<td>event callback function</td>
</tr>
<tr>
<td><strong>in</strong> 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.

7.30.5.5 palSetLineCallbackI()

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

Associates a callback to a line.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> line</td>
<td>line identifier</td>
</tr>
<tr>
<td><strong>in</strong> cb</td>
<td>event callback function</td>
</tr>
<tr>
<td><strong>in</strong> 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.

7.30.5.6 palWaitPadTimeoutS()

```c
msg_t palWaitPadTimeoutS ( 
    ioportid_t port, 
```
### 7.3.0 PAL Driver

```c
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>port identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pad</td>
<td>pad number within the port</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 state.

#### Return values

| MSG_OK      | if an edge has been detected. |
| MSG_TIMEOUT | if a timeout occurred before an edge could be detected. |
| MSG_RESET   | if the event has been disabled while the thread was waiting for an edge. |

### Function Class:

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

Here is the call graph for this function:

![Call Graph]

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

Returns

The operation state.

Return values

| MSG_OK | if an edge has been detected. |
| MSG_TIMEOUT | if a timeout occurred before an edge could be detected. |
| MSG_RESET | if the event has been disabled while the thread was waiting for an edge. |

Function Class:

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

Here is the call graph for this function:

```
osalSysLock
palWaitPadTimeout
osalSysUnlock
palWaitPadTimeoutS
osalThreadEnqueueTimeoutS
```

7.30.5.8 palWaitLineTimeoutS()

```c
msg_t palWaitLineTimeoutS ( 
    ioline_t line, 
    sysinterval_t timeout )
```

Waits for an edge on the specified line.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>line</th>
<th>line identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>operation timeout</td>
</tr>
</tbody>
</table>

Returns

The operation state.

Return values

- **MSG_OK** if an edge has been detected.
- **MSG_TIMEOUT** if a timeout occurred before an edge could be detected.
- **MSG_RESET** if the event has been disabled while the thread was waiting for an edge.

Function Class:

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

Here is the call graph for this function:

```
palWaitLineTimeoutS → osalThreadEnqueueTimeoutS
```

### 7.30.5.9 palWaitLineTimeout()

```c
msg_t palWaitLineTimeout ( 
    ioline_t line, 
    sysinterval_t timeout )
```

Waits for an edge on the specified line.

Parameters

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

![Call Graph](call_graph.png)

7.30.5.10  _pal_lld_init()

void _pal_lld_init (  
    void )

STM32 I/O ports configuration.

Function Class:

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

7.30.5.11  _pal_lld_setgroupmode()

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>in</th>
<th>port</th>
<th>the port identifier</th>
</tr>
</thead>
<tbody>
<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.
7.31 Peripheral Interfaces

HAL Abstract Peripheral Interfaces.

7.31.1 Detailed Description

HAL Abstract Peripheral Interfaces.

Modules

- Generic Flash Interface
  
  HAL Generic Flash Driver Interface.
7.32  PWM Driver

Generic PWM Driver.

7.32.1  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.

7.32.2  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).

7.32.3  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_FRACTION_TO_WIDTH(pwmp, denominator, numerator)`
  Converts from fraction to pulse width.
- `#define PWM_DEGREES_TO_WIDTH(pwmp, degrees)`
  Converts from degrees to pulse width.
- `#define PWM_PERCENTAGE_TO_WIDTH(pwmp, 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)`
  Returns a PWM channel status.
- `#define pwmEnablePeriodicNotificationI(pwmp)`
  Enables the periodic activation edge notification.
- `#define pwmDisablePeriodicNotificationI(pwmp)`
  Disables the periodic activation edge notification.
- `#define pwmEnableChannelNotificationI(pwmp, channel)`
  Enables a channel de-activation edge notification.
- `#define pwmDisableChannelNotificationI(pwmp, channel)`
  Disables a channel de-activation edge notification.
PLATFORM configuration options

- `#define PLATFORM_PWM_USE_PWM1 FALSE`
  
  PWMD1 driver enable switch.

Typedefs

- `typedef struct PWMDriver PWMDriver`
  
  Type of a structure representing a PWM driver.

- `typedef void(*pwmcallback_t)(PWMDriver* pwmp)`
  
  Type of a PWM notification callback.

- `typedef uint32_t pwmmode_t`
  
  Type of a PWM mode.

- `typedef uint8_t pwmchannel_t`
  
  Type of a PWM channel.

- `typedef uint32_t 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.

7.32.4 Macro Definition Documentation

7.32.4.1 PWM_OUTPUT_MASK

#define PWM_OUTPUT_MASK 0x0FU

Standard output modes mask.
### 7.32.4.2 PWM_OUTPUT_DISABLED

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

Output not driven, callback only.

### 7.32.4.3 PWM_OUTPUT_ACTIVE_HIGH

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

Positive PWM logic, active is logic level one.

### 7.32.4.4 PWM_OUTPUT_ACTIVE_LOW

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

Inverse PWM logic, active is logic level zero.

### 7.32.4.5 PWM_FRACTION_TO_WIDTH

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

**Value:**

```
((pwmcnt_t)(((pwmcnt_t)(pwmp)->period) * (pwmcnt_t)(numerator)) / (pwmcnt_t)(denominator))
```

Converts from fraction to pulse width.

**Note**

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>pwmp</th>
<th>pointer to a PWMDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>denominator</td>
<td>denominator of the fraction</td>
</tr>
<tr>
<td>in</td>
<td>numerator</td>
<td>numerator of the fraction</td>
</tr>
</tbody>
</table>
Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

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

### 7.32.4.6 PWM_DEGREES_TO_WIDTH

```c
#define PWM_DEGREES_TO_WIDTH(pwmp, degrees) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)
```

Converts from degrees to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify hundredths of degrees but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>pwmp</code></th>
<th>pointer to a <code>PWMDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>degrees</code></td>
<td>degrees as an integer between 0 and 36000</td>
</tr>
</tbody>
</table>

Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

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

### 7.32.4.7 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`.
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.

7.32.4.8 `pwmChangePeriodI`

```c
#define pwmChangePeriodI(
    pwmp,
    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.

### 7.32.4.9 pwmEnableChannelI

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

Value:
```
do {
    (pwmp)->enabled |= ((pwmchnmsk_t)1U « (pwmchnmsk_t)(channel));
    pwm_lld_enable_channel(pwmp, channel, width);
} while (false)
```

Enables a PWM channel.

Precondition

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

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

<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>
<tr>
<td><code>width</code></td>
<td>PWM pulse width as clock pulses number</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.32.4.10 pwmDisableChannelI

```c
#define pwmDisableChannelI(
    pwmp,
    channel )
```
Disables a PWM channel.

Precondition
The PWM unit must have been activated using \texttt{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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\texttt{pwmp}</td>
<td>pointer to a \texttt{PWMDriver} object</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{channel}</td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>

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

\subsection*{7.32.4.11 \texttt{pwmIsChannelEnabledI}}

\begin{verbatim}
#define pwmIsChannelEnabledI(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)
\end{verbatim}

Returns a PWM channel status.

Precondition
The PWM unit must have been activated using \texttt{pwmStart}().

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\texttt{pwmp}</td>
<td>pointer to a \texttt{PWMDriver} object</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{channel}</td>
<td>PWM channel identifier (0...channels-1)</td>
</tr>
</tbody>
</table>
Function Class:

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

### 7.32.4.12 \texttt{pwmEnablePeriodicNotificationI}

\begin{verbatim}
#define pwmEnablePeriodicNotificationI(pwmp) pwm_lld_enable_periodic_notification(pwmp)
\end{verbatim}

Enables the periodic activation edge notification.

**Precondition**

The PWM unit must have been activated using \texttt{pwmStart()}.

**Note**

If the notification is already enabled then the call has no effect.

**Parameters**

\begin{tabular}{|l|}
\hline
\textbf{in pwmp} pointer to a PWMDriver object \textbf{in}
\hline
\end{tabular}

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.

### 7.32.4.13 \texttt{pwmDisablePeriodicNotificationI}

\begin{verbatim}
#define pwmDisablePeriodicNotificationI(pwmp) pwm_lld_disable_periodic_notification(pwmp)
\end{verbatim}

Disables the periodic activation edge notification.

**Precondition**

The PWM unit must have been activated using \texttt{pwmStart()}.

**Note**

If the notification is already disabled then the call has no effect.
Parameters

| 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.

### 7.32.4.14 pwmEnableChannelNotificationI

```c
#define pwmEnableChannelNotificationI(pwmp, channel) pwm_lld_enable_channel_notification(pwmp, channel)
```

Enables a channel de-activation edge notification.

**Precondition**
- The PWM unit must have been activated using `pwmStart()`.
- The channel must have been activated using `pwmEnableChannel()`.

**Note**
If the notification is already enabled then the call has no effect.

Parameters

| in  | pwmp | pointer to a PWMDriver object |
| in  | channel | PWM channel identifier (0...channels-1) |

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

### 7.32.4.15 pwmDisableChannelNotificationI

```c
#define pwmDisableChannelNotificationI(pwmp, channel) pwm_lld_disable_channel_notification(pwmp, channel)
```

Disables a channel de-activation edge notification.
Precondition

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

Note

If the notification is already disabled then the call has no effect.

Parameters

| in | pwmp | pointer to a PWMDriver object |
| in | channel | PWM channel identifier (0...channels-1) |

Function Class:

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

7.32.4.16 PWM_CHANNELS

```c
#define PWM_CHANNELS 4
```

Number of PWM channels per PWM driver.

7.32.4.17 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.
7.32.4.18 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 `pwmStart()`.

Precondition

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

Postcondition

The PWM unit period is changed to the new value.

Note

The function has effect at the next cycle start.
If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in pwmp</code></td>
<td>pointer to a PWM Driver object</td>
</tr>
<tr>
<td><code>in period</code></td>
<td>new cycle time in ticks</td>
</tr>
</tbody>
</table>

Function Class:

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

7.32.5 Typedef Documentation

7.32.5.1 PWMDriver

typedef struct PWMDriver PWMDriver

Type of a structure representing a PWM driver.

7.32.5.2 pwmcallback_t

typedef void(* pwmcallback_t)(PWMDriver *pwmp)

Type of a PWM notification callback.
Parameters

| in | pwm | pointer to a PWMDriver object |

### 7.32.5.3 pwmmode_t

```c
typedef uint32_t pwmmode_t
```

Type of a PWM mode.

### 7.32.5.4 pwmchannel_t

```c
typedef uint8_t pwmchannel_t
```

Type of a PWM channel.

### 7.32.5.5 pwmchnmsk_t

```c
typedef uint32_t pwmchnmsk_t
```

Type of a channels mask.

### 7.32.5.6 pwmcnt_t

```c
typedef uint32_t pwmcnt_t
```

Type of a PWM counter.

### 7.32.6 Enumeration Type Documentation

#### 7.32.6.1 pwmstate_t

```c
enum pwmstate_t
```

Driver state machine possible states.
7.32.7 Function Documentation

7.32.7.1 pwmInit()

```c
void pwmInit()
{
    void
}
```

PWM Driver initialization.

Note

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

Function Class:

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

Here is the call graph for this function:

```
PwmInit -> pwm_lld_init -> pwmObjectInit
```

7.32.7.2 pwmObjectInit()

```c
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.

7.32.7.3 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 |
| in  | 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:
7.32.7.4  pwmStop()

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:

```
osalSysLock
osalSysUnlock
pwmStop
pwm_lld_stop
```

7.32.7.5  pwmChangePeriod()

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

| in | pwmp | pointer to a PWMDriver object |
| in | period | new cycle time 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:

```
pwmChangePeriod
   osalSysLock
   osalSysUnlock
```

7.32.7.6  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).
Parameters

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

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

Here is the call graph for this function:

```
pwmEnableChannel
    osalSysLock

osalSysUnlock
```

### 7.32.7.7 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>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:

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

Here is the call graph for this function:

```
pwmDisableChannel
    osalSysLock
    osalSysUnlock
```

### 7.32.7.8 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

| 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:

```
pwmEnablePeriodicNotification
  osalSysLock
osalSysUnlock
```

### 7.32.7.9 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
```

7.32.7.10  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
```

7.32.7.11  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>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:
   Normal API, 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
```

7.32.7.12  pwm_lld_init()

```c
void pwm_lld_init ( 
    void )
```

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
```

7.32.7.13  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.

### 7.32.7.14 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.

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

Function Class:

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

7.32.7.16  pwm_lld_disable_channel()

void pwm_lld_disable_channel (  
  PWMDriver * pwmp,  
  pwmchannel_t channel )

Disables a PWM channel and its notification.

Precondition

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

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

The function has effect at the next cycle start.

Parameters

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

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

7.32.7.17  pwm_lld_enable_periodic_notification()

void pwm_lld_enable_periodic_notification (  
  PWMDriver * pwmp )

Enables the periodic activation edge notification.
7.32 PWM Driver

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.

7.32.7.18 `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.

7.32.7.19 `pwm_lld_enable_channel_notification()`

```c
void pwm_lld_enable_channel_notification ( 
    PWMDriver * pwmp,
    pwmchannel_t channel )
```

Enables a channel de-activation edge notification.
Precondition

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

Note

If the notification is already enabled then the call has no effect.

Parameters

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

7.32.7.20  `pwm_lld_disable_channel_notification()`

```c
void pwm_lld_disable_channel_notification (  
    PWMDriver * pwmp,  
    pwmchannel_t channel )
```

Disables a channel de-activation edge notification.

Precondition

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

Note

If the notification is already disabled then the call has no effect.

Parameters

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

7.32.8  Variable Documentation
7.32.8.1 PWMD1

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.
7.33  I/O Bytes Queues

7.33.1  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 kinds 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.
• #define iqIsFull(iqp)
  Evaluates to true if the specified input queue is full.
• #define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)
  Input queue read.
• #define oqGetFull(oqp) (qSizeX(oqp) - qSpace(oqp))
  Returns the filled space into an output queue.
• #define oqGetEmpty(oqp) qSpace(oqp)
  Returns the empty space into an output queue.
• #define oqIsEmpty(oqp)
  Evaluates to true if the specified output queue is empty.
• #define oqIsFull(oqp) ((bool)(qSpace(oqp) == 0U))
  Evaluates to true if the specified output queue is full.
• #define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)
  Output queue write.

Typedefs

• typedef struct io_queue io_queue_t
  Type of a generic I/O queue structure.
• typedef void(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.

7.33.2 Macro Definition Documentation

7.33.2.1 Q_OK

#define Q_OK MSG_OK

Operation successful.

7.33.2.2 Q_TIMEOUT

#define Q_TIMEOUT MSG_TIMEOUT

Timeout condition.

7.33.2.3 Q_RESET

#define Q_RESET MSG_RESET

Queue has been reset.
7.33.2.4 Q_EMPTY

#define Q_EMPTY MSG_TIMEOUT

Queue empty.

7.33.2.5 Q_FULL

#define Q_FULL MSG_TIMEOUT

Queue full,

7.33.2.6 qSizeX

#define qSizeX(

Value:
/*lint -save -e9033 [10.8] The cast is safe.*/ 
((size_t)((qp)->q_top - (qp)->q_buffer)) \n/*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.

7.33.2.7 qSpaceI

#define qSpaceI(

Queue space.

Returns the used space if used on an input queue or the empty space if used on an output queue.
7.33.2.8 qGetLink

```c
#define qGetLink(qp) ((qp)->q_link)
```

Returns the queue application-defined link.

**Note**

This function can be called in any context.

**Parameters**

| in  | qp | pointer to a io_queue_t structure |

**Returns**

The application-defined link.

**Function Class:**

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

7.33.2.9 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.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>qp</th>
<th>pointer to a io_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>lk</td>
<td>The application-defined link.</td>
</tr>
</tbody>
</table>

Function Class:
Special function, this function has special requirements see the notes.

### 7.33.2.10 iqGetFullI

```c
#define iqGetFullI(iqp) qSpaceI(iqp)
```

Returns the filled space into an input queue.

Parameters

| in  | iq | pointer to an input_queue_t structure |

Returns
The number of full bytes in the queue.

Return values

| 0 | if the queue is empty. |

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

### 7.33.2.11 iqGetEmptyI

```c
#define iqGetEmptyI(iqp) (qSpaceX(iqp) - qSpaceI(iqp))
```

Returns the empty space into an input queue.

Parameters

| in  | iq | pointer to an input_queue_t structure |

ChibiOS/HAL
Returns

The number of empty bytes in the queue.

Return values

| 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.

### 7.33.2.12 iqIsEmptyI

```c
#define iqIsEmptyI(inqp) ((bool)(qSpaceI(inqp) == 0U))
```

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

**Parameters**

| in | inqp | 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.

### 7.33.2.13 iqIsFullI

```c
#define iqIsFullI(inqp) Value:

/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/
{(bool){{(inqp)->q_wrptr == (inqp)->q_rdptr} && ((inqp)->q_counter != 0U)}}
/*lint -restore*/
```

Evaluates to **true** if the specified input queue is full.
7.33 I/O Bytes Queues

Parameters

| in iqp | 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.

7.33.2.14 iqGet

#define iqGet(
    iqp ) iqGetTimeout(iqp, TIME_INFINITE)

Input queue read.

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

Parameters

| in iqp | pointer to an input_queue_t structure |

Returns

A byte value from the queue.

Return values

| MSG_RESET if the queue has been reset. |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.33.2.15 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.

7.33.2.16 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.

7.33.2.17  oqIsEmptyI

#define oqIsEmptyI(
    oqp )

Value:

/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/
{(bool){(oqp)->q_wrptr == (oqp)->q_rdptr) && (oqp)->q_counter != 0U}}
/*lint -restore*/

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.

7.33.2.18  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

| 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.

7.33.2.19 `oqPut`

```c
#define oqPut(
    oqp,
    b
) oqPutTimeout(oqp, b, TIME_INFINITE)
```

Output queue write.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>oqp</th>
<th>pointer to an <code>output_queue_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written in the queue</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the queue has been reset.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.33.3 Typedef Documentation
7.33 I/O Bytes Queues

7.33.1 io_queue_t

typedef struct io_queue io_queue_t

Type of a generic I/O queue structure.

7.33.2 qnotify_t

typedef void(* qnotify_t) (io_queue_t *qp)

Queue notification callback type.
Parameters

in  qp  the queue pointer

7.33.3 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.

7.33.4 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.

7.33.4 Function Documentation

7.33.4.1 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>iq</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.

7.33.4.2  oq_write()

static size_t oq_write (output_queue_t * oqp, const uint8_t * bp, size_t n ) [static]

Non-blocking output queue write.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or when the output queue has been filled.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>oqp</th>
<th>pointer to an output_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.

Function Class:

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

7.33.4.3  iqObjectInit()

void iqObjectInit (input_queue_t * iqp,
Initiates 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>iqP</td>
<td>pointer to an <code>input_queue_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to a memory area allocated as queue buffer</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of the queue buffer</td>
</tr>
<tr>
<td>in</td>
<td>infy</td>
<td>pointer to a callback function that is invoked when data is read from the queue. The value can be NULL.</td>
</tr>
<tr>
<td>in</td>
<td>link</td>
<td>application defined pointer</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

![Call Graph]

7.33.4.4 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iqp</code></td>
<td>Pointer to an <code>input_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:

![Call Graph](image)

#### 7.33.4.5 `iqPutI()`

```c
msg_t iqPutI ( 
    input_queue_t *iqp,
    uint8_t b )
```

**Input queue write.**

A byte value is written into the low end of an input queue. The operation completes immediately.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iqp</code></td>
<td>Pointer to an <code>input_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>Return Code</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:

![](call_graph.png)

### 7.33.4.6 iqGetI()

```c
msg_t iqGetI (
    input_queue_t *iqp )
```

Input queue non-blocking read.

This function reads a byte value from an input queue. The operation completes immediately.

**Note**

The callback is invoked after removing a character from the queue.

**Parameters**

| in     | iqp | pointer to an input_queue_t structure |

**Returns**

A byte value from the queue.

**Return values**

- `MSG_TIMEOUT` if the queue is empty.
- `MSG_RESET` if the queue has been reset.

Function Class:

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

```c
msg_t iqGetTimeout ( 
    input_queue_t  *iqp, 
    sysinterval_t  timeout )
```

Input queue read with timeout.

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

**Note**

The callback is invoked after removing a character from the queue.

**Parameters**

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

**Returns**

A byte value from the queue.

**Return values**

- `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:
7.33.4.8 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**

<table>
<thead>
<tr>
<th>in</th>
<th>iqp</th>
<th>pointer to an input_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

**Returns**

The number of bytes effectively transferred.

**Function Class:**

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](#)

7.33.4.9 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.
**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ip</th>
<th>pointer to an <code>input_queue_t</code> 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>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The number of bytes effectively transferred.

**Function Class:**

Normal API, 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
```

### 7.33.4.10 `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>out</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 a memory area allocated as queue buffer</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>size of the queue buffer</td>
</tr>
<tr>
<td>in</td>
<td>only</td>
<td>pointer to a callback function that is invoked when data is written to the queue. The value can be NULL.</td>
</tr>
<tr>
<td>in</td>
<td>link</td>
<td>application defined pointer</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
  oqObjectInit → osalThreadQueueObjectInit
```

7.33.4.11  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

| in  | oqp   | pointer to an output_queue_t structure |

Function Class:

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

![Call Graph Image]

### 7.33.4.12 oqPutI()

```c
msg_t oqPutI ( 
    output_queue_t ∗ oqp, 
    uint8_t b )
```

Output queue non-blocking write.

This function writes a byte value to an output queue. The operation completes immediately.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>oqp</th>
<th>pointer to an output_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte value to be written in the queue</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

| MSG_OK | if the operation succeeded. |
| MSG_TIMEOUT | if the queue is full. |
| MSG_RESET | if the queue has been reset. |

**Function Class:**

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

### 7.33.4.13 oqPutTimeout()

```c
msg_t oqPutTimeout ( 
    output_queue_t ∗ oqp, 
```
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 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><strong>in</strong> oqp</td>
<td>pointer to an output_queue_t structure</td>
</tr>
<tr>
<td><strong>in</strong> b</td>
<td>the byte value to be written in the queue</td>
</tr>
<tr>
<td><strong>in</strong> timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The 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 operation succeeded.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the specified time expired.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the queue has been reset.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:
7.33.4.14 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
```

7.33.4.15 oqWriteI()

```c
size_t oqWriteI ( output_queue_t * oqp,
                  const uint8_t * bp,
                  size_t n )
```

Output queue non-blocking write.

The function writes data from a buffer to an output queue. The operation completes immediately.
7.33 I/O Bytes Queues

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>oqp</th>
<th>pointer to an output_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.

Function Class:

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

Here is the call graph for this function:

```
7.33.4.16 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>\textit{oqp}</th>
<th>pointer to an \texttt{output_queue_t} structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\textit{bp}</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>\textit{n}</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
<tr>
<td>in</td>
<td>\textit{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>• \texttt{TIME_IMMEDIATE} immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• \texttt{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:
7.34 RTC Driver

Generic RTC Driver.

7.34.1 Detailed Description

Generic RTC Driver.

This module defines an abstract interface for a Real Time Clock Peripheral.

Precondition

In order to use the RTC driver the `HAL_USE_RTC` option must be enabled in `halconf.h`.

Macros

- `#define RTC_BASE_YEAR 1980U`
  - Base year of the calendar.
- `#define _rtc_driver_methods _base_pers_storage_methods`
  - BasePersistentStorage specific methods.
- `#define rtc_lld_driver_fields uint32_t dummy`
  - Implementation-specific RTCDriver fields.

Date/Time bit masks for FAT format

- `#define RTC_FAT_TIME_SECONDS_MASK 0x0000001FU`
- `#define RTC_FAT_TIME_MINUTES_MASK 0x000007E0U`
- `#define RTC_FAT_TIME_HOURS_MASK 0x0000F800U`
- `#define RTC_FAT_DATE_DAYS_MASK 0x001F0000U`
- `#define RTC_FAT_DATE_MONTHS_MASK 0x01E00000U`
- `#define RTC_FAT_DATE_YEARS_MASK 0xFE000000U`

Day of week encoding

- `#define RTC_DAY_SATURDAY 0U`
- `#define RTC_DAY_MONDAY 1U`
- `#define RTC_DAY_TUESDAY 2U`
- `#define RTC_DAY_WEDNESDAY 3U`
- `#define RTC_DAY_THURSDAY 4U`
- `#define RTC_DAY_FRIDAY 5U`
- `#define RTC_DAY_SATURDAY 6U`
- `#define RTC_DAY_SUNDAY 7U`

Implementation capabilities

- `#define RTC_SUPPORTS_CALLBACKS TRUE`
  - Callback support int the driver.
- `#define RTC_ALARMS 2`
  - Number of alarms available.
- `#define RTC_HAS_STORAGE FALSE`
  - Presence of a local persistent storage.
PLATFORM configuration options

- `#define PLATFORM_RTC_USE_RTC1 FALSE`
  
  RTCD1 driver enable switch.

### Typedefs

- `typedef struct RTCDriver RTCDriver`
  
  Type of a structure representing an RTC driver.

- `typedef unsigned int rtcalarm_t`
  
  Type of an RTC alarm number.

- `typedef void(∗rtccb_t)(RTCDriver ∗rtcp, rtcevent_t event)`
  
  Type of a generic RTC callback.

### Data Structures

- `struct RTCDateTime`
  
  Type of a structure representing an RTC date/time stamp.

- `struct RTCDriverVMT`
  
  RTCDriver virtual methods table.

- `struct RTCDriver`
  
  Structure representing an RTC driver.

- `struct RTCAlarm`
  
  Type of a structure representing an RTC alarm time stamp.

### Functions

- `void rtcInit (void)`
  
  RTC Driver initialization.

- `void rtcObjectInit (RTCDriver ∗rtcp)`
  
  Initializes a generic RTC driver object.

- `void rtcSetTime (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)`
  
  Set current time.

- `void rtcGetTime (RTCDriver ∗rtcp, RTCDateTime ∗timespec)`
  
  Get current time.

- `void rtcSetAlarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAlarm ∗alarmspec)`
  
  Set alarm time.

- `void rtcGetAlarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAlarm ∗alarmspec)`
  
  Get current alarm.

- `void rtcSetCallback (RTCDriver ∗rtcp, rtccb_t callback)`
  
  Enables or disables RTC callbacks.

- `void rtcConvertDateTimeToStructTm (const RTCDateTime ∗timespec, struct tm ∗timp, uint32_t ∗tv_msec)`
  
  Convert RTCDateTime to broken-down time structure.

- `void rtcConvertStructTmToDateTIme (const struct tm ∗timp, uint32_t tv_msec, RTCDateTime ∗timespec)`
  
  Convert broken-down time structure to RTCDateTime.

- `uint32_t rtcConvertDateTimeToFAT (const RTCDateTime ∗timespec)`
  
  Get current time in format suitable for usage in FAT file system.

- `void rtc_lld_init (void)`
  
  RTC driver identifier.
7.34 RTC Driver

- void rtc_lld_set_time (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)
  Set current time.
- void rtc_lld_get_time (RTCDriver ∗rtcp, RTCDateTime ∗timespec)
  Get current time.
- void rtc_lld_set_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAlarm ∗alarmspec)
  Set alarm time.
- void rtc_lld_get_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAlarm ∗alarmspec)
  Get alarm time.

Enumerations

- enum rtcevent_t
  Type of an RTC event.

7.34.2 Macro Definition Documentation

7.34.2.1 RTC_BASE_YEAR

#define RTC_BASE_YEAR 1980U

Base year of the calendar.

7.34.2.2 _rtc_driver_methods

#define _rtc_driver_methods _base_pers_storage_methods

BasePersistentStorage specific methods.

7.34.2.3 RTC_SUPPORTS_CALLBACKS

#define RTC_SUPPORTS_CALLBACKS TRUE

Callback support int the driver.

7.34.2.4 RTC_ALARMS

#define RTC_ALARMS 2

Number of alarms available.
7.34.2.5 RTC_HAS_STORAGE

#define RTC_HAS_STORAGE FALSE

Presence of a local persistent storage.

7.34.2.6 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.

7.34.2.7 rtc_lld_driver_fields

#define rtc_lld_driver_fields uint32_t dummy

Implementation-specific RTCDriver fields.

7.34.3 Typedef Documentation

7.34.3.1 RTCDriver

typedef struct RTCDriver RTCDriver

Type of a structure representing an RTC driver.

7.34.3.2 rtcalarm_t

typedef unsigned int rtcalarm_t

Type of an RTC alarm number.
7.34 RTC Driver

7.34.3 rtcb_t

typedef void(* rtcb_t) (RTCDriver *rtcp, rtcevent_t event)
Type of a generic RTC callback.

7.34.4 Enumeration Type Documentation

7.34.4.1 rtcevent_t

enum rtcevent_t
Type of an RTC event.

7.34.5 Function Documentation

7.34.5.1 rtcInit()

void rtcInit ()
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:

![Call Graph]

7.34.5.2 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.

7.34.5.3 rtcSetTime()

```c
void rtcSetTime (  
    RTCDriver * rtcp,  
    const RTCDateTime * timespec )
```

Set current time.

Note

This function can be called from any context but limitations could be imposed by the low level implementation.
It is guaranteed that the function can be called from thread context.
The function can be reentrant or not reentrant depending on the low level implementation.

Parameters

| 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:

```
rtcSetTime <- rtc_lld_set_time
```
### 7.34.5.4 rtcGetTime()

```c
void rtcGetTime (  
    RTCDriver * rtcp,  
    RTCDateTime * timespec )
```

Get current time.

**Note**

This function can be called from any context but limitations could be imposed by the low level implementation. It is guaranteed that the function can be called from thread context.

The function can be reentrant or not reentrant depending on the low level implementation.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>timespec</td>
<td>pointer to a RTCDateTime structure</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

![Call graph](image)

### 7.34.5.5 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

<table>
<thead>
<tr>
<th>in rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in alarm</td>
<td>alarm identifier</td>
</tr>
<tr>
<td>in 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
```

### 7.34.5.6 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

<table>
<thead>
<tr>
<th>in rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in alarm</td>
<td>alarm identifier</td>
</tr>
<tr>
<td>out 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()
```

### 7.34.5.7 rtcSetCallback()

```c
void rtcSetCallback ( 
    RTCDriver ∗ rtcp, 
    rtc_cb_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**

| In | rtp | pointer to RTC driver structure |
| In | callback | callback function pointer or NULL |

**Function Class:**

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

### 7.34.5.8 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 RTCDateTime structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>timp</td>
<td>pointer to a broken-down time structure</td>
</tr>
<tr>
<td>out</td>
<td>tv_msec</td>
<td>pointer to milliseconds value or NULL</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.34.5.9 rtcConvertStructTmToDateTime()

    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 RTCDateTime structure</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.34.5.10 rtcConvertDateTimeToFAT()

    uint32_t rtcConvertDateTimeToFAT (  
        const RTCDateTime * timespec  
    )

Get current time in format suitable for usage in FAT file system.

Note

The information about day of week and DST is lost in DOS format, the second field loses its least significant bit.

Parameters

| out | timespec | pointer to a RTCDateTime structure |
Returns
FAT date/time value.

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

### 7.34.5.11 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:

![Call graph](rtc_lld_init rtcObjectInit)

### 7.34.5.12 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.
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.

### 7.34.5.13 rtc_lld_get_time()

```c
void rtc_lld_get_time (  
    RTCDriver * rtcp,  
    RTCDateTime * timespec )
```

Get current time.

Note
The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>timespec</td>
<td>pointer to a RTCDateTime structure</td>
</tr>
</tbody>
</table>

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

### 7.34.5.14 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 perform any checks of alarm time validity.
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>alarm</td>
<td>alarm identifier. Can be 1 or 2.</td>
</tr>
<tr>
<td>in</td>
<td>alarmspec</td>
<td>pointer to a RTCAlarm structure.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.34.5.15 rtc_lld_get_alarm()

```c
void rtc_lld_get_alarm ( 
    RTCDriver * rtcp, 
    rtcalarm_t alarm, 
    RTCAlarm * alarmspec )
```

Get alarm time.

**Note**

The function can be called from any context.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>rtcp</th>
<th>pointer to RTC driver structure</th>
</tr>
</thead>
<tbody>
<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:

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

Generic SD Card Driver.

7.35.1 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.

7.35.2 Driver State Machine

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

7.35.3 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` SDriver 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`
  PWMD1 driver enable switch.

Typedefs

- `typedef uint32_t sdcmode_t`
  Type of card flags.
- `typedef uint32_t sdcflags_t`
  SDC Driver condition flags type.
- `typedef struct SDCDriver SDCDriver`
  Type of a structure representing an SDC driver.

Data Structures

- `struct 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 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_set_bus_width (SDCDriver *sdcp)
  Sets bus width for SDC.
- static 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.

- 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.
7.35.4 Macro Definition Documentation

7.35.4.1 SDC_INIT_RETRY

#define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.35.4.2 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.

7.35.4.3 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.

7.35.4.4 SDC_INIT_OCR_V20

#define SDC_INIT_OCR_V20 0x50FF8000U

OCR initialization constant for V20 cards.
7.35.4.5  SDC_INIT_OCR

#define SDC_INIT_OCR 0x80100000U

OCR initialization constant for non-V20 cards.

7.35.4.6  sdcIsCardInserted

#define sdcIsCardInserted(sdc) (sdc_lld_is_card_inserted(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.

7.35.4.7 sdcIsWriteProtected

```c
#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.
7.35 SDC Driver

7.35.4.8 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.

7.35.4.9 _sdc_driver_methods

#define _sdc_driver_methods _mmcsd_block_device_methods

SDCDriver specific methods.

7.35.5 Typedef Documentation

7.35.5.1 sdcmode_t

typedef uint32_t sdcmode_t

Type of card flags.

7.35.5.2 sdcflags_t

typedef uint32_t sdcflags_t

SDC Driver condition flags type.

7.35.5.3 SDCDriver

typedef struct SDCDriver SDCDriver

Type of a structure representing an SDC driver.
7.35.6  Enumeration Type Documentation

7.35.6.1  mmc_switch_t

enum mmc_switch_t
MMC switch mode.

7.35.6.2  sd_switch_t

enum sd_switch_t
SDC switch mode.

7.35.6.3  sd_switch_function_t

enum sd_switch_function_t
SDC switch function.

7.35.6.4  sdcbusmode_t

enum sdcbusmode_t
Type of SDIO bus mode.

7.35.6.5  sdcbusclk_t

enum sdcbusclk_t
Max supported clock.

7.35.7  Function Documentation

7.35.7.1  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

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
mode_detect  sdc_lld_send_cmd_short_crc
```

7.35.7.2 mmc_init()

```c
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

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
mmc_init -> sdc_lid_send_cmd_short
```

### 7.35.7.3  sdc_init()

```c
static bool sdc_init (  
    SCDriver * sdcp ) [static]
```

Init procedure for SDC.

**Parameters**

- **in sdcp** pointer to the SCDriver 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.

### 7.35.7.4  mmc_cmd6_construct()  

```c
static uint32_t mmc_cmd6_construct ( 
    mmc_switch_t access,  
    uint32_t idx, 
```
```c
uint32_t value,
uint32_t cmd_set) [static]
```

Constructs CMD6 argument for MMC.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>EXT_CSD access mode</td>
</tr>
<tr>
<td>idx</td>
<td>EXT_CSD byte number</td>
</tr>
<tr>
<td>value</td>
<td>value to be written in target field</td>
</tr>
<tr>
<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.

7.35.7.5  sdc_cmd6_construct()

static uint32_t sdc_cmd6_construct (sd_switch_t mode, sd_switch_function_t function, uint32_t value) [static]

Constructs CMD6 argument for SDC.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>switch/test mode</td>
</tr>
<tr>
<td>function</td>
<td>function number to be switched</td>
</tr>
<tr>
<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.

7.35.7.6  sdc_cmd6_extract_info()

static uint16_t sdc_cmd6_extract_info (sd_switch_function_t function, const uint8_t * buf) [static]

Extracts information from CMD6 answer.
7.35 SDC Driver

Parameters

| in | function | function number to be switched |
| in | buf | buffer with answer |

Returns

extracted answer.

Function Class:

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

7.35.7.7 sdc_cmd6_check_status()

static bool sdc_cmd6_check_status (  
    sd_switch_function_t function,  
    const uint8_t * buf )  

Checks status after switching using CMD6.

Parameters

| in | function | function number to be switched |
| in | buf | buffer with answer |

Returns

The operation status.

Return values

| HAL_SUCCESS | operation succeeded. |
| HAL_FAILED  | operation failed. |

Function Class:

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

7.35.7.8 sdc_detect_bus_clk()

static bool sdc_detect_bus_clk (  
    SCDriver * sdc,  
    sdbusclk_t * clk )  

Reads supported bus clock and switch SDC 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.

7.35.7.9  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.
7.35.7.10  **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></th>
<th>operation succeeded.</th>
<th>operation failed.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HAL_SUCCESS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HAL_FAILED</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.35.7.11  **sdc_set_bus_width()**

```c
static bool sdc_set_bus_width (  
    SDCDriver * sdcp )  [static]
```

Sets bus width for SDC.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th></th>
<th>operation succeeded.</th>
<th>operation failed.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HAL_SUCCESS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HAL_FAILED</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

7.35.7.12  mmc_set_bus_width()

static bool mmc_set_bus_width (  
    SDCDriver ∗ sdoc ∗ ) [static]

Sets bus width for MMC.

Parameters

\texttt{in sdoc} pointer to the \texttt{SDC_DRIVER} object

Returns
The operation status.

Return values

| \texttt{HAL_SUCCESS} | operation succeeded. |
| \texttt{HAL_FAILED}   | operation failed.    |

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

Here is the call graph for this function:

\begin{center}
\begin{tabular}{|c|c|}
\hline
\texttt{mmcm_set_bus_width} & \texttt{mmcm_cmd6_construct} \\
\hline
\end{tabular}
\end{center}

7.35.7.13  _sdc_wait_for_transfer_state()

bool _sdc_wait_for_transfer_state (  
    SDCDriver ∗ sdoc ∗ )

Wait for the card to complete pending operations.
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdp</td>
</tr>
</tbody>
</table>

pointer to the **SDCDriver** object

Returns

The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HAL_SUCCESS</strong></td>
<td>operation succeeded.</td>
</tr>
<tr>
<td><strong>HAL_FAILED</strong></td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>_sdc_wait_for_transfer</td>
<td>_state</td>
</tr>
<tr>
<td>_state</td>
<td>sdc_lld_send_cmd_short_crc</td>
</tr>
</tbody>
</table>

7.35.7.14  **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
```

### 7.35.7.15 sdcObjectInit()

```c
void sdcObjectInit ( SDCDriver * sdcp )
```

Initializes the standard part of a SDCDriver structure.

**Parameters**

| out | sdcp | pointer to the SDCDriver object |

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

### 7.35.7.16 sdcStart()

```c
void sdcStart ( SDCDriver * sdcp, const SDCConfig * config )
```

Configures and activates the SDC peripheral.

**Parameters**

| in | sdcp | pointer to the SDCDriver object |
| in | config | pointer to the SDCConfig object, can be NULL if the driver supports a default configuration or requires no 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:

```
sdcStart
    osalSysLock
    osalSysUnlock
    sdc_lld_start
```

### 7.35.7.17  sdcStop()

```c
void sdcStop ( SDCDriver * sdcp )
```

Deactivates 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:

![Call Graph]

7.35.7.18 sdcConnect()

```c
bool sdcConnect (  
    SDCDriver * sdcp )
```

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the **BLK_READY** state where it is possible to perform read and write operations.

Parameters

- **in** `sdcp` pointer to the `SDCDriver` object

Returns

- The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HAL_SUCCESS</strong></td>
<td>operation succeeded.</td>
</tr>
<tr>
<td><strong>HAL_FAILED</strong></td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.35.7.19  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></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:

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

Here is the call graph for this function:

7.35.7.20  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>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to read</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the read buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to read</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

| 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.

7.35.7.21 sdcWrite()

bool sdcWrite (  
    SDCDriver * sdcp,  
    uint32_t startblk,  
    const uint8_t * buf,  
    uint32_t n )

Writes one or more blocks.

Precondition

The driver must be in the BLK_READY state after a successful sdcConnect() invocation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to write</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the write buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to write</td>
</tr>
</tbody>
</table>
7.35 SDC Driver

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.

7.35.7.22 sdcGetAndClearErrors()

sdcflags_t sdcGetAndClearErrors (
  SDCDriver * sdcp )

Returns the errors mask associated to the previous operation.

Parameters

| in | sdcp | pointer to the SDCDriver object |

Returns

The errors mask.

Function Class:

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

Here is the call graph for this function:

7.35.7.23 sdcSync()

bool sdcSync (  
  SDCDriver * sdcp )

Waits for card idle condition.
Parameters

| in  | sdcp | pointer to the SDCDriver object |

Returns

The operation status.

Return values

| HAL_SUCCESS | the operation succeeded. |
| HAL_FAILED  | the operation failed.    |

Function Class:

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

Here is the call graph for this function:

![Call graph](call_graph.png)

### 7.35.7.24 sdcGetInfo()

```c
bool sdcGetInfo (    
    SDCDriver * sdcp,    
    BlockDeviceInfo * bdip )
```

Returns the media info.

Parameters

| in  | sdcp | pointer to the SDCDriver object |
| out | bdip | pointer to a BlockDeviceInfo structure |

Returns

The operation status.
**7.35 SDC Driver**

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.

---

### 7.35.7.25 sdcErase()

```c
bool sdcErase(
  SDCDriver * sdcp,
  uint32_t startblk,
  uint32_t endblk
)
```

Erases the supplied blocks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>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

|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.

---

### 7.35.7.26 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:

![Call Graph](image)

### 7.35.7.27 sdc_lld_start()

```c
void sdc_lld_start ( 
    SDCDriver * sdcp )
```

Configures and activates the SDC peripheral.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
</table>

Function Class:

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

### 7.35.7.28 sdc_lld_stop()

```c
void sdc_lld_stop ( 
    SDCDriver * sdcp )
```

Deactivates the SDC peripheral.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
</table>

Function Class:

Not an API, this function is for internal use only.
7.35.7.29  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.

7.35.7.30  sdc_lld_set_data_clk()

```c
void sdc_lld_set_data_clk ( SDCDriver * sdcp, sdcbusclk_t clk )
```

Sets the SDIO clock to data mode (25MHz or less).

**Parameters**

- `in sdcp` pointer to the `SDCDriver` object
- `in clk` the clock mode

**Function Class:**

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

7.35.7.31  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.

7.35.7.32  sdc_lld_set_bus_mode()

```c
void sdc_lld_set_bus_mode (  
    SCDriver ∗ 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 SCDriver 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.

7.35.7.33  sdc_lld_send_cmd_none()

```c
void sdc_lld_send_cmd_none (  
    SCDriver ∗ 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 SCDriver 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.

7.35.7.34  sdc_lld_send_cmd_short()

```c
bool sdc_lld_send_cmd_short (  
    SCDriver ∗ sdcp,
```
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

| in  | sdc | pointer to the SDCDriver object |
| in  | cmd | card command |
| in  | arg | command argument |
| out | resp | pointer to the response buffer (one word) |

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.

7.35.7.35   sdc_lld_send_cmd_short_crc()

bool sdc_lld_send_cmd_short_crc(
    SDCDriver *sdc,
    uint8_t cmd,
    uint32_t arg,
    uint32_t * resp
)

Sends an SDIO command with a short response expected and CRC.

Parameters

| in  | sdc | pointer to the SDCDriver object |
| in  | cmd | card command |
| in  | arg | command argument |
| out | resp | pointer to the response buffer (one word) |
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_SUCCESS</td>
<td>operation succeeded.</td>
</tr>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.35.7.36  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></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.
7.35.7.37  sdc_lld_read()

bool sdc_lld_read (  
    SDCDriver ∗ sdcp,  
    uint32_t startblk,  
    uint8_t ∗ buf,  
    uint32_t n )

Reads one or more blocks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to read</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the read buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to read</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>HAL_SUCCESS</th>
<th>operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_FAILED</td>
<td>operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.35.7.38  sdc_lld_write()

bool sdc_lld_write (  
    SDCDriver ∗ sdcp,  
    uint32_t startblk,  
    const uint8_t ∗ buf,  
    uint32_t n )

Writes one or more blocks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdcp</th>
<th>pointer to the SDCDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startblk</td>
<td>first block to write</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>pointer to the write buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of blocks to write</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

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

### 7.35.7.39  sdc_lld_sync()

```c
bool sdc_lld_sync ( SDCEngine * sdp )
```

Waits for card idle condition.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sdp</td>
</tr>
<tr>
<td></td>
<td>pointer to the SDCEngine object</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>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.

### 7.35.8  Variable Documentation

### 7.35.8.1  sdc_vmt

```c
const struct SDCDriverVMT sdc_vmt  [static]
```
Initial value:

```c
= {
  (size_t)0,
  (bool (*)(void *))sdc_lld_is_card_inserted,
  (bool (*)(void *))sdc_lld_is_write_protected,
  (bool (*)(void *))sdcConnect,
  (bool (*)(void *))sdcDisconnect,
  (bool (*)(uint32_t, uint8_t *, uint32_t))sdcRead,
  (bool (*)(uint32_t, const uint8_t *, uint32_t))sdcWrite,
  (bool (*)(void *))sdcSync,
  (bool (*)(uint32_t, BlockDeviceInfo *))sdcGetInfo
}
```

Virtual methods table.

### 7.35.8.2 SDCD1

**SDCDriver SDCD1**

SDCD1 driver identifier.
7.36 Serial Driver

Generic Serial Driver.

7.36.1 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`.

7.36.2 Driver State Machine

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

![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 sdGet(sdp) iqGet(&(sdp)->iqueue)
  Direct read from a SerialDriver.

- #define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->iqueue, t)
  Direct read from a SerialDriver with timeout specification.

- #define sdWriteI(sdp, b, n) oqWriteI(&(sdp)->oqueue, b, n)
  Direct blocking write to a SerialDriver.

- #define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)
  Direct blocking write to a SerialDriver.

- #define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->oqueue, b, n, t)
  Direct blocking write to a SerialDriver with timeout specification.

- #define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)
  Direct non-blocking write to a SerialDriver.

- #define sdReadI(sdp, b, n) iqReadI(&(sdp)->iqueue, b, n, TIME_INFINITE)
  Direct blocking read from a SerialDriver.
• \#define \texttt{sdRead}(\texttt{sdp}, b, n) \texttt{iqReadTimeout}(&(\texttt{sdp})->\texttt{queue}, b, n, \texttt{TIME\_INFINITE})
  Direct blocking read from a \texttt{SerialDriver}.
• \#define \texttt{sdReadTimeout}(\texttt{sdp}, b, n, t) \texttt{iqReadTimeout}(&(\texttt{sdp})->\texttt{queue}, b, n, t)
  Direct blocking read from a \texttt{SerialDriver} with timeout specification.
• \#define \texttt{sdAsynchronousRead}(\texttt{sdp}, b, n) \texttt{iqReadTimeout}(&(\texttt{sdp})->\texttt{queue}, b, n, \texttt{TIME\_IMMEDIATE})
  Direct non-blocking read from a \texttt{SerialDriver}.

**PLATFORM configuration options**

• \#define \texttt{PLATFORM\_SERIAL\_USE\_USART1} FALSE
  USART1 driver enable switch.

**Typedefs**

• typedef struct \texttt{SerialDriver} \texttt{SerialDriver}  
  Structure representing a serial driver.

**Data Structures**

• struct \texttt{SerialDriverVMT} 
  \texttt{SerialDriver} virtual methods table.
• struct \texttt{SerialDriver}  
  Full duplex serial driver class.
• struct \texttt{SerialConfig}  
  PLATFORM Serial Driver configuration structure.

**Functions**

• void \texttt{sdInit} (void)  
  Serial Driver initialization.
• void \texttt{sdObjectInit} (\texttt{SerialDriver} *\texttt{sdp}, \texttt{qnotify\_t} \texttt{inotify}, \texttt{qnotify\_t} \texttt{onotify})  
  Initializes a generic serial driver object.
• void \texttt{sdStart} (\texttt{SerialDriver} *\texttt{sdp}, const \texttt{SerialConfig} *\texttt{config})  
  Configures and starts the driver.
• void \texttt{sdStop} (\texttt{SerialDriver} *\texttt{sdp})  
  Stops the driver.
• void \texttt{sdIncomingDataI} (\texttt{SerialDriver} *\texttt{sdp}, uint8\_t \texttt{b})  
  Handles incoming data.
• \texttt{msg\_t} \texttt{sdRequestDataI} (\texttt{SerialDriver} *\texttt{sdp})  
  Handles outgoing data.
• bool \texttt{sdPutWouldBlock} (\texttt{SerialDriver} *\texttt{sdp})  
  Direct output check on a \texttt{SerialDriver}.
• bool \texttt{sdGetWouldBlock} (\texttt{SerialDriver} *\texttt{sdp})  
  Direct input check on a \texttt{SerialDriver}.
• \texttt{msg\_t} \texttt{sdControl} (\texttt{SerialDriver} *\texttt{sdp}, unsigned int \texttt{operation}, void *\texttt{arg})  
  Control operation on a serial port.
• void \texttt{sd\_lld\_init} (void)  
  Low level serial driver initialization.
• void \texttt{sd\_lld\_start} (\texttt{SerialDriver} *\texttt{sdp}, const \texttt{SerialConfig} *\texttt{config})  
  Low level serial driver configuration and (re)start.
• void \texttt{sd\_lld\_stop} (\texttt{SerialDriver} *\texttt{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.*

7.36.3 Macro Definition Documentation

7.36.3.1 SD_PARITY_ERROR

#define SD_PARITY_ERROR (eventflags_t)32

*Parity.*

7.36.3.2 SD_FRAMING_ERROR

#define SD_FRAMING_ERROR (eventflags_t)64

*Framing.*

7.36.3.3 SD_OVERRUN_ERROR

#define SD_OVERRUN_ERROR (eventflags_t)128

*Overflow.*

7.36.3.4 SD_NOISE_ERROR

#define SD_NOISE_ERROR (eventflags_t)256

*Line noise.*
7.36.3.5  **SD_BREAK_DETECTED**

```c
#define SD_BREAK_DETECTED (eventflags_t)512
```

LIN Break.

7.36.3.6  **SD_QUEUE_FULL_ERROR**

```c
#define SD_QUEUE_FULL_ERROR (eventflags_t)1024
```

Queue full.

7.36.3.7  **SERIAL_DEFAULT_BITRATE**

```c
#define SERIAL_DEFAULT_BITRATE 38400
```

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.36.3.8  **SERIAL_BUFFERS_SIZE**

```c
#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.

7.36.3.9  **_serial_driver_methods**

```c
#define _serial_driver_methods _base_asynchronous_channel_methods
```

SerialDriver specific methods.
7.36.3.10  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.

7.36.3.11  sdPut

#define sdPut(
sdp,
b) oqPut(&(sdp)->oqueue, b)

Direct write to a SerialDriver.

Note
This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

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

7.36.3.12  sdPutTimeout

#define sdPutTimeout(
 sdp,
b,
t) oqPutTimeout(&(sdp)->oqueue, b, t)

Direct write to a SerialDriver with timeout specification.

Note
This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.36.3.13 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.

7.36.3.14 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.

7.36.3.15 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.
7.36.3.16  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.

7.36.3.17  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.

7.36.3.18  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.
7.36.3.19  sdAsynchronousWrite

#define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->queue, 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.

7.36.3.20  sdReadI

#define sdReadI(sdp, b, n) iqReadI(&(sdp)->queue, 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.

7.36.3.21  sdRead

#define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->queue, 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.
7.36.3.22 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.

7.36.3.23 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.

7.36.3.24 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.
### 7.36.3.25 _serial_driver_data

```c
#define _serial_driver_data

Value:

```c
_sd_base_asynchronous_channel_data

/* Driver state.*/
sdstate_t state;

/* Input queue.*/
input_queue_t iqueue;

/* Output queue.*/
output_queue_t oqueue;

/* Input circular buffer.*/
uint8_t ib[SERIAL_BUFFERS_SIZE];

/* Output circular buffer.*/
uint8_t ob[SERIAL_BUFFERS_SIZE];

/* End of the mandatory fields.*/
```

**SerialDriver** specific data.

### 7.36.4 Typedef Documentation

#### 7.36.4.1 SerialDriver

```c
typedef struct SerialDriver SerialDriver
```

Structure representing a serial driver.

### 7.36.5 Enumeration Type Documentation

#### 7.36.5.1 sdstate_t

```c
enum sdstate_t
```

Driver state machine possible states.

**Enumerator**

<table>
<thead>
<tr>
<th>sdstate_t</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>
7.36.6 Function Documentation

7.36.6.1 sdInit()

```c
void sdInit ( )
```

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
```

7.36.6.2 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>sdp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>inotify</code></td>
</tr>
<tr>
<td>in</td>
<td><code>onotify</code></td>
</tr>
</tbody>
</table>
Function Class:

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

### 7.36.6.3 sdStart()

```c
void sdStart ( SerialDriver * sdp,
              const SerialConfig * config )
```

Configures and starts the driver.

**Parameters**

<table>
<thead>
<tr>
<th>Parmeter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sdp</strong></td>
<td>pointer to a SerialDriver object</td>
</tr>
<tr>
<td><strong>config</strong></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:

```plaintext
osalSysLock
-----------
|          |
| sdStart  |
|          |
|          |
| osalSysUnlock |
|              |
|              |
| sd_lld_start |
```

### 7.36.6.4 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdp</td>
<td>pointer to a <code>SerialDriver</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:

![Call Graph](image)

### 7.36.6.5 `sdIncomingDataI()`

```c
void sdIncomingDataI ( 
    SerialDriver * sdp, 
    uint8_t b )
```

Handles incoming data.

This function must be called from the input interrupt service routine in order to enqueue incoming data and generate the related events.

#### Note

The incoming data event is only generated when the input queue becomes non-empty.

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sdp</th>
<th>pointer to a SerialDriver structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>the byte to be written in the driver's Input Queue</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
sdIncomingDataI iqPutI osalThreadDequeueNextI
```

7.36.6.6  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:

```
sdRequestDataI → oqGetI → osalThreadDequeueNextI
```

### 7.36.6.7 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
```

### 7.36.6.8 \texttt{sdGetWouldBlock()}

```c
bool sdGetWouldBlock (SerialDriver * sdp)
```

Direct input check on a \texttt{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**

- \texttt{sdp}: pointer to a \texttt{SerialDriver} structure

**Returns**

The queue status.

**Return values**

- \texttt{false}: if the next write operation would not block.
- \texttt{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:

```
    sdGetWouldBlock
      ↓
    osalSysLock
```

7.36.6.9 sdControl()

\[
\text{msg_t sdControl (}
\text{  SerialDriver * sdp,}
\text{  unsigned int operation,}
\text{  void * arg )}
\]

Control operation on a serial port.

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>operation</td>
<td>control operation code</td>
</tr>
<tr>
<td>in, out</td>
<td>arg</td>
<td>operation argument</td>
</tr>
</tbody>
</table>

Returns

The control operation status.

Return values

| MSG_OK     | in case of success. |
| MSG_TIMEOUT | in case of operation timeout. |
| MSG_RESET   | in case of operation reset.  |

Function Class:

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

```c
void sd_lld_init (  
    void );
```

Low level serial driver initialization.

Function Class:

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

Here is the call graph for this function:

```
  sd_lld_init → sdObjectInit
```

7.36.6.11 sd_lld_start()

```c
void sd_lld_start (  
    SerialDriver * sdp,  
    const SerialConfig * config );
```

Low level serial driver configuration and (re)start.

Parameters

| in | sdp | pointer to a SerialDriver object |
| in | config | the architecture-dependent serial driver configuration. If this parameter is set to NULL then a default configuration is used. |

Function Class:

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

7.36.6.12 sd_lld_stop()

```c
void sd_lld_stop (  
    SerialDriver * sdp );
```

Low level serial driver stop.

De-initializes the USART, stops the associated clock, resets the interrupt vector.
Parameters

| in  | sdp | pointer to a SerialDriver object |

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

### 7.36.7 Variable Documentation

#### 7.36.7.1 SD1

*SerialDriver* SD1

USART1 serial driver identifier.

#### 7.36.7.2 default_config

*const SerialConfig* default_config [static]

**Initial value:**
```
{ 38400 }
```

Driver default configuration.
7.37 Serial over USB Driver

Serial over USB Driver.

7.37.1 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`.

7.37.2 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.`
**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.

7.37.3 Macro Definition Documentation

7.37.3.1 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.

7.37.3.2 SERIAL_USB_BUFFERS_NUMBER

#define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.37.3.3 _serial_usb_driver_data

#define _serial_usb_driver_data

Value:

_base_asynchronous_channel_data
/* Driver state.*/
  sdustate_t state;
/* Input buffers queue.*/
  input_buffers_queue_t ibqueue;
/* Output queue.*/
  output_buffers_queue_t obqueue;
/* Input buffer.*/
  uint8_t ib[BQ_BUFFER_SIZE(SERIAL_USB_BUFFERS_NUMBER,
    SERIAL_USB_BUFFERS_SIZE)];
/* Output buffer.*/
  uint8_t ob[BQ_BUFFER_SIZE(SERIAL_USB_BUFFERS_NUMBER,
    SERIAL_USB_BUFFERS_SIZE)];
/* End of the mandatory fields.*/
/* Current configuration data.*/
  const SerialUSBConfig *config;

SerialDriver specific data.
7.37 Serial over USB Driver

7.37.3.4 _serial_usb_driver_methods

#define _serial_usb_driver_methods _base_asynchronous_channel_methods

SerialUSBDriver specific methods.

7.37.4 Typedef Documentation

7.37.4.1 SerialUSBDriver

typedef struct SerialUSBDriver SerialUSBDriver

Structure representing a serial over USB driver.

7.37.5 Enumeration Type Documentation

7.37.5.1 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>

7.37.6 Function Documentation

7.37.6.1 ibnotify()

static void ibnotify ( 
    io_buffers_queue_t * bqp ) [static]

Notification of empty buffer released into the input buffers queue.
**7.37.6.2 obnotify()**

static void obnotify (  
    io_buffers_queue_t * bqp ) [static]  

Notification of filled buffer inserted into the output buffers queue.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bqp</td>
<td>the buffers queue pointer.</td>
</tr>
</tbody>
</table>

Here is the call graph for this function:

![Call Graph](image)

**7.37.6.3 sduInit()**

void sduInit (  
    void )  

Serial Driver initialization.

Note

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

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.
### 7.37.6.4 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.

---

### 7.37.6.5 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:
7.37.6.6  sduStop()

void sduStop (  
    SerialUSBDriver * sdup )

Stops the driver.

Any thread waiting on the driver’s queues will be awakened with the message MSG_RESET.

Parameters

| in | sdup | pointer to a SerialUSBDriver object |

Function Class:

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

Here is the call graph for this function:

```
osalThreadDequeueAll
osalOsRescheduleS
osalSysLock
osalSysUnlock
```

7.37.6.7  sduSuspendHookI()

void sduSuspendHookI (  
    SerialUSBDriver * sdup )

USB device suspend handler.

Generates a CHN_DISCONNECT event and puts queues in non-blocking mode, this way the application cannot get stuck in the middle of an I/O operations.

Note

If this function is not called from an ISR then an explicit call to osalOsRescheduleS() in necessary afterward.
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.

7.37.6.8 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.

7.37.6.9 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
  \   \--- oscalThreadDequeueAll
```

### 7.37.6.10  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**

<table>
<thead>
<tr>
<th>true</th>
<th>Message handled internally.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>Message not handled.</td>
</tr>
</tbody>
</table>
### 7.37.6.11 sduSOFHookI()

```c
def 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
    ↓
    ↓
    ↓
obqGetFullBufferI
    ↓
    ↓
    ↓
    sduSOFHookI
    ↓
    ↓
    ↓
    obqTryFlushI
    ↓
    ↓
    ↓
    usbStartTransmitI
    ↓
    ↓
    ↓
    usb_lld_start_in
```

### 7.37.6.12 sduDataTransmitted()

```c
def 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               |
Here is the call graph for this function:

![Call Graph](image)

### 7.37.6.13 sduDataReceived()

```c
void sduDataReceived ( 
    USBDriver * usbp, 
    usbep_t ep )
```

Default data received callback.

The application must use this function as callback for the OUT data endpoint.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>usbp</code> pointer to the <code>USBDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>ep</code> OUT endpoint number</td>
</tr>
</tbody>
</table>

Here is the call graph for this function:

![Call Graph](image)
7.37.6.14  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**

<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>

7.37.6.15  sduControl()

```c
msg_t sduControl ( 
    USBDriver * usbp, 
    unsigned int operation, 
    void * arg )
```

Control operation on a serial USB port.

**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 a USBDriver 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**

| 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.
7.38 SIO Driver

Generic SIO Driver.

7.38.1 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`.

7.38.2 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).
Macros

- `#define sioGetFlagsX(siop) sio_lld_get_flags(siop)`
  Returns the current set of flags and clears it.
- `#define sioRXIsEmptyX(siop) sio_lld_rx_is_empty(siop)`
  Determines the state of the RX FIFO.
- `#define sioTXIsEmptyX(siop) sio_lld_tx_is_empty(siop)`
  Determines the state of the TX FIFO.
- `#define sioRXGetX(siop) sio_lld_rx_get(siop)`
  Returns one frame from the RX FIFO.
- `#define sioTXPutX(siop, data) sio_lld_tx_put(siop, data)`
  Pushes one frame into the TX FIFO.
- `#define sioReadX(siop, buffer, size) sio_lld_read(siop, buffer, size)`
  Reads data from the RX FIFO.
- `#define sioWriteX(siop, buffer, size) sio_lld_write(siop, buffer, size)`
  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_rx_is_empty(siop) true`
  Determines the state of the RX FIFO.
- `#define sio_lld_tx_is_empty(siop) true`
  Determines the state of the TX FIFO.
- `#define sio_lld_rx_get(siop)`
  Returns one frame from the RX FIFO.
- `#define sio_lld_tx_put(siop, data)`
  Pushes one frame into the TX FIFO.

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.

PLATFORM configuration options

- `#define PLATFORM_SIO_USE_SIO1 FALSE`
  SIO driver enable switch.
Typedefs

- 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 uint32_t sioflags_t
  SIO driver condition flags type.
- typedef void(siocb_t)(SIODriver *siop)
  Generic SIO notification callback type.
- typedef void(sioecb_t)(SIODriver *siop, sioflags_t e)
  Receive error SIO notification callback type.

Data Structures

- struct hal_sio_config
  Driver configuration structure.
- struct hal_sio_driver
  Structure representing a SIO driver.

Functions

- void sioInit (void)
  SIO Driver initialization.
- void sioObjectInit (SIODriver *siop)
  Initializes the standard part of a SIODriver structure.
- void sioStart (SIODriver *siop, const SIOConfig *config)
  Configures and activates the SIO peripheral.
- void sioStop (SIODriver *siop)
  Deactivates the SIO peripheral.
- void sio_lld_init (void)
  Low level SIO driver initialization.
- void sio_lld_start (SIODriver *siop)
  Configures and activates the SIO peripheral.
- void sio_lld_stop (SIODriver *siop)
  Deactivates the SIO peripheral.
- 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 }
  Driver state machine possible states.

Variables

- SIODriver SIOD1
  SIO1 driver identifier.
7.38.3 Macro Definition Documentation

7.38.3.1 SIO_NO_ERROR

#define SIO_NO_ERROR 0

No pending conditions.

7.38.3.2 SIO_PARITY_ERROR

#define SIO_PARITY_ERROR 4

Parity error happened.

7.38.3.3 SIO_FRAMING_ERROR

#define SIO_FRAMING_ERROR 8

Framing error happened.

7.38.3.4 SIO_OVERRUN_ERROR

#define SIO_OVERRUN_ERROR 16

Overflow happened.

7.38.3.5 SIO_NOISE_ERROR

#define SIO_NOISE_ERROR 32

Noise on the line.
7.38.3.6 SIO_BREAK_DETECTED

#define SIO_BREAK_DETECTED 64

Break detected.

7.38.3.7 sioGetFlagsX

#define sioGetFlagsX(
    siop ) sio_lld_get_flags(siop)

Returns the current set of flags and clears it.

7.38.3.8 sioRXIsEmptyX

#define sioRXIsEmptyX(
    siop ) sio_lld_rx_is_empty(siop)

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:

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

7.38.3.9 sioTXIsFullX

#define sioTXIsFullX(
    siop ) sio_lld_tx_is_full(siop)

Determines the state of the TX FIFO.
7.38 SIO Driver

Parameters

| in | siop | pointer to the SIODriver object |

Returns

The TX FIFO state.

Return values

| false | if TX FIFO is not full |
| true  | if TX FIFO is full |

Function Class:

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

7.38.3.10 sioRXGetX

#define sioRXGetX(
    siop 
) sio_lld_rx_get(siop)

Returns one frame from the RX FIFO.

Note

If the FIFO is empty then the returned value is unpredictable.

Parameters

| in | siop | pointer to the SIODriver object |

Returns

The frame from RX FIFO.

Function Class:

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

7.38.3.11 sioTXPutX

#define sioTXPutX(
    siop, 
    data 
) sio_lld_tx_put(siop, data)

Pushes one frame into the TX FIFO.
Note

If the FIFO is full then the behavior is unpredictable.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>frame to be written</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.38.3.12 sioReadX

```c
#define sioReadX( siop, buffer, size ) sio_lld_read(siop, buffer, size)
```

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>size</td>
<td>maximum number of frames to read</td>
</tr>
</tbody>
</table>

Returns

The number of received frames.

Function Class:

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

#define sioWriteX(
    siop,
    buffer,
    size ) sio_lld_write(siop, buffer, size)

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

| in | siop | pointer to the SIODriver object |
| out | buffer | buffer containing the data to be transmitted |
| in | size | maximum number of frames to read |

Returns

The number of transmitted frames.

Function Class:

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

7.38.3.14  sioControlX

#define sioControlX(
    siop,
    operation,
    arg ) sio_lld_control(siop, operation, arg)

Control operation on a serial port.

Parameters

| in | siop | pointer to the SIODriver object |
| in | operation | control operation code |
| in,out | arg | operation argument |

Returns

The control operation status.
Return values

| MSG_OK       | in case of success. |
| MSG_TIMEOUT  | in case of operation timeout. |
| MSG_RESET    | in case of operation reset. |

Function Class:

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

### 7.38.3.15 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.

### 7.38.3.16 sio_lld_rx_is_empty

#define sio_lld_rx_is_empty(
   siop ) true

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.

7.38.3.17 sio_lld_tx_is_full

#define sio_lld_tx_is_full(
    siop ) true

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.

7.38.3.18 sio_lld_rx_get

#define sio_lld_rx_get(
    siop )

Returns one frame from the RX FIFO.

Note
   If the FIFO is empty then the returned value is unpredictable.

Parameters
   in  siop  pointer to the SIODriver object
Returns

The frame from RX FIFO.

Function Class:

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

7.38.3.19 sio_lld_tx_put

#define sio_lld_tx_put(
    siop,
    data
)

Pushes one frame into the TX FIFO.

Note

If the FIFO is full then the behavior is unpredictable.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>frame to be written</td>
</tr>
</tbody>
</table>

Function Class:

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

7.38.4 Typedef Documentation

7.38.4.1 SIODriver

typedef struct hal_sio_driver SIODriver

Type of structure representing a SIO driver.

7.38.4.2 SIOConfig

typedef struct hal_sio_config SIOConfig

Type of structure representing a SIO configuration.
7.38.4.3 sioflags_t

typedef uint32_t sioflags_t

SIO driver condition flags type.

7.38.4.4 siocb_t

typedef void(* siocb_t) (SIODriver *siop)

Generic SIO notification callback type.

Parameters

| in | siop | pointer to the SIODriver object |

7.38.4.5 sioecb_t

typedef void(* sioecb_t) (SIODriver *siop, sioflags_t e)

Receive error SIO notification callback type.

Parameters

| in | siop | pointer to the SIODriver object triggering the callback |
| in | e    | receive error mask |

7.38.5 Enumeration Type Documentation

7.38.5.1 siostate_t

enum siostate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIO_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SIO_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>SIO_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>ChibiOS/HAL</td>
<td></td>
</tr>
</tbody>
</table>
7.38.6 Function Documentation

7.38.6.1 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:

![Call Graph](image)

7.38.6.2 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.
7.38 SIO Driver

7.38.6.3 sioStart()

```c
void 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</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:

```
sioStart -> osalSysLock -> sioStop
```

---

7.38.6.4 sioStop()

```c
void sioStop (  
    SIODriver * siop )
```

Deactivates the SIO peripheral.

**Parameters**

| in   | siop  | pointer to the SIODriver 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)

### 7.38.6.5 `sio_lld_init()`

```c
void sio_lld_init ()
    void )
```

Low level SIO driver initialization.

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)

### 7.38.6.6 `sio_lld_start()`

```c
void sio_lld_start ( 
    SIODriver * siop )
```

Configures and activates the SIO peripheral.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
</table>

Function Class:

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

### 7.38.6.7 sio_lld_stop()

```c
void sio_lld_stop (SIODriver * siop)
```

Deactivates the SIO peripheral.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
</table>

Function Class:

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

### 7.38.6.8 sio_lld_control()

```c
msg_t sio_lld_control (SIODriver * siop,
                       unsigned int operation,
                       void * arg)
```

Control operation on a serial port.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>siop</th>
<th>pointer to the SIODriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td></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:

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

### 7.38.7 Variable Documentation

#### 7.38.7.1 SIOD1

**SIODriver** SIOD1

SIO1 driver identifier.
7.39 SPI Driver

Generic SPI Driver.

7.39.1 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.

7.39.2 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).

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.
PLATFOR\textit{M} 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\((\ast spicallback\_t) (SPIDriver \ast spip)\)
  
  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 \ast spip)
  
  Initializes the standard part of a SPI\_Driver structure.
- void spiStart (SPIDriver \ast spip, const SPIConfig \ast \text{config})
  
  Configures and activates the SPI peripheral.
- void spiStop (SPIDriver \ast spip)
  
  Deactivates the SPI peripheral.
- void spiSelect (SPIDriver \ast spip)
  
  Asserts the slave select signal and prepares for transfers.
- void spiUnselect (SPIDriver \ast spip)
  
  Deasserts the slave select signal.
- void spiStartIgnore (SPIDriver \ast spip, size\_t n)
  
  Ignores data on the SPI bus.
- void spiStartExchange (SPIDriver \ast spip, size\_t n, const void \ast txbuf, void \ast rxbuf)
  
  Exchanges data on the SPI bus.
- void spiStartSend (SPIDriver \ast spip, size\_t n, const void \ast txbuf)
  
  Sends data over the SPI bus.
- void spiStartReceive (SPIDriver \ast spip, size\_t n, void \ast rxbuf)
  
  Receives data from the SPI bus.
- void spiAbortI (SPIDriver \ast spip)
  
  aborts the ongoing SPI operation.
- void spiIgnore (SPIDriver \ast 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_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.

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.

7.39.3 Macro Definition Documentation
7.39.3.1 SPI_USE_WAIT

#define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note
    Disabling this option saves both code and data space.

7.39.3.2 SPI_USE_CIRCULAR

#define SPI_USE_CIRCULAR FALSE

Enables circular transfers APIs.

Note
    Disabling this option saves both code and data space.

7.39.3.3 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.

7.39.3.4 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.

7.39.3.5 spiIsBufferComplete

#define spiIsBufferComplete(spip) ((bool)((spip)->state == SPI_COMPLETE))

Buffer state.

Note
    This function is meant to be called from the SPI callback only.
## 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 SPI Driver 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.

## 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.
7.39 SPI Driver

Parameters

| in | spip | pointer to the SPI Driver 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.

7.39.3.8 spiStartIgnoreI

#define spiStartIgnoreI(
    spip,
    n
)

Value:

    (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 SPI Driver 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.

7.39.3.9 spiStartExchangeI

#define spiStartExchangeI(
    spip,
    ChibiOS/HAL
n,  
 txbuf,  
 rxbuf)

Value:  
{  
(spip)->state = SPI_ACTIVE;  
spi_lld_exchange(spip, n, txbuf, rxbuf);  
}

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition
A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition
At the end of the operation the configured callback is invoked.

Note
The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

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

7.39.3.10 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spip</code></td>
<td>pointer to the <code>SPIDriver</code> 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:

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

7.39.3.11 `spiStartReceiveI`

```c
#define spiStartReceiveI(spip, n, rxbuf)  
{  
  (spip)->state = SPI_ACTIVE;  
  spi_lld_receive(spip, n, rxbuf);  
}
```

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>spip</em></td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>n</em></td>
<td>number of words to receive</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><em>nbuf</em></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.

7.39.3.12 *spiPolledExchange*

```c
#define spiPolledExchange(
    spip,
    frame ) spi_lld_polled_exchange(spip, frame)
```

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

Note

This API is implemented as a macro in order to minimize latency.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>spip</em></td>
<td>pointer to the SPIDriver object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>frame</em></td>
<td>the data frame to send over the SPI bus</td>
</tr>
</tbody>
</table>

Returns

The received data frame from the SPI bus.

7.39.3.13 *_spi_wakeup_isr*

```c
#define _spi_wakeup_isr(
    spip )
```

Value:

```c
    {
        osalSysLockFromISR();
        osalThreadResumeI(&(spip)->thread, MSG_OK);
        osalSysUnlockFromISR();
    }
```

Wakes up the waiting thread.
Parameters

| in | spip | pointer to the SPIDriver object |

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

### 7.39.3.14 _spi_isr_code

```c
#define _spi_isr_code(
    spip )

Value:

```c
if ((spip)->config->end_cb) { 
    (spip)->state = SPI_COMPLETE;
    (spip)->config->endCb(spip);
    if ((spip)->state == SPI_COMPLETE) 
        (spip)->state = SPI_READY;
    } 
else 
    (spip)->state = SPI_READY;
    _spi_wakeup_isr(spip);
}
```

Common ISR code when circular mode is not supported.

This code handles the portable part of the ISR code:

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

Note

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

Parameters

| in | spip | pointer to the SPIDriver object |

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

#define _spi_isr_half_code( 
  spip )

Value:
{
  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 SPI Driver object

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

7.39.3.16 _spi_isr_full_code

#define _spi_isr_full_code( 
  spip )

Value:
{
  if ((spip)->config->end_cb) {
    (spip)->state = SPI_COMPLETE;
    (spip)->config->end_cb(spip);
    if ((spip)->state == SPI_COMPLETE)
      (spip)->state = SPI_ACTIVE;
  }
}

Full buffer filled ISR code in circular mode.

This code handles the portable part of the ISR code:

• Callback invocation.

• Driver state transitions.

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

| in  | spip | pointer to the SPIDriver object |

Function Class:

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

### 7.39.3.17 SPI_SUPPORTS_CIRCULAR

#define SPI_SUPPORTS_CIRCULAR TRUE

Circular mode support flag.

### 7.39.3.18 PLATFORM_SPI_USE_SPI1

#define PLATFORM_SPI_USE_SPI1 FALSE

SPI1 driver enable switch.

If set to TRUE the support for SPI1 is included.

**Note**

The default is FALSE.

### 7.39.3.19 spi_lld_driver_fields

#define spi_lld_driver_fields

**Value:**

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

Low level fields of the SPI driver structure.

### 7.39.3.20 spi_lld_config_fields

#define spi_lld_config_fields

**Value:**

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

Low level fields of the SPI configuration structure.
7.39.4 Typedef Documentation

7.39.4.1 SPIDriver

typedef struct hal_spi_driver SPIDriver

Type of a structure representing an SPI driver.

7.39.4.2 SPIConfig

typedef struct hal_spi_config SPIConfig

Type of a SPI driver configuration structure.

7.39.4.3 spicallback_t

typedef void(* spicallback_t) (SPIDriver *spip)

SPI notification callback type.

Parameters

| in  | spip | pointer to the SPIDriver object triggering the callback |

7.39.5 Enumeration Type Documentation

7.39.5.1 spistate_t

enum spistate_t

Driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>SPI_STOP</td>
<td>Stopped.</td>
</tr>
</tbody>
</table>
### 7.39.6 Function Documentation

#### 7.39.6.1 spiInit()

```c
void spiInit (  
    void )
```

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](call_graph.png)

#### 7.39.6.2 spiObjectInit()

```c
void spiObjectInit (  
    SPIDriver * sip )
```

Initializes the standard part of a SPIDriver structure.

**Table**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>SPI_ACTIVE</td>
<td>Exchanging data.</td>
</tr>
<tr>
<td>SPI_COMPLETE</td>
<td>Asynchronous operation complete.</td>
</tr>
</tbody>
</table>
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td>spip</td>
<td>pointer to the SPI()Driver object</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:

![Call graph](call_graph.png)

7.39.6.3 spiStart()

```c
void spiStart (  
    SPIDriver * spip,  
    const SPIConfig * config )
```

Configures and activates the SPI peripheral.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>spip</td>
<td>pointer to the SPI()Driver object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>config</td>
<td>pointer to the SPI()Config 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:

```
spiStart
osalSysLock
osalSysUnlock
spi_lld_start
```

### 7.39.6.4 spiStop()

```c
void spiStop ( SPIDriver * spip )
```

Deactivates the SPI peripheral.

**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 for spiStop function]

### 7.39.6.5 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:

![Call Graph for spiSelect function]
### 7.39.6.6 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)

### 7.39.6.7 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

<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 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:

```
spiStartIgnore
    - osalSysLock
    - spiStartIgnore
    - osalSysUnlock
```

### 7.39.6.8 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>spip</code></td>
</tr>
<tr>
<td>in</td>
<td><code>n</code></td>
</tr>
<tr>
<td>in</td>
<td><code>txbuf</code></td>
</tr>
<tr>
<td>out</td>
<td><code>rxbuf</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:

```plaintext
spiStartExchange
  osalSysLock
  spiStartExchange
  osalSysUnlock
```

### 7.39.6.9 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

| in | spip | pointer to the SPI Driver object |
| in | n    | number of words to send          |
| in | txbuf| the pointer to the transmit buffer |

Function Class:

Normal API, 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]

7.39.6.10  spiStartReceive()

void spiStartReceive (  
    SPIDriver * spip,  
    size_t n,  
    void * 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.
7.39 SPI Driver

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in spip</td>
<td>pointer to the SPI Driver object</td>
</tr>
<tr>
<td>in n</td>
<td>number of words to receive</td>
</tr>
<tr>
<td>out 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:

```
spiStartReceive
    ^
    | osalSysLock
  osalSysUnlock
```

7.39.6.11 spiAbortI()

```c
void spiAbortI (  
    SPI Driver * spip )
```

Aborts the ongoing SPI operation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in spip</td>
<td>pointer to the SPI Driver object</td>
</tr>
</tbody>
</table>
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:

```
spiAbortI
    osalThreadResumeI
        spiAbortI
            spi_lld_abort
```

### 7.39.6.12 `spiAbort()`

```c
void spiAbort ( SPIDriver * spip )
```

Aborts the ongoing SPI operation, if any.

**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](image)

### 7.39.6.13 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>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 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
```

7.39.6.14  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>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:

Normal API, 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
```

### 7.39.6.15 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
  osalThreadSuspendS
```

7.39.6.16  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
  ↓
spiReceive
  ↓
osalSysUnlock
  ↓
osalThreadSuspendS
```

### 7.39.6.17 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**

- `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
```

### 7.39.6.18 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
```
### 7.39.6.19 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:

![Call Graph](image)

### 7.39.6.20 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.

### 7.39.6.21 spi_lld_stop()

```c
void spi_lld_stop ( SPIDriver * spip )
```

Deactivates the SPI peripheral.
692 Module Documentation

Parameters

| in | spip | pointer to the SPIDriver object |

Function Class:

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

7.39.6.22 spi_lld_select()

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.

7.39.6.23 spi_lld_unselect()

void spi_lld_unselect (  
    SPIDriver * spip  )

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

| in | spip | pointer to the SPIDriver object |

Function Class:

Not an API, this function is for internal use only.
7.39.6.24  

**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 <strong>SPIDriver</strong> 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.

7.39.6.25  

**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 <strong>SPIDriver</strong> 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.

7.39.6.26 spi_lld_send()

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.

7.39.6.27 spi_lld_receive()

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>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.

### 7.39.6.28 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.

### 7.39.6.29 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.
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.

Function Class:

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

7.39.7 Variable Documentation

7.39.7.1 SPID1

SPIDriver SPID1

SPI1 driver identifier.
7.40 ST Driver

Generic System Tick Driver.

7.40.1 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 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.

- systime_t stGetCounter (void)
  
  Returns the time counter value.

- bool stsAlarmActive (void)
  
  Determines if the alarm zero is active.

- bool stsAlarmActiveN (unsigned alarm)
  
  Determines if the specified alarm is active.

- void stStartAlarmN (unsigned alarm, systime_t abstime, st_callback_t cb)
  
  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.

- 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.
7.40.2 Function Documentation

7.40.2.1 stInit()

```c
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.

7.40.2.2 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>in</th>
<th>abstime</th>
<th>the time to be set for the first alarm</th>
</tr>
</thead>
</table>
7.40 ST Driver

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

Here is the call graph for this function:

7.40.2.3 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:

7.40.2.4 stSetAlarm()

void stSetAlarm (  
    systime_t abstime )

Sets the alarm zero time.

Note
This functionality is only available in free running mode, the behavior in periodic mode is undefined.
Parameters

\begin{tabular}{|c|c|}
  \hline
  in & \textit{abstime} & the time to be set for the next alarm \\
  \hline
\end{tabular}

Function Class:

Normal API, 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 stSetAlarm()](call_graph_stSetAlarm.png)

7.40.2.5 \textbf{stGetAlarm()}

\begin{verbatim}
systime_t stGetAlarm (  
    void  )
\end{verbatim}

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:

![Call Graph for stGetAlarm()](call_graph_stGetAlarm.png)
7.40 ST Driver

7.40.2.6 stGetCounter()

```
systime_t stGetCounter (void)
```

Returns the time counter value.

**Note**

This functionality is only available in free running mode, the behaviour in periodic mode is undefined.

**Returns**

The counter value.

**Function Class:**

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

Here is the call graph for this function:

```
| stGetCounter | st_lld_get_counter |
```

7.40.2.7 stIsAlarmActive()

```
bool stIsAlarmActive (void)
```

Determines if the alarm zero is active.

**Returns**

The alarm status.

**Return values**

| false | if the alarm is not active. |
| true  | is the alarm is active |
Function Class:

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

Here is the call graph for this function:

```
    stIsAlarmActive
         ^                  ^
         |                  |
        --> st_lld_is_alarm_active
```

### 7.40.2.8 stIsAlarmActiveN()

```c
bool stIsAlarmActiveN (  
    unsigned alarm  )
```

Determines if the specified alarm is active.

**Parameters**

| in   | alarm | alarm channel number (1..ST_LLD_NUM_ALARMS) |

**Returns**

The alarm status.

**Return values**

- `false` if the alarm is not active.
- `true` if the alarm is active

Function Class:

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

### 7.40.2.9 stStartAlarmN()

```c
void stStartAlarmN (  
    unsigned alarm,  
    systime_t abstime,  
    st_callback_t cb  )
```

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>in</th>
<th>abstime</th>
<th>the time to be set for the first alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>alarm</td>
<td>alarm channel number (1..ST_LLD_NUM_ALARMS)</td>
</tr>
<tr>
<td>in</td>
<td>cb</td>
<td>alarm callback</td>
</tr>
</tbody>
</table>

Function Class:

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

7.40.2.10  stStopAlarmN()

void stStopAlarmN (
    unsigned alarm
)

Stops an additional alarm.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Parameters

| in  | alarm          | alarm channel number (1..ST_LLD_NUM_ALARMS) |

Function Class:

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

7.40.2.11  stSetAlarmN()

void stSetAlarmN (
    unsigned alarm,
    systime_t abstime
)

Sets an additional alarm time.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>alarm</th>
<th>alarm channel number (1..ST_LL办公楼_NUM_ALARMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></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.

### 7.40.2.12 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

<table>
<thead>
<tr>
<th></th>
<th>alarm</th>
<th>alarm channel number (1..ST_LL办公楼_NUM_ALARMS)</th>
</tr>
</thead>
</table>

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.

### 7.40.2.13 st_ll办公楼_init()

```c
void st_ll办公楼_init ( 
    void )
```

Low level ST driver initialization.

Function Class:

Not an API, this function is for internal use only.
7.40.2.14  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.

7.40.2.15  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.

7.40.2.16  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.
7.40.2.17  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.

7.40.2.18  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.

7.40.2.19  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.
7.41 TRNG Driver

Generic True Random Numbers Generator Driver.

7.41.1 Detailed Description

Generic True Random Numbers Generator Driver.

This module implements a generic TRNG driver.

Precondition

In order to use the TRNG driver the HAL_USE_TRNG option must be enabled in halconf.h.

Macros

- #define trng_lld_driver_fields
  Low level fields of the TRNG driver structure.
- #define trng_lld_config_fields
  Low level fields of the TRNG configuration structure.

PLATFORM configuration options

- #define PLATFORM_TRNG_USE_TRNG1 FALSE
  TRNGD1 driver enable switch.

Typedefs

- typedef struct hal_trng_driver TRNGDriver
  Type of a structure representing a TRNG driver.
- typedef struct hal_trng_config TRNGConfig
  Driver configuration structure.

Data Structures

- struct hal_trng_config
  Driver configuration structure.
- struct hal_trng_driver
  Structure representing a TRNG driver.
Functions

- void trngInit (void)
  TRNG Driver initialization.
- void trngObjectInit (TRNGDriver ∗trngp)
  Initializes the standard part of a TRNGDriver structure.
- 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.

7.41.2 Macro Definition Documentation

7.41.2.1 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.
7.41.2.2 trng_lld_driver_fields

#define trng_lld_driver_fields

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

Low level fields of the TRNG driver structure.

7.41.2.3 trng_lld_config_fields

#define trng_lld_config_fields

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

Low level fields of the TRNG configuration structure.

7.41.3 Typedef Documentation

7.41.3.1 TRNGDriver

typedef struct hal_trng_driver TRNGDriver

Type of a structure representing a TRNG driver.

7.41.3.2 TRNGConfig

typedef struct hal_trng_config TRNGConfig

Driver configuration structure.

Note
It could be empty on some architectures.

7.41.4 Enumeration Type Documentation

7.41.4.1 trngstate_t

enum trngstate_t

Driver state machine possible states.
7.41 TRNG Driver

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRNG_UNINIT</td>
<td>Not initialized.</td>
</tr>
<tr>
<td>TRNG_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>TRNG_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>TRNG_RUNNING</td>
<td>Generating random number.</td>
</tr>
</tbody>
</table>

7.41.5 Function Documentation

7.41.5.1 trngInit()

```c
void trngInit (  
    void )
```

TRNG Driver initialization.

Note

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

Function Class:

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

Here is the call graph for this function:

```
    trngInit -> trng_lld_init -> trngObjectInit
```

7.41.5.2 trngObjectInit()

```c
void trngObjectInit (  
    TRNGDriver * trngp )
```

Initializes the standard part of a TRNGDriver structure.
Parameters

| out  | trngp  | pointer to the TRNGDriver object |

Function Class:

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

### 7.41.5.3 trngStart()

```c
void trngStart (  
    TRNGDriver * trngp,  
    const TRNGConfig * config )
```

Configures and activates the TRNG peripheral.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>trngp</th>
<th>pointer to the TRNGDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the TRNGConfig object or NULL for default configuration</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, 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`
- `osalSysUnlock`
- `trngStart`
- `trng_lld_start`

### 7.41.5.4 trngStop()

```c
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:

```
trngStop
osalSysLock
osalSysUnlock
trng_lld_stop
```

7.41.5.5 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:

![Call graph](call_graph_trngGenerate_trng_lld_generate.png)

7.41.5.6 trng_lld_init()

```c
void trng_lld_init (  
    void  )
```

Low level TRNG driver initialization.

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

Here is the call graph for this function:

![Call graph](call_graph_trng_lld_init_trngObjectInit.png)

7.41.5.7 trng_lld_start()

```c
void trng_lld_start (  
    TRNGDriver * trngp  )
```

Configures and activates the TRNG peripheral.
7.41 TRNG Driver

Parameters

| in | trngp | pointer to the TRNGDriver object |

Function Class:

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

7.41.5.8 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.

7.41.5.9 trng_lld_generate()

```c
bool trng_lld_generate ( 
    TRNGDriver * trngp, 
    size_t size, 
    uint8_t * out )
```

True random numbers generator.

Note

The function is blocking and likely performs polled waiting inside the low level implementation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>trngp</th>
<th>pointer to the TRNGDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of output buffer</td>
</tr>
<tr>
<td>out</td>
<td>out</td>
<td>output buffer</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if a random number has been generated.</td>
</tr>
<tr>
<td>true</td>
<td>if an HW error occurred.</td>
</tr>
</tbody>
</table>

Function Class:

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

7.41.6 Variable Documentation

7.41.6.1 TRNGD1

`TRNGDriver TRNGD1`

TRNGD1 driver identifier.
7.42 UART Driver

Generic UART Driver.

7.42.1 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.

7.42.2 Driver State Machine

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

The follow diagram describes the transmitter state machine, this diagram is valid while the driver is in the UART\_READY state. This state machine is automatically reset to the TX\_IDLE state each time the driver enters the UART\_READY state.

![Transmitter State Machine Diagram]

7.42.2.2 Receiver sub State Machine

The follow diagram describes the receiver state machine, this diagram is valid while the driver is in the UART\_READY state. This state machine is automatically reset to the RX\_IDLE state each time the driver enters the UART\_READY state.

![Receiver State Machine Diagram]

UART status flags

- \#define UART\_NO\_ERROR 0

   No pending conditions.
• #define UART_PARITY_ERROR 4
  Parity error happened.

• #define UART_FRAMING_ERROR 8
  Framing error happened.

• #define UART_OVERRUN_ERROR 16
  Overflow happened.

• #define UART_NOISE_ERROR 32
  Noise on the line.

• #define UART_BREAK_DETECTED 64
  Break detected.

UART error conditions
  • #define UART_ERR_NOT_ACTIVE (size_t)-1

UART configuration options
  • #define UART_USE_WAIT FALSE
    Enables synchronous APIs.
  • #define UART_USE_MUTUAL EXCLUSION FALSE
    Enables the uartAcquireBus() and uartReleaseBus() APIs.

Low level driver helper macros
  • #define _uart_wakeup_tx1_isr(uartp)
    Wakes up the waiting thread in case of early TX complete.
  • #define _uart_wakeup_tx2_isr(uartp)
    Wakes up the waiting thread in case of late TX complete.
  • #define _uart_wakeup_rx_complete_isr(uartp)
    Wakes up the waiting thread in case of RX complete.
  • #define _uart_wakeup_rx_error_isr(uartp)
    Wakes up the waiting thread in case of RX error.
  • #define _uart_wakeup_rx_cm_isr(uartp)
    Wakes up the waiting thread in case of RX character match.
  • #define _uart_wakeup_rx_timeout_isr(uartp)
    Wakes up the waiting thread in case of RX timeout.
  • #define _uart_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.
PLATFOM 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 uartStartSendl (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 uartStopSendl (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 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.

7.42.3 Macro Definition Documentation
7.42.3.1 UART_NO_ERROR

#define UART_NO_ERROR 0

No pending conditions.

7.42.3.2 UART_PARITY_ERROR

#define UART_PARITY_ERROR 4

Parity error happened.

7.42.3.3 UART_FRAMING_ERROR

#define UART_FRAMING_ERROR 8

Framing error happened.

7.42.3.4 UART_OVERRUN_ERROR

#define UART_OVERRUN_ERROR 16

Overflow happened.

7.42.3.5 UART_NOISE_ERROR

#define UART_NOISE_ERROR 32

Noise on the line.

7.42.3.6 UART_BREAK_DETECTED

#define UART_BREAK_DETECTED 64

Break detected.
7.42 UART Driver

7.42.3.7 UART_USE_WAIT

#define UART_USE_WAIT FALSE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.42.3.8 UART_USE_MUTUAL_EXCLUSION

#define UART_USE_MUTUAL_EXCLUSION FALSE

Enables the \texttt{uartAcquireBus()} and \texttt{uartReleaseBus()} APIs.

Note
Disabling this option saves both code and data space.

7.42.3.9 _uart_wakeup_tx1_isr

#define _uart_wakeup_tx1_isr(
   uartp )

\textbf{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.

\textbf{Parameters}
\begin{itemize}
   \item \textbf{in} \texttt{uartp} \texttt{pointer to the UARTDriver object}
\end{itemize}

\textbf{Function Class:}

Not an API, this function is for internal use only.
7.42.3.10 _uart_wakeup_tx2_isr

#define _uart_wakeup_tx2_isr(
    uartp )

Value:
{
    \ 
    if ((uartp)->early == false) {
        osalSysLockFromISR();
        osalThreadResumeI((uartp)->threadtx, MSG_OK);
        osalSysUnlockFromISR();
    }
}

Wakes up the waiting thread in case of late TX complete.

Parameters

| in  | uartp | pointer to the UARTDriver object |

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

7.42.3.11 _uart_wakeup_rx_complete_isr

#define _uart_wakeup_rx_complete_isr(
    uartp )

Value:
{
    osalSysLockFromISR();
    osalThreadResumeI((uartp)->threadrx, MSG_OK);
    osalSysUnlockFromISR();
}

Wakes up the waiting thread in case of RX complete.

Parameters

| in  | uartp | pointer to the UARTDriver object |

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

7.42.3.12 _uart_wakeup_rx_error_isr

#define _uart_wakeup_rx_error_isr(
    uartp )
7.42 UART Driver

Value:

```c
osalSysLockFromISR();
osalThreadResumeI(&(uartp)->threadrx, MSG_RESET);
osalSysUnlockFromISR();
```

Wakes up the waiting thread in case of RX error.

Parameters

`in uartp` pointer to the `UARTDriver` object

Function Class:

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

7.42.3.13 _uart_wakeup_rx_cm_isr

#define _uart_wakeup_rx_cm_isr(
    uartp )

Value:

```c
osalSysLockFromISR();
osalThreadResumeI(&(uartp)->threadrx, MSG_TIMEOUT);
osalSysUnlockFromISR();
```

Wakes up the waiting thread in case of RX character match.

Parameters

`in uartp` pointer to the `UARTDriver` object

Function Class:

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

7.42.3.14 _uart_wakeup_rx_timeout_isr

#define _uart_wakeup_rx_timeout_isr(
    uartp )

Value:

```c
osalSysLockFromISR();
osalThreadResumeI(&(uartp)->threadrx, MSG_TIMEOUT);
osalSysUnlockFromISR();
```

Wakes up the waiting thread in case of RX timeout.
### 7.42.3.15 _uart_tx1_isr_code

```c
#define _uart_tx1_isr_code( uartp )

Value:

\[
\begin{align*}
(uartp)->txstate &= UART_TX_COMPLETE; \\
if ( (uartp)->config->txend1_cb != NULL ) { \\
    (uartp)->config->txend1_cb(uartp); \\
} \\
if ( (uartp)->txstate == UART_TX_COMPLETE) { \\
    (uartp)->txstate = UART_TX_IDLE; \\
} \\
_uart_wakeup_tx1_isr(uartp);
\]

Common ISR code for early TX.

This code handles the portable part of the ISR code:

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

**Note**

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

### Parameters

- **in** `uartp`: pointer to the `UARTDriver` object

### Function Class:

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

### 7.42.3.16 _uart_tx2_isr_code

```c
#define _uart_tx2_isr_code( uartp )
```

This code handles the portable part of the ISR code:
Value:
{
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
in uartp pointer to the UARTDriver object

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

7.42.3.17 _uart_rx_complete_isr_code

#define _uart_rx_complete_isr_code(
uartp )

Value:
{
(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.

7.42.3.18  _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.

7.42.3.19  _uart_rx_idle_code

#define _uart_rx_idle_code(
    uartp
)
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.

7.42.3.20 _uart_timeout_isr_code

#define _uart_timeout_isr_code(
    uartp
)

Timeout ISR code for receiver.

This code handles the portable part of the ISR code:

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

Note

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

```
in | uartp | pointer to the UARTDriver object
```

Function Class:

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

### 7.42.3.21 _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.

Parameters

```
in | uartp | pointer to the UARTDriver object
```

Function Class:

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

### 7.42.3.22 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.

### 7.42.4 Typedef Documentation

#### 7.42.4.1 uartflags_t

```c
typedef uint32_t uartflags_t
```

UART driver condition flags type.

#### 7.42.4.2 UARTDriver

```c
typedef struct UARTDriver UARTDriver
```

Type of structure representing an UART driver.

#### 7.42.4.3 uartcb_t

```c
typedef void(∗ uartcb_t) (UARTDriver ∗ uartp)
```

Generic UART notification callback type.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in uartp</td>
<td>UARTDriver pointer</td>
<td>pointer to the UARTDriver object</td>
</tr>
</tbody>
</table>

#### 7.42.4.4 uartccb_t

```c
typedef void(∗ uartccb_t) (UARTDriver ∗ uartp, uint16_t c)
```

Character received UART notification callback type.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in uartp</td>
<td>UARTDriver pointer</td>
<td>pointer to the UARTDriver object triggering the callback</td>
</tr>
<tr>
<td>in c</td>
<td>uint16_t</td>
<td>received character</td>
</tr>
</tbody>
</table>
7.42.4.5  **uartcb_t**

```c
typedef void(* uartcb_t)(UARTDriver *uartp, uartflags_t e)
```

Receive error UART notification callback type.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>uartp</th>
<th>pointer to the UARTDriver object triggering the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>e</td>
<td>receive error mask</td>
</tr>
</tbody>
</table>

7.42.5  **Enumeration Type Documentation**

7.42.5.1  **uartstate_t**

```c
enum uartstate_t
```

Driver state machine possible states.

**Enumerator**

| UART_UNINIT | Not initialized. |
| UART_STOP   | Stopped.         |
| UART_READY  | Ready.          |

7.42.5.2  **uarttxstate_t**

```c
enum uarttxstate_t
```

Transmitter state machine states.

**Enumerator**

| UART_TX_IDLE | Not transmitting. |
| UART_TX_ACTIVE| Transmitting.     |
| UART_TX_COMPLETE| Buffer complete. |
7.42.5.3 uartrxstate_t

enum uartrxstate_t

Receiver state machine states.

Enumerator

<table>
<thead>
<tr>
<th>UART_RX_IDLE</th>
<th>Not receiving.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_RX_ACTIVE</td>
<td>Receiving.</td>
</tr>
<tr>
<td>UART_RX_COMPLETE</td>
<td>Buffer complete.</td>
</tr>
</tbody>
</table>

7.42.6 Function Documentation

7.42.6.1 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]

7.42.6.2 uartObjectInit()

void uartObjectInit (  
    UARTDriver * uartp  )

Initializes the standard part of a UARTDriver structure.
Parameters

| out | uartp | pointer to the UARTDriver object |

Function Class:

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

Here is the call graph for this function:

uartObjectInit → osalMutexObjectInit

7.42.6.3 uartStart()

void uartStart (UARTDriver * uartp,
               const UARTConfig * config )

Configures and activates the UART peripheral.

Parameters

| in  | uartp | pointer to the UARTDriver object |
| in  | config | pointer to the UARTConfig 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:

7.42.6.4 uartStop()

void uartStop (UARTDriver * uartp )

Deactivates the UART peripheral.

Parameters

| in  | uartp | pointer to the UARTDriver object |

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

![Call Graph Diagram]

#### 7.42.6.5 uartStartSend()

```c
void uartStartSend ( 
    UARTDriver * uarp, 
    size_t n, 
    const void * txbuf )
```

Starts a transmission on the UART peripheral.

**Note**

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uarp</code></td>
<td>pointer to the <code>UARTDriver</code> object</td>
</tr>
<tr>
<td><code>n</code></td>
<td>number of data frames to send</td>
</tr>
<tr>
<td><code>txbuf</code></td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>
7.42 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:

```
systemLock
uartStartSend
osalSysUnlock
uart_lld_start_send
```

7.42.6.6 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>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>
Module Documentation

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
```

### 7.42.6.7 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.
Function Class:

Normal API, 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
```

7.42.6.8 `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:

```
uartStopSendI -> uart_lld_stop_send
```

### 7.42.6.9 uartStartReceive()

```c
void uartStartReceive ( UARTDriver * uartp,
               size_t n,
               void * rxbuf )
```

Starts a receive operation on the UART peripheral.

**Note**

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

**Parameters**

<table>
<thead>
<tr>
<th>In</th>
<th><code>uartp</code></th>
<th>Pointer to the <code>UARTDriver</code> object</th>
</tr>
</thead>
<tbody>
<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:

![Call Graph]

### 7.42.6.10 uartStartReceive1()

```c
void uartStartReceive1 (UARTDriver * uartp, size_t n, void * rxbuf )
```

Starts a receive operation on the UART peripheral.

**Note**

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

This function has to be invoked from a lock zone.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</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 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
```

### 7.42.6.11 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
```

### 7.42.6.12 `uartStopReceiveI()`

```c
size_t uartStopReceiveI (UARTDriver * uartp)
```

Stops any ongoing receive operation.

**Note**

Stopping a receive operation also suppresses the receive callbacks.

This function has to be invoked from a lock zone.

**Parameters**

| in  | `uartp` | pointer to the `UARTDriver` object |

**Returns**

The number of data frames not received by the stopped receive operation.

**Return values**

- `UART_ERR_NOT_ACTIVE` if there was no receive operation in progress.

**Function Class:**

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

```
uartStopReceiveI  uart_lld_stop_receive
```

### 7.42.6.13  `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></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uartp</code></td>
<td>pointer to the <code>UARTDriver</code> object</td>
</tr>
<tr>
<td><code>np</code></td>
<td>number of data frames to transmit, on exit the number of frames actually transmitted</td>
</tr>
<tr>
<td><code>txbuf</code></td>
<td>the pointer to the transmit buffer</td>
</tr>
<tr>
<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.
Function Class:
Normal API, 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
```

### 7.42.6.14 uartSendFullTimeout()

```c
msg_t uartSendFullTimeout (UARTDriver *uartp, size_t *np, const void *txbuf, sysinterval_t timeout)
```

Performs a transmission on the UART peripheral.

**Note**

The function returns when the specified number of frames have been physically transmitted or on timeout.

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

This function implements a software timeout, it does not use any underlying HW timeout mechanism.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>uartp</code>  pointer to the <code>UARTDriver</code> object</td>
</tr>
<tr>
<td>in, out</td>
<td><code>np</code>  number of data frames to transmit, on exit the number of frames actually transmitted</td>
</tr>
<tr>
<td>in</td>
<td><code>txbuf</code>  the pointer to the transmit buffer</td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code> operation timeout</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the operation completed successfully.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation timed out.</td>
</tr>
</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:

```
uartSendFullTimeout
  osalSysLock
  osalThreadSuspendTimeoutS
  uart_lld_start_send
```

7.42.6.15  \texttt{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 \texttt{uint8}_t arrays for data sizes below or equal to 8 bits else it is organized as \texttt{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>uartp</th>
<th>pointer to the UARTDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>np</td>
<td>number of data frames to receive, on exit the number of frames actually received</td>
</tr>
<tr>
<td>in</td>
<td>rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
<tr>
<td>in</td>
<td>timeout</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
  |_________
  |         |
  |         |
uartReceiveTimeout
  |__________|
  |         |
  |         |
  |         |
  |         |
  |         |
  |         |
  |         |
uart_lld_start_receive
```

7.42.6.16 uartAcquireBus()

```c
void uartAcquireBus (
    UARTDriver * uartp )
```

Gains exclusive access to the UART bus.

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

Precondition

In order to use this function the option UART_USE_MUTUAL_EXCLUSION must be enabled.

Parameters

| in  | uartp | pointer to the UARTDriver object |
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
uartAcquireBus → osalMutexLock
```

### 7.42.6.17 uartReleaseBus()

```c
void uartReleaseBus (UARTDriver * uartp)
```

Releases exclusive access to the UART bus.

**Precondition**
In order to use this function the option `UART_USE_MUTUAL_EXCLUSION` must be enabled.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>uartp</code></td>
<td>pointer to the <code>UARTDriver</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:

```
uartReleaseBus → osalMutexUnlock
```
7.42.6.18  uart_lld_init()

```c
void uart_lld_init (  
    void )
```

Low level UART driver initialization.

**Function Class:**

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

Here is the call graph for this function:

```mermaid
diagram flowchart
start:uart_lld_init
    - uartObjectInit
    - osalMutexObjectInit
end
```

7.42.6.19  uart_lld_start()

```c
void uart_lld_start (  
    UARTDriver * uarp )
```

Configures and activates the UART peripheral.

**Parameters**

| in  | uarp | pointer to the UARTDriver object |

**Function Class:**

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

7.42.6.20  uart_lld_stop()

```c
void uart_lld_stop (  
    UARTDriver * uarp )
```

Deactivates the UART peripheral.
7.42.6.21 uart_lld_start_send()

```c
void uart_lld_start_send (UARTDriver * uartp,
                          size_t n,
                          const void * txbuf )
```

Starts a transmission on the UART peripheral.

**Note**

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

**Parameters**

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

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

7.42.6.22 uart_lld_stop_send()

```c
size_t uart_lld_stop_send (UARTDriver * uartp )
```

Stops any ongoing transmission.

**Note**

Stopping a transmission also suppresses the transmission callbacks.
Parameters

| in | uartp | pointer to the UARTDriver object |

Returns

The number of data frames not transmitted by the stopped transmit operation.

Function Class:

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

7.42.6.23 uart_lld_start_receive()

void uart_lld_start_receive (UARTDriver ∗ uartp, size_t n, void ∗ rxbuf)

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

| in | uartp | pointer to the UARTDriver object |
| in | n | number of data frames to send |
| out | rxbuf | the pointer to the receive buffer |

Function Class:

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

7.42.6.24 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.

7.42.7 Variable Documentation

7.42.7.1 UARTD1

UARTDriver UARTD1

UART1 driver identifier.
7.43 USB Driver

Generic USB Driver.

7.43.1 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`.

7.43.2 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).

7.43.3 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.
7.43.3.1 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.

7.43.3.2 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:
7.43.3 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`  
  Method for set address acknowledge.
- `#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW`  
  The address can be changed immediately upon packet reception.
- `#define usb_lld_get_frame_number(usbp) 0`  
  Returns the current frame number.
- `#define usb_lld_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`  
  Returns the exact size of a receive transaction.
- `#define usb_lld_connect_bus(usbp)`  
  Connects the USB device.
- `#define usb_lld_disconnect_bus(usbp)`  
  Disconnect the USB device.
- `#define usb_lld_wakeup_host(usbp)`  
  Start of host wake-up procedure.

Helper macros for USB descriptors

- `#define USB_DESC_INDEX(i) ((uint8_t)(i))`  
  Helper macro for index values into descriptor strings.
- `#define USB_DESC_BYTE(b) ((uint8_t)(b))`  
  Helper macro for byte values into descriptor strings.
- `#define USB_DESC_WORD(w)`  
  Helper macro for word values into descriptor strings.
- `#define USB_DESC_BCD(bcd)`  
  Helper macro for BCD values into descriptor strings.
- `#define USB_DESC_DEVICE_SIZE 18U`
• #define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)
  Device Descriptor helper macro.
• #define USB_DESC_CONFIGURATION_SIZE 9U
  Configuration Descriptor size.
• #define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)
  Configuration Descriptor helper macro.
• #define USB_DESC_INTERFACE_SIZE 9U
  Interface Descriptor size.
• #define USB_DESC_INTERFACE(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)
  Interface Descriptor helper macro.
• #define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U
  Interface Association Descriptor size.
• #define USB_DESC_INTERFACE_ASSOCIATION(bFirstInterface, bInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, 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, usb event callback.
- `#define _usb_isr_invoke_sof_cb(usbp)`  
  Common ISR code, SOF callback.
- `#define _usb_isr_invoke_setup_cb(usbp, ep)`  
  Common ISR code, setup packet callback.
- `#define _usb_isr_invoke_in_cb(usbp, ep)`  
  Common ISR code, IN endpoint callback.
- `#define _usb_isr_invoke_out_cb(usbp, ep)`  
  Common ISR code, OUT endpoint event.

PLATFORM configuration options

- `#define PLATFORM_USB_USE_USB1 FALSE`  
  USB driver enable switch.

Typedefs

- `typedef struct USBDriver USBDriver`  
  Type of a structure representing an USB driver.
- `typedef uint8_t usbep_t`  
  Type of an endpoint identifier.
- `typedef void(∗usbcallback_t) (USBDriver ∗usbp)`  
  Type of an USB generic notification callback.
- `typedef void(∗usbepcallback_t) (USBDriver ∗usbp, usbep_t ep)`  
  Type of an USB endpoint callback.
- `typedef void(∗usbeventcb_t) (USBDriver ∗usbp, usbevent_t event)`  
  Type of an USB event notification callback.
- `typedef bool(∗usbreqhandler_t) (USBDriver ∗usbp)`  
  Type of a requests handler callback.

Data Structures

- `struct USBDescriptor`  
  Type of an USB descriptor.
- `struct USBInEndpointState`  
  Type of an IN endpoint state structure.
- `struct USBOutEndpointState`  
  Type of an OUT endpoint state structure.
- `struct USBEndpointConfig`  
  Type of an USB endpoint configuration structure.
- `struct USBConfig`  
  Type of an USB driver configuration structure.
- `struct USBDriver`  
  Structure representing an USB driver.
Functions

- static void set_address (USBDriver ∗usbp)
  SET ADDRESS transaction callback.
- static bool default_handler (USBDriver ∗usbp)
  Standard requests handler.
- void usbnInit (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 USBEndpointCfg ∗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)
• void *usb_lld_set_address (USBDriver *usbp)
  Sets the USB address.

• void *usb_lld_init_endpoint (USBDriver *usbp, usbp_t ep)
  Enables an endpoint.

• void *usb_lld_disable_endpoints (USBDriver *usbp)
  Disables all the active endpoints except the endpoint zero.

• usbpstatus_t usb_lld_get_status_out (USBDriver *usbp, usbp_t ep)
  Returns the status of an OUT endpoint.

• usbpstatus_t usb_lld_get_status_in (USBDriver *usbp, usbp_t ep)
  Returns the status of an IN endpoint.

• void *usb_lld_read_setup (USBDriver *usbp, usbp_t ep, uint8_t *buf)
  Reads a setup packet from the dedicated packet buffer.

• void *usb_lld_prepare_receive (USBDriver *usbp, usbp_t ep)
  Prepares for a receive operation.

• void *usb_lld_prepare_transmit (USBDriver *usbp, usbp_t ep)
  Prepares for a transmit operation.

• void *usb_lld_start_out (USBDriver *usbp, usbp_t ep)
  Starts a receive operation on an OUT endpoint.

• void *usb_lld_start_in (USBDriver *usbp, usbp_t ep)
  Starts a transmit operation on an IN endpoint.

• void *usb_lld_stall_out (USBDriver *usbp, usbp_t ep)
  Brings an OUT endpoint in the stalled state.

• void *usb_lld_stall_in (USBDriver *usbp, usbp_t ep)
  Brings an IN endpoint in the stalled state.

• void *usb_lld_clear_out (USBDriver *usbp, usbp_t ep)
  Brings an OUT endpoint in the active state.

• void *usb_lld_clear_in (USBDriver *usbp, usbp_t ep)
  Brings an IN endpoint in the active state.

Enumerations

• enum usbstate_t {
  USB_UNINIT = 0, USB_STOP = 1, USB_READY = 2, USB_SELECTED = 3,
  USB_ACTIVE = 4, USB_SUSPENDED = 5 }
  Type of a driver state machine possible states.

• enum usbpstatus_t { EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE =
  2 }
  Type of an endpoint status.

• enum usbp0state_t {
  USB_EP0_STP_WAITING = 0U, USB_EP0_IN_TX = USB_IN_STATE | 1U, USB_EP0_IN_WAITING_TX0 =
  USB_IN_STATE | 2U, USB_EP0_IN_SENDING_STS = USB_IN_STATE | 3U,
  USB_EP0_OUT_WAITING_STS = USB_OUT_STATE | 4U, USB_EP0_OUT_RX = USB_OUT_STATE | 5U,
  USB_EP0_ERROR = 6U }
  Type of an endpoint zero state machine states.

• enum usbevent_t {
  USB_EVENT_RESET = 0, USB_EVENT_ADDRESS = 1, USB_EVENT_CONFIGURED = 2, USB_EVENT_UNCONFIGURED =
  3,
  USB_EVENT_SUSPEND = 4, USB_EVENT_WAKEUP = 5, USB_EVENT_STALLED = 6 }
  Type of an enumeration of the possible USB events.
Variables

• `const typedef USBDescriptor *(usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`
  Type of an USB descriptor-retrieving callback.

• `USBDriver USBD1`  
  USB1 driver identifier.

• `union {
        USBInEndpointState in 
        IN EP0 state.
        USBOutEndpointState out 
        OUT EP0 state.
    } ep0_state
  
  EP0 state.

• `static const USBEndpointConfig ep0config`  
  EP0 initialization structure.

7.43.4 Macro Definition Documentation

7.43.4.1 USB_DESC_INDEX

```c
#define USB_DESC_INDEX( i ) ((uint8_t)(i))
```

Helper macro for index values into descriptor strings.

7.43.4.2 USB_DESC_BYTE

```c
#define USB_DESC_BYTE( b ) ((uint8_t){(b)})
```

Helper macro for byte values into descriptor strings.

7.43.4.3 USB_DESC_WORD

```c
#define USB_DESC_WORD( w )

Value:
(\(\{\text{uint8_t}\}{(\text{w} \& 255U)}\),
(\(\{\text{uint8_t}\}{(\text{w} \gg 8) \& 255U)}\))
```

Helper macro for word values into descriptor strings.
7.43.4.4 USB_DESC_BCD

```c
#define USB_DESC_BCD(
    bcd)

Value:
(uint8_t)((bcd) & 255U),
(uint8_t)(((bcd) >> 8) & 255)
```

Helper macro for BCD values into descriptor strings.

7.43.4.5 USB_DESC_DEVICE

```c
#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_WODER(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.

7.43.4.6 USB_DESC_CONFIGURATION_SIZE

```c
#define USB_DESC_CONFIGURATION_SIZE 9U
```

Configuration Descriptor size.
### 7.43.4.7 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.

### 7.43.4.8 USB_DESC_INTERFACE_SIZE

```c
#define USB_DESC_INTERFACE_SIZE 9U
```

Interface Descriptor size.

### 7.43.4.9 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.
7.43.4.10 USB_DESC_INTERFACE_ASSOCIATION_SIZE

#define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U

Interface Association Descriptor size.

7.43.4.11 USB_DESC_INTERFACE_ASSOCIATION

#define USB_DESC_INTERFACE_ASSOCIATION(
  bFirstInterface,
  bInterfaceCount,
  bFunctionClass,
  bFunctionSubClass,
  bFunctionProtocol,
  iInterface
)

Value:
USB_DESC_BYTE(USB_DESC_INTERFACE_ASSOCIATION_SIZE),
USB_DESC_BYTE(USB_DESCRIPTOR_INTERFACE_ASSOCIATION),
USB_DESC_BYTE(bFirstInterface),
USB_DESC_BYTE(bInterfaceCount),
USB_DESC_BYTE(bFunctionClass),
USB_DESC_BYTE(bFunctionSubClass),
USB_DESC_BYTE(bFunctionProtocol),
USB_DESC_INDEX(iInterface)

Interface Association Descriptor helper macro.

7.43.4.12 USB_DESC_ENDPOINT_SIZE

#define USB_DESC_ENDPOINT_SIZE 7U

Endpoint Descriptor size.

7.43.4.13 USB_DESC_ENDPOINT

#define USB_DESC_ENDPOINT(
  bEndpointAddress,
  bmAttributes,
  wMaxPacketSize,
  bInterval
)

Value:
USB_DESC_BYTE(USB_DESC_ENDPOINT_SIZE),
USB_DESC_BYTE(USB_DESCRIPTOR_ENDPOINT),
USB_DESC_BYTE(bEndpointAddress),
USB_DESC_BYTE(bmAttributes),
USB_DESC_WORD(wMaxPacketSize),
USB_DESC_BYTE(bInterval)

Endpoint Descriptor helper macro.
### 7.43.4.14 USB_EP_MODE_TYPE

#define USB_EP_MODE_TYPE 0x0003U

Endpoint type mask.

### 7.43.4.15 USB_EP_MODE_TYPE_CTRL

#define USB_EP_MODE_TYPE_CTRL 0x0000U

Control endpoint.

### 7.43.4.16 USB_EP_MODE_TYPE_ISOC

#define USB_EP_MODE_TYPE_ISOC 0x0001U

Isochronous endpoint.

### 7.43.4.17 USB_EP_MODE_TYPE_BULK

#define USB_EP_MODE_TYPE_BULK 0x0002U

Bulk endpoint.

### 7.43.4.18 USB_EP_MODE_TYPE_INTR

#define USB_EP_MODE_TYPE_INTR 0x0003U

Interrupt endpoint.

### 7.43.4.19 USB_USE_WAIT

#define USB_USE_WAIT FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

### 7.43.4.20 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.

7.43.4.21 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.

7.43.4.22 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.
### 7.43.4.23 usbGetFrameNumberX

```
#define usbGetFrameNumberX(usbp) usb_lld_get_frame_number(usbp)
```

Returns the current frame number.

**Parameters**

- `in usbp` pointer to the `USBDriver` object

**Returns**

The current frame number.

**Function Class:**

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

### 7.43.4.24 usbGetTransmitStatusI

```
#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.
7.43.4.25  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

<table>
<thead>
<tr>
<th>false</th>
<th>Endpoint ready.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Endpoint receiving.</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.

7.43.4.26  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.

7.43.4.27 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 USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>pointer to a buffer for the transaction data</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to be transferred</td>
</tr>
<tr>
<td>in</td>
<td>endcb</td>
<td>callback to be invoked after the transfer or NULL</td>
</tr>
</tbody>
</table>

Function Class:

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

7.43.4.28 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

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>buffer where to copy the packet data</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.43.4.29 _usb_isr_invoke_event_cb

```
#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

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<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.

### 7.43.4.30 _usb_isr_invoke_sof_cb

```
#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.

7.43.4.31  `_usb_isr_invoke_setup_cb`

```c
#define _usb_isr_invoke_setup_cb(
    usbp,
    ep
)
```

Value:
```
{ (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.

7.43.4.32  `_usb_isr_invoke_in_cb`

```c
#define _usb_isr_invoke_in_cb(
    usbp,
    ep
)
```

Value:
```
{ (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.
7.43 USB Driver

Parameters

<table>
<thead>
<tr>
<th></th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Function Class:

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

7.43.4.33 _usb_isr_invoke_out_cb

#define _usb_isr_invoke_out_cb(
    usbp,
    ep )

Value:

{ (usbp)->receiving &= ~(1 « (ep)); \
  if ((usbp)->epc[ep]->out_cb != NULL) { \
    (usbp)->epc[ep]->out_cb(usbp, ep); \
  } \
  osalSysLockFromISR(); \
  osalThreadResumeI(&(usbp)->epc[ep]->out_state->thread, \
                   usbGetReceiveTransactionSizeX(usbp, ep)); \
  osalSysUnlockFromISR(); 
}

Common ISR code, OUT endpoint event.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Function Class:

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

7.43.4.34 USB_MAX_ENDPOINTS

#define USB_MAX_ENDPOINTS 4

Maximum endpoint address.
7.43.4.35  USB_EP0_STATUS_STAGE

#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW

Status stage handling method.

7.43.4.36  USB_SET_ADDRESS_MODE

#define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS

The address can be changed immediately upon packet reception.

7.43.4.37  USB_SET_ADDRESS_ACK_HANDLING

#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW

Method for set address acknowledge.

7.43.4.38  PLATFORM_USB_USE_USB1

#define PLATFORM_USB_USE_USB1 FALSE

USB driver enable switch.

If set to TRUE the support for USB1 is included.

Note

The default is FALSE.

7.43.4.39  usb_lld_get_frame_number

#define usb_lld_get_frame_number(   
        usbp ) 0

Returns the current frame number.

Parameters

| in | usbp | pointer to the USBDriver object |
7.43 USB Driver

Returns

The current frame number.

Function Class:

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

7.43.4.40  usb_lld_get_transaction_size

#define usb_lld_get_transaction_size(
    usbp,
    ep ) ((usbp)->epc[ep]->out_state->rxcnt)

Returns the exact size of a receive transaction.

The received size can be different from the size specified in usbStartReceiveI() because the last packet could have a size different from the expected one.

Precondition

The OUT endpoint must have been configured in transaction mode in order to use this function.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>usbp</th>
<th>pointer to the USBDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ep</td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

Returns

Received data size.

Function Class:

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

7.43.4.41  usb_lld_connect_bus

#define usb_lld_connect_bus(
    usbp )

Connects the USB device.

Function Class:

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

```c
#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.

---

**7.43.43 usb_lld_wakeup_host**

```c
#define usb_lld_wakeup_host(
    usbp )
```

Start of host wake-up procedure.

**Function Class:**

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

---

**7.43.5 Typedef Documentation**

**7.43.5.1 USBDriver**

```c
typedef struct USBDriver USBDriver
```

Type of a structure representing an USB driver.

**7.43.5.2 usbep_t**

```c
typedef uint8_t usbep_t
```

Type of an endpoint identifier.

**7.43.5.3 usbcallback_t**

```c
typedef void(* usbcallback_t) (USBDriver *usbp)
```

Type of an USB generic notification callback.
7.43 USB Driver

Parameters

| in  | usbp | pointer to the USBDriver object triggering the callback |

7.43.5.4 usbepcallback_t

typedef void(* usbepcallback_t) (USBDriver *usbp, usbep_t ep)

Type of an USB endpoint callback.

Parameters

| in  | usbp | pointer to the USBDriver object triggering the callback |
| in  | ep   | endpoint number |

7.43.5.5 usbeventcb_t

typedef void(* usbeventcb_t) (USBDriver *usbp, usbevent_t event)

Type of an USB event notification callback.

Parameters

| in  | usbp | pointer to the USBDriver object triggering the callback |
| in  | event | event type |

7.43.5.6 usbreqhandler_t

typedef bool(* usbreqhandler_t) (USBDriver *usbp)

Type of a requests handler callback.

The request is encoded in the usb_setup buffer.

Parameters

| in  | usbp | pointer to the USBDriver object triggering the callback |
Returns

The request handling exit code.

### Return values

<table>
<thead>
<tr>
<th>false</th>
<th>Request not recognized by the handler.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Request handled.</td>
</tr>
</tbody>
</table>

#### 7.43.6 Enumeration Type Documentation

##### 7.43.6.1 usbstate_t

```c
enum usbstate_t
```

Type of a driver state machine possible states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_UNINIT</td>
<td>Not initialized.</td>
</tr>
<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>

##### 7.43.6.2 usbepstatus_t

```c
enum usbepstatus_t
```

Type of an endpoint status.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP_STATUS_DISABLED</td>
<td>Endpoint not active.</td>
</tr>
<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>
7.43 USB Driver

7.43.6.3 usbep0state_t

enum usbep0state_t

Type of an endpoint zero state machine states.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_EP0_STP_WAITING</td>
<td>Waiting for SETUP data.</td>
</tr>
<tr>
<td>USB_EP0_IN_TX</td>
<td>Transmitting.</td>
</tr>
<tr>
<td>USB_EP0_IN_WAITING_TX0</td>
<td>Waiting transmit 0.</td>
</tr>
<tr>
<td>USB_EP0_IN_SENDING_STS</td>
<td>Sending status.</td>
</tr>
<tr>
<td>USB_EP0_OUT_WAITING_STS</td>
<td>Waiting status.</td>
</tr>
<tr>
<td>USB_EP0_OUT_RX</td>
<td>Receiving.</td>
</tr>
<tr>
<td>USB_EP0_ERROR</td>
<td>Error, EP0 stalled.</td>
</tr>
</tbody>
</table>

7.43.6.4 usbevent_t

enum usbevent_t

Type of an enumeration of the possible USB events.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB_EVENT_RESET</td>
<td>Driver has been reset by host.</td>
</tr>
<tr>
<td>USB_EVENT_ADDRESS</td>
<td>Address assigned.</td>
</tr>
<tr>
<td>USB_EVENT_CONFIGURED</td>
<td>Configuration selected.</td>
</tr>
<tr>
<td>USB_EVENT_UNCONFIGURED</td>
<td>Configuration removed.</td>
</tr>
<tr>
<td>USB_EVENT_SUSPEND</td>
<td>Entering suspend mode.</td>
</tr>
<tr>
<td>USB_EVENT_WAKEUP</td>
<td>Leaving suspend mode.</td>
</tr>
<tr>
<td>USB_EVENT_STALLED</td>
<td>Endpoint 0 error, stalled.</td>
</tr>
</tbody>
</table>
7.43.7 Function Documentation

7.43.7.1 set_address()

```c
static void set_address (
    USBDriver * usbp ) [static]
```

SET ADDRESS transaction callback.

Parameters

| in | usbp | pointer to the USBDriver object |

Here is the call graph for this function:

```
set_address  ->  usb_lld_set_address
```

7.43.7.2 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

| in | usbp | pointer to the USBDriver object |

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>

### 7.43.7.3 usbInit()

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:

```
  usbInit  ->  usb_lld_init  ->  usbObjectInit
```

### 7.43.7.4 usbObjectInit()

void usbObjectInit (  
   USBDriver * usbp  )

Initializes the standard part of a `USBDriver` structure.

**Parameters**

| out | usbp | pointer to the `USBDriver` object |

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

7.43.7.5 usbStart()

void usbStart (  
	USBDriver ∗ usbp,  
	const USBConfig ∗ config )

Configures and activates the USB peripheral.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in usbp</td>
<td>pointer to the USBDriver object</td>
</tr>
<tr>
<td>in 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](image)

7.43.7.6 usbStop()

void usbStop (  
	USBDriver ∗ usbp )

Deactivates the USB peripheral.
Parameters

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

Function Class:

Normal API, 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
    usbStop
    osalThreadResume
    usb_lld_stop
```

### 7.43.7.7 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

<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>
<tr>
<td>in</td>
<td>epcp</td>
<td>the endpoint configuration</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.43.7.8 `usbDisableEndpointsI()`

```c
void usbDisableEndpointsI (  
    USBDriver * usbp  )
```

Disables all the active endpoints.

This function disables all the active endpoints except the endpoint zero.

**Note**

This function must be invoked in response of a SET 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)
### 7.43.7.9  `usbStartReceive()`

```c
void usbStartReceive (    
    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**

- **in** `usbp` pointer to the `USBDriver` object
- **in** `ep` endpoint number
- **out** `buf` buffer where to copy the received data
- **in** `n` transaction size. It is recommended a multiple of the packet size because the excess is discarded.

**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)

### 7.43.7.10  `usbStartTransmit()`

```c
void usbStartTransmit (    
    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>usb</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:

```
usbStartTransmitI -|-> usb_lld_start_in
```

### 7.43.7.11 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>usb</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:

```
usbReceive
osalSysLock
osalSysUnlock
```

7.43.7.12 `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

| in | `usbp` | pointer to the `USBDriver` object |
| in | `ep` | endpoint number |
| in | `buf` | buffer where to fetch the data to be transmitted |
| in | `n` | transaction size |

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:

```
<table>
<thead>
<tr>
<th>Call Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>usbTransmit</td>
</tr>
<tr>
<td>osalSysLock</td>
</tr>
</tbody>
</table>
```

### 7.43.7.13 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
```

### 7.43.7.14 `usbStallTransmitI()`

```c
bool usbStallTransmitI ( 
    USBDriver * usbp,
    usbep_t ep )
```

Stalls an IN endpoint.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>usbp</code></th>
<th>pointer to the <code>USBDriver</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>ep</code></td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

#### Returns

The operation status.

#### Return values

| `false` | Endpoint stalled. |
| `true`  | Endpoint busy, not stalled. |
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
```

7.43.7.15  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.

7.43.7.16  _usb_reset()

```c
void _usb_reset (  
    USBDriver * usbp )
```

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.
7.43 USB Driver

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:

```
_usb_reset
  osalSysLockFromISR
osalThreadResumeI
```

7.43.7.17 _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

| in | usbp | pointer to the USBDriver object |
7.43.7.18 _usb_wakeup()

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.

7.43.7.19 _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>usbp</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.

7.43.7.20 _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>usbp</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.

7.43.7.21 _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>usbp</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.

Here is the call graph for this function:

```
osalSysLockFromISR
osalSysUnlockFromISR
usb_lld_stall_in
usb_lld_stall_out
usbStartTransmit
```  

### 7.43.7.22 usb_lld_init()

```c
void usb_lld_init (  
    void  )
```

Low level USB driver initialization.

Function Class:

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

Here is the call graph for this function:

```
usb_lld_init
usbObjectInit
```

### 7.43.7.23 usb_lld_start()

```c
void usb_lld_start (  
    USBDriver * usbp )
```

Configures and activates the USB peripheral.
7.43 USB Driver

Parameters

| in  | usbp | pointer to the USBDriver object |

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

7.43.7.24 usb_lld_stop()

```c
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.

7.43.7.25 usb_lld_reset()

```c
void usb_lld_reset (    USBDriver * usbp )
```

USB low level reset routine.

Parameters

| in  | usbp | pointer to the USBDriver object |
7.43.7.26  **usb_lld_set_address()**

```c
void usb_lld_set_address ( 
    USBDriver * usbp )
```

Sets the USB address.

**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 <strong>USBDriver</strong> object</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.43.7.27  **usb_lld_init_endpoint()**

```c
void usb_lld_init_endpoint ( 
    USBDriver * usbp,
    usbep_t ep )
```

Enables an endpoint.

**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 <strong>USBDriver</strong> 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.

7.43.7.28   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.

7.43.7.29   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 |
|     | 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.                |
7.43.7.30  usb_lld_get_status_in()

usbpstatus_t  usb_lld_get_status_in (  
    USBDriver  ∗  usbp,  
    usbep_t  ep  )

Returns the status of an IN endpoint.

Parameters

| in  | usbp   | pointer to the USBDriver object |
| in  | ep     | endpoint number |

Returns

The endpoint status.

Return values

| EP_STATUS_DISABLED | The endpoint is not active. |
| EP_STATUS_STALLED  | The endpoint is stalled.    |
| EP_STATUS_ACTIVE   | The endpoint is active.     |

Function Class:

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

7.43.7.31  usb_lld_read_setup()

void usb_lld_read_setup (  
    USBDriver  ∗  usbp,  
    usbep_t  ep,  
    uint8_t  ∗  buf  )

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the setup_cb callback in order to read the received setup packet.

Precondition

In order to use this function the endpoint must have been initialized as a control endpoint.

Postcondition

The endpoint is ready to accept another packet.
Parameters

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

### 7.43.7.32 usbd_prepare_receive()

```c
void usbd_prepare_receive ( 
    USBDriver * usbp, 
    usbep_t ep )
```

Prepares for a receive operation.

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>

Function Class:

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

### 7.43.7.33 usbd_prepare_transmit()

```c
void usbd_prepare_transmit ( 
    USBDriver * usbp, 
    usbep_t ep )
```

Prepares for a transmit operation.

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>

Function Class:

Not an API, this function is for internal use only.
7.43.7.34  **usb_lld_start_out()**

```c
void usb_lld_start_out ( 
    USBDriver ∗ usbp, 
    usbep_t ep )
```

Starts a receive operation on an OUT endpoint.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>usbp</strong></td>
<td>pointer to the <strong>USBDriver</strong> object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>ep</strong></td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.43.7.35  **usb_lld_start_in()**

```c
void usb_lld_start_in ( 
    USBDriver ∗ usbp, 
    usbep_t ep )
```

Starts a transmit operation on an IN endpoint.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>usbp</strong></td>
<td>pointer to the <strong>USBDriver</strong> object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>ep</strong></td>
<td>endpoint number</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.43.7.36  **usb_lld_stall_out()**

```c
void usb_lld_stall_out ( 
    USBDriver ∗ usbp, 
    usbep_t ep )
```

Brings an OUT endpoint in the stalled state.

**Parameters**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>usbp</strong></td>
<td>pointer to the <strong>USBDriver</strong> object</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>ep</strong></td>
<td>endpoint number</td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

7.43.7.37  usb_lld_stall_in()

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.

7.43.7.38  usb_lld_clear_out()

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.

7.43.7.39  usb_lld_clear_in()

void usb_lld_clear_in ( 
    USBDriver ∗ usbp, 
    usbep_t ep )

Brings an IN endpoint in the active state.
Parameters

| in  | usbp       | pointer to the USBDriver object |
| in  | ep         | endpoint number                 |

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

7.43.8 Variable Documentation

7.43.8.1 usbgetdescriptor_t

const typedef USBDescriptor*(*usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)

Type of an USB descriptor-retrieving callback.

7.43.8.2 USBD1

USBDriver USBD1

USB1 driver identifier.

7.43.8.3 ep0_state

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.

7.43.8.4 in

USBInEndpointState { ... } in

IN EP0 state.
7.43.8.5 out

USBOutEndpointState { ... } out

OUT EP0 state.

7.43.8.6 ep0config

const USBEndpointConfig ep0config [static]

Initial value:

= {
  USB_EP_MODE_TYPE_CTRL,
  _usb_ep0setup,
  _usb_ep0is,
  _usb_ep0out,
  0x40,
  0x40,
  &ep0_state.in,
  &ep0_state.out
}

EP0 initialization structure.
7.44 USB CDC Header

USB CDC Support Header.

7.44.1 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.*
7.45 WDG Driver

Generic WDG Driver.

7.45.1 Detailed Description

Generic WDG Driver.

This module defines an abstract interface for a watchdog timer.

Precondition

In order to use the WDG driver the HAL_USE_WDG option must be enabled in halconf.h.

Macros

- #define wdgReset(wdgp) wdg_lld_reset(wdgp)
  Resets WDG's counter.

Configuration options

- #define PLATFORM_WDG_USE_WDG1 FALSE
  WDG1 driver enable switch.

Typedefs

- typedef struct WDGDriver WDGDriver
  Type of a structure representing an WDG driver.

Data Structures

- struct WDGConfig
  Driver configuration structure.
- struct WDGDriver
  Structure representing an WDG driver.

Functions

- void wdgInit (void)
  WDG Driver initialization.
- 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.
- void wdg_lld_init (void)
  Low level WDG driver initialization.
- void wdg_lld_start (WDGDriver *wdgp)
  Configures and activates the WDG peripheral.
- void wdg_lld_stop (WDGDriver *wdgp)
  Deactivates the WDG peripheral.
- void wdg_lld_reset (WDGDriver *wdgp)
  Reloads WDG's counter.
Enumerations

- enum wdgstate_t { WDG_UNINIT = 0, WDG_STOP = 1, WDG_READY = 2 }

Driver state machine possible states.

7.45.2 Macro Definition Documentation

7.45.2.1 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.

7.45.2.2 PLATFORM_WDG_USE_WDG1

#define PLATFORM_WDG_USE_WDG1 FALSE

WDG1 driver enable switch.

Note

The default is FALSE.

7.45.3 Typedef Documentation

7.45.3.1 WDGDriver

typedef struct WDGDriver WDGDriver

Type of a structure representing an WDG driver.
7.45.4 Enumeration Type Documentation

7.45.4.1 wdgstate_t

enum 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>

7.45.5 Function Documentation

7.45.5.1 wdgInit()

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:
7.45.5.2  wdgStart()

```c
void wdgStart (  
    WDGDriver * wdgp,  
    const WDGConfig * config )
```

Configures and activates the WDG peripheral.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>wdg</th>
<th>pointer to the WDGDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config</td>
<td>pointer to the WDGConfig 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:

```
wdgStart
    osalSysLock
    osalSysUnlock
    wdg_lld_start
```

7.45.5.3  wdgStop()

```c
void wdgStop (  
    WDGDriver * wdgp )
```

Deactivates the WDG peripheral.

**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:

```
wdgStop
   ↙
osalSysLock

wdg_lld_stop
   ↘
osalSysUnlock
```

7.45.5.4 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

wdg_lld_stop
   ↘
osalSysUnlock
```
7.45.5.5  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.

7.45.5.6  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.

7.45.5.7  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.
7.45.5.8  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.
7.46 WSPI Driver

Generic WSPI Driver.

7.46.1 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)`

ChibiOS/HAL
• #define WSPI_CFG_ADDR_MODE_NONE (0LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_ONE_LINE (1LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_TWO_LINES (2LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_FOUR_LINES (3LU << 8LU)
• #define WSPI_CFG_ADDR_MODE_EIGHT_LINES (4LU << 8LU)
• #define WSPI_CFG_ADDR_DTR (1LU << 11LU)
• #define WSPI_CFG_ADDR_SIZE_MASK (3LU << 12LU)
• #define WSPI_CFG_ADDR_SIZE_8 (0LU << 12LU)
• #define WSPI_CFG_ADDR_SIZE_16 (1LU << 12LU)
• #define WSPI_CFG_ADDR_SIZE_24 (2LU << 12LU)
• #define WSPI_CFG_ADDR_SIZE_32 (3LU << 12LU)
• #define WSPI_CFG_ALT_MODE_MASK (7LU << 16LU)
• #define WSPI_CFG_ALT_MODE_NONE (0LU << 16LU)
• #define WSPI_CFG_ALT_MODE_ONE_LINE (1LU << 16LU)
• #define WSPI_CFG_ALT_MODE_TWO_LINES (2LU << 16LU)
• #define WSPI_CFG_ALT_MODE_FOUR_LINES (3LU << 16LU)
• #define WSPI_CFG_ALT_MODE_EIGHT_LINES (4LU << 16LU)
• #define WSPI_CFG_ALT_DTR (1LU << 19LU)
• #define WSPI_CFG_ALT_SIZE_MASK (3LU << 20LU)
• #define WSPI_CFG_ALT_SIZE_8 (0LU << 20LU)
• #define WSPI_CFG_ALT_SIZE_16 (1LU << 20LU)
• #define WSPI_CFG_ALT_SIZE_24 (2LU << 20LU)
• #define WSPI_CFG_ALT_SIZE_32 (3LU << 20LU)
• #define WSPI_CFG_DATA_MODE_MASK (7LU << 24LU)
• #define WSPI_CFG_DATA_MODE_NONE (0LU << 24LU)
• #define WSPI_CFG_DATA_MODE_ONE_LINE (1LU << 24LU)
• #define WSPI_CFG_DATA_MODE_TWO_LINES (2LU << 24LU)
• #define WSPI_CFG_DATA_MODE_FOUR_LINES (3LU << 24LU)
• #define WSPI_CFG_DATA_MODE_EIGHT_LINES (4LU << 24LU)
• #define WSPI_CFG_DATA_DTR (1LU << 27LU)
• #define WSPI_CFG_DQS_ENABLE (1LU << 29LU)
• #define WSPI_CFG_SIOO (1LU << 31LU)
• #define WSPI_CFG_ALL_DTR

Macro Functions

• #define wspiStartCommandI(wspip, cmdp)
  Sends a command without data phase.
• #define wspiStartSendI(wspip, cmdp, n, txbuf)
  Sends data over the WSPI bus.
• #define wspiStartReceiveI(wspip, cmdp, n, rxbuf)
  Receives data from the WSPI bus.
• #define wspiMapFlashI(wspip, cmdp, addrp) wspi_lld_map_flash(wspip, cmdp, addrp)
  Maps in memory space a WSPI flash device.
• #define wspiUnmapFlashI(wspip)
  Maps in memory space a WSPI flash device.

Low level driver helper macros

• #define _wspi_wakeup_isr(wspip, msg)
  Wakes up the waiting thread.
• #define _wspi_isr_code(wspip)
  Common ISR code.
• #define _wspi_error_code(wspip)
  Common error ISR code.
WSPI implementation capabilities

- #define WSPI_SUPPORTS_MEMMAP TRUE
- #define WSPI_DEFAULT_CFG_MASKS TRUE

Configuration options

- #define PLATFORM_WSPI_USE_WSPI1 FALSE
  WSPID1 driver enable switch.

Typedefs

- typedef struct hal_wspi_driver WSPIDriver
  Type of a structure representing an WSPI driver.
- typedef struct hal_wspi_config WSPIConfig
  Type of a structure representing an WSPI driver configuration.
- typedef void(*wspicallback_t)(WSPIDriver *wspip)
  Type of a WSPI notification callback.

Data Structures

- struct wspi_command_t
  Type of a WSPI command descriptor.
- struct hal_wspi_config
  Driver configuration structure.
- struct hal_wspi_driver
  Structure representing an WSPI driver.

Functions

- void wspiInit (void)
  WSPI Driver initialization.
- void wspiObjectInit (WSPIDriver *wspip)
  Initializes the standard part of a WSPIDriver structure.
- 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.
• void wspi_lld_init (void)
  Low level WSPI driver initialization.
• void wspi_lld_start (WSPIDriver *wspip)
  Configures and activates the WSPI peripheral.
• void wspi_lld_stop (WSPIDriver *wspip)
  Deactivates the WSPI peripheral.
• void wspi_lld_command (WSPIDriver *wspip, const wspi_command_t *cmdp)
  Sends a command without data phase.
• void wspi_lld_send (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)
  Sends a command with data over the WSPI bus.
• void wspi_lld_receive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf)
  Sends a command then receives data over the WSPI bus.
• void wspi_lld_map_flash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp)
  Maps in memory space a WSPI flash device.
• void wspi_lld_unmap_flash (WSPIDriver *wspip)
  Unmaps from memory space a WSPI flash device.

Enumerations

• enum wspistate_t {
  WSPI_UNINIT = 0, WSPI_STOP = 1, WSPI_READY = 2, WSPI_SEND = 3,
  WSPI_RECEIVE = 4, WSPI_COMPLETE = 5, WSPI_MEMMAP = 6
}
  Driver state machine possible states.

Variables

• WSPIDriver WSPID1
  WSPID1 driver identifier.

7.46.2 Macro Definition Documentation
7.46.2.1 WSPI_USE_WAIT

#define WSPI_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.46.2.2 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.

7.46.2.3 wspiStartCommandI

#define wspiStartCommandI(
  wspip,
  cmdp )

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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
</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.

7.46.2.4 wspiStartSendI

#define wspiStartSendI(
    wspip,
    cmdp,
    n,
    txbuf )

Value:

{ 
    osalDbgAssert(((cmdp)->cfg & WSPI_CFG_DATA_MODE_MASK) !=
        WSPI_CFG_DATA_MODE_NONE,
        "data mode required");
    (wspip)->state = WSPI_SEND;
    wspi_lld_send(wspip, cmdp, n, txbuf);
}

Sends data over the WSPI bus.

This asynchronous function starts a transmit operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

| in   wspip | pointer to the WSPIDriver object |
| in   cmdp  | pointer to the command descriptor |
| in   n    | number of bytes to send or zero if no data phase |
| in   txbuf| the pointer to the transmit buffer |

Function Class:

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

7.46.2.5 wspiStartReceiveI

#define wspiStartReceiveI(
    wspip,
    cmdp,
    n,
    rxbuf )

Sends data over the WSPI bus.

This asynchronous function starts a transmit operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

| in   wspip | pointer to the WSPIDriver object |
| in   cmdp  | pointer to the command descriptor |
| in   n    | number of bytes to send or zero if no data phase |
| in   rxbuf| the pointer to the receive buffer |

Function Class:

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

```c
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**

| in | wspip       | pointer to the WSPIDriver object |
| in | cmdp        | pointer to the command descriptor |
| in | n           | number of bytes to receive or zero if no data phase |
| out| rxbuf       | the pointer to the receive buffer |

**Function Class:**

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

### 7.46.2.6 wspiMapFlashI

```c
#define wspiMapFlashI(wspip, cmdp, addrp) wspi_lld_map_flash(wspip, cmdp, addrp)
```

Maps in memory space a WSPI flash device.

**Precondition**

The memory flash device must be initialized appropriately before mapping it in memory space.

**Parameters**

| in | wspip       | pointer to the WSPIDriver object |
| in | cmdp        | pointer to the command descriptor |
| out| addrp       | pointer to the memory start address of the mapped flash or NULL |
Function Class:

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

### 7.46.2.7 wspiUnmapFlashI

```c
#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.

### 7.46.2.8 _wspi_wakeup_isr

```c
#define _wspi_wakeup_isr(
    wspip, msg)
```

Value:

```c
{
    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.

7.46.2.9 _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.

7.46.2.10 _wspi_error_code

#define _wspi_error_code(
    wspip )

Value:
{    

ChibiOS/HAL
if ((wspip)->config->error_cb) {
    (wspip)->state = WSPI_COMPLETE;
    (wspip)->config->error_cb(wspip);
    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.

7.46.2.11 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.

7.46.2.12 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.
7.46.2.13 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.

7.46.3 Typedef Documentation

7.46.3.1 WSPIDriver
typedef struct hal_wspi_driver WSPIDriver

Type of a structure representing a WSPI driver.

7.46.3.2 WSPIConfig
typedef struct hal_wspi_config WSPIConfig

Type of a structure representing an WSPI driver configuration.

7.46.3.3 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 |

7.46.4 Enumeration Type Documentation

7.46.4.1 wspistate_t
enum wspistate_t

ChibiOS/HAL
Driver state machine possible states.

**Enumerator**

<table>
<thead>
<tr>
<th>WSPI_UNINIT</th>
<th>Not initialized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSPI_STOP</td>
<td>Stopped.</td>
</tr>
<tr>
<td>WSPI_READY</td>
<td>Ready.</td>
</tr>
<tr>
<td>WSPI_SEND</td>
<td>Sending data.</td>
</tr>
<tr>
<td>WSPI.Receive</td>
<td>Receiving data.</td>
</tr>
<tr>
<td>WSPI_COMPLETE</td>
<td>Asynchronous operation complete.</td>
</tr>
<tr>
<td>WSPI_MEMMAP</td>
<td>In memory mapped mode.</td>
</tr>
</tbody>
</table>

### 7.46.5 Function Documentation

#### 7.46.5.1 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:
7.46.5.2 wspiObjectInit()

void wspiObjectInit (  
    WSPIDriver * wspip )

Initializes the standard part of a WSPIDriver structure.
Parameters

| out | wspip | pointer to the WSPIDriver object |

Function Class:

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

Here is the call graph for this function:

```
    wspiObjectInit -> osalMutexObjectInit
```

### 7.46.5.3 wspiStart()

```c
void wspiStart (  
    WSPIDriver * wspip,  
    const WSPIConfig * config )
```

Configures and activates the WSPI peripheral.

Parameters

| in | wspip | pointer to the WSPIDriver object |
| in | config | pointer to the WSPIConfig object |
Function Class:

Normal API, 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]

7.46.5.4 wspiStop()

```c
void wspiStop ( WSPIDriver *wspip )
```

Deactivates the WSPI peripheral.

Note

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

Parameters

| in | wspip | pointer to the WSPIDriver object |
Function Class:

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

Here is the call graph for this function:

```
osiSysLock
wspiStop
osiSysUnlock
wspi_lld_stop
```

### 7.46.5.5 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><code>in</code></td>
<td><code>wspip</code> pointer to the WSPIDriver object</td>
</tr>
<tr>
<td><code>in</code></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
```

### 7.46.5.6 `wspiStartSend()`

```c
void wspiStartSend (  
    WSPIDriver * wspip,  
    const wspi_command_t * cmdp,  
    size_t n,  
    const uint8_t * txbuf )
```

Sends a command with data over the WSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
<td>pointer to the <code>WSPIDriver</code> object</td>
</tr>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of bytes to send</td>
</tr>
<tr>
<td>in</td>
<td>txbuf</td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
wspiStartSend
    osalSysLock
wspiStartSend
    osalSysUnlock
```

### 7.46.5.7 wspiStartReceive()

```c
void wspiStartReceive (  
    WSPIDriver * wspip,  
    const wspi_command_t * cmdp,  
    size_t n,  
    uint8_t * rxbuf)
```

Sends a command then receives data over the WSPI bus.

**Postcondition**

At the end of the operation the configured callback is invoked.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wspip</td>
</tr>
<tr>
<td>in</td>
<td>cmdp</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
</tr>
</tbody>
</table>
7.46 WSPI 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:

```
wspiStartReceive
osalSysLock
osalSysUnlock
```

7.46.5.8  wsplCommand()

```c
bool wsplCommand (  
    WSPIDriver * wspip,
    const wspl_command_t * cmdp )
```

Sends a command without data phase.

Precondition

In order to use this function the option WSPI_USE_WAIT must be enabled.

In order to use this function the driver must have been configured without callbacks (end_cb = NULL).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>wspip</th>
<th>pointer to the WSPIDriver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if the operation failed because HW issues.</td>
</tr>
</tbody>
</table>
7.46.5.9  wspiSend()

bool wspiSend (  
    WSPIDriver * wspip,  
    const wspi_command_t * cmdp,  
    size_t n,  
    const uint8_t * txbuf)

Sends a command with data over the WSPI bus.

Precondition
In order to use this function the option WSPI_USE_WAIT must be enabled.
In order to use this function the driver must have been configured without callbacks (end_cb = NULL).

Parameters

| 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 |

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.

7.46.5.10  wspiReceive()

bool wspiReceive (  
    WSPIDriver * wspip,  
    const wspi_command_t * cmdp,  
    size_t n,  
    uint8_t * rxbuf)

Sends a command then receives data over the WSPI bus.
Precondition

In order to use this function the option `WSPI_USE_WAIT` must be enabled.
In order to use this function the driver must have been configured without callbacks (`end_cb = 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>in</td>
<td>n</td>
<td>number of bytes to send</td>
</tr>
<tr>
<td>out</td>
<td>rxbuf</td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

| 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.

7.46.5.11  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 NULL</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.46.5.12 `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:

![Call Graph](call_graph_image)

7.46.5.13 `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.
7.46.5.14 wspiReleaseBus()

`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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in wspip</td>
<td>pointer to the WSPIDriver object</td>
</tr>
</tbody>
</table>

**Function Class:**

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

**Call Graph:**

```
 wspiReleaseBus  osalMutexUnlock
```

ChibiOS/HAL
7.46.5.15  \texttt{wspi\_lld\_init()}

\begin{verbatim}
void wspi_lld_init (  
    \textbf{void} )
\end{verbatim}

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:

\begin{center}
\begin{tikzpicture}
    \node (wspi_lld_init) {wspi\_lld\_init};
    \node (wspiObjectInit) [right of=wspi_lld_init] {wspiObjectInit};
    \node (osalMutexObjectInit) [right of=wspiObjectInit] {osalMutexObjectInit};
    \path[->] (wspi_lld_init) edge (wspiObjectInit);
    \path[->] (wspiObjectInit) edge (osalMutexObjectInit);
\end{tikzpicture}
\end{center}

7.46.5.16  \texttt{wspi\_lld\_start()}

\begin{verbatim}
void wspi_lld_start (  
    \textbf{WSPIDriver} \textbf{*} wspip )
\end{verbatim}

Configures and activates the WSPI peripheral.

Parameters
\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textbf{wspin} & \textbf{pointer to the WSPIDriver object} \\
\hline
\end{tabular}

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

7.46.5.17  \texttt{wspi\_lld\_stop()}

\begin{verbatim}
void wspi_lld_stop (  
    \textbf{WSPIDriver} \textbf{*} wspip )
\end{verbatim}

Deactivates the WSPI peripheral.
Parameters

| in  | wspip  | pointer to the WSPIDriver object |

Function Class:

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

7.46.5.18 wspi_lld_command()

```c
void wspi_lld_command ( 
    WSPIDriver * wspip, 
    const wspi_command_t * cmdp )
```

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

| in  | wspip  | pointer to the WSPIDriver object |
| in  | cmdp   | pointer to the command descriptor |

Function Class:

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

7.46.5.19 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>wspip</code></td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>cmdp</code></td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code></td>
<td>number of bytes to send</td>
</tr>
<tr>
<td>in</td>
<td><code>txbuf</code></td>
<td>the pointer to the transmit buffer</td>
</tr>
</tbody>
</table>

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

### 7.46.5.20 wspi_lld_receive()

```c
void wspi_lld_receive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf )
```

Sends a command then receives data over the WSPI bus.

### Postcondition
At the end of the operation the configured callback is invoked.

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>wspip</code></td>
<td>pointer to the WSPIDriver object</td>
</tr>
<tr>
<td>in</td>
<td><code>cmdp</code></td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>in</td>
<td><code>n</code></td>
<td>number of bytes to send</td>
</tr>
<tr>
<td>out</td>
<td><code>rxbuf</code></td>
<td>the pointer to the receive buffer</td>
</tr>
</tbody>
</table>

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

### 7.46.5.21 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.
7.46 WSPI Driver

Parameters

<table>
<thead>
<tr>
<th></th>
<th>wspip</th>
<th>pointer to the WSPI Driver object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmdp</td>
<td>pointer to the command descriptor</td>
</tr>
<tr>
<td>out</td>
<td>addrp</td>
<td>pointer to the memory start address of the mapped flash or NULL</td>
</tr>
</tbody>
</table>

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

7.46.5.22 wspi_lld_unmap_flash()

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

|      | wspip | pointer to the WSPI Driver object |

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

7.46.6 Variable Documentation

7.46.6.1 WSPID1

WSPIDriver WSPID1

WSPID1 driver identifier.
7.47  HAL

Hardware Abstraction Layer.

7.47.1  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.

7.47.2  HAL Device Drivers Architecture

The HAL contains several kind of modules:

- Normal Device Drivers
- Complex Device Drivers
- Interfaces
- Inner Code

7.47.3  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.
7.47.3.1 Diagram

![Diagram of Application, High Level Driver, Low Level Driver, and Microcontroller Hardware with HAL shared low level code]

7.47.4 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.

7.47.5 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.

7.47.6 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.
7.48 Configuration

HAL Configuration.

7.48.1 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.
7.48 Configuration

• #define SERIAL_DEFAULT_BITRATE 38400
  Default bit rate.
• #define SERIAL_BUFFERS_SIZE 16
  Serial buffers size.
• #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.

7.48.2 Macro Definition Documentation

7.48.2.1 HAL_USE_PAL

#define HAL_USE_PAL TRUE

Enables the PAL subsystem.

7.48.2.2 HAL_USE_ADC

#define HAL_USE_ADC TRUE

Enables the ADC subsystem.
7.48.2.3 HAL_USE_CAN

#define HAL_USE_CAN TRUE

Enables the CAN subsystem.

7.48.2.4 HAL_USE_CRY

#define HAL_USE_CRY TRUE

Enables the cryptographic subsystem.

7.48.2.5 HAL_USE_DAC

#define HAL_USE_DAC TRUE

Enables the DAC subsystem.

7.48.2.6 HAL_USE_EFL

#define HAL_USE_EFL TRUE

Enables the EFlash subsystem.

7.48.2.7 HAL_USE_GPT

#define HAL_USE_GPT TRUE

Enables the GPT subsystem.

7.48.2.8 HAL_USE_I2C

#define HAL_USE_I2C TRUE

Enables the I2C subsystem.
7.48.2.9  HAL_USE_I2S

#define HAL_USE_I2S TRUE

Enables the I2S subsystem.

7.48.2.10  HAL_USE_ICU

#define HAL_USE_ICU TRUE

Enables the ICU subsystem.

7.48.2.11  HAL_USE_MAC

#define HAL_USE_MAC TRUE

Enables the MAC subsystem.

7.48.2.12  HAL_USE_MMC_SPI

#define HAL_USE_MMC_SPI TRUE

Enables the MMC_SPI subsystem.

7.48.2.13  HAL_USE_PWM

#define HAL_USE_PWM TRUE

Enables the PWM subsystem.

7.48.2.14  HAL_USE_RTC

#define HAL_USE_RTC TRUE

Enables the RTC subsystem.
7.48.2.15  **HAL_USE_SDC**

```c
#define HAL_USE_SDC TRUE
```

Enables the SDC subsystem.

7.48.2.16  **HAL_USE_SERIAL**

```c
#define HAL_USE_SERIAL TRUE
```

Enables the SERIAL subsystem.

7.48.2.17  **HAL_USE_SERIAL_USB**

```c
#define HAL_USE_SERIAL_USB TRUE
```

Enables the SERIAL over USB subsystem.

7.48.2.18  **HAL_USE_SIO**

```c
#define HAL_USE_SIO TRUE
```

Enables the SIO subsystem.

7.48.2.19  **HAL_USE_SPI**

```c
#define HAL_USE_SPI TRUE
```

Enables the SPI subsystem.

7.48.2.20  **HAL_USE_TRNG**

```c
#define HAL_USE_TRNG TRUE
```

Enables the TRNG subsystem.
7.48 Configuration

7.48.2.21 HAL_USE_UART

#define HAL_USE_UART TRUE

Enables the UART subsystem.

7.48.2.22 HAL_USE_USB

#define HAL_USE_USB TRUE

Enables the USB subsystem.

7.48.2.23 HAL_USE_WDG

#define HAL_USE_WDG TRUE

Enables the WDG subsystem.

7.48.2.24 HAL_USE_WSPI

#define HAL_USE_WSPI TRUE

Enables the WSPI subsystem.

7.48.2.25 PAL_USE_CALLBACKS

#define PAL_USE_CALLBACKS FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.
7.48.2.26  PAL_USE_WAIT

#define PAL_USE_WAIT FALSE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.48.2.27  ADC_USE_WAIT

#define ADC_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.48.2.28  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.

7.48.2.29  CAN_USE_SLEEP_MODE

#define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

7.48.2.30  CAN_ENFORCE_USE_CALLBACKS

#define CAN_ENFORCE_USE_CALLBACKS FALSE

Enforces the driver to use direct callbacks rather than OSAL events.
7.48.2.31 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.

7.48.2.32 HAL_CRY_ENFORCE_FALLBACK

#define HAL_CRY_ENFORCE_FALLBACK FALSE

Makes the driver forcibly use the fall-back implementations.

7.48.2.33 DAC_USE_WAIT

#define DAC_USE_WAIT TRUE

Enables synchronous APIs.

Note
Disabling this option saves both code and data space.

7.48.2.34 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.
7.48.2.35 I2C_USE_MUTUAL_EXCLUSION

#define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.48.2.36 MAC_USE_ZERO_COPY

#define MAC_USE_ZERO_COPY TRUE

Enables the zero-copy API.

7.48.2.37 MAC_USE_EVENTS

#define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.48.2.38 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.

7.48.2.39 SDC_INIT_RETRY

#define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.
7.48.2.40 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.

7.48.2.41 SDC_NICE_WAITING

#define SDC_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.

7.48.2.42 SDC_INIT_OCR_V20

#define SDC_INIT_OCR_V20 0x50FF8000U

OCR initialization constant for V20 cards.

7.48.2.43 SDC_INIT_OCR

#define SDC_INIT_OCR 0x80100000U

OCR initialization constant for non-V20 cards.

7.48.2.44 SERIAL_DEFAULT_BITRATE

#define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.
7.48.2.45 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.

7.48.2.46 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.

7.48.2.47 SERIAL_USB BUFFERS NUMBER

#define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.48.2.48 SPI_USE_WAIT

#define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.
7.48.2.49  **SPI_USE_CIRCULAR**

```c
#define SPI_USE_CIRCULAR FALSE
```

Enables circular transfers APIs.

**Note**

Disabling this option saves both code and data space.

7.48.2.50  **SPI_USE_MUTUAL_EXCLUSION**

```c
#define SPI_USE_MUTUAL_EXCLUSION TRUE
```

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

**Note**

Disabling this option saves both code and data space.

7.48.2.51  **SPI_SELECT_MODE**

```c
#define SPI_SELECT_MODE SPI_SELECT_MODE_PAD
```

Handling method for SPI CS line.

**Note**

Disabling this option saves both code and data space.

7.48.2.52  **UART_USE_WAIT**

```c
#define UART_USE_WAIT TRUE
```

Enables synchronous APIs.

**Note**

Disabling this option saves both code and data space.
**7.48.2.53 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.

**7.48.2.54 USB_USE_WAIT**

#define USB_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

**7.48.2.55 WSPI_USE_WAIT**

#define WSPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

**7.48.2.56 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.
7.49 Normal Drivers

HAL Normal Drivers.

7.49.1 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.
- ECL 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.
7.50 Complex Drivers

HAL Complex Drivers.

7.50.1 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.
7.51 Interfaces and Classes

HAL Interfaces and Classes.

7.51.1 Detailed Description

HAL Interfaces and Classes.

Modules

- Blocks
- Streams
- Base Object
7.52 Inner Code

HAL Inner Code.

7.52.1 Detailed Description

HAL Inner Code.

Modules

- I/O Buffers Queues
- MMC/SD Block Device
- I/O Bytes Queues
7.53 Support Code

HAL Support Code.

7.53.1 Detailed Description

HAL Support Code.

Modules

- MII/RMII Header
  MI/RII Support Header.
- USB CDC Header
  USB CDC Support Header.
7.54 OSAL

Operating System Abstraction Layer.

7.54.1 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)((uint32_t)(secs) ∗ (uint32_t)OSAL_ST_FREQUENCY))
  Seconds to system ticks.
- #define OSAL_MS2I(msecs)
  Milliseconds to system ticks.
- #define OSAL_US2I(usecs)
  Microseconds to system ticks.

Time conversion utilities for the realtime counter

- #define OSAL_S2RTC(freq, sec) ((freq) ∗ (sec))
  Seconds to realtime counter.
- #define OSAL_MS2RTC(freq, msec) (rtcnt_t)(((freq) + 999UL) / 1000UL) ∗ (msec)
  Milliseconds to realtime counter.
- #define OSAL_US2RTC(freq, usec) (rtcnt_t)(((freq) + 999999UL) / 1000000UL) ∗ (usec)
  Microseconds to realtime counter.
Sleep macros using absolute time

- `#define osalThreadSleepSeconds(secs) osalThreadSleep(OSAL_S2I(secs))`
  Delays the invoking thread for the specified number of seconds.
- `#define osalThreadSleepMilliseconds(msecs) osalThreadSleep(OSAL_MS2I(msecs))`
  Delays the invoking thread for the specified number of milliseconds.
- `#define osalThreadSleepMicroseconds(usecs) osalThreadSleep(OSAL_US2I(usecs))`
  Delays the invoking thread for the specified number of microseconds.

Typedefs

- `typedef uint32_t syssts_t`
  Type of a system status word.
- `typedef int32_t msg_t`
  Type of a message.
- `typedef uint32_t systime_t`
  Type of system time counter.
- `typedef uint32_t sysinterval_t`
  Type of system time interval.
- `typedef uint32_t rtcnt_t`
  Type of realtime counter.
- `typedef void thread_reference_t`
  Type of a thread reference.
- `typedef uint32_t eventflags_t`
  Type of an event flags mask.
- `typedef struct event_source event_source_t`
  Type of an event flags object.
- `typedef void(eventcallback_t) (event_source_t *esp)`
  Type of an event source callback.
- `typedef uint32_t mutex_t`
  Type of a mutex.

Data Structures

- `struct event_source`
  Events source object.
- `struct threads_queue_t`
  Type of a thread queue.

Functions

- `void 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 osalThreadResumeS (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 osalThreadDequeueNextS (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up one thread from the queue, if any.
• void osalThreadDequeueAllS (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up all threads from the queue.
• void osalEventBroadcastFlagsS (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 osalTimeIsInRangeX(systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time window.
• static void osalThreadQueueObjectInit(threads_queue_t *tqp)
  Initializes a threads queue object.
• static void osalEventObjectInit(event_source_t *esp)
  Initializes an event source object.
• static void osalMutexObjectInit(mutex_t *mp)
  Initializes a mutex_t object.

Variables

• const char *osal_halt_msg
  Pointer to a halt error message.

7.54.2 Macro Definition Documentation

7.54.2.1 OSAL_ST_RESOLUTION

#define OSAL_ST_RESOLUTION 32

Size in bits of the systick_t type.

7.54.2.2 OSAL_ST_FREQUENCY

#define OSAL_ST_FREQUENCY 1000

Required systick frequency or resolution.

7.54.2.3 OSAL_ST_MODE

#define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC

Systick mode required by the underlying OS.
7.54.2.4 OSAL_IRQ_PRIORITY_LEVELS

#define OSAL_IRQ_PRIORITY_LEVELS 16U

Total priority levels.
Implementation not mandatory.

7.54.2.5 OSAL_IRQ_MAXIMUM_PRIORITY

#define OSAL_IRQ_MAXIMUM_PRIORITY 0U

Highest IRQ priority for HAL drivers.
Implementation not mandatory.

7.54.2.6 OSAL_DBG_ENABLE_ASSERTS

#define OSAL_DBG_ENABLE_ASSERTS FALSE

Enables OSAL assertions.

7.54.2.7 OSAL_DBG_ENABLE_CHECKS

#define OSAL_DBG_ENABLE_CHECKS FALSE

Enables OSAL functions parameters checks.

7.54.2.8 osalDbgAssert

#define osalDbgAssert (c, remark)

Value:

\- lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
/*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
if (OSAL_DBG_ENABLE_ASSERTS != FALSE) {
  if (!(c)) {
    /*lint -restore*/
    osalSysHalt(__func__);
    }
  }
} while (false)

Condition assertion.
If the condition check fails then the OSAL panics with a message and halts.

Note

    The condition is tested only if the OSAL_ENABLE_ASSERTIONS switch is enabled.

    The remark string is not currently used except for putting a comment in the code about the assertion.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>c</th>
<th>the condition to be verified to be true</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>remark</td>
<td>a remark string</td>
</tr>
</tbody>
</table>

Function Class:

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

7.54.2.9 osalDbgCheck

```c
#define osalDbgCheck(
    c
)
```

Value:

```c
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.

7.54.2.10 osalDbgCheckClassI

```c
#define osalDbgCheckClassI( )
```

I-Class state check.

Note

Implementation is optional.
7.54.2.11 osalDbgCheckClassS

#define osalDbgCheckClassS()

S-Class state check.

Note

Implementation is optional.

7.54.2.12 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.

7.54.2.13 OSAL_IRQ_PROLOGUE

#define OSAL_IRQ_PROLOGUE()

IRQ prologue code.
This macro must be inserted at the start of all IRQ handlers.

7.54.2.14 OSAL_IRQ_EPILOGUE

#define OSAL_IRQ_EPILOGUE()

IRQ epilogue code.
This macro must be inserted at the end of all IRQ handlers.

7.54.2.15 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
7.54.2.16 OSAL_S2I

#define OSAL_S2I(
    secs ) ((sysinterval_t)((uint32_t)(secs) * (uint32_t)OSAL_ST_FREQUENCY))

Seconds to system ticks.
Converts from seconds to system ticks number.

Note
The result is rounded upward to the next tick boundary.

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.

7.54.2.17 OSAL_MS2I

#define OSAL_MS2I(
    msecs )

Value:

{{sysinterval_t}{{((uint32_t)msecs) * 
    ((uint32_t)OSAL_ST_FREQUENCY)) - 1UL} / 1000UL} + 1UL}]

Milliseconds to system ticks.
Converts from milliseconds to system ticks number.

Note
The result is rounded upward to the next tick boundary.

Parameters

| in | msecs | number of milliseconds |
Returns

The number of ticks.

Function Class:

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

7.54.2.18 OSAL_US2I

#define OSAL_US2I(
    usecs )

Value:

\[
\frac{((\text{uint32_t})(\text{usecs}) \times (\text{uint32_t})\text{OSAL_ST_FREQUENCY}) - 1UL}{1000000UL} + 1UL
\]

Microseconds to system ticks.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

| in | usecs | number of microseconds |

Returns

The number of ticks.

Function Class:

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

7.54.2.19 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.

### 7.54.2.20 OSAL_MS2RTC

```c
#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.
### 7.54.2.21 OSAL_US2RTC

```c
#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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>clock frequency, in Hz, of the realtime counter</td>
</tr>
<tr>
<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.

### 7.54.2.22 osalThreadSleepSeconds

```c
#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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secs</td>
<td>time in seconds, must be different from 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.

### 7.54.2.23 osalThreadSleepMilliseconds

```c
#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.

### 7.54.2.24 osalThreadSleepMicroseconds

```c
#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.
7.54.3 Typedef Documentation

7.54.3.1 syssts_t

typedef uint32_t syssts_t

Type of a system status word.

7.54.3.2 msg_t

typedef int32_t msg_t

Type of a message.

7.54.3.3 systime_t

typedef uint32_t systime_t

Type of system time counter.

7.54.3.4 sysinterval_t

typedef uint32_t sysinterval_t

Type of system time interval.

7.54.3.5 rtcnt_t

typedef uint32_t rtcnt_t

Type of realtime counter.
7.54.3.6 thread_reference_t

typedef void* thread_reference_t

Type of a thread reference.

7.54.3.7 eventflags_t

typedef uint32_t eventflags_t

Type of an event flags mask.

7.54.3.8 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.

7.54.3.9 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.

7.54.3.10 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.
7.54.4 Function Documentation

7.54.4.1 osalInit()

```c
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.

7.54.4.2 osalSysHalt()

```c
void osalSysHalt (  
    const char * reason  )
```

System halt with error message.

Parameters

| in | reason | the halt message pointer |

Function Class:

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

Here is the call graph for this function:

ChibiOS/HAL
7.54.4.3 osalSysPolledDelayX()

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.

7.54.4.4 osalOsTimerHandlerI()

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.

7.54.4.5 osalOsRescheduleS()

void osalOsRescheduleS (  
    void  )

Checks if a reschedule is required and performs it.

Note
I-Class functions invoked from thread context must not reschedule by themselves, an explicit reschedule using this function is required in this scenario.
Not implemented in this simplified OSAL.

Function Class:
This is an S-Class API, this function can be invoked from within a system lock zone by threads only.
7.54.4.6 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.

7.54.4.7 osalThreadSleepS()

```c
void osalThreadSleepS ( sysinterval_t time )
```

Suspends the invoking thread for the specified time.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>time</code></th>
<th>the delay in system ticks, the special values are handled as follow:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> 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.

7.54.4.8 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>• <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> 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.

7.54.4.9 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.

7.54.4.10 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>• TIME_INFINITE the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE the thread is not enqueued and the function returns MSG_TIMEOUT as if a timeout occurred.</td>
</tr>
</tbody>
</table>

Returns

The 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.

7.54.4.11 osalThreadResumeI()

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.
7.54.4.12 osalThreadResumeS()

```c
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.

7.54.4.13 osalThreadEnqueueTimeoutS()

```c
msg_t osalThreadEnqueueTimeoutS (  
    threads_queue_t  * tqp,  
    sysinterval_t    timeout )
```

Enqueues the caller thread.

The caller thread is enqueued and put to sleep until it is dequeued or the specified timeouts expires.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>tqp</th>
<th>pointer to the threads queue object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the timeout in system ticks, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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 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.

### 7.54.4.14 osalThreadDequeueNextI()

```c
void osalThreadDequeueNextI ( 
    threads_queue_t * tqp,
    msg_t msg )
```

Dequeues and wakes up one thread from the queue, if any.

**Parameters**

<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.

### 7.54.4.15 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.
### 7.54.4.16 osalEventBroadcastFlagsI() 

```c
void osalEventBroadcastFlagsI (  
    event_source_t * esp,  
    eventflags_t flags  
)
```

Add flags to an event source object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>esp</code></td>
<td>pointer to the event flags object</td>
</tr>
<tr>
<td><code>flags</code></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.

### 7.54.4.17 osalEventBroadcastFlags() 

```c
void osalEventBroadcastFlags (  
    event_source_t * esp,  
    eventflags_t flags  
)
```

Add flags to an event source object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>esp</code></td>
<td>pointer to the event flags object</td>
</tr>
<tr>
<td><code>flags</code></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:

```
osalEventBroadcastFlagsI
osalEventBroadcastFlags
osalSysLock
osalSysUnlock
```

### 7.54.4.18 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.
7.54.4.19 osalMutexLock()

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.

7.54.4.20 osalMutexUnlock()

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.

7.54.4.21 osalSysDisable()

static void osalSysDisable (       
      void ) [inline], [static]

Disables interrupts globally.
Function Class:

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

7.54.4.22 osalSysEnable()

static void osalSysEnable (  
    void ) [inline], [static]

Enables interrupts globally.

Function Class:

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

7.54.4.23 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.

7.54.4.24 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.
7.54.4.25 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.

7.54.4.26 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.

7.54.4.27 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.
### osalSysRestoreStatusX()

```c
static void osalSysRestoreStatusX ( 
    syssts_t sts ) [inline], [static]
```

Restores the specified execution status and leaves a critical zone.

**Note**

A call to `chSchRescheduleS()` is automatically performed if exiting the critical zone and if not in ISR context.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sts</code></td>
<td>the system status to be restored.</td>
</tr>
</tbody>
</table>

**Function Class:**

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

---

### osalTimeAddX()

```c
static systime_t osalTimeAddX ( 
    systime_t systime, 
    sysinterval_t interval ) [inline], [static]
```

Adds an interval to a system time returning a system time.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>systime</code></td>
<td>base system time</td>
</tr>
<tr>
<td><code>interval</code></td>
<td>interval to be added</td>
</tr>
</tbody>
</table>

**Returns**

The new system time.

**Function Class:**

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

---

### osalTimeDiffX()

```c
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.

7.54.4.31 osalTimeIsInRangeX()

static bool osalTimeIsInRangeX (  
    systime_t time,  
    systime_t start,  
    systime_t end ) [inline], [static]

Checks if the specified time is within the specified time window.

Note

When start==end then the function returns always false because the time window has zero size.
This function can be called from any context.

Parameters

<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.
7.54.4.32 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.

7.54.4.33 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.

7.54.4.34 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.
7.54.5 Variable Documentation

7.54.5.1 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.

7.54.5.2 osal_halt_msg [2/2]

const char* osal_halt_msg

Pointer to a halt error message.

Note

The message is meant to be retrieved by the debugger after the system halt caused by an unexpected error.
8.1 BaseAsynchronousChannel Struct Reference

Base asynchronous channel class.

#include <hal_channels.h>

Inheritance diagram for BaseAsynchronousChannel:
Data Fields

- const struct BaseAsynchronousChannelVMT * vmt
  
  Virtual Methods Table.

8.1.1 Detailed Description

Base asynchronous channel class.

This class extends BaseChannel by adding event sources fields for asynchronous I/O for use in an event-driven environment.

8.1.2 Field Documentation

8.1.2.1 vmt

const struct BaseAsynchronousChannelVMT * BaseAsynchronousChannel::vmt

Virtual Methods Table.
8.2 BaseAsynchronousChannelVMT Struct Reference

BaseAsynchronousChannel virtual methods table.

#include <hal_channels.h>

Inheritance diagram for BaseAsynchronousChannelVMT:
8.2.1 Detailed Description

`BaseAsynchronousChannel` virtual methods table.

8.3 BaseBlockDevice Struct Reference

Base block device class.

#include <hal_ioblock.h>
Inheritance diagram for BaseBlockDevice:
Collaboration diagram for BaseBlockDevice:

![Collaboration Diagram]

**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.
Base channel class.

#include <hal_channels.h>

Inheritance diagram for BaseChannel:
Data Fields

- const struct BaseChannelVMT * vmt

Virtual Methods Table.

8.5.1 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.

8.5.2 Field Documentation

8.5.2.1 vmt

const struct BaseChannelVMT * BaseChannel::vmt

Virtual Methods Table.
8.6 BaseChannelVMT Struct Reference

`BaseChannel` virtual methods table.

```c
#include <hal_channels.h>
```

Inheritance diagram for BaseChannelVMT:
Collaboration diagram for BaseChannelVMT:

```
#include <hal_flash.h>
```

### 8.6.1 Detailed Description

BaseChannel virtual methods table.

### 8.7 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.

8.7.1 Detailed Description

Base flash class.

8.7.2 Field Documentation

8.7.2.1 vmt

const struct BaseFlashVMT * BaseFlash::vmt

Virtual Methods Table.
8.8 BaseFlashVMT Struct Reference

**BaseFlash** virtual methods table.

```c
#include <hal_flash.h>
```

Inheritance diagram for BaseFlashVMT:

![Inheritance Diagram](image)

Collaboration diagram for BaseFlashVMT:

![Collaboration Diagram](image)

8.8.1 Detailed Description

**BaseFlash** virtual methods table.

8.9 BaseObject Struct Reference

Base stream class.

```c
#include <hal_objects.h>
```
8.9 BaseObject Struct Reference

Inheritance diagram for BaseObject:

Collaboration diagram for BaseObject:

Data Fields

- const struct BaseObjectVMT * vmt
  
  Virtual Methods Table.

8.9.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

ChibiOS/HAL
8.9.2 Field Documentation

8.9.2.1 vmt

const struct BaseObjectVMT * BaseObject::vmt

Virtual Methods Table.

8.10 BaseObjectVMT Struct Reference

BaseObject virtual methods table.

#include <hal_objects.h>

Collaboration diagram for BaseObjectVMT:

8.10.1 Detailed Description

BaseObject virtual methods table.

8.11 BasePersistentStorage Struct Reference

Base persistent storage class.

#include <hal_persistent.h>
Inheritance diagram for BasePersistentStorage:

```
BaseObject
  + vmt

BasePersistentStorage
  + vmt
```

Collaboration diagram for BasePersistentStorage:

```
BaseObjectVMT
+ vmt

BaseObject

BasePersistentStorageVMT
+ vmt

BasePersistentStorage
```

Data Fields

- const struct BasePersistentStorageVMT * vmt

Virtual Methods Table.
8.11.1 Detailed Description

Base persistent storage class.

8.11.2 Field Documentation

8.11.2.1 vmt

```c
const struct BasePersistentStorageVMT* BasePersistentStorage::vmt
```

Virtual Methods Table.

8.12 BasePersistentStorageVMT Struct Reference

`BasePersistentStorage` virtual methods table.

```c
#include <hal_persistent.h>
```

Collaboration diagram for `BasePersistentStorageVMT`:

```
BasePersistentStorageVMT
```

8.12.1 Detailed Description

`BasePersistentStorage` virtual methods table.
8.13 BaseSequentialStream Struct Reference

Base stream class.

#include <hal_streams.h>

Inheritance diagram for BaseSequentialStream:
Data Fields

- const struct BaseSequentialStreamVMT * vmt
  
  Virtual Methods Table.

8.13.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

8.13.2 Field Documentation

8.13.2.1 vmt

const struct BaseSequentialStreamVMT * BaseSequentialStream::vmt

Virtual Methods Table.
8.14  BaseSequentialStreamVMT Struct Reference

BaseSequentialStream virtual methods table.

#include <hal_streams.h>

Inheritance diagram for BaseSequentialStreamVMT:

Collaboration diagram for BaseSequentialStreamVMT:

8.14.1  Detailed Description

BaseSequentialStream virtual methods table.
8.15 BlockDeviceInfo Struct Reference

Block device info.

#include <hal_ioblock.h>

Collaboration diagram for BlockDeviceInfo:

![Collaboration diagram for BlockDeviceInfo](image)

Data Fields

- `uint32_t blk_size
  
  Block size in bytes.

- `uint32_t blk_num
  
  Total number of blocks.

8.15.1 Detailed Description

Block device info.

8.15.2 Field Documentation

8.15.2.1 blk_size

`uint32_t BlockDeviceInfo::blk_size`

Block size in bytes.
8.15.2.2 `blk_num`

```c
uint32_t BlockDeviceInfo::blk_num
```

Total number of blocks.

8.16 **CANConfig Struct Reference**

Driver configuration structure.

```
#include <hal_can_lld.h>
```

Collaboration diagram for CANConfig:

![Collaboration diagram for CANConfig](image)

8.16.1 **Detailed Description**

Driver configuration structure.

8.17 **CANDriver Struct Reference**

Structure representing an CAN driver.

```
#include <hal_can_lld.h>
```
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.

8.17.1 Detailed Description

Structure representing an CAN driver.
8.17.2 Field Documentation

8.17.2.1 state

canstate_t CANDriver::state

Driver state.

8.17.2.2 config

const CANConfig* CANDriver::config

Current configuration data.

8.17.2.3 txqueue

threads_queue_t CANDriver::txqueue

Transmission threads queue.

8.17.2.4 rxqueue

threads_queue_t CANDriver::rxqueue

Receive threads queue.

8.17.2.5 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.
8.17.2.6 txempty_event

\texttt{event\_source\_t} \texttt{CANDriver::txempty\_event}

One or more transmission mailbox become available.

\textbf{Note}

The flags associated to the listeners will indicate which transmit mailboxes become empty.

8.17.2.7 error_event

\texttt{event\_source\_t} \texttt{CANDriver::error\_event}

A CAN bus error happened.

\textbf{Note}

The flags associated to the listeners will indicate the error(s) that have occurred.

8.17.2.8 sleep_event

\texttt{event\_source\_t} \texttt{CANDriver::sleep\_event}

Entering sleep state event.

8.17.2.9 wakeup_event

\texttt{event\_source\_t} \texttt{CANDriver::wakeup\_event}

Exiting sleep state event.
8.18 CANRxFrame Struct Reference

CAN received frame.

```
#include <hal_can_lld.h>
```

Collaboration diagram for CANRxFrame:

![Collaboration diagram](image)

**Data Fields**

- `uint8_t FMI`
  
  Filter id.

- `uint16_t TIME`
  
  Time stamp.

- `uint8_t DLC:4`
  
  Data length.

- `uint8_t RTR:1`
  
  Frame type.

- `uint8_t IDE:1`
  
  Identifier type.

- `uint32_t SID:11`
  
  Standard identifier.

- `uint32_t EID:29`
  
  Extended identifier.

- `uint8_t data8 [8]`
  
  Frame data.
8.18.1 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.

8.18.2 Field Documentation

8.18.2.1 FMI

uint8_t CANRxFrame::FMI

Filter id.

8.18.2.2 TIME

uint16_t CANRxFrame::TIME

Time stamp.

8.18.2.3 DLC

uint8_t CANRxFrame::DLC

Data length.
8.18.2.4 RTR

uint8_t CANRxFrame::RTR

Frame type.

8.18.2.5 IDE

uint8_t CANRxFrame::IDE

Identifier type.

8.18.2.6 SID

uint32_t CANRxFrame::SID

Standard identifier.

8.18.2.7 EID

uint32_t CANRxFrame::EID

Extended identifier.

8.18.2.8 data8

uint8_t CANRxFrame::data8[8]

Frame data.

8.18.2.9 data16

uint16_t CANRxFrame::data16[4]

Frame data.
8.18.2.10 data32

```c
uint32_t CANRxFrame::data32[2]
```
Frame data.

8.19 CANTxFrame Struct Reference

CAN transmission frame.

```c
#include <hal_can_lld.h>
```
Collaboration diagram for CANTxFrame:

**Data Fields**

- `uint8_t DLC:4`
  
  Data length.

- `uint8_t RTR:1`
  
  Frame type.

- `uint8_t IDE:1`
  
  Identifier type.

- `uint32_t SID:11`
  
  Standard identifier.

- `uint32_t EID:29`
  
  Extended identifier.

- `uint8_t data8 [8]`
8.19 CANTxFrame Struct Reference

Frame data.

• uint16_t data16 [4]
  Frame data.

• uint32_t data32 [2]
  Frame data.

8.19.1 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.

8.19.2 Field Documentation

8.19.2.1 DLC

uint8_t CANTxFrame::DLC

Data length.

8.19.2.2 RTR

uint8_t CANTxFrame::RTR

Frame type.

8.19.2.3 IDE

uint8_t CANTxFrame::IDE

Identifier type.
8.19.2.4 SID

uint32_t CANTxFrame::SID

Standard identifier.

8.19.2.5 EID

uint32_t CANTxFrame::EID

Extended identifier.

8.19.2.6 data8

uint8_t CANTxFrame::data8[8]

Frame data.

8.19.2.7 data16

uint16_t CANTxFrame::data16[4]

Frame data.

8.19.2.8 data32

uint32_t CANTxFrame::data32[2]

Frame data.
8.20 cdc_linecoding_t Struct Reference

Type of Line Coding structure.

```
#include <hal_usb_cdc.h>
```

Collaboration diagram for cdc_linecoding_t:

```
cdc_linecoding_t
+ dwDTERate
+ bCharFormat
+ bParityType
+ bDataBits
```

8.20.1 Detailed Description

Type of Line Coding structure.

8.21 CRYConfig Struct Reference

Driver configuration structure.

```
#include <hal_crypto_lld.h>
```

Collaboration diagram for CRYConfig:

```
CRYConfig
+ dummy
```

8.21.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.
8.22 CRYDriver Struct Reference

Structure representing an CRY driver.

```c
#include <hal_crypto_lld.h>
```

Collaboration diagram for CRYDriver:

![Collaboration Diagram](image)

Data Fields

- `crystate_t state`
  
  Driver state.

- `const CRYConfig * config`
  
  Current configuration data.

8.22.1 Detailed Description

Structure representing an CRY driver.

8.22.2 Field Documentation

8.22.2.1 state

```c
crystate_t CRYDriver::state
```

Driver state.
8.22.2.2 config

const CRYConfig* CRYDriver::config

Current configuration data.

8.23 EFlashDriverVMT Struct Reference

EFlash virtual methods table.

#include <hal_efl.h>

Inheritance diagram for EFlashDriverVMT:

![Inheritance diagram]

Collaboration diagram for EFlashDriverVMT:

![Collaboration diagram]
8.23.1 Detailed Description

EFlash virtual methods table.

8.24 event_source Struct Reference

Events source object.

#include <osal.h>

Collaboration diagram for event_source:

![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.

8.24.1 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.

8.24.2 Field Documentation
8.24.2.1 flags

volatile eventflags_t event_source::flags

Stored event flags.

8.24.2.2 cb

eventcallback_t event_source::cb

Event source callback.

8.24.2.3 param

void* event_source::param

User defined field.

8.25 FileStream Struct Reference

Base file stream class.

#include <hal_files.h>
Inheritance diagram for FileStream:
Collaboration diagram for FileStream:

![Collaboration diagram for FileStream]

**Data Fields**

- const struct FileStreamVMT * vmt
  
  Virtual Methods Table.

### 8.25.1 Detailed Description

Base file stream class.

This class represents a generic file data stream.

### 8.25.2 Field Documentation

#### 8.25.2.1 vmt

const struct FileStreamVMT* FileStream::vmt

Virtual Methods Table.
8.26 FileStreamVMT Struct Reference

FileStream virtual methods table.

#include <hal_files.h>

Inheritance diagram for FileStreamVMT:

Collaboration diagram for FileStreamVMT:

8.26.1 Detailed Description

FileStream virtual methods table.
8.27 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.

8.27.1 Detailed Description

Type of a flash device descriptor.
8.27.2 Field Documentation

8.27.2.1 attributes

`uint32_t flash_descriptor_t::attributes`

Device attributes.

8.27.2.2 page_size

`uint32_t flash_descriptor_t::page_size`

Size of write page.

8.27.2.3 sectors_count

`flash_sector_t flash_descriptor_t::sectors_count`

Number of sectors in the device.

8.27.2.4 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.

8.27.2.5 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.
8.27.6  address

uint8_t* flash_descriptor_t::address

Flash address if memory mapped or zero.

Note
Conventionally, non memory mapped devices have address NULL.

8.27.7  size

uint32_t flash_descriptor_t::size

Flash size.

8.28  flash_sector_descriptor_t Struct Reference

Flash sector descriptor.

#include <hal_flash.h>

Collaboration diagram for flash_sector_descriptor_t:

```
flash_sector_descriptor_t
+ offset
+ size
```

Data Fields

- flash_offset_t offset
  Sector offset.
- uint32_t size
  Sector size.

8.28.1  Detailed Description

Flash sector descriptor.
8.28.2 Field Documentation

8.28.2.1 offset

\texttt{flash_offset_t} \texttt{flash_sector_descriptor_t::offset}

Sector offset.

8.28.2.2 size

\texttt{uint32_t} \texttt{flash_sector_descriptor_t::size}

Sector size.

8.29 GPTConfig Struct Reference

Driver configuration structure.

\verb+#include <hal_gpt_lld.h>#+

Collaboration diagram for GPTConfig:

![Collaboration Diagram](image)

Data Fields

- \texttt{gptfreq_t} frequency
  
  \textit{Timer clock in Hz.}

- \texttt{gptcallback_t} callback
  
  \textit{Timer callback pointer.}
8.29.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.29.2 Field Documentation

8.29.2.1 frequency


gptfreq_t GPTConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.29.2.2 callback


gptcallback_t GPTConfig::callback

Timer callback pointer.

Note

This callback is invoked on GPT counter events.
8.30 GPTDriver Struct Reference

Structure representing a GPT driver.

```c
#include <hal_gpt_lld.h>
```

Collaboration diagram for GPTDriver:

![Collaboration Diagram](image)

#### Data Fields

- `gptstate_t state`
  
  Driver state.

- `const GPTConfig * config`
  
  Current configuration data.

8.30.1 Detailed Description

Structure representing a GPT driver.

8.30.2 Field Documentation

8.30.2.1 state

```c
GPTDriver::state
```

Driver state.
8.30.2.2 config

const GPTConfig* GPTDriver::config

Current configuration data.

8.31 hal_adc_config Struct Reference

Driver configuration structure.

#include <hal_adc.h>

Collaboration diagram for hal_adc_config:

8.31.1 Detailed Description

Driver configuration structure.

8.32 hal_adc_configuration_group Struct Reference

Conversion group configuration structure.

#include <hal_adc.h>
Collaboration diagram for `hal_adc_configuration_group`:

```
hal_adc_configuration_group
  + circular
  + num_channels
  + adc_lld_configuration_group_fields

hal_adc_driver
  + state
  + samples
  + depth
  + thread
  + mutex
  + adc_lld_driver_fields

hal_adc_config
  + adc_lld_config_fields
    + config

+grpp  +error_cb
+end_cb
+config
```

### 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`.

### 8.32.1 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.
8.32.2 Field Documentation

8.32.2.1 circular

`bool hal_adc_configuration_group::circular`

Enables the circular buffer mode for the group.

8.32.2.2 num_channels

`adc_channels_num_t hal_adc_configuration_group::num_channels`

Number of the analog channels belonging to the conversion group.

8.32.2.3 end_cb

`adccallback_t hal_adc_configuration_group::end_cb`

Callback function associated to the group or NULL.

8.32.2.4 error_cb

`adcerrorcallback_t hal_adc_configuration_group::error_cb`

Error callback or NULL.
#hal_adc_driver Struct Reference

Structure representing an ADC driver.

```c
#include <hal_adc.h>
```

Collaboration diagram for hal_adc_driver:

```plaintext
hal_adc_driver
+ state
+ samples
+ depth
+ thread
+ mutex
+ adc_lld_driver_fields
    + hal_adc_config
        + adc_lld_config_fields
            + config
                + error_cb
                + end_cb
                + grpp
                + circular
                + num_channels
                + adc_lld_configuration
                    + group_fields
                        + config
```

## Data Fields

- `adcstate_t state`
  
  Driver state.

- `const ADCConfig * config`
  
  Current configuration data.

- `adcsample_t * samples`
  
  Current samples buffer pointer or NULL.

- `size_t depth`
  
  Current samples buffer depth or 0.
• const ADCConversionGroup * grpp
  Current conversion group pointer or NULL.
• thread_reference_t thread
  Waiting thread.
• mutex_t mutex
  Mutex protecting the peripheral.

8.33.1 Detailed Description

Structure representing an ADC driver.

8.33.2 Field Documentation

8.33.2.1 state

adcstate_t hal_adc_driver::state

Driver state.

8.33.2.2 config

const ADCConfig* hal_adc_driver::config

Current configuration data.

8.33.2.3 samples

adcsample_t* hal_adc_driver::samples

Current samples buffer pointer or NULL.

8.33.2.4 depth

size_t hal_adc_driver::depth

Current samples buffer depth or 0.
8.33.2.5  grpp

const ADCConversionGroup* hal_adc_driver::grpp

Current conversion group pointer or NULL.

8.33.2.6  thread

thread_reference_t hal_adc_driver::thread

Waiting thread.

8.33.2.7  mutex

mutex_t hal_adc_driver::mutex

Mutex protecting the peripheral.

8.34  hal_dac_config Struct Reference

Driver configuration structure.

#include <hal_dac.h>

Collaboration diagram for hal_dac_config:

```
hal_dac_config
+ dac_pll_config_fields
```

8.34.1  Detailed Description

Driver configuration structure.
8.35 hal_dac_conversion_group Struct Reference

DAC Conversion group structure.

#include <hal_dac.h>

Collaboration diagram for hal_dac_conversion_group:

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.

8.35.1 Detailed Description

DAC Conversion group structure.

8.35.2 Field Documentation
8.35.2.1 num_channels

uint32_t hal_dac_conversion_group::num_channels

Number of DAC channels.

8.35.2.2 end_cb

daccallback_t hal_dac_conversion_group::end_cb

Operation complete callback or NULL.

8.35.2.3 error_cb

dacerrorcallback_t hal_dac_conversion_group::error_cb

Error handling callback or NULL.

8.36 hal_dac_driver Struct Reference

Structure representing a DAC driver.

#include <hal_dac.h>
Collaboration diagram for hal_dac_driver:

```
hal_dac_driver
+ state
+ samples
+ depth
+ thread
+ mutex
+ dac_lld_driver_fields

hal_dac_conversion
_group
+ ... dac_lld_conversion
_group_fields

 +error_cb
+end_cb

hal_dac_config
+ dac_lld_config_fields

 +config
 +grpp
```

**Data Fields**

- **dacstate_t state**
  
  Driver state.

- **const DACConversionGroup * grpp**
  
  Conversion group.

- **dacsamples_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.
8.36.1 Detailed Description

Structure representing a DAC driver.

8.36.2 Field Documentation

8.36.2.1 state

dacstate_t hal_dac_driver::state

Driver state.

8.36.2.2 grpp

const DACConversionGroup* hal_dac_driver::grpp

Conversion group.

8.36.2.3 samples

dacsample_t* hal_dac_driver::samples

Samples buffer pointer.

8.36.2.4 depth

size_t hal_dac_driver::depth

Samples buffer size.

8.36.2.5 config

const DACConfig* hal_dac_driver::config

Current configuration data.
8.36.2.6 thread

thread_reference_t hal_dac_driver::thread

Waiting thread.

8.36.2.7 mutex

mutex_t hal_dac_driver::mutex

Mutex protecting the bus.

8.37 hal_efl_config Struct Reference

Type of a structure representing a flash driver configuration.

#include <hal_efl.h>

Collaboration diagram for hal_efl_config:

8.37.1 Detailed Description

Type of a structure representing a flash driver configuration.
8.38 hal_efl_driver Struct Reference

Structure representing an embedded flash driver.

```c
#include <hal_efl.h>
```

Inheritance diagram for hal_efl_driver:
Collaboration diagram for hal_efl_driver:

Data Fields

- const struct EFlashDriverVMT * vmt
  
  SNORDriver Virtual Methods Table.

8.38.1 Detailed Description

Structure representing an embedded flash driver.

8.38.2 Field Documentation

8.38.2.1 vmt

const struct EFlashDriverVMT* hal_efl_driver::vmt

SNORDriver Virtual Methods Table.
8.39 hal_i2s_config Struct Reference

Driver configuration structure.

#include <hal_i2s.h>

Collaboration diagram for hal_i2s_config:

Data Fields

- const void * tx_buffer
  Transmission buffer pointer.
- void * rx_buffer
  Receive buffer pointer.
- size_t size
  TX and RX buffers size as number of samples.
- i2scallback_t end_cb
  Callback function called during streaming.

8.39.1 Detailed Description

Driver configuration structure.

8.39.2 Field Documentation
8.39.2.1 tx_buffer

const void* hal_i2s_config::tx_buffer

Transmission buffer pointer.

Note
Can be NULL if TX is not required.

8.39.2.2 rx_buffer

void* hal_i2s_config::rx_buffer

Receive buffer pointer.

Note
Can be NULL if RX is not required.

8.39.2.3 size

size_t hal_i2s_config::size

TX and RX buffers size as number of samples.

8.39.2.4 end_cb

i2scallback_t hal_i2s_config::end_cb

Callback function called during streaming.
8.40 hal_i2s_driver Struct Reference

Structure representing an I2S driver.

```c
#include <hal_i2s.h>
```

Collaboration diagram for hal_i2s_driver:

![Collaboration Diagram](image)

### Data Fields

- `i2sstate_t state`
  
  Driver state.

- `const I2SConfig *config`
  
  Current configuration data.

### 8.40.1 Detailed Description

Structure representing an I2S driver.

### 8.40.2 Field Documentation

#### 8.40.2.1 state

```c
i2sstate_t hal_i2s_driver::state
```

Driver state.
8.40.2.2 config

const I2SConfig* hal_i2s_driver::config

Current configuration data.

8.41 hal_sio_config Struct Reference

Driver configuration structure.

#include <hal_sio_lld.h>

Collaboration diagram for hal_sio_config:

Data Fields

- siocb_t rxne_cb
  Receive buffer filled callback.
- siocb_t txnf_cb
  End of transmission buffer callback.
- siocb_t txend_cb
  Physical end of transmission callback.
- sioecb_t rxevt_cb
  Receive event callback.
8.41.1 Detailed Description

Driver configuration structure.

Note
Implementations may extend this structure to contain more, architecture dependent, fields.

8.41.2 Field Documentation

8.41.2.1 rxne_cb

siocb_t hal_sio_config::rxne_cb

Receive buffer filled callback.

Note
Can be NULL.

8.41.2.2 txnf_cb

siocb_t hal_sio_config::txnf_cb

End of transmission buffer callback.

Note
Can be NULL.

8.41.2.3 txend_cb

siocb_t hal_sio_config::txend_cb

Physical end of transmission callback.

Note
Can be NULL.
8.41.2.4  rxevt_cb

sioecb_t hal_sio_config::rxevt_cb

Receive event callback.

Note

Can be NULL.

8.42  hal_sio_driver Struct Reference

Structure representing a SIO driver.

#include <hal_sio_lld.h>

Collaboration diagram for hal_sio_driver:

Data Fields

- siostate_t state
  
  Driver state.
- const SIOConfig * config
  
  Current configuration data.

8.42.1  Detailed Description

Structure representing a SIO driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.
8.42.2 Field Documentation

8.42.2.1 state

siostate_t hal_sio_driver::state

Driver state.

8.42.2.2 config

const SIOConfig* hal_sio_driver::config

Current configuration data.

8.43 hal_spi_config Struct Reference

Driver configuration structure.

#include <hal_spi.h>

Collaboration diagram for hal_spi_config:
Data Fields

- bool circular
  Enables the circular buffer mode.
- spicallback_t end_cb
  Operation complete callback or NULL.
- ioline_t ssline
  The chip select line.
- ioportid_t ssport
  The chip select port.
- ioportmask_t ssmask
  The chip select port mask.
- uint_fast8_t sspad
  The chip select pad number.

8.43.1 Detailed Description

Driver configuration structure.

8.43.2 Field Documentation

8.43.2.1 circular

bool hal_spi_config::circular

Enables the circular buffer mode.

8.43.2.2 end_cb

spicallback_t hal_spi_config::end_cb

Operation complete callback or NULL.

8.43.2.3 ssline

ioline_t hal_spi_config::ssline

The chip select line.
8.43.2.4 ssport

ioportid_t hal_spi_config::ssport

The chip select port.

8.43.2.5 ssmask

ioportmask_t hal_spi_config::ssmask

The chip select port mask.

8.43.2.6 sspad

uint_fast8_t hal_spi_config::sspad

The chip select pad number.

8.44 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.

8.44.1 Detailed Description

Structure representing an SPI driver.

8.44.2 Field Documentation

8.44.2.1 state

spistate_t hal_spi_driver::state

Driver state.

8.44.2.2 config

const SPIConfig* hal_spi_driver::config

Current configuration data.

8.44.2.3 thread

thread_reference_t hal_spi_driver::thread

Waiting thread.

8.44.2.4 mutex

mutex_t hal_spi_driver::mutex

Mutex protecting the peripheral.
8.45  **hal_trng_config Struct Reference**

Driver configuration structure.

```c
#include <hal_trng.h>
```

Collaboration diagram for hal_trng_config:

---

8.45.1  **Detailed Description**

Driver configuration structure.

8.46  **hal_trng_driver Struct Reference**

Structure representing a TRNG driver.

```c
#include <hal_trng.h>
```

Collaboration diagram for hal_trng_driver:

---
Data Fields

- `trngstate_t state`
  
  *Driver state.*

- `const TRNGConfig * config`
  
  *Current configuration data.*

### 8.46.1 Detailed Description

Structure representing a TRNG driver.

### 8.46.2 Field Documentation

#### 8.46.2.1 state

```c
trngstate_t hal_trng_driver::state
```

*Driver state.*

#### 8.46.2.2 config

```c
const TRNGConfig* hal_trng_driver::config
```

*Current configuration data.*

### 8.47 hal_wspi_config Struct Reference

Driver configuration structure.

```c
#include <hal_wspi.h>
```
Collaboration diagram for `hal_wspi_config`:

```
hal_wspi_config
  + wspi_lld_config_fields

+config +error_cb
+end_cb

hal_wspi_driver
  + state
  + thread
  + mutex
  + wspi_lld_driver_fields
```

### Data Fields

- **wspicallback_t end_cb**
  
  *Operation complete callback or NULL.*

- **wspicallback_t error_cb**

  *Operation error callback or NULL.*

### 8.47.1 Detailed Description

Driver configuration structure.

### 8.47.2 Field Documentation

#### 8.47.2.1 end_cb

```
wspicallback_t hal_wspi_config::end_cb
```

*Operation complete callback or NULL.*
### 8.47.2.2 error_cb

`wspicallback_t hal_wspi_config::error_cb`

Operation error callback or NULL.

### 8.48 hal_wspi_driver Struct Reference

Structure representing an WSPI driver.

```c
#include <hal_wspi.h>
```

Collaboration diagram for hal_wspi_driver:

![Collaboration diagram for hal_wspi_driver](#)

#### Data Fields

- `wspistate_t state`
  
  Driver state.

- `const WSPICfg * config`
  
  Current configuration data.

- `thread_reference_t thread`
  
  Waiting thread.

- `mutex_t mutex`
  
  Mutex protecting the peripheral.

### 8.48.1 Detailed Description

Structure representing an WSPI driver.
8.48.2 Field Documentation

8.48.2.1 state

wspistate_t hal_wspi_driver::state
Driver state.

8.48.2.2 config

const WSPIConfig* hal_wspi_driver::config
Current configuration data.

8.48.2.3 thread

thread_reference_t hal_wspi_driver::thread
Waiting thread.

8.48.2.4 mutex

mutex_t hal_wspi_driver::mutex
Mutex protecting the peripheral.

8.49 HMACSHA256Context Struct Reference

Type of a HMAC_SHA256 context.
#include <hal_crypto_lld.h>

Collaboration diagram for HMACSHA256Context:

```
HMACSHA256Context
+ dummy
```
8.49.1 Detailed Description

Type of a HMAC_SHA256 context.

8.50 HMACSHA512Context Struct Reference

Type of a HMAC_SHA512 context.

```c
#include <hal_crypto_lld.h>
```

Collaboration diagram for HMACSHA512Context:

![Collaboration diagram for HMACSHA512Context](image)

8.50.1 Detailed Description

Type of a HMAC_SHA512 context.

8.51 I2CConfig Struct Reference

Type of I2C driver configuration structure.

```c
#include <hal_i2c_lld.h>
```

Collaboration diagram for I2CConfig:

![Collaboration diagram for I2CConfig](image)
8.51.1 Detailed Description

Type of I2C driver configuration structure.

Note
Implementations may extend this structure to contain more, architecture dependent, fields.

8.52 I2CDriver Struct Reference

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.

8.52.1 Detailed Description

Structure representing an I2C driver.
8.52.2 Field Documentation

8.52.2.1 state

i2cstate_t I2CDriver::state

Driver state.

8.52.2.2 config

const I2CConfig* I2CDriver::config

Current configuration data.

8.52.2.3 errors

i2cflags_t I2CDriver::errors

Error flags.

8.53 ICUConfig Struct Reference

Driver configuration structure.

#include <hal_icu_lld.h>

Collaboration diagram for ICUConfig:
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.

8.53.1 Detailed Description

Driver configuration structure.

Note
It could be empty on some architectures.

8.53.2 Field Documentation

8.53.2.1 mode

icumode_t ICUConfig::mode

Driver mode.

8.53.2.2 frequency

icufreq_t ICUConfig::frequency

Timer clock in Hz.

Note
The low level can use assertions in order to catch invalid frequency specifications.
8.53.2.3 width_cb

`icucallback_t ICUConfig::width_cb`

Callback for pulse width measurement.

8.53.2.4 period_cb

`icucallback_t ICUConfig::period_cb`

Callback for cycle period measurement.

8.53.2.5 overflow_cb

`icucallback_t ICUConfig::overflow_cb`

Callback for timer overflow.

8.54 ICUDriver Struct Reference

Structure representing an ICU driver.

```
#include <hal_icu_lld.h>
```

Collaboration diagram for ICUDriver:
Data Fields

- icustate_t state
  Driver state.
- const ICUConfig * config
  Current configuration data.

8.54.1 Detailed Description

Structure representing an ICU driver.

8.54.2 Field Documentation

8.54.2.1 state

icustate_t ICUDriver::state

Driver state.

8.54.2.2 config

const ICUConfig* ICUDriver::config

Current configuration data.

8.55 io_buffers_queue Struct Reference

Structure of a generic buffers queue.

#include <hal_buffers.h>
Collaboration diagram for `io_buffers_queue`:

```
+ suspended
+ bcounter
+ bwrptr
+ brdptr
+ btop
+ bsize
+ bn
+ buffers
+ ptr
+ top
+ link
+notify
threads_queue_t
+ tr
+ waiting
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.
Boundary for R/W sequential access.

- bqnotify_t notify
  Data notification callback.
- void * link
  Application defined field.

### 8.55.1 Detailed Description

Structure of a generic buffers queue.

### 8.55.2 Field Documentation

#### 8.55.2.1 waiting

```c
threads_queue_t io_buffers_queue::waiting
```

Queue of waiting threads.

#### 8.55.2.2 suspended

```c
bool io_buffers_queue::suspended
```

Queue suspended state flag.

#### 8.55.2.3 bcounter

```c
volatile size_t io_buffers_queue::bcounter
```

Active buffers counter.

#### 8.55.2.4 bwrptr

```c
uint8_t* io_buffers_queue::bwrptr
```

Buffer write pointer.
8.55.2.5 brdptr

\texttt{uint8_t * io_buffers_queue::brdptr}

Buffer read pointer.

8.55.2.6 btop

\texttt{uint8_t * io_buffers_queue::btop}

Pointer to the buffers boundary.

8.55.2.7 bsize

\texttt{size_t io_buffers_queue::bsize}

Size of buffers.

\textbf{Note}

The buffer size must be not lower than \texttt{sizeof(size_t) + 2} because the first bytes are used to store the used size of the buffer.

8.55.2.8 bn

\texttt{size_t io_buffers_queue::bn}

Number of buffers.

8.55.2.9 buffers

\texttt{uint8_t * io_buffers_queue::buffers}

Queue of buffer objects.
8.55.2.10  ptr

```c
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.

8.55.2.11  top

```c
uint8_t* io_buffers_queue::top
```

Boundary for R/W sequential access.

8.55.2.12  notify

```c
bqnotify_t io_buffers_queue::notify
```

Data notification callback.

8.55.2.13  link

```c
void* io_buffers_queue::link
```

Application defined field.
8.56 io_queue Struct Reference

Generic I/O queue structure.

```c
#include <hal_queues.h>
```

Collaboration diagram for io_queue:

![Collaboration Diagram]

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.
8.56.1 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.

8.56.2 Field Documentation

8.56.2.1 q_waiting

threads_queue_t io_queue::q_waiting

Queue of waiting threads.

8.56.2.2 q_counter

volatile size_t io_queue::q_counter

Resources counter.

8.56.2.3 q_buffer

uint8_t* io_queue::q_buffer

Pointer to the queue buffer.

8.56.2.4 q_top

uint8_t* io_queue::q_top

Pointer to the first location after the buffer.
8.56.2.5  q_wrptr

uint8_t* io_queue::q_wrptr

Write pointer.

8.56.2.6  q_rdptr

uint8_t* io_queue::q_rdptr

Read pointer.

8.56.2.7  q_notify

qnotify_t io_queue::q_notify

Data notification callback.

8.56.2.8  q_link

void* io_queue::q_link

Application defined field.

8.57  IOBus Struct Reference

I/O bus descriptor.

#include <hal_pal.h>

Collaboration diagram for IOBus:

<table>
<thead>
<tr>
<th>IOBus</th>
<th>+ portid</th>
<th>+ mask</th>
<th>+ offset</th>
</tr>
</thead>
</table>

ChibiOS/HAL
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.

8.57.1 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.

8.57.2 Field Documentation

8.57.2.1 portid

ioportid_t IOBus::portid

Port identifier.

8.57.2.2 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.

8.57.2.3 offset

uint_fast8_t IOBus::offset

Offset, within the port, of the least significant bit of the bus.
Driver configuration structure.

```
#include <hal_mac_lld.h>
```

Collaboration diagram for MACConfig:

![Collaboration diagram for MACConfig](image)

### Data Fields

- `uint8_t * mac_address`
  
  MAC address.

### 8.58.1 Detailed Description

Driver configuration structure.

### 8.58.2 Field Documentation

#### 8.58.2.1 mac_address

```
uint8_t* MACConfig::mac_address
```

MAC address.
8.59  MACDriver Struct Reference

Structure representing a MAC driver.

#include <hal_mac_lld.h>

Collaboration diagram for MACDriver:

```
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.
```

8.59.1 Detailed Description

Structure representing a MAC driver.

8.59.2 Field Documentation
8.59.2.1 state

\texttt{macstate_t MACDriver::state}

Driver state.

8.59.2.2 config

\texttt{const MACConfig* MACDriver::config}

Current configuration data.

8.59.2.3 tdqueue

\texttt{threads_queue_t MACDriver::tdqueue}

Transmit semaphore.

8.59.2.4 rdqueue

\texttt{threads_queue_t MACDriver::rdqueue}

Receive semaphore.

8.59.2.5 rdevent

\texttt{event_source_t MACDriver::rdevent}

Receive event.

8.60 MACReceiveDescriptor Struct Reference

Structure representing a receive descriptor.

\texttt{#include <hal_mac_lld.h>}

Collaboration diagram for MACReceiveDescriptor:
Data Fields

- `size_t offset`
  
  Current read offset.

- `size_t size`
  
  Available data size.

8.60.1 Detailed Description

Structure representing a receive descriptor.

8.60.2 Field Documentation

8.60.2.1 offset

```
size_t MACReceiveDescriptor::offset
```

Current read offset.

8.60.2.2 size

```
size_t MACReceiveDescriptor::size
```

Available data size.

8.61 MACTransmitDescriptor Struct Reference

Structure representing a transmit descriptor.

```
#include <hal_mac_lld.h>
```

Collaboration diagram for MACTransmitDescriptor:
8.62 MemoryStream Struct Reference

Data Fields

- size_t offset
  
  Current write offset.

- size_t size
  
  Available space size.

8.61.1 Detailed Description

Structure representing a transmit descriptor.

8.61.2 Field Documentation

8.61.2.1 offset

size_t MACTransmitDescriptor::offset

Current write offset.

8.61.2.2 size

size_t MACTransmitDescriptor::size

Available space size.

8.62 MemoryStream Struct Reference

Memory stream object.

#include <memstreams.h>
Inheritance diagram for MemoryStream:
Collaboration diagram for MemoryStream:

Data Fields

- const struct MemStreamVMT* vmt

  Virtual Methods Table.

8.62.1 Detailed Description

Memory stream object.

8.62.2 Field Documentation

8.62.2.1 vmt

const struct MemStreamVMT* MemoryStream::vmt

Virtual Methods Table.
8.63 MemStreamVMT Struct Reference

MemStream virtual methods table.

```c
#include <memstreams.h>
```

Collaboration diagram for MemStreamVMT:

![Collaboration diagram](image1)

8.63.1 Detailed Description

MemStream virtual methods table.

8.64 mfs_bank_header_t Union Reference

Type of a bank header.

```c
#include <hal_mfs.h>
```

Collaboration diagram for mfs_bank_header_t:

![Collaboration diagram](image2)
8.64.1 Detailed Description

Type of a bank header.

Note
The header resides in the first 16 bytes of a bank.

8.64.2 Field Documentation

8.64.2.1 magic1

```c
uint32_t mfs_bank_header_t::magic1
```

Bank magic 1.

8.64.2.2 magic2

```c
uint32_t mfs_bank_header_t::magic2
```

Bank magic 2.

8.64.2.3 counter

```c
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.

8.64.2.4 reserved1

```c
uint16_t mfs_bank_header_t::reserved1
```

Reserved field.
8.64.2.5 crc

`uint16_t mfs_bank_header_t::crc`

Header CRC.

8.65 mfs_data_header_t Union Reference

Type of a data block header.

```
#include <hal_mfs.h>
```

Collaboration diagram for mfs_data_header_t:

```
+ magic1
+ magic2
+ id
+ crc
+ size
+ fields
+ hdr8
+ hdr32
```

8.65.1 Detailed Description

Type of a data block header.

This structure is placed before each written data block.

8.65.2 Field Documentation

8.65.2.1 magic1

`uint32_t mfs_data_header_t::magic1`

Data header magic 1.
8.65.2.2 magic2

uint32_t mfs_data_header_t::magic2

Data header magic 2.

8.65.2.3 id

uint16_t mfs_data_header_t::id

Record identifier.

8.65.2.4 crc

uint16_t mfs_data_header_t::crc

Data CRC.

8.65.2.5 size

uint32_t mfs_data_header_t::size

Data size.

Note

The next record is located at MFS_ALIGN_NEXT(size).

8.66 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.*

8.66.1 Detailed Description

Type of a buffered write/erase operation within a transaction.

8.66.2 Field Documentation

8.66.2.1 offset

```c
flash_offset_t mfs_transaction_op_t::offset
```

*Written header offset.*

8.66.2.2 size

```c
size_t mfs_transaction_op_t::size
```

*Written data size.*

8.66.2.3 id

```c
mfs_id_t mfs_transaction_op_t::id
```

*Record identifier.*
Type of a MFS configuration structure.

```c
#include <hal_mfs.h>
```

Collaboration diagram for MFSConfig:

### 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.*
8.67.1 Detailed Description

Type of a MFS configuration structure.

8.67.2 Field Documentation

8.67.2.1 flashp

\texttt{BaseFlash* MFSConfig::flashp}

Flash driver associated to this MFS instance.

8.67.2.2 erased

\texttt{uint32_t MFSConfig::erased}

Erased value.

8.67.2.3 bank_size

\texttt{flash_offset_t MFSConfig::bank_size}

Banks size.

8.67.2.4 bank0_start

\texttt{flash_sector_t MFSConfig::bank0_start}

Base sector index for bank 0.

8.67.2.5 bank0_sectors

\texttt{flash_sector_t MFSConfig::bank0_sectors}

Number of sectors for bank 0.

Note

The total size of bank0 sectors must be greater or equal to \texttt{bank\_size}.
8.67.2.6 bank1_start

`flash_sector_t MFSConfig::bank1_start`

Base sector index for bank 1.

8.67.2.7 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`.

8.68 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.

8.68.1 Detailed Description

Type of an MFS instance.

8.68.2 Field Documentation

8.68.2.1 state

`mfs_state_t MFSDriver::state`

Driver state.
## 8.68 MFSDriver Struct Reference

### 8.68.2.2 config

const MFSConfig* MFSDriver::config

Current configuration data.

### 8.68.2.3 current_bank

mfs_bank_t MFSDriver::current_bank

Bank currently in use.

### 8.68.2.4 current_counter

uint32_t MFSDriver::current_counter

Usage counter of the current bank.

### 8.68.2.5 next_offset

flash_offset_t MFSDriver::next_offset

Pointer to the next free position in the current bank.

### 8.68.2.6 used_space

flash_offset_t MFSDriver::used_space

Used space in the current bank without considering erased records.

### 8.68.2.7 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.
8.68.2.8  tr_next_offset

flash_offset_t  MFSDriver::tr_next_offset

Next write offset for current transaction.

8.68.2.9  tr_limit_offset

flash_offset_t  MFSDriver::tr_limit_offset

Maximum offset for the transaction.

8.68.2.10  tr_nops

uint32_t  MFSDriver::tr_nops

Number of buffered operations in current transaction.

8.68.2.11  tr_ops

mfs_transaction_op_t  MFSDriver::tr_ops[MFS_CFG_TRANSACTION_MAX]

Buffered operations in current transaction.

8.68.2.12  buffer

union { ... }  MFSDriver::buffer

Transient buffer.
MMC/SD over SPI driver configuration structure.

#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.

8.69.1 Detailed Description

MMC/SD over SPI driver configuration structure.
8.69.2 Field Documentation

8.69.2.1 spip

`SPIDriver* MMCConfig::spip`

SPI driver associated to this MMC driver.

8.69.2.2 lscfg

`const SPIConfig* MMCConfig::lscfg`

SPI low speed configuration used during initialization.

8.69.2.3 hscfg

`const SPIConfig* MMCConfig::hscfg`

SPI high speed configuration used during transfers.

8.70 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.

8.70.1 Detailed Description

Structure representing a MMC/SD over SPI driver.

8.70.2 Field Documentation
**8.70.2.1 vmt**

const struct MMCDriverVMT* MMCDriver::vmt

Virtual Methods Table.

**8.70.2.2 config**

const _mmcsd_block_device_data MMCConfig* MMCDriver::config

Current configuration data.

**8.70.2.3 block_addresses**

bool MMCDriver::block_addresses

Addresses use blocks instead of bytes.

---

### 8.71 MMCDriverVMT Struct Reference

**MMCDriver** virtual methods table.

```c
#include <hal_mmc_spi.h>
```

Inheritance diagram for MMCDriverVMT:
8.71.1 Detailed Description

**MMCDriver** virtual methods table.

8.72 **MMCSDBlockDevice Struct Reference**

MCC/SD block device class.

```c
#include <hal_mmcsd.h>
```
Inheritance diagram for MMCSDBlockDevice:
Collaboration diagram for MMCSDBlockDevice:

Data Fields

- const struct MMCSDBlockDeviceVMT * vmt
  Virtual Methods Table.

8.72.1 Detailed Description

MCC/SD block device class.

This class represents a, block-accessible, MMC/SD device.

8.72.2 Field Documentation

8.72.2.1 vmt

const struct MMCSDBlockDeviceVMT * MMCSDBlockDevice::vmt

Virtual Methods Table.
# include <hal_mmcsd.h>

Inheritance diagram for MMCSDBlockDeviceVMT:

Collaboration diagram for MMCSDBlockDeviceVMT:
8.73.1 Detailed Description

`MMCSDBlockDevice` virtual methods table.

8.74 NullStream Struct Reference

Null stream object.

```
#include <nullstreams.h>
```

Inheritance diagram for NullStream:
Collaboration diagram for NullStream:

```
BaseObject
    BaseSequentialStreamVMT
        NullStreamVMT
            NullStream
```

Data Fields

- const struct NullStreamVMT * vmt
  
  Virtual Methods Table.

8.74.1 Detailed Description

Null stream object.

8.74.2 Field Documentation

8.74.2.1 vmt

const struct NullStreamVMT* NullStream::vmt

Virtual Methods Table.
8.75 NullStreamVMT Struct Reference

**NullStream** virtual methods table.

```c
#include <nullstreams.h>
```

Collaboration diagram for NullStreamVMT:

![Collaboration diagram for NullStreamVMT](image)

8.75.1 Detailed Description

**NullStream** virtual methods table.

8.76 PALConfig Struct Reference

Generic I/O ports static initializer.

```c
#include <hal_pal_lld.h>
```

Collaboration diagram for PALConfig:

![Collaboration diagram for PALConfig](image)

8.76.1 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.
8.77 palevent_t Struct Reference

Type of a PAL event record.

#include <hal_pal.h>

Collaboration diagram for palevent_t:

```
Data Fields

- threads_queue_t threads
  Threads queued for an event.
- palcallback_t cb
  Event callback.
- void * arg
  Event callback argument.

8.77.1 Detailed Description

Type of a PAL event record.

8.77.2 Field Documentation

8.77.2.1 threads

threads_queue_t palevent_t::threads

Threads queued for an event.
8.77.2.2 cb

`palcallback_t palevent_t::cb`

Event callback.

8.77.2.3 arg

`void* palevent_t::arg`

Event callback argument.

8.78 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.

8.78.1 Detailed Description

Type of a PWM driver channel configuration structure.

8.78.2 Field Documentation

8.78.2.1 mode

`pwmmode_t` `PWMChannelConfig::mode`

Channel active logic level.

8.78.2.2 callback

`pwmcallback_t` `PWMChannelConfig::callback`

Channel callback pointer.

Note

This callback is invoked on the channel compare event. If set to `NULL` then the callback is disabled.
8.79  PWMConfig Struct Reference

Type of a PWM driver configuration structure.

```c
#include <hal_pwm_lld.h>
```

Collaboration diagram for PWMConfig:

![Collaboration Diagram](image)

### Data Fields

- `uint32_t frequency`
  
  *Timer clock in Hz.*

- `pwmcnt_t period`
  
  *PWM period in ticks.*

- `pwmcallback_t callback`
  
  *Periodic callback pointer.*

- `PWMChannelConfig channels [PWM_CHANNELS]`
  
  *Channels configurations.*

#### 8.79.1 Detailed Description

Type of a PWM driver configuration structure.
8.79.2 Field Documentation

8.79.2.1 frequency

uint32_t PWMConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.79.2.2 period

pwmcnt_t PWMConfig::period

PWM period in ticks.

Note

The low level can use assertions in order to catch invalid period specifications.

8.79.2.3 callback

pwmcallback_t PWMConfig::callback

Periodic callback pointer.

Note

This callback is invoked on PWM counter reset. If set to NULL then the callback is disabled.

8.79.2.4 channels

PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]

Channels configurations.
8.80 PWMDriver Struct Reference

Structure representing a PWM driver.

```c
#include <hal_pwm_lld.h>
```

Collaboration diagram for PWMDriver:

![Collaboration Diagram](image)

**Data Fields**

- `pwmstate_t state`
  
  `Driver state.`

- `const PWMConfig * config`
  
  `Current driver configuration data.`

- `pwmcnt_t period`
  
  `Current PWM period in ticks.`

- `pwmchnmsk_t enabled`
  
  `Mask of the enabled channels.`

- `pwmchannel_t channels`
  
  `Number of channels in this instance.`
8.80 PWMDriver Struct Reference

8.80.1 Detailed Description

Structure representing a PWM driver.

8.80.2 Field Documentation

8.80.2.1 state

$pwmstate_t$ PWMDriver::state

Driver state.

8.80.2.2 config

$const PWMConfig *$ PWMDriver::config

Current driver configuration data.

8.80.2.3 period

$pwmcnt_t$ PWMDriver::period

Current PWM period in ticks.

8.80.2.4 enabled

$pwmchnmsk_t$ PWMDriver::enabled

Mask of the enabled channels.

8.80.2.5 channels

$pwmchannel_t$ PWMDriver::channels

Number of channels in this instance.
8.81  RTCAlarm Struct Reference

Type of a structure representing an RTC alarm time stamp.

#include <hal_rtc_lld.h>

Collaboration diagram for RTCAlarm:

8.81.1 Detailed Description

Type of a structure representing an RTC alarm time stamp.

8.82  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.

### 8.82.1 Detailed Description

Type of a structure representing an RTC date/time stamp.

### 8.82.2 Field Documentation

#### 8.82.2.1 year

```
uint32_t RTCDateTime::year
```

Years since 1980.

#### 8.82.2.2 month

```
uint32_t RTCDateTime::month
```

Months 1..12.
8.82.2.3 dstflag

uint32_t RTCDateTime::dstflag

DST correction flag.

8.82.2.4 dayofweek

uint32_t RTCDateTime::dayofweek

Day of week 1..7.

8.82.2.5 day

uint32_t RTCDateTime::day

Day of the month 1..31.

8.82.2.6 millisecond

uint32_t RTCDateTime::millisecond

Milliseconds since midnight.

8.83 RTCDriver Struct Reference

Structure representing an RTC driver.

#include <hal_rtc.h>
Data Fields

- const struct `RTCDriverVMT` *vmt

  Virtual Methods Table.

### 8.83.1 Detailed Description

Structure representing an RTC driver.

### 8.83.2 Field Documentation

#### 8.83.2.1 vmt

```
const struct `RTCDriverVMT`* RTCDriver::vmt
```

Virtual Methods Table.
8.84 RTCDriverVMT Struct Reference

RTCDriver virtual methods table.

#include <hal_rtc.h>

Inheritance diagram for RTCDriverVMT:
Collaboration diagram for RTCDriverVMT:

```
BaseSequentialStream  FileStreamVMT
                  +vmt

FileStream

RTCDriverVMT
```

Additional Inherited Members

8.84.1 Detailed Description

RTCDriver virtual methods table.

8.85 SDCConfig Struct Reference

Driver configuration structure.
#include <hal_sdc_lld.h>

Collaboration diagram for SDCConfig:

```
SDCConfig
+ bus_width
```
Data Fields

- `sdcbusmode_t bus_width`

  Bus width.

8.85.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.85.2 Field Documentation

8.85.2.1 bus_width

`sdcbusmode_t SDCConfig::bus_width`

Bus width.

8.86 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.

8.86.1 Detailed Description

Structure representing an SDC driver.
8.86.2 Field Documentation

8.86.2.1 vmt

```c
const struct SDCDriverVMT* SDCDriver::vmt
```

Virtual Methods Table.

8.86.2.2 config

```c
const _mmcsd_block_device_data SDCConfig* SDCDriver::config
```

Current configuration data.

8.86.2.3 cardmode

```c
sdcmode_t SDCDriver::cardmode
```

Various flags regarding the mounted card.

8.86.2.4 errors

```c
sdcflags_t SDCDriver::errors
```

Errors flags.

8.86.2.5 rca

```c
uint32_t SDCDriver::rca
```

Card RCA.

8.86.2.6 buf

```c
uint8_t SDCDriver::buf[MMCSD_BLOCK_SIZE]
```

Buffer for internal operations.
SDCDriver virtual methods table.

#include <hal_sdc_lld.h>

Inheritance diagram for SDCDriverVMT:
Collaboration diagram for SDCDriverVMT:

8.87.1 Detailed Description

SDCDriver virtual methods table.

8.88 SerialConfig Struct Reference

PLATFORM Serial Driver configuration structure.

#include <hal_serial_lld.h>

Collaboration diagram for SerialConfig:
Data Fields

- `uint32_t speed`
  
  Bit rate.

8.88.1 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.

8.88.2 Field Documentation

8.88.2.1 speed

`uint32_t SerialConfig::speed`

Bit rate.

8.89 SerialDriver Struct Reference

Full duplex serial driver class.

#include <hal_serial.h>
Inheritance diagram for SerialDriver:
Collaboration diagram for SerialDriver:

![Collaboration Diagram]

**Data Fields**

- const struct SerialDriverVMT * vmt
  
  *Virtual Methods Table.*

### 8.89.1 Detailed Description

Full duplex serial driver class.

This class extends *BaseAsynchronousChannel* by adding physical I/O queues.

### 8.89.2 Field Documentation

#### 8.89.2.1 vmt

*const struct SerialDriverVMT* * SerialDriver::vmt

*Virtual Methods Table.*
8.90 SerialDriverVMT Struct Reference

SerialDriver virtual methods table.

#include <hal_serial.h>

Inheritance diagram for SerialDriverVMT:
Collaboration diagram for SerialDriverVMT:

8.90.1 Detailed Description

`SerialDriver` virtual methods table.

8.91 SerialUSBConfig Struct Reference

Serial over USB Driver configuration structure.

```c
#include <hal_serial_usb.h>
```

ChibiOS/HAL
Data Structure Documentation

Collaboration diagram for SerialUSBConfig:

```
USBDriver
+ state
+ transmitting
+ receiving
+ in_params
+ out_params
+ ep0State
+ ep0Next
+ ep0n
+ setup
+ status
+ address
+ configuration
+ saved_state

SerialUSBConfig
+ bulk_in
+ bulk_out
+ int_in

USBConfig
+ get_descriptor_cb
+ requests_hook_cb
+ event_cb
+ sof_cb

USBEndpointConfig
+ ep_mode
+ in_maxsize
+ out_maxsize
+ out_cb
+ in_cb
+ setup_cb
+ epc
+ ep0endcb

Data Fields

- **USBDriver** * usbp
  
  USB driver to use.
- **usbep_t bulk_in**
  
  Bulk IN endpoint used for outgoing data transfer.
- **usbep_t bulk_out**
  
  Bulk OUT endpoint used for incoming data transfer.
- **usbep_t int_in**
  
  Interrupt IN endpoint used for notifications.

8.91.1 Detailed Description

Serial over USB Driver configuration structure.

An instance of this structure must be passed to *sduStart()* in order to configure and start the driver operations.

8.91.2 Field Documentation
8.91.2.1  usbp

```c
USBDriver* SerialUSBConfig::usbp
```

USB driver to use.

8.91.2.2  bulk_in

```c
usbep_t SerialUSBConfig::bulk_in
```

Bulk IN endpoint used for outgoing data transfer.

8.91.2.3  bulk_out

```c
usbep_t SerialUSBConfig::bulk_out
```

Bulk OUT endpoint used for incoming data transfer.

8.91.2.4  int_in

```c
usbep_t SerialUSBConfig::int_in
```

Interrupt IN endpoint used for notifications.

**Note**

If set to zero then the INT endpoint is assumed to be not present, USB descriptors must be changed accordingly.
Full duplex serial driver class.

#include <hal_serial_usb.h>

Inheritance diagram for SerialUSBDriver:
Collaboration diagram for SerialUSB_driver:

Data Fields

- const struct SerialUSB_driverVMT * vmt
  Virtual Methods Table.

8.92.1 Detailed Description

Full duplex serial driver class.

This class extends BaseAsynchronousChannel by adding physical I/O queues.

8.92.2 Field Documentation

8.92.2.1 vmt

const struct SerialUSB_driverVMT* SerialUSB_driver::vmt

Virtual Methods Table.
8.93 SerialUSBDriverVMT Struct Reference

SerialDriver virtual methods table.

#include <hal_serial_usb.h>

Inheritance diagram for SerialUSBDriverVMT:
8.93.1 Detailed Description

SerialDriver virtual methods table.

8.94 SHA1Context Struct Reference

Type of a SHA1 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA1Context:
8.94.1 Detailed Description

Type of a SHA1 context.

8.95 SHA256Context Struct Reference

Type of a SHA256 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA256Context:

8.95.1 Detailed Description

Type of a SHA256 context.

8.96 SHA512Context Struct Reference

Type of a SHA512 context.

#include <hal_crypto_lld.h>

Collaboration diagram for SHA512Context:
8.96.1 Detailed Description

Type of a SHA512 context.

8.97 SNORConfig Struct Reference

Type of a SNOR configuration structure.

#include <hal_serial_nor.h>

Collaboration diagram for SNORConfig:

8.97.1 Detailed Description

Type of a SNOR configuration structure.

8.98 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.

8.98.1 Detailed Description

Type of SNOR flash class.

8.98.2 Field Documentation
8.98.2.1 vmt

const struct SNORDriverVMT* SNORDriver::vmt

SNORDriver Virtual Methods Table.

8.98.2.2 config

const _base_flash_data SNORConfig* SNORDriver::config

Current configuration data.

8.98.2.3 device_id

uint8_t SNORDriver::device_id[20]

Device ID and unique ID.

8.99 SNORDriverVMT Struct Reference

SNOR virtual methods table.

#include <hal_serial_nor.h>

Inheritance diagram for SNORDriverVMT:
8.100 threads_queue_t Struct Reference

Type of a thread queue.

#include <osal.h>

Collaboration diagram for threads_queue_t:

8.100.1 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.
8.101 UARTConfig Struct Reference

Driver configuration structure.

#include <hal_uart_lld.h>

Collaboration diagram for UARTConfig:

Data Fields

- uartcb_t txend1_cb
  
  \textit{End of transmission buffer callback.}

- uartcb_t txend2_cb
  
  \textit{Physical end of transmission callback.}

- uartcb_t txend_cb
  
  \textit{Receive buffer filled callback.}

- uartcbb_t rxchar_cb
  
  \textit{Character received while out if the UART\_RECEIVE state.}

- uartcb_t rxerr_cb
  
  \textit{Receive error callback.}

8.101.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.
8.101.2 Field Documentation

8.101.2.1 txend1_cb

```c
uartcb_t UARTConfig::txend1_cb
```
End of transmission buffer callback.

8.101.2.2 txend2_cb

```c
uartcb_t UARTConfig::txend2_cb
```
Physical end of transmission callback.

8.101.2.3 rxend_cb

```c
uartcb_t UARTConfig::rxend_cb
```
Receive buffer filled callback.

8.101.2.4 rxchar_cb

```c
uartccb_t UARTConfig::rxchar_cb
```
Character received while out if the UART_RECEIVE state.

8.101.2.5 rxerr_cb

```c
uartecb_t UARTConfig::rxerr_cb
```
Receive error callback.
8.102 UARTDriver Struct Reference

Structure representing an UART driver.

#include <hal_uart_lld.h>

Collaboration diagram for UARTDriver:

Data Fields

- **uartstate_t state**  
  Driver state.
- **uarttxstate_t txstate**  
  Transmitter state.
- **uarrxstate_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.
8.102 UARTDriver Struct Reference

8.102.1 Detailed Description

Structure representing an UART driver.

Note
Implementations may extend this structure to contain more, architecture dependent, fields.

8.102.2 Field Documentation

8.102.2.1 state

**uartstate_t** UARTDriver::state

Driver state.

8.102.2.2 txstate

**uarttxstate_t** UARTDriver::txstate

Transmitter state.

8.102.2.3 rxstate

**uarrxstate_t** UARTDriver::rxstate

Receiver state.

8.102.2.4 config

**const UARTConfig** UARTDriver::config

Current configuration data.
8.102.2.5 early

bool UARTDriver::early

Synchronization flag for transmit operations.

8.102.2.6 threadrx

thread_reference_t UARTDriver::threadrx

Waiting thread on RX.

8.102.2.7 threadtx

thread_reference_t UARTDriver::threadtx

Waiting thread on TX.

8.102.2.8 mutex

mutex_t UARTDriver::mutex

Mutex protecting the peripheral.

8.103 unpacked_mmc_cid_t Struct Reference

Unpacked CID register from MMC.

#include <hal_mmcsd.h>

Collaboration diagram for unpacked_mmc_cid_t:

```
unpacked_mmc_cid_t
+ mid
+ oid
+ pnm
+ prv_n
+ prv_m
+ psn
+ mdt_m
+ mdt_y
+ crc
```
8.103.1 Detailed Description

Unpacked CID register from MMC.

8.104 unpacked_mmc_csd_t Struct Reference

Unpacked CSD register from MMC.

#include <hal_mmcsd.h>

Collaboration diagram for unpacked_mmc_csd_t:

8.104.1 Detailed Description

Unpacked CSD register from MMC.

8.105 unpacked_sdc_cid_t Struct Reference

Unpacked CID register from SDC.

#include <hal_mmcsd.h>
Collaboration diagram for unpacked_sdc_cid_t:

```
unpacked_sdc_cid_t
+ mid
+ oid
+ pnm
+ prv_n
+ prv_m
+ psn
+ mdt_m
+ mdt_y
+ crc
```

8.105.1 Detailed Description

Unpacked CID register from SDC.

8.106 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:

```
unpacked_sdc_csd_10_t
+ csd_structure
+ taac
+ nsac
+ tran_speed
+ ccc
+ read_bl_lsn
+ read_bl_partial
+ write_blk_misalign
+ read_blk_misalign
+ dsr_imp
and 19 more...
```
8.106.1 Detailed Description

Unpacked CSD v1.0 register from SDC.

8.107 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...
```

8.107.1 Detailed Description

Unpacked CSD v2.0 register from SDC.

8.108 USBConfig Struct Reference

Type of an USB driver configuration structure.

#include <hal_usb_lld.h>
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.
8.108 USBConfig Struct Reference

8.108.1 Detailed Description

Type of an USB driver configuration structure.

8.108.2 Field Documentation

8.108.2.1 event_cb

usbeventcb_t USBConfig::event_cb

USB events callback.

This callback is invoked when an USB driver event is registered.

8.108.2.2 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.

8.108.2.3 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.

8.108.2.4 sof_cb

usbcallback_t USBConfig::sof_cb

Start Of Frame callback.
8.109  USBDescriptor Struct Reference

Type of an USB descriptor.

#include <hal_usb.h>

Collaboration diagram for USBDescriptor:

```
+ ud_size
+ ud_string
```

Data Fields

- size_t ud_size
  
  *Descriptor size in unicode characters.*
- const uint8_t * ud_string
  
  *Pointer to the descriptor.*

8.109.1  Detailed Description

Type of an USB descriptor.

8.109.2  Field Documentation

8.109.2.1  ud_size

size_t USBDescriptor::ud_size

Descriptor size in unicode characters.

8.109.2.2  ud_string

const uint8_t* USBDescriptor::ud_string

Pointer to the descriptor.
8.110 USBDriver Struct Reference

Structure representing an USB driver.

```c
#include <hal_usb_lld.h>
```

Collaboration diagram for USBDriver:

Data Fields

- `usbstate_t state`
  Driver state.
- `const USBConfig *config`
  Current configuration data.
- `uint16_t transmitting`
  Bit map of the transmitting IN endpoints.
- `uint16_t receiving`
  Bit map of the receiving OUT endpoints.
- `const USBEndpointConfig *epc[USB_MAX_ENDPOINTS+1]`
  Active endpoints configurations.
- `void *in_params[USB_MAX_ENDPOINTS]`
  Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.
- `void *out_params[USB_MAX_ENDPOINTS]`
  Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.
- `usbep0state_t ep0state`
  Endpoint 0 state.
- `uint8_t *ep0next`
  Next position in the buffer to be transferred through endpoint 0.
- `size_t ep0n`
Number of bytes yet to be transferred through endpoint 0.

- `usbcallback_t ep0endcb`
  Endpoint 0 end transaction callback.
- `uint8_t setup [8]`
  Setup packet buffer.
- `uint16_t status`
  Current USB device status.
- `uint8_t address`
  Assigned USB address.
- `uint8_t configuration`
  Current USB device configuration.
- `usbstate_t saved_state`
  State of the driver when a suspend happened.

### 8.110.1 Detailed Description

Structure representing an USB driver.

### 8.110.2 Field Documentation

#### 8.110.2.1 state

`usbstate_t USBDriver::state`

Driver state.

#### 8.110.2.2 config

`const USBConfig* USBDriver::config`

Current configuration data.

#### 8.110.2.3 transmitting

`uint16_t USBDriver::transmitting`

Bit map of the transmitting IN endpoints.
8.110.2.4 receiving

uint16_t USBDriver::receiving

Bit map of the receiving OUT endpoints.

8.110.2.5 epc

const USBEndpointConfig* USBDriver::epc[USB_MAX_ENDPOINTS+1]

Active endpoints configurations.

8.110.2.6 in_params

void* USBDriver::in_params[USB_MAX_ENDPOINTS]

Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.

Note
The base index is one, the endpoint zero does not have a reserved element in this array.

8.110.2.7 out_params

void* USBDriver::out_params[USB_MAX_ENDPOINTS]

Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.

Note
The base index is one, the endpoint zero does not have a reserved element in this array.

8.110.2.8 ep0state

usbep0state_t USBDriver::ep0state

Endpoint 0 state.
8.110.2.9  ep0next

uint8_t* USBDriver::ep0next

Next position in the buffer to be transferred through endpoint 0.

8.110.2.10  ep0n

size_t USBDriver::ep0n

Number of bytes yet to be transferred through endpoint 0.

8.110.2.11  ep0endcb

usbcallback_t USBDriver::ep0endcb

Endpoint 0 end transaction callback.

8.110.2.12  setup

uint8_t USBDriver::setup[8]

Setup packet buffer.

8.110.2.13  status

uint16_t USBDriver::status

Current USB device status.

8.110.2.14  address

uint8_t USBDriver::address

Assigned USB address.
8.110.2.15 configuration

```c
uint8_t USBDriver::configuration
```

Current USB device configuration.

8.110.2.16 saved_state

```c
usbstate_t USBDriver::saved_state
```

State of the driver when a suspend happened.

8.111 USBEndpointConfig Struct Reference

Type of an USB endpoint configuration structure.

```
#include <hal_usb_lld.h>
```
Data Fields

- uint32_t ep_mode
  Type and mode of the endpoint.
- usbepcallback_t setup_cb
  Setup packet notification callback.
- usbepcallback_t in_cb
IN endpoint notification callback.
- `usbepcallback_t out_cb`
  OUT endpoint notification callback.
- `uint16_t in_maxsize`
  IN endpoint maximum packet size.
- `uint16_t out_maxsize`
  OUT endpoint maximum packet size.
- `USBInEndpointState * in_state`
  `USBEndpointState` associated to the IN endpoint.
- `USBOutEndpointState * out_state`
  `USBEndpointState` associated to the OUT endpoint.

8.111.1 Detailed Description

Type of an USB endpoint configuration structure.

Note
Platform specific restrictions may apply to endpoints.

8.111.2 Field Documentation

8.111.2.1 ep_mode

`uint32_t USBEndpointConfig::ep_mode`

Type and mode of the endpoint.

8.111.2.2 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.
8.111.2.3 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.

8.111.2.4 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.

8.111.2.5 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.

8.111.2.6 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.

8.111.2.7 in_state

USBInEndpointState* USBEndpointConfig::in_state

USBEndpointState associated to the IN endpoint.
This structure maintains the state of the IN endpoint.

8.111.2.8 out_state

USBOutEndpointState* USBEndpointConfig::out_state

USBEndpointState associated to the OUT endpoint.
This structure maintains the state of the OUT endpoint.
8.112 USBInEndpointState Struct Reference

Type of an IN endpoint state structure.

#include <hal_usb_lld.h>

Collaboration diagram for USBInEndpointState:

```
<table>
<thead>
<tr>
<th>USBInEndpointState</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ txsize</td>
</tr>
<tr>
<td>+ txcnt</td>
</tr>
<tr>
<td>+ txbuf</td>
</tr>
<tr>
<td>+ thread</td>
</tr>
</tbody>
</table>
```

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.

8.112.1 Detailed Description

Type of an IN endpoint state structure.

8.112.2 Field Documentation

8.112.2.1 txsize

size_t USBInEndpointState::txsize

Requested transmit transfer size.
8.112.2.2  txcnt

size_t USBInEndpointState::txcnt

Transmitted bytes so far.

8.112.2.3  txbuf

const uint8_t* USBInEndpointState::txbuf

Pointer to the transmission linear buffer.

8.112.2.4  thread

thread_reference_t USBInEndpointState::thread

Waiting thread.

8.113  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.

8.113.1 Detailed Description

Type of an OUT endpoint state structure.

8.113.2 Field Documentation

8.113.2.1 rxsize

`size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

8.113.2.2 rxcnt

`size_t USBOutEndpointState::rxcnt`

Received bytes so far.

8.113.2.3 rxbuff

`uint8_t * USBOutEndpointState::rxbuff`

Pointer to the receive linear buffer.

8.113.2.4 thread

`thread_reference_t USBOutEndpointState::thread`

Waiting thread.
8.114  **WDGConfig Struct Reference**

Driver configuration structure.

```c
#include <hal_wdg_lld.h>
```

Collaboration diagram for WDGConfig:

![Collaboration diagram for WDGConfig](image)

8.114.1  **Detailed Description**

Driver configuration structure.

**Note**

It could be empty on some architectures.

8.115  **WDGDriver Struct Reference**

Structure representing an WDG driver.

```c
#include <hal_wdg_lld.h>
```

Collaboration diagram for WDGDriver:

![Collaboration diagram for WDGDriver](image)
Data Fields

- `wdgstate_t state`
  
  Driver state.
- `const WDGConfig* config`
  
  Current configuration data.

8.115.1 Detailed Description

Structure representing an WDG driver.

8.115.2 Field Documentation

8.115.2.1 state

```c
wdgstate_t WDGDriver::state
```

Driver state.

8.115.2.2 config

```c
const WDGConfig* WDGDriver::config
```

Current configuration data.

8.116 wspi_command_t Struct Reference

Type of a WSPI command descriptor.

```c
#include <hal_wspi.h>
```

Collaboration diagram for `wspi_command_t`:

```
+ wspi_command_t
  + cfg
  + cmd
  + addr
  + alt
  + dummy
```
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.

8.116.1 Detailed Description

Type of a WSPI command descriptor.

8.116.2 Field Documentation

8.116.2.1 cfg

`uint32_t wspi_command_t::cfg`
Transfer configuration field.

8.116.2.2 cmd

`uint32_t wspi_command_t::cmd`
Command phase data.

8.116.2.3 addr

`uint32_t wspi_command_t::addr`
Address phase data.

8.116.2.4 alt

`uint32_t wspi_command_t::alt`
Alternate phase data.

8.116.2.5 dummy

`uint32_t wspi_command_t::dummy`
Number of dummy cycles to be inserted.
Chapter 9

File Documentation

9.1 chprintf.c File Reference

Mini printf-like functionality.

```
#include "hal.h"
#include "chprintf.h"
#include "memstreams.h"
```

Functions

- int chvprintf (BaseSequentialStream *chp, const char *fmt, va_list ap)
  
  System formatted output function.

- int chprintf (BaseSequentialStream *chp, const char *fmt,...)
  
  System formatted output function.

- int chsnprintf (char *str, size_t size, const char *fmt,...)
  
  System formatted output function.

- int chvsnprintf (char *str, size_t size, const char *fmt, va_list ap)
  
  System formatted output function.

9.1.1 Detailed Description

Mini printf-like functionality.

9.2 chprintf.h File Reference

Mini printf-like functionality.

```
#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 chvprintf (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.

9.2.1 Detailed Description

Mini printf-like functionality.

9.3 hal.c File Reference

HAL subsystem code.

#include "hal.h"

Functions

- void halInit (void)
  
  HAL initialization.

9.3.1 Detailed Description

HAL subsystem code.
9.4 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_mmcsd.h"
#include "hal_persistent.h"
#include "hal_flash.h"
#include "hal_buffers.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 1**
  Stable release flag.

ChibiOS/HAL version identification

- **#define HAL_VERSION "7.1.5"**
HAL version string.

- #define CH_HAL_MAJOR 7
  HAL version major number.
- #define CH_HAL_MINOR 1
  HAL version minor number.
- #define CH_HAL_PATCH 5
  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.

9.4.1 Detailed Description

HAL subsystem header.

9.5 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.

9.5.1 Detailed Description

ADC Driver code.

9.6 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 adcsIsBufferComplete(adcp) ((bool)((adcp)->state == ADC_COMPLETE))
  Buffer state.

Low level driver helper macros

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

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

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

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

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

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

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

**Typedefs**

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

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

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

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

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

**Enumerations**

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

**Functions**

- `void adcInit (void)`  
  
  ADC Driver initialization.

- `voidadcObjectInit (ADCDriver ∗adcp)`  
  
  Initiates the standard part of an ADC configuration.

- `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)`  
  
  Deactivates the ADC peripheral.
Stops an ongoing conversion.

- void adcStopConversionI (ADCDriver *adcp)

Performs an ADC conversion.

- msg_t adcConvert (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
- void adcAcquireBus (ADCDriver *adcp)
  Gains exclusive access to the ADC peripheral.
- void adcReleaseBus (ADCDriver *adcp)
  Releases exclusive access to the ADC peripheral.

9.6.1 Detailed Description

ADC Driver macros and structures.

9.7 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.

9.7.1 Detailed Description

PLATFORM ADC subsystem low level driver source.
9.8 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.

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.

Enumerations

- enum adcerror_t { ADC_ERR_DMAFAILURE = 0, ADC_ERR_OVERFLOW = 1, ADC_ERR_AWD = 2 }
  
  Possible ADC failure causes.

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.

9.8.1 Detailed Description

PLATFORM ADC subsystem low level driver header.
9.9 hal_buffers.c File Reference

I/O Buffers code.

#include <string.h>
#include "hal.h"

Functions

- **void ibqObjectInit (input_buffers_queue_t *ibqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t infy, void *link)**
  
  Initializes an input buffers queue object.

- **void ibqResetI (input_buffers_queue_t *ibqp)**
  
  Resets an input buffers queue.

- **uint8_t * ibqGetEmptyBufferI (input_buffers_queue_t *ibqp)**

  Gets the next empty buffer from the queue.

- **void ibqPostFullBufferI (input_buffers_queue_t *ibqp, size_t size)**

  Posts a new filled buffer to the queue.

- **msg_t ibqGetFullBufferTimeout (input_buffers_queue_t *ibqp, sysinterval_t timeout)**

  Gets the next filled buffer from the queue.

- **msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t *ibqp, sysinterval_t timeout)**

  Gets the next filled buffer from the queue.

- **void ibqReleaseEmptyBuffer (input_buffers_queue_t *ibqp)**

  Releases the buffer back in the queue.

- **void ibqReleaseEmptyBufferS (input_buffers_queue_t *ibqp)**

  Releases the buffer back in the queue.

- **msg_t ibqGetTimeout (input_buffers_queue_t *ibqp, sysinterval_t timeout)**

  Input queue read with timeout.

- **size_t ibqReadTimeout (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.

9.9.1 Detailed Description

I/O Buffers code.

9.10 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 *bqp)
  Double buffer notification callback type.
- typedef io_buffers_queue_t input_buffers_queue_t
  Type of an input buffers queue.
- typedef io_buffers_queue_t output_buffers_queue_t
  Type of an output buffers queue.

## Functions

- void ibqObjectInit (input_buffers_queue_t *ibqp, bool suspended, uint8_t *bp, size_t size, size_t n, bqnotify_t info, 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.
- 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.

### 9.10.1 Detailed Description

I/O Buffers macros and structures.

### 9.11 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 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.
9.11.1 Detailed Description

CAN Driver code.

9.12 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.

9.12.1 Detailed Description

CAN Driver macros and structures.

9.13 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)
  
  *Enforces leaving the sleep mode.*

Variables

- CANDriver **CAND1**
  
  *CAN1 driver identifier.*

9.13.1 Detailed Description

PLATFORM CAN subsystem low level driver source.

9.14  **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_lldAbort (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.

9.14.1 Detailed Description

PLATFORM CAN subsystem low level driver header.
9.15 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 \ _base_asynchronous_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 chnAddFlagsI(ip, flags)
  Adds status flags to the listener's flags mask.

9.15.1 Detailed Description

I/O channels access.

This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

9.16 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.

9.16.1 Detailed Description

Cryptographic Driver code.

9.17 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)`
  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)`
  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.
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.

### 9.17.1 Detailed Description

Cryptographic Driver macros and structures.
9.18 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, 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.

9.18.1 Detailed Description

PLATFORM cryptographic subsystem low level driver source.

9.19 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 a 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)

- 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)

Initializes 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-ECB.

- 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-ECB.

- 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, const uint8_t *iv)

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, const uint8_t *iv)

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.

9.19.1 Detailed Description

PLATFORM cryptographic subsystem low level driver header.

9.20 hal_dac.c File Reference

DAC Driver code.

#include "hal.h"
Functions

- void `dacInit` (void)
  
  DAC Driver initialization.

- void `dacObjectInit` (DACDriver ∗dacr)
  
  Initializes the standard part of a DACDriver structure.

- void `dacStart` (DACDriver ∗dacr, const DACConfig ∗config)
  
  Configures and activates the DAC peripheral.

- void `dacStop` (DACDriver ∗dacr)
  
  Deactivates the DAC peripheral.

- void `dacPutChannelX` (DACDriver ∗dacr, dacchannel_t channel, dacsample_t sample)
  
  Outputs a value directly on a DAC channel.

- void `dacStartConversion` (DACDriver ∗dacr, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)
  
  Starts a DAC conversion.

- void `dacStartConversionI` (DACDriver ∗dacr, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)
  
  Starts a DAC conversion.

- void `dacStopConversion` (DACDriver ∗dacr)
  
  Stops an ongoing conversion.

- void `dacStopConversionI` (DACDriver ∗dacr)
  
  Stops an ongoing conversion.

- msg_t `dacConvert` (DACDriver ∗dacr, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)
  
  Performs a DAC conversion.

- void `dacAcquireBus` (DACDriver ∗dacr)
  
  Gains exclusive access to the DAC bus.

- void `dacReleaseBus` (DACDriver ∗dacr)
  
  Releases exclusive access to the DAC bus.

9.20.1 Detailed Description

DAC Driver code.

9.21 hal_dac.h File Reference

DAC Driver macros and structures.

#include "hal_dac_lld.h"

Data Structures

- struct `hal_dac_conversion_group`
  
  DAC Conversion group structure.

- struct `hal_dac_config`
  
  Driver configuration structure.

- struct `hal_dac_driver`
  
  Structure representing a DAC driver.
Macros

DAC configuration options

- `#define DAC_USE_WAIT TRUE`  
  Enables synchronous APIs.
- `#define DAC_USE_MUTUAL_EXCLUSION TRUE`  
  Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

Low level driver helper macros

- `#define dacIsBufferComplete(dacp) ((bool)((dacp)->state == DAC_COMPLETE))`  
  Buffer state.
- `#define _dac_wait_s(dacp) osalThreadSuspendS(&(dacp)->thread)`  
  Waits for operation completion.
- `#define _dac_reset_i(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)`  
  Resumes a thread waiting for a conversion completion.
- `#define _dac_reset_s(dacp) osalThreadResumeS(&(dacp)->thread, MSG_RESET)`  
  Resumes a thread waiting for a conversion completion.
- `#define _dac_wakeup_isr(dacp)`  
  Wakes up the waiting thread.
- `#define _dac_timeout_isr(dacp)`  
  Wakes up the waiting thread with a timeout message.
- `#define _dac_isr_half_code(dacp)`  
  Common ISR code, half buffer event.
- `#define _dac_isr_full_code(dacp)`  
  Common ISR code, full buffer event.
- `#define _dac_isr_error_code(dacp, err)`  
  Common ISR code, error event.

Typedefs

- `typedef struct hal_dac_driver DACDriver`  
  Type of a structure representing an DAC driver.
- `typedef struct hal_dac_config DACConfig`  
  Type of a structure representing an DAC driver configuration.
- `typedef struct hal_dac_conversion_group DACConversionGroup`  
  Type of a DAC conversion group.
- `typedef void(*daccallback_t)(DACDriver *dacp)`  
  DAC notification callback type.
- `typedef void(*dacerrorcallback_t)(DACDriver *dacp, dacerror_t err)`  
  DAC error callback type.

Enumerations

- `enum dacstate_t { DAC_UNINIT = 0, DAC_STOP = 1, DAC_READY = 2, DAC_ACTIVE = 3, DAC_COMPLETE = 4, DAC_ERROR = 5 }`  
  Driver state machine possible states.
Functions

- `void dacInit (void)`
  DAC Driver initialization.
- `void dacObjectInit (DACDriver *dcp)`
  Initializes the standard part of a DACDriver structure.
- `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.

9.21.1 Detailed Description

DAC Driver macros and structures.

9.22 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.

9.22.1 Detailed Description

PLATFORM DAC subsystem low level driver source.

9.23 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 *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._

### 9.23.1 Detailed Description

PLATFORM DAC subsystem low level driver header.

### 9.24 hal_efl.c File Reference

Embedded Flash Driver code.

```c
#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._

### 9.24.1 Detailed Description

Embedded Flash Driver code.

### 9.25 hal_efl.h File Reference

Embedded Flash Driver macros and structures.

```c
#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.

9.25.1 Detailed Description

Embedded Flash Driver macros and structures.

9.26 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 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.

9.26.1 Detailed Description

PLATFORM Embedded Flash subsystem low level driver source.

9.27 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.

9.27.1 Detailed Description

PLATFORM Embedded Flash subsystem low level driver header.

9.28 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 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 fileStreamSetPosition(ip, offset) ((ip)->vmt->setposition(ip, offset))**
  
  Moves the file current pointer to an absolute position.

- **#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 fileStreamGetError(ip) ((ip)->vmt->geterror(ip))**
  
  Returns an implementation dependent error code.

Typedefs

- **typedef uint32_t fileoffset_t**
  
  File offset type.

9.28.1 Detailed Description

Data files.

This header defines abstract interfaces useful to access generic data files in a standardized way.

9.29 hal_flash.c File Reference

Generic flash driver class code.

```c
#include "hal.h"
#include "hal_flash.h"
```
Functions

- **flash_error_t flashWaitErase (BaseFlash *devp)**
  Waits until the current erase operation is finished.
- **flash_offset_t flashGetSectorOffset (BaseFlash *devp, flash_sector_t sector)**
  Returns the offset of a sector.
- **uint32_t flashGetSectorSize (BaseFlash *devp, flash_sector_t sector)**
  Returns the size of a sector.

9.29.1 Detailed Description

Generic flash driver class code.

9.30 hal_flash.h File Reference

Generic flash driver class header.

Data Structures

- **struct flash_sector_descriptor_t**
  Flash sector descriptor.
- **struct flash_descriptor_t**
  Type of a flash device descriptor.
- **struct BaseFlashVMT**
  BaseFlash virtual methods table.
- **struct BaseFlash**
  Base flash class.

Macros

- **#define _base_flash_methods_alone**
  BaseFlash specific methods.
- **#define _base_flash_methods**
  BaseFlash specific methods with inherited ones.
- **#define _base_flash_data**
  BaseFlash specific data.

Flash attributes

- **#define FLASH_ATTR_ERASED_IS_ONE 0x00000001U**
  Defines one as the erased bit state.
- **#define FLASH_ATTR_MEMORY_MAPPED 0x00000002U**
  The memory is accessible in a memory mapped mode.
- **#define FLASH_ATTR_REWRITABLE 0x00000004U**
  Programmed pages can be programmed again.
- **#define FLASH_ATTR_ECC_CAPABLE 0x00000008U**
  The memory is protected by an ECC mechanism.
- **#define FLASH_ATTR_ECC_ZERO_LINE_CAPABLE 0x00000010U**
The device is able to overwrite zero to a line.

- \#define FLASH\_ATTR\_SUSPEND\_ERASE\_CAPABLE 0x00000020U
  
  The device is able to suspend erase operations.

Macro Functions (BaseFlash)

- \#define getBaseFlash(ip) ((BaseFlash *)&(ip)->vmt)
  
  Instance getter.
- \#define flashGetDescriptor(ip) (ip)->vmt->get_descriptor(ip)
  
  Gets the flash descriptor structure.
- \#define flashRead(ip, offset, n, rp) (ip)->vmt->read(ip, offset, n, rp)
  
  Read operation.
- \#define flashProgram(ip, offset, n, pp) (ip)->vmt->program(ip, offset, n, pp)
  
  Program operation.
- \#define flashStartEraseAll(ip) (ip)->vmt->start_erase_all(ip)
  
  Starts a whole-device erase operation.
- \#define flashStartEraseSector(ip, sector) (ip)->vmt->start_erase_sector(ip, sector)
  
  Starts an sector erase operation.
- \#define flashQueryErase(ip, msec) (ip)->vmt->query_erase(ip, msec)
  
  Queries the driver for erase operation progress.
- \#define flashVerifyErase(ip, sector) (ip)->vmt->verify_erase(ip, sector)
  
  Returns the erase state of a sector.

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.

9.30.1 Detailed Description

Generic flash driver class header.
9.31 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.

9.31.1 Detailed Description

GPT Driver code.

9.32 hal_gpt.h File Reference

GPT Driver macros and structures.

#include "hal_gpt_lld.h"
Macros

- #define gptChangeIntervalI(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.
9.32.1 Detailed Description

GPT Driver macros and structures.

9.33 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.

9.33.1 Detailed Description

PLATFORM GPT subsystem low level driver source.

9.34 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.

9.34.1 Detailed Description

PLATFORM GPT subsystem low level driver header.

9.35 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 tbytes, uint8_t *rxbuf, size_t rbytes, sysinterval_t timeout)
  Sends data via the I2C bus.
- msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rbytes, 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.

9.35.1 Detailed Description

I2C Driver code.

9.36 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, tbytes, rxbuf, rbytes)
  Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.
- #define i2cMasterReceive(i2cp, addr, rxbuf, rbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rbytes, 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.
9.36.1 Detailed Description

I2C Driver macros and structures.

9.37 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.

9.37.1 Detailed Description

PLATFORM I2C subsystem low level driver source.

9.38 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.

9.38.1 Detailed Description

PLATFORM I2C subsystem low level driver header.

9.39 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.

9.39.1 Detailed Description

I2S Driver code.

9.40 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.

9.40.1 Detailed Description

I2S Driver macros and structures.

9.41 hal_i2s_lld.c File Reference

PLATFORM I2S subsystem low level driver source.

#include "hal.h"
9.42 hal_i2s_lld.h File Reference

Functions

- void i2s_lld_init (void)
  Low level I2S driver initialization.
- void i2s_lld_start (I2SDriver *i2sp)
  Configures and activates the I2S peripheral.
- void i2s_lld_stop (I2SDriver *i2sp)
  Deactivates the I2S peripheral.
- void i2s_lld_start_exchange (I2SDriver *i2sp)
  Starts a I2S data exchange.
- void i2s_lld_stop_exchange (I2SDriver *i2sp)
  Stops the ongoing data exchange.

Variables

- I2SDriver I2SD1
  I2S2 driver identifier.

9.41.1 Detailed Description

PLATFORM I2S subsystem low level driver source.

9.42 hal_i2s_lld.h File Reference

PLATFORM I2S subsystem low level driver header.

Macros

- #define i2s_lld_driver_fields
  Low level fields of the I2S driver structure.
- #define i2s_lld_config_fields
  Low level fields of the I2S configuration structure.

PLATFORM configuration options

- #define PLATFORM_I2S_USE_I2S1 FALSE
  I2SD1 driver enable switch.

Functions

- void i2s_lld_init (void)
  Low level I2S driver initialization.
- void i2s_lld_start (I2SDriver *i2sp)
  Configures and activates the I2S peripheral.
- void i2s_lld_stop (I2SDriver *i2sp)
  Deactivates the I2S peripheral.
- void i2s_lld_start_exchange (I2SDriver *i2sp)
  Starts a I2S data exchange.
- void i2s_lld_stop_exchange (I2SDriver *i2sp)
  Stops the ongoing data exchange.
9.42.1 Detailed Description

PLATFORM I2S subsystem low level driver header.

9.43 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.

9.43.1 Detailed Description

ICU Driver code.

9.44 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 icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)
  Check on notifications status.
- #define icuGetWidthX(icup) icu_lld_get_width(icup)
  Returns the width of the latest pulse.
- #define icuGetPeriodX(icup) icu_lld_get_period(icup)
  Returns the width of the latest cycle.

Low level driver helper macros

- #define _icu_isr_invoke_width_cb(icup)
  Common ISR code, ICU width event.
- #define _icu_isr_invoke_period_cb(icup)
  Common ISR code, ICU period event.
- #define _icu_isr_invoke_overflow_cb(icup)
  Common ISR code, ICU timer overflow event.

Typedefs

- typedef struct ICUDriver ICUDriver
  Type of a structure representing an ICU driver.
- typedef void(icucallback_t) (ICUDriver *icup)
  ICU notification callback type.

Enumerations

- enum icustate_t {
  ICU_UNINIT = 0, ICU_STOP = 1, ICU_READY = 2, ICU_WAITING = 3,
  ICU_ACTIVE = 4
}
  Driver state machine possible states.

Functions

- void icuInit (void)
  ICU Driver initialization.
- void icuObjectInit (ICUDriver *icup)
  Initializes the standard part of a ICUDriver structure.
- 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.

9.44.1 Detailed Description

ICU Driver macros and structures.

9.45 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.
9.45.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.46 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
  
  ICUD1 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.

9.46.1 Detailed Description

PLATFORM ICU subsystem low level driver header.

9.47 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.

9.47.1 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.

9.48 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.

9.48.1 Detailed Description

PLATFORM HAL subsystem low level driver source.
9.49  hal_lld.h File Reference

PLATFORM HAL subsystem low level driver header.

Macros

Platform identification macros

• #define PLATFORM_NAME "templates"

Functions

• void hal_lld_init (void)
  Low level HAL driver initialization.

9.49.1 Detailed Description

PLATFORM HAL subsystem low level driver header.

9.50  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.
9.50.1 Detailed Description

MAC Driver code.

9.51 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.
• void macReleaseTransmitDescriptor (MACDriver *macp)
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
• void macPollLinkStatus (MACDriver *macp)
  Updates and returns the link status.

9.51.1 Detailed Description

MAC Driver macros and structures.

9.52 hal_mac_lld.c File Reference

PLATFORM MAC subsystem low level driver source.

#include <string.h>
#include "hal.h"
#include "hal_mii.h"

Functions

• void mac_lld_init (void)
  Low level MAC initialization.
• void mac_lld_start (MACDriver *macp)
  Configures and activates the MAC peripheral.
• void mac_lld_stop (MACDriver *macp)
  Deactivates the MAC peripheral.
• msg_t mac_lld_get_transmit_descriptor (MACDriver *macp)
  Returns a transmission descriptor.
• msg_t mac_lld_release_transmit_descriptor (MACDriver *macp)
  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)
  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.

Variables

• MACDriver ETHD1
  MAC1 driver identifier.

9.52.1 Detailed Description

PLATFORM MAC subsystem low level driver source.

9.53 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 (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.

9.53.1 Detailed Description

PLATFORM MAC subsystem low level driver header.

9.54 hal_mfs.c File Reference

Managed Flash Storage module code.

#include <string.h>
#include "hal.h"
#include "hal_mfs.h"

Macros

- **#define ALIGNED_REC_SIZE(n) (flash_offset_t)MFS_ALIGN_NEXT(sizeof (mfs_data_header_t) + (size_t(n))**
  
  Data record size aligned.
- **#define ALIGNED_DHDR_SIZE ALIGNED_REC_SIZE(0)**
  
  Data record header size aligned.
- **#define ALIGNED_SIZEOF(t) (((sizeof (t) - 1U) | MFS_ALIGN_MASK) + 1U)**
  
  Aligned size of a type.
- **#define PAIR(a, b) (((unsigned)(a) << 2U) | (unsigned)(b))**
  
  Combines two values (0..3) in one (0..15).
- **#define RET_ON_ERROR(err)**
  
  Error check helper.
Functions

- static mfs_error_t mfs_flash_read (MFSDriver *mfsp, flash_offset_t offset, size_t n, uint8_t *rp)
  Flash read.

- static mfs_error_t mfs_flash_write (MFSDriver *mfsp, flash_offset_t offset, size_t n, const uint8_t *wp)
  Flash write.

- static mfs_error_t mfs_flash_copy (MFSDriver *mfsp, flash_offset_t doffset, flash_offset_t soffset, uint32_t n)
  Flash copy.

- static mfs_error_t mfs_bank_erase (MFSDriver *mfsp, mfs_bank_t bank)
  Erases and verifies all sectors belonging to a bank.

- static mfs_error_t mfs_bank_verify_erase (MFSDriver *mfsp, mfs_bank_t bank)
  Erases and verifies all sectors belonging to a bank.

- static mfs_error_t mfs_bank_write_header (MFSDriver *mfsp, mfs_bank_t bank, uint32_t cnt)
  Writes the validation header in a bank.

- static mfs_bank_state_t mfs_bank_check_header (MFSDriver *mfsp)
  Checks integrity of the header in the shared buffer.

- static mfs_error_t mfs_bank_scan_records (MFSDriver *mfsp, mfs_bank_t bank, bool *wflagp)
  Scans blocks searching for records.

- static mfs_error_t mfs_bank_get_state (MFSDriver *mfsp, mfs_bank_t bank, mfs_bank_state_t *statep, uint32_t *cntp)
  Determines the state of a bank.

- 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, const MFSConfig *config)
  Configures and activates a MFS driver.

- void mfsObjectInit (MFSDriver *mfsp)
  Initializes an instance.

- 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.
9.54.1 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.

9.55 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)
9.56 hal_mii.h File Reference

**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 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.

9.55.1 Detailed Description

Managed Flash Storage module header.

9.56 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_DCOUNTER 0x12
- #define MII_FCSCOUNTER 0x13
- #define MII_NWAYTEST 0x14
- #define MII_REERRCOUNTER 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_RFALT 0x0010
- #define BMSR_ANEGCOMPLETE 0x0020
- #define BMSR_MFRUPPCAP 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`
9.56.1 Detailed Description

MII macros and structures.

9.57 hal_mmc_spi.c File Reference

MMC over SPI driver code.

```c
#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 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 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)
9.58 hal_mmc_spi.h File Reference

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 \( x^7 + x^3 + 1 \)).

9.57.1 Detailed Description
MMC over SPI driver code.

9.58 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.

- **boolmmcGetInfo (MMCDriver *mmcp, BlockDeviceInfo *bdip)**
  
  Returns the media info.

- **bool mmcErase (MMCDriver *mmcp, uint32_t startblk, uint32_t endblk)**
  
  Erases blocks.

### 9.58.1 Detailed Description

MMC over SPI driver header.

### 9.59 hal_mmcasd.c File Reference

MMC/SD cards common code.

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

- `uint32_t _mmcsd_get_slice (const uint32_t *data, uint32_t end, uint32_t start)`
  Gets a bit field from a words array.
- `uint32_t _mmcsd_get_capacity (const uint32_t *csd)`
  Extract card capacity from a CSD.
- `uint32_t _mmcsd_get_capacity_ext (const uint8_t *ext_csd)`
  Extract MMC card capacity from EXT_CSD.
- `void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice *sdcp, unpacked_sdc_cid_t *cidsdc)`
  Unpacks SDC CID array in structure.
- `void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice *sdcp, unpacked_mmc_cid_t *cidmmc)`
  Unpacks MMC CID array in structure.
- `void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice *sdcp, unpacked_mmc_csd_t *csdmmc)`
  Unpacks MMC CSD array in structure.
- `void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_10_t *csd10)`
  Unpacks SDC CSD v1.0 array in structure.
- `void _mmcsd_unpack_csd_v20 (const MMCSDBlockDevice *sdcp, unpacked_sdc_csd_20_t *csd20)`
  Unpacks SDC CSD v2.0 array in structure.

9.59.1 Detailed Description

MMC/SD cards common code.

9.60 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_READ_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 12U, 12U
• #define MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE 13U, 13U
• #define MMCSD_CSD_20_COPY_SLICE 14U, 14U
• #define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 15U, 15U
• #define MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE 21U, 21U
• #define MMCSD_CSD_20_WRITE_BL_LEN_SLICE 25U, 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_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_PNM5_SLICE  111U, 104U
• #define MMCSD_CID_SDC_MID_SLICE  127U, 120U
• #define MMCSD_CID_MMC_CRC_SLICE  7U, 1U
• #define MMCSD_CID_MMC_MDT_Y_SLICE  11U, 8U
• #define MMCSD_CID_MMC_MDT_M_SLICE  15U, 12U
• #define MMCSD_CID_MMC_PSN_SLICE  47U, 16U
• #define MMCSD_CID_MMC_PRV_M_SLICE  51U, 48U
• #define MMCSD_CID_MMC_PRV_N_SLICE  55U, 52U
• #define MMCSD_CID_MMC_PNM0_SLICE  63U, 56U
• #define MMCSD_CID_MMC_PNM1_SLICE  71U, 64U
• #define MMCSD_CID_MMC_PNM2_SLICE  79U, 72U
• #define MMCSD_CID_MMC_PNM3_SLICE  87U, 80U
• #define MMCSD_CID_MMC_PNM4_SLICE  95U, 88U
• #define MMCSD_CID_MMC_PNM5_SLICE  103U, 96U
• #define MMCSD_CID_MMC_OID_SLICE  111U, 104U
• #define MMCSD_CID_MMC_MID_SLICE  127U, 120U

R1 response utilities

• #define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)
Evaluates to true if the R1 response contains error flags.

- **#define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)**
  
  Returns the status field of an R1 response.

- **#define MMCSD_R1_IS_CARD_LOCKED(r1) (((((r1) >> 21U) & 1U) != 0U)**
  
  Evaluates to true if the R1 response indicates a locked card.

### Macro Functions

- **#define mmcsdGetCardCapacity(ip) ((ip) - capacity)**
  
  Returns the card capacity in blocks.

### Functions

- **uint32_t _mmcsd_get_slice (const uint32_t *data, uint32_t end, uint32_t start)**
  
  Gets a bit field from a words array.

- **uint32_t _mmcsd_get_capacity (const uint32_t *csd)**
  
  Extract card capacity from a CSD.

- **uint32_t _mmcsd_get_capacity_ext (const uint8_t *ext_csd)**
  
  Extract MMC card capacity from EXT_CSD.

- **void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice *sdcp, unpacked_sdc_cid_t *cidsdc)**
  Unpacks SDC CID array in structure.

- **void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice *sdcp, unpacked_mmc_cid_t *cidmmc)**
  Unpacks MMC CID array in structure.

- **void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice *sdcp, unpacked_mmc_csd_t *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.

### 9.60.1 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.

### 9.61 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) ((type)((size_t)(ip)) - (ip)->vmt->instance_offset)
  
  Returns the instance pointer starting from an interface pointer.

9.61.1 Detailed Description

Base object.

This header defines a base object that is the root for the inheritance system.

9.62 hal_pal.c File Reference

I/O Ports Abstraction Layer code.

#include "hal.h"

Functions

- iopormap_t palReadBus (const IOBus *bus)
  
  Read from an I/O bus.

- void palWriteBus (const IOBus *bus, iopormap_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.
9.62.1 Detailed Description

I/O Ports Abstraction Layer code.

9.63 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) palClearPad(PAL_PORT(line), PAL_PAD(line))
   Clears a line logic state to PAL_LOW.

#define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))
   Toggles a line logic state.

#define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)
   Line mode setup.

#define palEnablePadEventI(port, pad, mode)
   Pad event enable.

#define palDisablePadEventI(port, pad)
   Pad event disable.

#define palEnablePadEvent(port, pad, mode)
   Pad event enable.

#define palDisablePadEvent(port, pad, mode)
   Pad event disable.

#define palEnableLineEventI(line, mode) palEnablePadEventI(PAL_PORT(line), PAL_PAD(line), mode)
   Line event enable.

#define palDisableLineEventI(line) palDisablePadEventI(PAL_PORT(line), PAL_PAD(line))
   Line event disable.

#define palEnableLineEvent(line, mode)
   Line event enable.

#define palDisableLineEvent(line)
   Line event disable.

#define palIsPadEventEnabledX(port, pad) false
   Pad event enable check.

#define palIsLineEventEnabledX(line)
   pal_lld_ispadeventenabled(PAL_PORT(line), PAL_PAD(line))
   Line event enable check.
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.

**Typedefs**

- `typedef void(*palcallback_t)(void *arg)`
  Type of a PAL event callback.

**Functions**

- `ioportmask_t palReadBus(const IOBus *bus)`
  Read from an I/O bus.
- `void palWriteBus(const IOBus *bus, ioportmask_t bits)`
  Write to an I/O bus.
- `void palSetBusMode(const IOBus *bus, iomode_t mode)`
  Programs a bus with the specified mode.
- `void palSetPadCallbackI(ioportid_t port, iopadid_t pad, palcallback_t cb, void *arg)`
  Associates a callback to a port/pad.
- `void palSetLineCallbackI(ioline_t line, palcallback_t cb, void *arg)`
  Associates a callback to a line.
- `msg_t palWaitPadTimeoutS(ioportid_t port, iopadid_t pad, sysinterval_t timeout)`
  Waits for an edge on the specified port/pad.
- `msg_t palWaitPadTimeout(ioportid_t port, iopadid_t pad, sysinterval_t timeout)`
  Waits for an edge on the specified port/pad.
- `msg_t palWaitLineTimeoutS(ioline_t line, sysinterval_t timeout)`
  Waits for an edge on the specified line.
- `msg_t palWaitLineTimeout(ioline_t line, sysinterval_t timeout)`
  Waits for an edge on the specified line.

### 9.63.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures.

### 9.64 hal_pal_lld.c File Reference

PLATFORM PAL subsystem low level driver source.

```c
#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.
9.64.1 Detailed Description

PLATFORM PAL subsystem low level driver source.

9.65 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_writeport(port, bits)
  Writes a bits mask on a I/O port.
- #define pal_lld_setport(port, bits)
  Sets a bits mask on a I/O port.
- #define pal_lld_clearport(port, bits)
  clears a bits mask on a I/O port.
- #define pal_lld_toggleport(port, bits)
  Toggles a bits mask on a I/O port.
- #define pal_lld_readgroup(port, mask, offset) 0U
  Reads a group of bits.
- #define pal_lld_writegroup(port, mask, offset, bits)
  Writes a group of bits.
- #define pal_lld_setgroupmode(port, mask, offset, mode) __pal_lld_setgroupmode(port, mask \&\& offset, mode)
  Pads group mode setup.
- #define pal_lld_readpad(port, pad) PAL_LOW
  Reads a logical state from an I/O pad.
- #define pal_lld_writepad(port, pad, bit)
  Writes a logical state on an output pad.
- #define pal_lld_setpad(port, pad)
  Sets a pad logical state to PAL_HIGH.
- #define pal_lld_clearpad(port, pad)
  Clears a pad logical state to PAL_LOW.
- #define pal_lld_togglepad(port, pad)

Toggles a pad logical state.

- `#define pal_lld_setpadmode(port, pad, mode)`
  Pad mode setup.
- `#define pal_lld_get_pad_event(port, pad) &_pal_events[0]; (void)(port); (void)pad`
  Returns a PAL event structure associated to a pad.
- `#define pal_lld_get_line_event(line) &_pal_events[0]; (void)line`
  Returns a PAL event structure associated to a line.

**Port related definitions**

- `#define PAL_IOPORTS_WIDTH 16U`
  Width, in bits, of an I/O port.
- `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)`
  Whole port mask.

**Line handling macros**

- `#define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad)))`
  Forms a line identifier.
- `#define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFF0U))`
  Decodes a port identifier from a line identifier.
- `#define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))`
  Decodes a pad identifier from a line identifier.
- `#define PAL_NOLINE 0U`
  Value identifying an invalid line.

**Typedefs**

- `typedef uint32_t ioportmask_t`
  Digital I/O port sized unsigned type.
- `typedef uint32_t iomode_t`
  Digital I/O modes.
- `typedef uint32_t ioline_t`
  Type of an I/O line.
- `typedef uint32_t ioportid_t`
  Port Identifier.
- `typedef uint32_t iopadid_t`
  Type of an pad identifier.

**Functions**

- `void _pal_lld_init (void)`
  STM32 I/O ports configuration.
- `void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)`
  Pads mode setup.

**9.65.1 Detailed Description**

PLATFORM PAL subsystem low level driver header.
9.66 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 psGetSize(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.

9.66.1 Detailed Description

Generic persistent storage class header.
9.67 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, pwmcount_t period)
  Changes the period the PWM peripheral.
- void pwmEnableChannel (PWMDriver *pwmp, pwmchannel_t channel, pwmcount_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.

9.67.1 Detailed Description

PWM Driver code.

9.68 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(*)(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 of 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.

9.68.1 Detailed Description

PWM Driver macros and structures.

9.69 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.

9.69.1 Detailed Description

PLATFORM PWM subsystem low level driver source.

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

Functions

- `void pwm_lld_init (void)`  
  Low level PWM driver initialization.
- `void pwm_lld_start (PWMDriver *pwmp)`  
  Configures and activates the PWM peripheral.
- `void pwm_lld_stop (PWMDriver *pwmp)`  
  Deactivates the PWM peripheral.
- `void pwm_lld_enable_channel (PWMDriver *pwmp, pwmchannel_t channel, pwmcnt_t width)`  
  Enables a PWM channel.
- `void pwm_lld_disable_channel (PWMDriver *pwmp, pwmchannel_t channel)`  
  Disables a PWM channel and its notification.
- `void pwm_lld_enable_periodic_notification (PWMDriver *pwmp)`  
  Enables the periodic activation edge notification.
- `void pwm_lld_disable_periodic_notification (PWMDriver *pwmp)`  
  Disables the periodic activation edge notification.
- `void pwm_lld_enable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)`  
  Enables a channel de-activation edge notification.
- `void pwm_lld_disable_channel_notification (PWMDriver *pwmp, pwmchannel_t channel)`  
  Disables a channel de-activation edge notification.

9.70.1 Detailed Description

PLATFORM PWM subsystem low level driver header.
9.71 \textit{hal\_queues.c} File Reference

I/O Queues code.

#include <string.h>
#include "hal.h"

Functions

- static size_t \textit{iq\_read} (input\_queue\_t \*iqp, uint8\_t \*bp, size\_t n)
  
  Non-blocking input queue read.

- static size_t \textit{oq\_write} (output\_queue\_t \*oqp, const uint8\_t \*bp, size\_t n)
  
  Non-blocking output queue write.

- void \textit{iqObjectInit} (input\_queue\_t \*iqp, uint8\_t \*bp, size\_t size, qnotify\_t infy, void \*link)
  
  Initializes an input queue.

- void \textit{iqResetI} (input\_queue\_t \*iqp)
  
  Resets an input queue.

- msg\_t \textit{iq\_PutI} (input\_queue\_t \*iqp, uint8\_t b)
  
  Input queue write.

- msg\_t \textit{iq\_GetI} (input\_queue\_t \*iqp)
  
  Input queue non-blocking read.

- msg\_t \textit{iq\_GetTimeout} (input\_queue\_t \*iqp, sysinterval\_t timeout)
  
  Input queue read with timeout.

- size\_t \textit{iq\_ReadI} (input\_queue\_t \*iqp, uint8\_t \*bp, size\_t n)
  
  Input queue non-blocking read.

- size\_t \textit{iq\_ReadTimeout} (input\_queue\_t \*iqp, uint8\_t \*bp, size\_t n, sysinterval\_t timeout)
  
  Input queue read with timeout.

- void \textit{oqObjectInit} (output\_queue\_t \*oqp, uint8\_t \*bp, size\_t size, qnotify\_t onfy, void \*link)
  
  Initializes an output queue.

- void \textit{oqResetI} (output\_queue\_t \*oqp)
  
  Resets an output queue.

- msg\_t \textit{oq\_PutI} (output\_queue\_t \*oqp, uint8\_t b)
  
  Output queue write.

- msg\_t \textit{oq\_PutTimeout} (output\_queue\_t \*oqp, uint8\_t b, sysinterval\_t timeout)
  
  Output queue write with timeout.

- msg\_t \textit{oq\_GetI} (output\_queue\_t \*oqp)
  
  Output queue read.

- size\_t \textit{oq\_WriteI} (output\_queue\_t \*oqp, const uint8\_t \*bp, size\_t n)
  
  Output queue non-blocking write.

- size\_t \textit{oq\_WriteTimeout} (output\_queue\_t \*oqp, const uint8\_t \*bp, size\_t n, sysinterval\_t timeout)
  
  Output queue write with timeout.

9.71.1 Detailed Description

I/O Queues code.
9.72 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 iqIsFullI(iqp) ((bool)(qSpaceI(iqp) == 0U))
  
  Evaluates to true if the specified input queue is full.

• #define iqIsEmptyI(iqp)
  
  Evaluates to true if the specified input queue is empty.

• #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 oqIsFullI(oqp)
  
  Evaluates to true if the specified output queue is full.

• #define oqIsEmptyI(oqp) ((bool)(qSpaceI(oqp) == 0U))
  
  Evaluates to true if the specified output queue is empty.

• #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-leading 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.

**9.72.1 Detailed Description**

I/O Queues macros and structures.
9.73 hal_rtc.c File Reference

RTC Driver code.

#include "hal.h"

Functions

• void rtcInit (void)
  RTC Driver initialization.
• void rtcObjectInit (RTCDriver ∗rtcp)
  Initializes a generic RTC driver object.
• void rtcSetTime (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)
  Set current time.
• void rtcGetTime (RTCDriver ∗rtcp, RTCDateTime ∗timespec)
  Get current time.
• void rtcSetAlarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAlarm ∗alarmspec)
  Set alarm time.
• void rtcGetAlarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAlarm ∗alarmspec)
  Get current alarm.
• void rtcSetCallback (RTCDriver ∗rtcp, rtccb_t callback)
  Enables or disables RTC callbacks.
• void rtcConvertDateTimeToStructTm (const RTCDateTime ∗timespec, struct tm ∗timp, uint32_t ∗tv_msec)
  Convert RTCDateTime to broken-down time structure.
• void rtcConvertStructTmToDateTime (const struct tm ∗timp, uint32_t tv_msec, RTCDateTime ∗timespec)
  Convert broken-down time structure to RTCDateTime.
• uint32_t rtcConvertDateTimeToFAT (const RTCDateTime ∗timespec)
  Get current time in format suitable for usage in FAT file system.

9.73.1 Detailed Description

RTC Driver code.

9.74 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.
9.74.1 Detailed Description

RTC Driver macros and structures.

9.75 hal_rtc_lld.c File Reference

PLATFORM RTC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void rtc_lld_init (void)
  
  RTC driver identifier.
- void rtc_lld_set_time (RTCDriver *rtcp, const RTCDateTime *timespec)
  
  Set current time.
- void rtc_lld_get_time (RTCDriver *rtcp, RTCDateTime *timespec)
  
  Get current time.
- void rtc_lld_set_alarm (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAAlarm *alarmspec)
  
  Set alarm time.
- void rtc_lld_get_alarm (RTCDriver *rtcp, rtcalarm_t alarm, RTCAAlarm *alarmspec)
  
  Get alarm time.

9.75.1 Detailed Description

PLATFORM RTC subsystem low level driver source.

9.76 hal_rtc_lld.h File Reference

PLATFORM RTC subsystem low level driver header.

Data Structures

- struct RTCAAlarm
  
  Type of a structure representing an RTC alarm time stamp.
Macros

• #define rtc_ld_driver_fields uint32_t dummy
  Implementation-specific RTC Driver fields.

Implementation capabilities

• #define RTC_SUPPORTS_CALLBACKS TRUE
  Callback support in the driver.
• #define RTC_ALARMS 2
  Number of alarms available.
• #define RTC_HAS_STORAGE FALSE
  Presence of a local persistent storage.

PLATFORM configuration options

• #define PLATFORM_RTC_USE_RTC1 FALSE
  RTCD1 driver enable switch.

Typedefs

• typedef void(∗ rtccb_t)(RTCDriver ∗rtcp, rtcevent_t event)
  Type of a generic RTC callback.

Enumerations

• enum rtcevent_t
  Type of an RTC event.

Functions

• void rtc_lld_init (void)
  RTC driver identifier.
• void rtc_lld_set_time (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)
  Set current time.
• void rtc_lld_get_time (RTCDriver ∗rtcp, RTCDateTime ∗timespec)
  Get current time.
• void rtc_lld_set_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAAlarm ∗alarmspec)
  Set alarm time.
• void rtc_lld_get_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAAlarm ∗alarmspec)
  Get alarm time.

9.76.1 Detailed Description

PLATFORM RTC subsystem low level driver header.
9.77 hal_sdc.c File Reference

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 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.

**Variables**

• static const struct SDCDriverVMT sdc_vmt

Virtual methods table.

**9.77.1 Detailed Description**

SDC Driver code.

**9.78 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 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.

- bool _sdc_wait_for_transfer_state (SDCDriver *sdcp)
  
  Wait for the card to complete pending operations.

9.78.1 Detailed Description

SDC Driver macros and structures.

9.79 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 *sdp)
  
  Configures and activates the SDC peripheral.

- void sdc_lld_stop (SDCDriver *sdp)
  
  Deactivates the SDC peripheral.

- void sdc_lld_start_clk (SDCDriver *sdp)
  
  Starts the SDIO clock and sets it to init mode (400kHz or less).

- void sdc_lld_set_data_clk (SDCDriver *sdp, sdcbusclk_t clk)
  
  Sets the SDIO clock to data mode (25MHz or less).

- void sdc_lld_stop_clk (SDCDriver *sdp)
  
  Stops the SDIO clock.

- void sdc_lld_set_bus_mode (SDCDriver *sdp, sdcbusmode_t mode)
  
  Switches the bus to 4 bits mode.

- void sdc_lld_send_cmd_none (SDCDriver *sdp, uint8_t cmd, uint32_t arg)
  
  Sends an SDIO command with no response expected.

- bool sdc_lld_send_cmd_short (SDCDriver *sdp, 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 *sdp, 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 *sdp, 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 *sdp, uint32_t startblk, uint8_t *buf, uint32_t n)
  
  Reads one or more blocks.

- bool sdc_lld_write (SDCDriver *sdp, uint32_t startblk, const uint8_t *buf, uint32_t n)
  
  Writes one or more blocks.

- bool sdc_lld_sync (SDCDriver *sdp)
  
  Waits for card idle condition.

Variables

- SDCDriver SDCD1
  
  SDCD1 driver identifier.

9.79.1 Detailed Description

PLATFORM SDC subsystem low level driver source.

9.80 hal_sdc_lld.h File Reference

PLATFORM SDC subsystem low level driver header.
Data Structures

- struct SDCConfig
  Driver configuration structure.
- struct SDCDriverVMT
  SDCDriver virtual methods table.
- struct SDCDriver
  Structure representing an SDC driver.

Macros

- #define _sdc_driver_methods _mmcsd_block_device_methods
  SDCDriver specific methods.

PLATFORM configuration options

- #define PLATFORM_SDC_USE_SDC1 FALSE
  PWMD1 driver enable switch.

Typedefs

- typedef uint32_t sdcmode_t
  Type of card flags.
- typedef uint32_t sdcflags_t
  SDC Driver condition flags type.
- typedef struct SDCDriver SDCDriver
  Type of a structure representing an SDC driver.

Functions

- void sdc_lld_init (void)
  Low level SDC driver initialization.
- void sdc_lld_start (SDCDriver *sdcp)
  Configures and activates the SDC peripheral.
- void sdc_lld_stop (SDCDriver *sdcp)
  Deactivates the SDC peripheral.
- void sdc_lld_start_clk (SDCDriver *sdcp)
  Starts the SDIO clock and sets it to init mode (400kHz or less).
- void sdc_lld_set_data_clk (SDCDriver *sdcp, sdcbusclk_t clk)
  Sets the SDIO clock to data mode (25MHz or less).
- void sdc_lld_stop_clk (SDCDriver *sdcp)
  Stops the SDIO clock.
- void sdc_lld_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)
  Switches the bus to 4 bits mode.
- void sdc_lld_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)
  Sends an SDIO command with no response expected.
- bool sdc_lld_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)
  Sends an SDIO command with a short response expected.
- bool sdc_lld_send_cmd_short_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)
Sends an SDIO command with a short response expected and CRC.

- bool sdc_lld_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)
  Sends an SDIO command with a long response expected and CRC.

- bool sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)
  Reads one or more blocks.

- bool sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)
  Writes one or more blocks.

- bool sdc_lld_sync (SDCDriver *sdcp)
  Waits for card idle condition.

9.80.1 Detailed Description

PLATFORM SDC subsystem low level driver header.

9.81 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.

9.81.1 Detailed Description

Serial Driver code.
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 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)->queue)
  Direct read from a SerialDriver.
• #define sdGet(sdp) iqGet(&(sdp)->queue)
  Direct read from a SerialDriver.
• #define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->queue, t)
  Direct read from a SerialDriver with timeout specification.
• #define sdWriteI(sdp, b, n) oqWriteI(&(sdp)->queue, b, n)
  Direct blocking write to a SerialDriver.
• #define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->queue, b, n, TIME_INFINITE)
  Direct blocking write to a SerialDriver.
• #define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->queue, b, n, t)
  Direct blocking write to a SerialDriver with timeout specification.
• #define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->queue, b, n, TIME_IMMEDIATE)
  Direct non-blocking write to a SerialDriver.
• #define sdReadI(sdp, b, n) iqReadI(&(sdp)->queue, b, n, TIME_INFINITE)
  Direct blocking read from a SerialDriver.
• #define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->queue, b, n, TIME_INFINITE)
  Direct blocking read from a SerialDriver.
• #define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->queue, b, n, t)
  Direct blocking read from a SerialDriver with timeout specification.
• #define sdAsynchronousRead(sdp, b, n) iqReadTimeout(&(sdp)->queue, 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.
9.82.1 Detailed Description

Serial Driver macros and structures.

9.83 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.

9.83.1 Detailed Description

PLATFORM serial subsystem low level driver source.

9.84 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.

9.84.1 Detailed Description

PLATFORM serial subsystem low level driver header.

9.85 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.

9.85.1 Detailed Description

Serial NOR serial flash driver code.

9.86 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_addr_dummy_receive (BUSDriver *busp, uint32_t cmd, 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 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 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.
9.86.1 Detailed Description

Serial NOR driver header.

9.87 hal_serial_usb.c File Reference

Serial over USB Driver code.

#include "hal.h"

Functions

- static void ibnotify (io_buffers_queue_t *bqp)
  
  Notification of empty buffer released into the input buffers queue.

- static void obnotify (io_buffers_queue_t *bqp)
  
  Notification of filled buffer inserted into the output buffers queue.

- void sduInit (void)
  
  Serial Driver initialization.

- void sduObjectInit (SerialUSBDriver *dup)
  
  Initializes a generic full duplex driver object.

- void sduStart (SerialUSBDriver *dup, const SerialUSBConfig *config)
  
  Configures and starts the driver.

- void sduStop (SerialUSBDriver *dup)
  
  Stops the driver.

- void sduSuspendHookI (SerialUSBDriver *dup)
  
  USB device suspend handler.

- void sduWakeupHookI (SerialUSBDriver *dup)
  
  USB device wakeup handler.

- void sduConfigureHookI (SerialUSBDriver *dup)
  
  USB device configured handler.

- bool sduRequestsHook (USBDriver *usb)
  
  Default requests hook.

- void sduSOFHookI (SerialUSBDriver *dup)
  
  SOF handler.

- void sduDataTransmitted (USBDriver *usb, usbep_t ep)
  
  Default data transmitted callback.

- void sduDataReceived (USBDriver *usb, usbep_t ep)
  
  Default data received callback.

- void sduInterruptTransmitted (USBDriver *usb, usbep_t ep)
  
  Default data received callback.

- msg_t sduControl (USBDriver *usb, unsigned int operation, void *arg)
  
  Control operation on a serial USB port.

9.87.1 Detailed Description

Serial over USB Driver code.
Serial over USB Driver macros and structures.

```
#include "hal_usb_cdc.h"
```

## Data Structures

- **struct SerialUSBConfig**
  
  Serial over USB Driver configuration structure.

- **struct SerialUSBDriverVMT**
  
  SerialDriver virtual methods table.

- **struct SerialUSBDriver**
  
  Full duplex serial driver class.

## Macros

- **#define _serial_usb_driver_data**
  
  SerialDriver specific data.

- **#define _serial_usb_driver_methods _base_asynchronous_channel_methods**
  
  SerialUSBDriver specific methods.

## SERIAL_USB configuration options

- **#define SERIAL_USB_BUFFERS_SIZE 256**
  
  Serial over USB buffers size.

- **#define SERIAL_USB_BUFFERS_NUMBER 2**
  
  Serial over USB number of buffers.

## Typedefs

- **typedef struct SerialUSBDriver SerialUSBDriver**
  
  Structure representing a serial over USB driver.

## Enumerations

- **enum sdustate_t { SDU_UNINIT = 0, SDU_STOP = 1, SDU_READY = 2 }**
  
  Driver state machine possible states.
Functions

- void sduInit (void)
  Serial Driver initialization.
- void sduObjectInit (SerialUSBDriver ∗sdup)
  Initializes a generic full duplex driver object.
- 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.

9.88.1 Detailed Description
Serial over USB Driver macros and structures.

9.89 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.
- void sioStart (SIODriver ∗siop, const SIOConfig ∗config)
  Configures and activates the SIO peripheral.
- void sioStop (SIODriver ∗siop)
  Deactivates the SIO peripheral.
9.89.1 Detailed Description

SIO Driver code.

9.90 hal_sio.h File Reference

SIO Driver macros and structures.

#include "hal_sio_lld.h"

Macros

- #define sioGetFlagsX(siop) sio_lld_get_flags(siop)
  Returns the current set of flags and clears it.
- #define sioRXIsEmptyX(siop) sio_lld_rx_is_empty(siop)
  Determines the state of the RX FIFO.
- #define sioTXIsFullX(siop) sio_lld_tx_is_full(siop)
  Determines the state of the TX FIFO.
- #define sioRXGetX(siop) sio_lld_rx_get(siop)
  Returns one frame from the RX FIFO.
- #define sioTXPutX(siop, data) sio_lld_tx_put(siop, data)
  Pushes one frame into the TX FIFO.
- #define sioReadX(siop, buffer, size) sio_lld_read(siop, buffer, size)
  Reads data from the RX FIFO.
- #define sioWriteX(siop, buffer, size) sio_lld_write(siop, buffer, size)
  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.
Typedefs

- 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.

Enumerations

- enum siostate_t { SIO_UNINIT = 0, SIO_STOP = 1, SIO_READY = 2 }
  Driver state machine possible states.

Functions

- void sioInit (void)
  SIO Driver initialization.
- void sioObjectInit (SIODriver *siop)
  Initializes the standard part of a SIODriver structure.
- void sioStart (SIODriver *siop, const SIOConfig *config)
  Configures and activates the SIO peripheral.
- void sioStop (SIODriver *siop)
  Deactivates the SIO peripheral.

9.90.1 Detailed Description

SIO Driver macros and structures.

9.91 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.
- void sio_lld_start (SIODriver *siop)
  Configures and activates the SIO peripheral.
- void sio_lld_stop (SIODriver *siop)
  Deactivates the SIO peripheral.
- msg_t sio_lld_control (SIODriver *siop, unsigned int operation, void *arg)
  Control operation on a serial port.
Variables

• **SIODriver SIOD1**
  
  SIO1 driver identifier.

### 9.91.1 Detailed Description

PLATFORM SIO subsystem low level driver source.

### 9.92 hal_sio_lld.h File Reference

PLATFORM SIO subsystem low level driver header.

Data Structures

• **struct hal_sio_config**
  
  Driver configuration structure.

• **struct hal_sio_driver**
  
  Structure representing a SIO driver.

Macros

• #define sio_lld_rx_is_empty(siop) true
  
  Determines the state of the RX FIFO.

• #define sio_lld_tx_is_full(siop) true
  
  Determines the state of the TX FIFO.

• #define sio_lld_rx_get(siop)
  
  Returns one frame from the RX FIFO.

• #define sio_lld_tx_put(siop, data)
  
  Pushes one frame into the TX FIFO.

PLATFORM configuration options

• #define PLATFORM_SIO_USE_SIO1 FALSE
  
  SIO driver enable switch.

Typedefs

• typedef uint32_t sioflags_t
  
  SIO driver condition flags type.

• typedef void(* siocb_t) (SIODriver *siop)
  
  Generic SIO notification callback type.

• typedef void(* sioecb_t) (SIODriver *siop, sioflags_t e)
  
  Receive error SIO notification callback type.
9.93 hal_spi.c File Reference

Functions

- void sio_lld_init (void)
  Low level SIO driver initialization.
- void sio_lld_start (SIODriver *siop)
  Configures and activates the SIO peripheral.
- void sio_lld_stop (SIODriver *siop)
  Deactivates the SIO peripheral.
- msg_t sio_lld_control (SIODriver *siop, unsigned int operation, void *arg)
  Control operation on a serial port.

9.92.1 Detailed Description

PLATFORM SIO subsystem low level driver header.

9.93 hal_spi.c File Reference

SPI Driver code.

#include "hal.h"

Functions

- void spiInit (void)
  SPI Driver initialization.
- voidspiObjectInit (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.

9.93.1 Detailed Description

SPI Driver code.

9.94 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
• #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.

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.

9.94.1 Detailed Description

SPI Driver macros and structures.

9.95 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.

9.95.1 Detailed Description

PLATFORM SPI subsystem low level driver source.

9.96 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.

9.96.1 Detailed Description

PLATFORM SPI subsystem low level driver header.

9.97 hal_st.c File Reference

ST Driver code.

#include "hal.h"

Functions

- void stInit (void)
  
  ST Driver initialization.

- 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.
9.98 hal_st.h File Reference

ST Driver macros and structures.

`#include "hal_st_lld.h"

9.97.1 Detailed Description

ST Driver code.

9.98 hal_st.h File Reference

ST Driver macros and structures.

Functions

- `void stInit (void)`
  
  ST Driver initialization.

- `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.

9.98.1 Detailed Description

ST Driver macros and structures.

This header is designed to be include-able without having to include other files from the HAL.
9.99  hal_st_lld.c File Reference

PLATFORM ST subsystem low level driver source.

#include "hal.h"

Functions

- void st_lld_init (void)
  
  \textit{Low level ST driver initialization.}

9.99.1 Detailed Description

PLATFORM ST subsystem low level driver source.

9.100  hal_st_lld.h File Reference

PLATFORM ST subsystem low level driver header.

Functions

- void st_lld_init (void)
  
  \textit{Low level ST driver initialization.}
- static systime_t st_lld_get_counter (void)
  
  \textit{Returns the time counter value.}
- static void st_lld_start_alarm (systime_t abstime)
  
  \textit{Starts the alarm.}
- static void st_lld_stop_alarm (void)
  
  \textit{Stops the alarm interrupt.}
- static void st_lld_set_alarm (systime_t abstime)
  
  \textit{Sets the alarm time.}
- static systime_t st_lld_get_alarm (void)
  
  \textit{Returns the current alarm time.}
- static bool st_lld_is_alarm_active (void)
  
  \textit{Determines if the alarm is active.}

9.100.1 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.
Data streams.

Data Structures

- struct BaseSequentialStreamVMT
  BaseSequentialStream virtual methods table.
- struct BaseSequentialStream
  Base stream class.

Macros

- #define _base_sequential_stream_methods
  BaseSequentialStream specific methods.
- #define _base_sequential_stream_data _base_object_data
  BaseSequentialStream 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.

9.101.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

9.102 hal_trng.c File Reference

TRNG Driver code.

#include "hal.h"
Functions

- void trngInit (void)
  TRNG Driver initialization.
- void trngObjectInit (TRNGDriver *trngp)
  Initializes the standard part of a TRNGDriver structure.
- 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.

9.102.1 Detailed Description

TRNG Driver code.

9.103 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.

9.103.1 Detailed Description

TRNG Driver macros and structures.

9.104 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.

9.104.1 Detailed Description

PLATFORM TRNG subsystem low level driver source.
9.105  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.

9.105.1  Detailed Description

PLATFORM TRNG subsystem low level driver header.

9.106  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 uartStartSendl (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 uartStopSendl (UARTDriver *uartp)
  Stops any ongoing transmission.
- void uartStartReceive (UARTDriver *uartp, size_t n, void *rxbuf)
  Starts a receive operation on the UART peripheral.
- void uartStartReceivev (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 uartStopReceivev (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.

9.106.1 Detailed Description

UART Driver code.

9.107 hal_uart.h File Reference

UART Driver macros and structures.

#include "hal_uart_lld.h"
Macros

UART status flags

- `#define UART_NO_ERROR 0`
  No pending conditions.

- `#define UART_PARITY_ERROR 4`
  Parity error happened.

- `#define UART_FRAMING_ERROR 8`
  Framing error happened.

- `#define UART_OVERRUN_ERROR 16`
  Overflow happened.

- `#define UART_NOISE_ERROR 32`
  Noise on the line.

- `#define UART_BREAK_DETECTED 64`
  Break detected.

UART error conditions

- `#define UART_ERR_NOT_ACTIVE (size_t)-1`

UART configuration options

- `#define UART_USE_WAIT FALSE`
  Enables synchronous APIs.

- `#define UART_USE_MUTUAL_EXCLUSION FALSE`
  Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Low level driver helper macros

- `#define _uart_wakeup_tx1_isr(uartp)`
  Wakes up the waiting thread in case of early TX complete.

- `#define _uart_wakeup_tx2_isr(uartp)`
  Wakes up the waiting thread in case of late TX complete.

- `#define _uart_wakeup_rx_complete_isr(uartp)`
  Wakes up the waiting thread in case of RX complete.

- `#define _uart_wakeup_rx_error_isr(uartp)`
  Wakes up the waiting thread in case of RX error.

- `#define _uart_wakeup_rx_cm_isr(uartp)`
  Wakes up the waiting thread in case of RX character match.

- `#define _uart_wakeup_rx_timeout_isr(uartp)`
  Wakes up the waiting thread in case of RX timeout.

- `#define _uart_tx1_isr_code(uartp)`
  Common ISR code for early TX.

- `#define _uart_tx2_isr_code(uartp)`
  Common ISR code for late TX.

- `#define _uart_rx_complete_isr_code(uartp)`
  Common ISR code for RX complete.

- `#define _uart_rx_error_isr_code(uartp, errors)`
  Common ISR code for RX error.

- `#define _uart_rx_idle_code(uartp)`
  Common ISR code for RX on idle.

- `#define _uart_timeout_isr_code(uartp)`
  Timeout ISR code for receiver.

- `#define _uart_rx_char_match_isr_code(uartp)`
  Character match ISR code for receiver.
Enumerations

- enum uartstate_t { UART_UNINIT = 0, UART_STOP = 1, UART_READY = 2 }
  Driver state machine possible states.
- enum uartxstate_t { UART_TX_IDLE = 0, UART_TX_ACTIVE = 1, UART_TX_COMPLETE = 2 }
  Transmitter state machine states.
- enum uartrxstate_t { UART_RX_IDLE = 0, UART_RX_ACTIVE = 1, UART_RX_COMPLETE = 2 }
  Receiver state machine states.

Functions

- void uartInit (void)
  UART Driver initialization.
- void uartObjectInit (UARTDriver *uartp)
  Initializes the standard part of a UARTDriver structure.
- 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 transmission 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.

9.107.1 Detailed Description

UART Driver macros and structures.
9.108  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_stopReceive (UARTDriver *uartp)
  Stops any ongoing receive operation.

Variables

- UARTDriver UARTD1
  UART1 driver identifier.

9.108.1  Detailed Description

PLATFORM UART subsystem low level driver source.

9.109  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.

9.109.1 Detailed Description

PLATFORM UART subsystem low level driver header.

9.110 hal_usb.c File Reference

USB Driver code.

#include <string.h>
#include "hal.h"
Functions

- static void set_address (USBDriver *usbp)
  SET ADDRESS transaction callback.
- static bool default_handler (USBDriver *usbp)
  Standard requests handler.
- void usbInit (void)
  USB Driver initialization.
- void usbObjectInit (USBDriver *usbp)
  Initializes the standard part of a USBDriver structure.
- 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.

9.110.1 Detailed Description

USB Driver code.
9.111 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, ilInterface)
  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, ilInterface)
  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 usbdriverstatel(usb) ((usb)->state)  
  Returns the driver state.
• #define usbdconnectbus(usb) usbdLD_connect_bus(usb)  
  Connects the USB device.
• #define usbddisconnectbus(usb) usbdLD_disconnect_bus(usb)  
  Disconnect the USB device.
• #define usbdgetframenumbeX(usb) usbdLD_get_frame_number(usb)  
  Returns the current frame number.
• #define usbgettransmitstatusi(usb, ep) (((usb)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)  
  Returns the status of an IN endpoint.
• #define usbgetreceivestatusi(usb, ep) (((usb)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)  
  Returns the status of an OUT endpoint.
• #define usbgetreceivetransactionsizeX(usb, ep) usbdLD_get_transaction_size(usb, ep)  
  Returns the exact size of a receive transaction.
• #define usbdsetuptransfer(usb, buf, n, endcb)  
  Request transfer setup.
• #define usbrdreadsetup(usb, ep, buf) usbdLD_read_setup(usb, ep, buf)  
  Reads a setup packet from the dedicated packet buffer.

Low level driver helper macros

• #define _usb_isr_invoke_event_cb(usb, evt)  
  Common ISR code, usb event callback.
• #define _usb_isr_invoke_sof_cb(usb)  
  Common ISR code, SOF callback.
• #define _usb_isr_invoke_setup_cb(usb, ep)  
  Common ISR code, setup packet callback.
• #define _usb_isr_invoke_in_cb(usb, ep)  
  Common ISR code, IN endpoint callback.
• #define _usb_isr_invoke_out_cb(usb, 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 ∗usb)  
  Type of an USB generic notification callback.
• typedef void(∗ usbdcallback_t) (USBDriver ∗usb, usbep_t ep)  
  Type of an USB endpoint callback.
• typedef void(∗ usbeventcb_t) (USBDriver ∗usb, usbevent_t event)  
  Type of an USB event notification callback.
• typedef bool(∗ usbreqhandler_t) (USBDriver ∗usb)  
  Type of a requests handler callback.
Enumerations

- **enum usbstate_t**
  
  ```c
  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**
  
  ```c
  EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE = 2 }
  ```

  Type of an endpoint status.

- **enum usbep0state_t**
  
  ```c
  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**
  
  ```c
  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.

Functions

- **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.

Variables

- const typedef USBDescriptor *(* usbdgetdescriptor_t )(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)

  Type of an USB descriptor-retrieving callback.

9.111.1 Detailed Description

USB Driver macros and structures.

9.112 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

## 9.112.1 Detailed Description

USB CDC macros and structures.

## 9.113 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.
9.113.1 Detailed Description

PLATFORM USB subsystem low level driver source.

9.113.2 Variable Documentation

9.113.2.1 in

USBInEndpointState in

IN EP0 state.

9.113.2.2 out

USBOutEndpointState out

OUT EP0 state.

9.114 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.

### 9.114.1 Detailed Description

PLATFORM USB subsystem low level driver header.

### 9.115 hal_wdg.c File Reference

WDG Driver code.

```c
#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.

### 9.115.1 Detailed Description

WDG Driver code.

### 9.116 hal_wdg.h File Reference

WDG Driver macros and structures.

```c
#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.

### 9.116.1 Detailed Description

WDG Driver macros and structures.

### 9.117 hal_wdg_lld.c File Reference

WDG Driver subsystem low level driver source template.

```c
#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.

### 9.117.1 Detailed Description

WDG Driver subsystem low level driver source template.
WGD Driver subsystem low level driver header template.

Data Structures

- struct WDGConfig
  
  Driver configuration structure.

- struct WDGDriver
  
  Structure representing an WGD driver.

Macros

- Configuration options
  
  #define PLATFORM_WDG_USE_WDG1 FALSE
  
  WDG1 driver enable switch.

Typedefs

- typedef struct WDGDriver WDGDriver

  Type of a structure representing an WGD driver.

Functions

- void wdg_lld_init (void)

  Low level WGD driver initialization.

- void wdg_lld_start (WDGDriver *wdgp)

  Configures and activates the WGD peripheral.

- void wdg_lld_stop (WDGDriver *wdgp)

  Deactivates the WGD peripheral.

- void wdg_lld_reset (WDGDriver *wdgp)

  Reloads WDG's counter.

9.118.1 Detailed Description

WGD Driver subsystem low level driver header template.

9.119 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.

9.119.1 Detailed Description

WSPI Driver code.

9.120 hal_wspi.h File Reference

WSPI Driver macros and structures.

#include "hal_wspi_lld.h"

Data Structures

- struct wspi_command_t
  Type of a WSPI command descriptor.
- struct hal_wspi_config
  Driver configuration structure.
- struct hal_wspi_driver
  Structure representing an WSPI driver.
Macros

WSPI configuration options

- #define WSPI_USE_WAIT TRUE
  Enables synchronous APIs.
- #define WSPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the wspiAcquireBus() and wspiReleaseBus() APIs.

Transfer options

Note
The low level driver has the option to override the following definitions and use its own ones. In must take care to use the same name for the same function or compatibility is not ensured.

- #define WSPI_CFG_CMD_MODE_MASK (7LU << 0LU)
- #define WSPI_CFG_CMD_MODE_NONE (0LU << 0LU)
- #define WSPI_CFG_CMD_MODE_ONE_LINE (1LU << 0LU)
- #define WSPI_CFG_CMD_MODE_TWO_LINES (2LU << 0LU)
- #define WSPI_CFG_CMD_MODE_FOUR_LINES (3LU << 0LU)
- #define WSPI_CFG_CMD_MODE_EIGHT_LINES (4LU << 0LU)
- #define WSPI_CFG_CMD_DTR (1LU << 3LU)
- #define WSPI_CFG_CMD_SIZE_MASK (3LU << 4LU)
- #define WSPI_CFG_CMD_SIZE_8 (0LU << 4LU)
- #define WSPI_CFG_CMD_SIZE_16 (1LU << 4LU)
- #define WSPI_CFG_CMD_SIZE_24 (2LU << 4LU)
- #define WSPI_CFG_CMD_SIZE_32 (3LU << 4LU)
- #define WSPI_CFG_ADDR_MODE_MASK (7LU << 8LU)
- #define WSPI_CFG_ADDR_MODE_NONE (0LU << 8LU)
- #define WSPI_CFG_ADDR_MODE_ONE_LINE (1LU << 8LU)
- #define WSPI_CFG_ADDR_MODE_TWO_LINES (2LU << 8LU)
- #define WSPI_CFG_ADDR_MODE_FOUR_LINES (3LU << 8LU)
- #define WSPI_CFG_ADDR_MODE_EIGHT_LINES (4LU << 8LU)
- #define WSPI_CFG_ADDR_DTR (1LU << 11LU)
- #define WSPI_CFG_ADDR_SIZE_MASK (3LU << 12LU)
- #define WSPI_CFG_ADDR_SIZE_8 (0LU << 12LU)
- #define WSPI_CFG_ADDR_SIZE_16 (1LU << 12LU)
- #define WSPI_CFG_ADDR_SIZE_24 (2LU << 12LU)
- #define WSPI_CFG_ADDR_SIZE_32 (3LU << 12LU)
- #define WSPI_CFG_ALT_MODE_MASK (7LU << 16LU)
- #define WSPI_CFG_ALT_MODE_NONE (0LU << 16LU)
- #define WSPI_CFG_ALT_MODE_ONE_LINE (1LU << 16LU)
- #define WSPI_CFG_ALT_MODE_TWO_LINES (2LU << 16LU)
- #define WSPI_CFG_ALT_MODE_FOUR_LINES (3LU << 16LU)
- #define WSPI_CFG_ALT_MODE_EIGHT_LINES (4LU << 16LU)
- #define WSPI_CFG_ALT_DTR (1LU << 19LU)
- #define WSPI_CFG_ALT_SIZE_MASK (3LU << 20LU)
- #define WSPI_CFG_ALT_SIZE_8 (0LU << 20LU)
- #define WSPI_CFG_ALT_SIZE_16 (1LU << 20LU)
- #define WSPI_CFG_ALT_SIZE_24 (2LU << 20LU)
- #define WSPI_CFG_ALT_SIZE_32 (3LU << 20LU)
- #define WSPI_CFG_DATA_MODE_MASK (7LU << 24LU)
- #define WSPI_CFG_DATA_MODE_NONE (0LU << 24LU)
- #define WSPI_CFG_DATA_MODE_ONE_LINE (1LU << 24LU)
- #define WSPI_CFG_DATA_MODE_TWO_LINES (2LU << 24LU)
- #define WSPI_CFG_DATA_MODE_FOUR_LINES (3LU << 24LU)
- #define WSPI_CFG_DATA_MODE_EIGHT_LINES (4LU << 24LU)
- #define WSPI_CFG_DATA_DTR (1LU << 27LU)
- #define WSPI_CFG_DQS_ENABLE (1LU << 29LU)
- #define WSPI_CFG_SIOO (1LU << 31LU)
- #define WSPI_CFG_ALL_DTR
Macro Functions

- `#define wspiStartCommandI(wspip, cmdp)`
  Sends a command without data phase.
- `#define wspiStartSendI(wspip, cmdp, n, txbuf)`
  Sends data over the WSPI bus.
- `#define wspiStartReceiveI(wspip, cmdp, n, rxbuf)`
  Receives data from the WSPI bus.
- `#define wspiMapFlashI(wspip, cmdp, addrp) wspi_lld_map_flash(wspip, cmdp, addrp)`
  Maps in memory space a WSPI flash device.
- `#define wspiUnmapFlashI(wspip)`
  Maps in memory space a WSPI flash device.

Low level driver helper macros

- `#define _wspi_wakeup_isr(wspip, msg)`
  Wakes up the waiting thread.
- `#define _wspi_isr_code(wspip)`
  Common ISR code.
- `#define _wspi_error_code(wspip)`
  Common error ISR code.

Typedefs

- `typedef struct hal_wspi_driver WSPIDriver` 
  Type of a structure representing 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` 
  
  ```c
  enum wspistate_t {
  WSPI_UNINIT = 0, WSPI_STOP = 1, WSPI_READY = 2, WSPI_SEND = 3, 
  WSPI_RECEIVE = 4, WSPI_COMPLETE = 5, WSPI_MEMMAP = 6 }
  ```
  Driver state machine possible states.

Functions

- `void wspiInit (void)`
  WSPI Driver initialization.
- `void wspiObjectInit (WSPIDriver ∗ wspip)`
  Initializes the standard part of a WSPIDriver structure.
- `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.

### 9.120.1 Detailed Description

WSPI Driver macros and structures.

### 9.121 hal_wspi_lld.c File Reference

PLATFORM WSPI subsystem low level driver source.

```c
#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

- **WSPIDriver WSPID1**
  WSPID1 driver identifier.

9.121.1 Detailed Description

PLATFORM WSPI subsystem low level driver source.

9.122 hal_wspi_lld.h File Reference

PLATFORM WSPI subsystem low level driver header.

Macros

- `#define wspi_lld_driver_fields`
  Low level fields of the WSPI driver structure.
- `#define wspi_lld_config_fields`
  Low level fields of the WSPI configuration structure.

WSPI implementation capabilities

- `#define WSPI_SUPPORTS_MEMMAP TRUE`
- `#define WSPI_DEFAULT_CFG_MASKS TRUE`

Configuration options

- `#define PLATFORM_WSPI_USE_WSPI1 FALSE`
  WSPID1 driver enable switch.

Functions

- `void wspi_lld_init (void)`
  Low level WSPI driver initialization.
- `void wspi_lld_start (WSPIDriver *wspip)`
  Configures and activates the WSPI peripheral.
- `void wspi_lld_stop (WSPIDriver *wspip)`
  Deactivates the WSPI peripheral.
- `void wspi_lld_command (WSPIDriver *wspip, const wspi_command_t *cmdp)`
  Sends a command without data phase.
- `void wspi_lld_send (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, const uint8_t *txbuf)`
  Sends a command with data over the WSPI bus.
- `void wspi_lld_receive (WSPIDriver *wspip, const wspi_command_t *cmdp, size_t n, uint8_t *rxbuf)`
  Sends a command then receives data over the WSPI bus.
- `void wspi_lld_map_flash (WSPIDriver *wspip, const wspi_command_t *cmdp, uint8_t **addrp)`
  Maps in memory space a WSPI flash device.
- `void wspi_lld_unmap_flash (WSPIDriver *wspip)`
  Unmaps from memory space a WSPI flash device.
9.122.1 Detailed Description

PLATFORM WSPI subsystem low level driver header.

9.123 halconf.h File Reference

HAL configuration header.

#include "mcuconf.h"

Macros

• #define HAL_USE_PAL TRUE
  Enables the PAL subsystem.
• #define HAL_USE_ADC TRUE
  Enables the ADC subsystem.
• #define HAL_USE_CAN TRUE
  Enables the CAN subsystem.
• #define HAL_USE_CRY TRUE
  Enables the cryptographic subsystem.
• #define HAL_USE_DAC TRUE
  Enables the DAC subsystem.
• #define HAL_USE_EFL TRUE
  Enables the EFlash subsystem.
• #define HAL_USE_GPT TRUE
  Enables the GPT subsystem.
• #define HAL_USE_I2C TRUE
  Enables the I2C subsystem.
• #define HAL_USE_I2S TRUE
  Enables the I2S subsystem.
• #define HAL_USE_ICU TRUE
  Enables the ICU subsystem.
• #define HAL_USE_MAC TRUE
  Enables the MAC subsystem.
• #define HAL_USE_MMC_SPI TRUE
  Enables the MMC_SPI subsystem.
• #define HAL_USE_PWM TRUE
  Enables the PWM subsystem.
• #define HAL_USE_RTC TRUE
  Enables the RTC subsystem.
• #define HAL_USE_SDC TRUE
  Enables the SDC subsystem.
• #define HAL_USE_SERIAL TRUE
  Enables the SERIAL subsystem.
• #define HAL_USE_SERIAL_USB TRUE
  Enables the SERIAL over USB subsystem.
• #define HAL_USE_SIO TRUE
Enables the SIO subsystem.

- `#define HAL_USE_SPI TRUE` Enables the SPI subsystem.

- `#define HAL_USE_TRNG TRUE` Enables the TRNG subsystem.

- `#define HAL_USE_UART TRUE` Enables the UART subsystem.

- `#define HAL_USE_USB TRUE` Enables the USB subsystem.

- `#define HAL_USE_WDG TRUE` Enables the WDG subsystem.

- `#define HAL_USE_WSPI TRUE` Enables the WSPI subsystem.

- `#define PAL_USE_CALLBACKS FALSE` Enables synchronous APIs.

- `#define PAL_USE_WAIT FALSE` Enables synchronous APIs.

- `#define ADC_USE_WAIT TRUE` Enables synchronous APIs.

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

- `#define CAN_USE_SLEEP_MODE TRUE` Sleep mode related APIs inclusion switch.

- `#define CAN_ENFORCE_USE_CALLBACKS FALSE` Enforces the driver to use direct callbacks rather than OSAL events.

- `#define HAL_CRY_USE_FALLBACK FALSE` Enables the SW fall-back of the cryptographic driver.

- `#define HAL_CRY_ENFORCE_FALLBACK FALSE` Makes the driver forcibly use the fall-back implementations.

- `#define DAC_USE_WAIT TRUE` Enables synchronous APIs.

- `#define DAC_USE_MUTUAL_EXCLUSION TRUE` Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

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

- `#define MAC_USE_ZERO_COPY TRUE` Enables the zero-copy API.

- `#define MAC_USE_EVENTS TRUE` Enables an event sources for incoming packets.

- `#define MMC_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 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.

9.123.1 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.

9.124 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.
9.124.1 Detailed Description

Memory streams code.

9.125 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.

9.125.1 Detailed Description

Memory streams structures and macros.

9.126 nullstreams.c File Reference

Null streams code.

#include "hal.h"
#include "nullstreams.h"

Functions

- void nullObjectInit (NullStream∗ nsp)
  
  Null stream object initialization.
9.126.1 Detailed Description

Null streams code.

9.127 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_sequential_stream_data`
  
  `NullStream` specific data.

**Functions**

- `void nullObjectInit (NullStream *nsp)`
  
  `Null stream object initialization`.

9.127.1 Detailed Description

Null streams structures and macros.

9.128 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.
9.128.1 Detailed Description

OSAL module code.

9.129 osal.h File Reference

OSAL module header.

```c
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

Data Structures

- `struct event_source`
  
  Events source object.

- `struct threads_queue_t`
  
  Type of a thread queue.

Macros

- `#define OSAL_DBG_ENABLE_ASSERTS FALSE`
  
    Enables OSAL assertions.

- `#define OSAL_DBG_ENABLE_CHECKS FALSE`
  
    Enables OSAL functions parameters checks.

Common constants

- `#define FALSE 0`
- `#define TRUE 1`
- `#define OSAL_SUCCESS false`
- `#define OSAL_FAILED true`

Messages

- `#define MSG_OK (msg_t)0`
- `#define MSG_TIMEOUT (msg_t)-1`
- `#define MSG_RESET (msg_t)-2`

Special time constants

- `#define TIME_IMMEDIATE ((sysinterval_t)0)`
- `#define TIME_INFINITE ((sysinterval_t)-1)`

Systick modes.

- `#define OSAL_ST_MODE_NONE 0`
- `#define OSAL_ST_MODE_PERIODIC 1`
- `#define OSAL_ST_MODE_FREERUNNING 2`
Systick parameters.

- `#define OSAL_ST_RESOLUTION 32`
  Size in bits of the `systick_t` type.
- `#define OSAL_ST_FREQUENCY 1000`
  Required systick frequency or resolution.
- `#define OSAL_ST_MODE OSAL_ST_MODE_PERIODIC`
  Systick mode required by the underlying OS.

IRQ-related constants

- `#define OSAL_IRQ_PRIORITY_LEVELS 16U`
  Total priority levels.
- `#define OSAL_IRQ_MAXIMUM_PRIORITY 0U`
  Highest IRQ priority for HAL drivers.

Debug related macros

- `#define osalDbgAssert(c, remark)`
  Condition assertion.
- `#define osalDbgCheck(c)`
  Function parameters check.
- `#define osalDbgCheckClassI()`
  I-Class state check.
- `#define osalDbgCheckClassS()`
  S-Class state check.

IRQ service routines wrappers

- `#define OSAL_IRQ_IS_VALID_PRIORITY(n) (((n) > OSAL_IRQ_MAXIMUM_PRIORITY) && ((n) < OSAL_IRQ_PRIORITY_LEVELS))`
  Priority level verification macro.
- `#define OSAL_IRQ_PROLOGUE()`
  IRQ prologue code.
- `#define OSAL_IRQ_EPILOGUE()`
  IRQ epilogue code.
- `#define OSAL_IRQ_HANDLER(id) void id(void)`
  IRQ handler function declaration.

Time conversion utilities

- `#define OSAL_S2I(secs) ((sysinterval_t)((uint32_t)(secs) * (uint32_t)OSAL_ST_FREQUENCY))`
  Seconds to system ticks.
- `#define OSAL_MS2I(msecs)`
  Milliseconds to system ticks.
- `#define OSAL_US2I(usecs)`
  Microseconds to system ticks.

Time conversion utilities for the realtime counter

- `#define OSAL_S2RTC(freq, sec) ((((freq) * (sec)) / 1000UL) + 999UL)`
  Seconds to realtime counter.
- `#define OSAL_MS2RTC(freq, msec) (((freq) * (msec)) / 1000000UL)`
  Milliseconds to realtime counter.
- `#define OSAL_US2RTC(freq, usec) (((freq) * (usec)) / 1000000000UL)`
  Microseconds to realtime counter.

Sleep macros using absolute time

- `#define osalThreadSleepSeconds(secs) osalThreadSleep(OSAL_S2I(secs))`
  Delays the invoking thread for the specified number of seconds.
- `#define osalThreadSleepMilliseconds(msecs) osalThreadSleep(OSAL_MS2I(msecs))`
  Delays the invoking thread for the specified number of milliseconds.
- `#define osalThreadSleepMicroseconds(usecs) osalThreadSleep(OSAL_US2I(usecs))`
  Delays the invoking thread for the specified number of microseconds.
Typedefs

- `typedef uint32_t syssts_t`
  Type of a system status word.
- `typedef int32_t msg_t`
  Type of a message.
- `typedef uint32_t systime_t`
  Type of system time counter.
- `typedef uint32_t sysinterval_t`
  Type of system time interval.
- `typedef 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)`
  Wakes up a thread waiting on a thread reference object.
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)
  Adds flags to an event source object.
- void osalEventBroadcastFlags (event_source_t *esp, eventflags_t flags)
  Adds 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 osalTimeIsInRangeX (systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time window.
- static void osalThreadQueueObjectInit (threads_queue_t *tqp)
  Initializes a threads queue object.
- static void osalEventObjectInit (event_source_t *esp)
  Initializes an event source object.
- static void osalMutexObjectInit (mutex_t *mp)
  Initializes mutex_t object.

Variables

- const char * osal_halt_msg
  Pointer to a halt error message.

9.129.1 Detailed Description

OSAL module header.
Index

_CHIBIOS_HAL_
  HAL Driver, 24

_IOBUS_DATA_
  PAL Driver, 467

_adc_isr_error_code
  ADC Driver, 37

_adc_isr_full_code
  ADC Driver, 36

_adc_isr_half_code
  ADC Driver, 35

_adc_reset_i
  ADC Driver, 34

_adc_reset_s
  ADC Driver, 34

_adc_timeout_isr
  ADC Driver, 35

_adc_wakeup_isr
  ADC Driver, 35

_base_asynchronous_channel_data
  Abstract I/O Channel Interface, 315

_base_asynchronous_channel_methods
  Abstract I/O Channel Interface, 315

_base_block_device_data
  Abstract I/O Block Device, 341

_base_block_device_methods
  Abstract I/O Block Device, 341

_base_channel_data
  Abstract I/O Channel Interface, 309

_base_channel_methods
  Abstract I/O Channel Interface, 309

_base_flash_data
  Generic Flash Interface, 221

_base_flash_methods
  Generic Flash Interface, 220

_base_flash_methods_alone
  Generic Flash Interface, 220

_base_object_data
  Base Object, 455

_base_object_methods
  Base Object, 455

_base_pers_storage_methods
  Abstract Persistent Storage Interface, 326

_base_pers_storage_methods_alone
  Abstract Persistent Storage Interface, 326

_base_persistent_storage_data
  Abstract Persistent Storage Interface, 326

_base_sequential_stream_data
  Abstract Streams Interface, 304

_base_sequential_stream_methods
  Abstract Streams Interface, 303

_can_error_isr
  CAN Driver, 86

_can_rx_full_isr
  CAN Driver, 85

_can_tx_empty_isr
  CAN Driver, 85

_can_wakeup_isr
  CAN Driver, 86

_dac_isr_error_code
  DAC Driver, 188

_dac_isr_full_code
  DAC Driver, 187

_dac_isr_half_code
  DAC Driver, 186

_dac_reset_i
  DAC Driver, 185

_dac_reset_s
  DAC Driver, 185

_dac_timeout_isr
  DAC Driver, 186

_dac_wait_s
  DAC Driver, 184

_dac_wakeup_isr
  DAC Driver, 184

_efl_driver_data
  EFL Driver, 207

_efl_flash_methods
  EFL Driver, 206

_efl_flash_methods_alone
  EFL Driver, 206

_file_stream_data
  Abstract Files Interface, 319

_file_stream_methods
  Abstract Files Interface, 318

_gpt_isr_invoke_cb
  GPT Driver, 232

_i2c_wakeup_error_isr
  I2C Driver, 252

_i2c_wakeup_isr
  I2C Driver, 251

_i2s_isr_full_code
  I2S Driver, 268

_i2s_isr_half_code
  I2S Driver, 267

_icu_isr_invoke_overflow_cb
  ICU Driver, 285

_icu_isr_invoke_period_cb
  ICU Driver, 284
Abstract Files Interface, 317
  _file_stream_data, 319
  _file_stream_methods, 318
  FILE_EOF, 318
  FILE_ERROR, 318
  FILE_OK, 318
  fileoffset_t, 324
  fileStreamClose, 321
  fileStreamGet, 321
  fileStreamGetError, 322
  fileStreamGetPosition, 323
  fileStreamGetSize, 323
  fileStreamPut, 320
  fileStreamRead, 320
  fileStreamSetPosition, 324
  fileStreamWrite, 319

Abstract I/O Block Device, 340
  _base_block_device_data, 341
  _base_block_device_methods, 341
  BLK_ACTIVE, 348
  BLK_CONNECTING, 348
  BLK_DISCONNECTING, 348
  BLK_READING, 348
  BLK_READY, 348
  BLK_STOP, 348
  BLK_SYNCING, 348
  BLK_UNINIT, 348
  BLK_WRITING, 348
  blkConnect, 344
  blkDisconnect, 345
  blkGetDriverState, 342
  blkGetInfo, 347
  blkIsInserted, 343
  blkIsTransferring, 342
  blkIsWriteProtected, 343
  blkRead, 345
  blkstate_t, 348
  blkSync, 346
  blkWrite, 346

Abstract I/O Channel Interface, 307
  _base_asynchronous_channel_data, 315
  _base_asynchronous_channel_methods, 315
  _base_channel_data, 309
  _base_channel_methods, 309
  CHN_CONNECTED, 314
  CHN_CTL_INVALID, 308
  CHN_CTL_NOP, 308
  CHN_CTL_TX_WAIT, 309
  CHN_DISCONNECTED, 314
  CHN_INPUT_AVAILABLE, 314
  CHN_NO_ERROR, 314
  CHN_OUTPUT_EMPTY, 314
  CHN_TRANSMISSION_END, 315
  chnAddFlagsI, 316
  chnControl, 313
  chnGetEventSource, 315
  chnGetTimeout, 310
  chnPutTimeout, 309
  chnRead, 312
  chnReadTimeout, 312
  chnWrite, 311
  chnWriteTimeout, 311

Abstract Persistent Storage Interface, 325
  _base_pers_storage_methods, 326
  _base_pers_storage_methods_alone, 326
  _base_persistent_storage_data, 326
  getBasePersistentStorage, 326
  ps_error_t, 328
  ps_offset_t, 328
  psGetStorageSize, 326
  psRead, 327
  psWrite, 327

Abstract Streams Interface, 303
  _base_sequential_stream_data, 304
  _base_sequential_stream_methods, 303
  streamGet, 306
  streamPut, 305
  streamRead, 305
  streamWrite, 304

ADC Driver, 29
  _adc_isr_error_code, 37
  _adc_isr_full_code, 36
  _adc_isr_half_code, 35
  _adc_reset_i, 34
  _adc_reset_s, 34
  _adc_timeout_isr, 35
  _adc_wakeup_isr, 35
  ADC_ACTIVE, 41
  adc_channels_num_t, 40
  ADC_COMPLETE, 41
  ADC_ERR_AWD, 41
  ADC_ERR_DMAFAILURE, 41
  ADC_ERR_OVERFLOW, 41
  ADC_ERROR, 41
  adc_lld_config_fields, 38
  adc_lld_configuration_group_fields, 38
  adc_lld_driver_fields, 38
  adc_lld_init, 50
  adc_lld_start, 50
  adc_lld_start_conversion, 51
  adc_lld_stop, 50
  adc_lld_stop_conversion, 51
  ADC_READY, 41
  ADC_STOP, 41
  ADC_UNINIT, 41
  ADC_USE_MUTUAL_EXCLUSION, 33
  ADC_USE_WAIT, 33
  adcAcquireBus, 48
  adccallback_t, 39
  ADCConfig, 39
  ADCConversionGroup, 39
  adcConvert, 47
  ADCD1, 51
  ADCDriver, 39
adcerror_t, 41
adcerrorcallback_t, 40
adcInit, 41
adcIsBufferComplete, 33
adcObjectInit, 42
adcReleaseBus, 49
adcsample_t, 40
adcStart, 42
adcStartConversion, 44
adcStartConversionI, 45
adcstate_t, 40
adcStop, 43
adcStopConversion, 46
adcStopConversionI, 46
PLATFORM_ADC_USE_ADC1, 38

ADC_ACTIVE
ADC Driver, 41
adc_channels_num_t
ADC Driver, 40
ADC_COMPLETE
ADC Driver, 41
ADC_ERR_AWD
ADC Driver, 41
ADC_ERR_DMAFAILURE
ADC Driver, 41
ADC_ERR_OVERFLOW
ADC Driver, 41
ADC_ERROR
ADC Driver, 41
adc_lld_config_fields
ADC Driver, 38
adc_lld_configuration_group_fields
ADC Driver, 38
adc_lld_driver_fields
ADC Driver, 38
adc_lld_init
ADC Driver, 50
adc_lld_start
ADC Driver, 50
adc_lld_start_conversion
ADC Driver, 51
adc_lld_stop
ADC Driver, 50
adc_lld_stop_conversion
ADC Driver, 51
ADC_READY
ADC Driver, 41
ADC_STOP
ADC Driver, 41
ADC_UNINIT
ADC Driver, 41
ADC_USE_MUTUAL_EXCLUSION
ADC Driver, 33
Configuration, 848
ADC_USE_WAIT
ADC Driver, 33
Configuration, 848
adcAcquireBus

ADC Driver, 48
adccallback_t
ADC Driver, 39
ADCConfig
ADC Driver, 39
ADCConversionGroup
ADC Driver, 39
adcConvert
ADC Driver, 47
ADCD1
ADC Driver, 51
ADCDriver
ADC Driver, 39
adcerror_t
ADC Driver, 41
adcerrorcallback_t
ADC Driver, 40
adcInit
ADC Driver, 41
adcIsBufferComplete
ADC Driver, 33
adcObjectInit
ADC Driver, 42
adcReleaseBus
ADC Driver, 49
adcsample_t
ADC Driver, 40
adcStart
ADC Driver, 42
adcStartConversion
ADC Driver, 44
adcStartConversionI
ADC Driver, 45
adcstate_t
ADC Driver, 40
adcStop
ADC Driver, 43
adcStopConversion
ADC Driver, 46
adcStopConversionI
ADC Driver, 46
addr
USBDriver, 1062

wspi_command_t, 1072
address
flash_descriptor_t, 936

MII/RMII Header, 401
ADVERTISE_100BASE4
MII/RMII Header, 400
ADVERTISE_100FULL
MII/RMII Header, 400
ADVERTISE_100HALF
MII/RMII Header, 400
ADVERTISE_10FULL
MII/RMII Header, 400
ADVERTISE_10HALF
MII/RMII Header, 400
ADVERTISE_CSMA
MII/RMII Header, 400
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>state, 919</td>
<td></td>
<td></td>
</tr>
<tr>
<td>txempty_event, 919</td>
<td></td>
<td></td>
</tr>
<tr>
<td>txqueue, 919</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wakeup_event, 920</td>
<td></td>
<td></td>
</tr>
<tr>
<td>canInit</td>
<td>CAN Driver, 88</td>
<td></td>
</tr>
<tr>
<td>canmbx_t</td>
<td>CAN Driver, 87</td>
<td></td>
</tr>
<tr>
<td>canObjectInit</td>
<td>CAN Driver, 89</td>
<td></td>
</tr>
<tr>
<td>canReceive</td>
<td>CAN Driver, 85</td>
<td></td>
</tr>
<tr>
<td>canReceiveTimeout</td>
<td>CAN Driver, 95</td>
<td></td>
</tr>
<tr>
<td>CANRxFrame, 921</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data16, 923</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data32, 923</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data8, 923</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLC, 922</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EID, 923</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMI, 922</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDE, 923</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTR, 922</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SID, 923</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME, 922</td>
<td></td>
<td></td>
</tr>
<tr>
<td>canSleep</td>
<td>CAN Driver, 96</td>
<td></td>
</tr>
<tr>
<td>canStart</td>
<td>CAN Driver, 89</td>
<td></td>
</tr>
<tr>
<td>canstate_t</td>
<td>CAN Driver, 88</td>
<td></td>
</tr>
<tr>
<td>canStop</td>
<td>CAN Driver, 88</td>
<td></td>
</tr>
<tr>
<td>canTransmit</td>
<td>CAN Driver, 85</td>
<td></td>
</tr>
<tr>
<td>canTransmitTimeout</td>
<td>CAN Driver, 93</td>
<td></td>
</tr>
<tr>
<td>canTryAbortX</td>
<td>CAN Driver, 93</td>
<td></td>
</tr>
<tr>
<td>canTryReceiveI</td>
<td>CAN Driver, 92</td>
<td></td>
</tr>
<tr>
<td>canTryTransmitI</td>
<td>CAN Driver, 91</td>
<td></td>
</tr>
<tr>
<td>CANTxFrame, 924</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data16, 926</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data32, 926</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data8, 926</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLC, 925</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EID, 926</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDE, 925</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTR, 925</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SID, 925</td>
<td></td>
<td></td>
</tr>
<tr>
<td>canWakeup</td>
<td>CAN Driver, 97</td>
<td></td>
</tr>
<tr>
<td>cardmode</td>
<td>SDCDriver, 1028</td>
<td></td>
</tr>
<tr>
<td>cb</td>
<td>event_source, 931</td>
<td></td>
</tr>
<tr>
<td>palevent_t, 1013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdc_linecoding_t, 927</td>
<td></td>
<td></td>
</tr>
<tr>
<td>config</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wspli_command_t, 1072</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH_HAL_MAJOR</td>
<td>HAL Driver, 24</td>
<td></td>
</tr>
<tr>
<td>CH_HAL_MINOR</td>
<td>HAL Driver, 25</td>
<td></td>
</tr>
<tr>
<td>CH_HAL_PATCH</td>
<td>HAL Driver, 25</td>
<td></td>
</tr>
<tr>
<td>CH_HAL_STABLE</td>
<td>HAL Driver, 24</td>
<td></td>
</tr>
<tr>
<td>channels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWMDriver, 1019</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHN_CONNECTED</td>
<td>Abstract I/O Channel Interface, 314</td>
<td></td>
</tr>
<tr>
<td>CHN_CTL_INVALID</td>
<td>Abstract I/O Channel Interface, 308</td>
<td></td>
</tr>
<tr>
<td>CHN_CTL_NOP</td>
<td>Abstract I/O Channel Interface, 308</td>
<td></td>
</tr>
<tr>
<td>CHN_CTL_TX_WAIT</td>
<td>Abstract I/O Channel Interface, 309</td>
<td></td>
</tr>
<tr>
<td>CHN_DISCONNECTED</td>
<td>Abstract I/O Channel Interface, 314</td>
<td></td>
</tr>
<tr>
<td>CHN_INPUT_AVAILABLE</td>
<td>Abstract I/O Channel Interface, 314</td>
<td></td>
</tr>
<tr>
<td>CHN_NO_ERROR</td>
<td>Abstract I/O Channel Interface, 314</td>
<td></td>
</tr>
<tr>
<td>CHN_OUTPUT_EMPTY</td>
<td>Abstract I/O Channel Interface, 314</td>
<td></td>
</tr>
<tr>
<td>CHN_TRANSMISSION_END</td>
<td>Abstract I/O Channel Interface, 315</td>
<td></td>
</tr>
<tr>
<td>chnAddFlagsI</td>
<td>Abstract I/O Channel Interface, 316</td>
<td></td>
</tr>
<tr>
<td>chnControl</td>
<td>Abstract I/O Channel Interface, 313</td>
<td></td>
</tr>
<tr>
<td>chnGetEventSource</td>
<td>Abstract I/O Channel Interface, 315</td>
<td></td>
</tr>
<tr>
<td>chnGetTimeout</td>
<td>Abstract I/O Channel Interface, 310</td>
<td></td>
</tr>
<tr>
<td>chnPutTimeout</td>
<td>Abstract I/O Channel Interface, 309</td>
<td></td>
</tr>
<tr>
<td>chnRead</td>
<td>Abstract I/O Channel Interface, 312</td>
<td></td>
</tr>
<tr>
<td>chnReadTimeout</td>
<td>Abstract I/O Channel Interface, 312</td>
<td></td>
</tr>
<tr>
<td>chnWrite</td>
<td>Abstract I/O Channel Interface, 311</td>
<td></td>
</tr>
<tr>
<td>chnWriteTimeout</td>
<td>Abstract I/O Channel Interface, 311</td>
<td></td>
</tr>
<tr>
<td>chprintf</td>
<td>Output Formatter Utility, 335</td>
<td></td>
</tr>
<tr>
<td>chprintf.c, 1073</td>
<td></td>
<td></td>
</tr>
<tr>
<td>chprintf.h, 1073</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHPRINTF_USE_FLOAT</td>
<td>Output Formatter Utility, 334</td>
<td></td>
</tr>
<tr>
<td>chsnprintf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ChibiOS/HAL
INDEX 1239

Output Formatter Utility, 336

chvprintf
Output Formatter Utility, 334

chvsnprintf
Output Formatter Utility, 338

circular
hal_adc_configuration_group, 943
hal_spi_config, 961

cmd
wspi_command_t, 1072

Complex Drivers, 857

config
CANDriver, 919
CRYDriver, 928
GPTDriver, 940
hal_adc_driver, 945
hal_dac_driver, 950
hal_i2s_driver, 956
hal_sio_driver, 960
hal_spi_driver, 963
hal_trng_driver, 965
hal_wspi_driver, 968
I2CDriver, 971
ICUDriver, 974
MACDriver, 985
MFSDriver, 998
MMCDriver, 1005
PWMDriver, 1019
SDCDriver, 1028
SNORDriver, 1046
UARTDriver, 1051
USBDriver, 1060
WDGDriver, 1071

Configuration, 841

ADC_USE_MUTUAL_EXCLUSION, 848
ADC_USE_WAIT, 848
CAN_ENFORCE_USE_CALLBACKS, 848
CAN_USE_SLEEP_MODE, 848
DAC_USE_MUTUAL_EXCLUSION, 849
DAC_USE_WAIT, 849
HAL_CRY_ENFORCE_FALLBACK, 849
HAL_CRY_USE_FALLBACK, 848
HAL_USE_ADC, 843
HAL_USE_CAN, 843
HAL_USE_CRY, 844
HAL_USE_DAC, 844
HAL_USE_EFL, 844
HAL_USE_GPT, 844
HAL_USE_I2C, 844
HAL_USE_I2S, 844
HAL_USE_ICU, 845
HAL_USE_MAC, 845
HAL_USE_MMC_SPI, 845
HAL_USE_PAL, 843
HAL_USE_PWM, 845
HAL_USE_RTC, 845
HAL_USE_SDC, 845
HAL_USE_SERIAL, 846
HAL_USE_SERIAL_USB, 846
HAL_USE_SIO, 846
HAL_USE_SPI, 846
HAL_USE_TRNG, 846
HAL_USE_UART, 846
HAL_USE_USB, 847
HAL_USE_WDG, 847
HAL_USE_WSPI, 847
I2C_USE_MUTUAL_EXCLUSION, 849
MAC_USE_EVENTS, 850
MAC_USE_ZERO_COPY, 850
MMC_NICE_WAITING, 850
PAL_USE_CALLBACKS_COPY, 850
PAL_USE_CALLBACKS, 847
PAL_USE_WAIT, 847
SDC_INIT_OCR, 851
SDC_INIT_OCR_V20, 851
SDC_INIT_RETRY, 850
SDC_MMC_SUPPORT, 850
SDC_NICE_WAITING, 851
SERIAL_BUFFERS_SIZE, 851
SERIAL_DEFAULT_BITRATE, 851
SERIAL_USB_BUFFERS_NUMBER, 852
SERIAL_USB_BUFFERS_SIZE, 852
SPI_SELECT_MODE, 853
SPI_USE_CIRCULAR, 852
SPI_USE_MUTUAL_EXCLUSION, 853
SPI_USE_WAIT, 852
UART_USE_MUTUAL_EXCLUSION, 853
UART_USE_WAIT, 853
USB_USE_WAIT, 854
WSPI_USE_MUTUAL_EXCLUSION, 854
WSPI_USE_WAIT, 854

configuration
USBDriver, 1062

counter
mfs_bank_header_t, 991
crc
mfs_bank_header_t, 991
mfs_data_header_t, 993
crc7
MMC over SPI Driver, 409
crc7_lookup_table
MMC over SPI Driver, 429
cry_algo_aes
Crypto Driver, 112
cry_algo_des
Crypto Driver, 112
cry_algo_hmac
Crypto Driver, 112
CRY_ERR_AUTH_FAILED
Crypto Driver, 112
CRY_ERR_INV_ALGO
Crypto Driver, 111
CRY_ERR_INV_KEY_ID
Crypto Driver, 112
CRY_ERR_INV_KEY_SIZE
Crypto Driver, 111
CRY_ERR_INV_KEY_TYPE
Crypto Driver, 111

ChibiOS/HAL
cryEncryptAES
  Crypto Driver, 116
cryEncryptAES_CBC
  Crypto Driver, 120
cryEncryptAES_CBC
  Crypto Driver, 122
cryEncryptAES_CTX
  Crypto Driver, 124
cryEncryptAES_ECB
  Crypto Driver, 124
cryEncryptAES_CBC
  Crypto Driver, 118
cryEncryptAES_GCM
  Crypto Driver, 126
cryEncryptDES
  Crypto Driver, 130
cryEncryptDES_CBC
  Crypto Driver, 134
cryEncryptDES_ECB
  Crypto Driver, 132
crerror_t
  Crypto Driver, 111
cryHMACSHA256Final
  Crypto Driver, 147
cryHMACSHA256Init
  Crypto Driver, 146
cryHMACSHA256Update
  Crypto Driver, 147
cryHMACSHA512Final
  Crypto Driver, 150
cryHMACSHA512Init
  Crypto Driver, 148
cryHMACSHA512Update
  Crypto Driver, 149
cryInit
  Crypto Driver, 112
cry_key_t
  Crypto Driver, 110
cryLoadAESTransientKey
  Crypto Driver, 115
cryLoadDESTransientKey
  Crypto Driver, 129
cryLoadHMACTransientKey
  Crypto Driver, 145
cryObjectInit
  Crypto Driver, 113
Crypto Driver, 104
bitsize_t
  110
cry_algo_aes
  112
cry_algo_des
  112
cryalgo_t
  112
cry_decrypt_AES_ECB
  155
cry_decrypt_AES_CFB
  159
cry_decrypt_AES_CTR
  160
cry_decrypt_AES_CBC
  162
cry_decrypt_DES
  165
cry_decrypt_DES_CBC
  168
cry_decrypt_DES_ECB
  167
cry_decrypt_AES
  153
cry_decrypt_AES_CBC
  156
cry_decrypt_AES_CFB
  158
cry_decrypt_AES_CTR
  160
cry_decrypt_AES_CBC
  154
cry_decrypt_AES_GCM
  161
cry_decrypt_DES
  164
cry_decrypt_DES_CBC
  168
cry_decrypt_DES_ECB
  166
cry_hmac_loadkey
  175
cry_HMACSHA256_final
  177
cry_HMACSHA256_init
  176
cry_HMACSHA256_update
  179
cry_HMACSHA512_final
  179
cry_HMACSHA512_init
  178
cry_HMACSHA512_update
  178
cry_sha1_init
  169
cry_sha1_update
  170
cry_sha256_final
  173
cry_sha256_init
  171
cry_sha256_update
  172
cry_sha512_final
  174
cry_sha512_init
  173
cry_sha512_update
  174
cry_init
  151
cry_start
  151
cry_stop
  152
CRY_NOERROR
  111
CRY_READY
  111
CRY_DRIVER
  111
CryAlgorithm_t
  112
CRYD1
  180
cry_decryptAES
  117
cry_decryptAES_CBC
  121
cry_decryptAES_CFB
  123
cry_decryptAES_CTR
  125
cry_decryptAES_ECB
  119
cry_decryptAES_GCM
  127
cryDecryptAES
  131
cryDecryptAES_CBC
  135
cryDecryptAES_ECB
  133
CRYDriver
  111
cry_encryptAES
  116
cry_encryptAES_CBC
  120
cry_encryptAES_CFB
  122
cry_encryptAES_CTR
  124
cry_encryptAES_ECB
  118
cry_encryptAES_GCM
  126
cryEncryptDES, 130
cryEncryptDES_CBC, 134
cryEncryptDES_ECB, 132
cryerror_t, 111
cryHMACSHA256Final, 147
cryHMACSHA256Init, 146
cryHMACSHA256Update, 147
cryHMACSHA512Final, 150
cryHMACSHA512Init, 148
cryHMACSHA512Update, 149
cryInit, 112
crykey_t, 110
cryLoadAESTransientKey, 115
cryLoadDESTransientKey, 129
cryLoadHMACTransientKey, 145
cryObjectInit, 113
crySHA1Final, 138
crySHA1Init, 136
crySHA1Update, 137
crySHA256Final, 141
crySHA256Init, 139
crySHA256Update, 140
crySHA512Final, 144
crySHA512Init, 142
crySHA512Update, 143
cryStart, 113
crystate_t, 111
cryStop, 114
HAL_CRY_ENFORCE_FALLBACK, 110
HAL_CRY_USE_FALLBACK, 109
PLATFORM_CRY_USE_CRY1, 110
crySHA1Final
Crypto Driver, 138
crySHA1Init
Crypto Driver, 136
crySHA1Update
Crypto Driver, 137
crySHA256Final
Crypto Driver, 141
crySHA256Init
Crypto Driver, 139
crySHA256Update
Crypto Driver, 140
crySHA512Final
Crypto Driver, 144
crySHA512Init
Crypto Driver, 142
crySHA512Update
Crypto Driver, 143
cryStart
Crypto Driver, 113
crystate_t
Crypto Driver, 111
cryStop
Crypto Driver, 114
current_bank
MFSDriver, 999
current_counter
MFSDriver, 999
DAC Driver, 181
dac_isr_error_code, 188
dac_isr_full_code, 187
dac_isr_half_code, 186
dac_reset_i, 185
dac_reset_s, 185
dac_timeout_isr, 186
dac_wait_s, 184
dac_wakeup_isr, 186
DAC_ACTIVE, 191
DAC_COMPLETE, 191
DAC_ERR_DMAFAILURE, 192
DAC_ERR_UNDERSFLOW, 192
DAC_ERROR, 191
dac_lld_config_fields, 189
dac_lld_conversion_group_fields, 189
dac_lld_driver_fields, 189
dac_lld_init, 201
dac_lld_put_channel, 203
dac_lld_start, 202
dac_lld_start_conversion, 203
dac_lld_stop, 202
dac_lld_stop_conversion, 204
DAC_MAX_CHANNELS, 188
DAC_READY, 191
DAC_STOP, 191
DAC_UNINIT, 191
DAC_USE_MUTUAL_ExCLUSION, 183
DAC_USE_WAIT, 183
dacAcquireBus, 200
daccallback_t, 190
dacchannel_t, 191
DACConfig, 190
DACConversionGroup, 190
dacConvert, 199
DACD1, 204
DACDriver, 190
dacerror_t, 191
dacerrorcallback_t, 190
dacInit, 192
dacIsBufferComplete, 184
dacObjectInit, 192
dacPutChannelX, 195
dacReleaseBus, 201
dacsample_t, 191
dacStart, 193
dacStartConversion, 196
dacStartConversionI, 196
dacstate_t, 191
dacStop, 194
dacStopConversion, 197
dacStopConversionI, 198
PLATFORM_DAC_USE_DAC1, 189
DAC_ACTIVE
DAC Driver, 191
DAC_COMPLETE
DAC Driver, 191
DAC_ERR_DMAFAILURE
DAC Driver, 192

DAC_ERR_UNDERFLOW
DAC Driver, 192

DAC_ERROR
DAC Driver, 191
dac_lld_config_fields
DAC Driver, 189
dac_lld_conversion_group_fields
DAC Driver, 189
dac_lld_driver_fields
DAC Driver, 189
dac_lld_init
DAC Driver, 201
dac_lld_put_channel
DAC Driver, 203
dac_lld_start
DAC Driver, 202
dac_lld_start_conversion
DAC Driver, 203
dac_lld_stop
DAC Driver, 202
dac_lld_stop_conversion
DAC Driver, 204
DAC_MAX_CHANNELS
DAC Driver, 188
DAC_READY
DAC Driver, 191
DAC_STOP
DAC Driver, 191
DAC_UNINIT
DAC Driver, 191
DAC_USE_MUTUAL_EXCLUSION
Configuration, 849
DAC Driver, 183
DAC_USE_WAIT
Configuration, 849
DAC Driver, 183
dacAcquireBus
DAC Driver, 200
daccallback_t
DAC Driver, 190
dacchannel_t
DAC Driver, 191
DACConfig
DAC Driver, 190
DACConversionGroup
DAC Driver, 190
dacConvert
DAC Driver, 199
DACD1
DAC Driver, 204
DACDriver
DAC Driver, 190
dacerror_t
DAC Driver, 191
dacerrorcallback_t
DAC Driver, 190
dacInit
DAC Driver, 192
dacIsBufferComplete
DAC Driver, 184
dacObjectInit
DAC Driver, 192
dacPutChannelX
DAC Driver, 195
dacReleaseBus
DAC Driver, 201
dacsample_t
DAC Driver, 191
dacStart
DAC Driver, 193
dacStartConversion
DAC Driver, 196
dacStartConversionI
DAC Driver, 196
dacstate_t
DAC Driver, 191
dacStop
DAC Driver, 194
dacStopConversion
DAC Driver, 197
dacStopConversionI
DAC Driver, 198
data16
CANRxFrame, 923
CANTxFrame, 926
data32
CANRxFrame, 923
CANTxFrame, 926
data8
CANRxFrame, 923
CANTxFrame, 926
day
RTCDateTime, 1022
dayOfWeek
RTCDateTime, 1022
default_config
Serial Driver, 631
default_handler
USB Driver, 778
depth
hal_adc_driver, 945
hal_dac_driver, 950
descriptors
MFSDDriver, 999
detect_bus_clk
SDC Driver, 592
device_id
SNORDriver, 1046
DLC
CANRxFrame, 922
CANTxFrame, 925
dstflag
RTCDateTime, 1021
dummy
early UARTDriver, 1051
EFL Driver, 205
__efl_driver_data, 207
__efl_flash_methods, 206
__efl_flash_methods_alone, 206
efl_lld_config_fields, 207
efl_lld_driver_fields, 207
efl_lld_get_descriptor, 211
efl_lld_init, 210
efl_lld_program, 212
efl_lld_query_erase, 214
efl_lld_read, 212
efl_lld_start, 210
efl_lld_start_erase_all, 213
efl_lld_start_erase_sector, 214
efl_lld_stop, 211
efl_lld_verify_erase, 215
EFlashConf, 208
EFlashDriver, 208
EFLD1, 215
eflInit, 208
eflObjectInit, 208
eflStart, 209
eflStop, 209
PLATFORM_EFL_USE_EFL1, 207
efl_lld_config_fields
  EFL Driver, 207
efl_lld_driver_fields
  EFL Driver, 207
efl_lld_get_descriptor
  EFL Driver, 211
efl_lld_init
  EFL Driver, 210
efl_lld_program
  EFL Driver, 212
efl_lld_query_erase
  EFL Driver, 214
efl_lld_read
  EFL Driver, 212
efl_lld_start
  EFL Driver, 210
efl_lld_start_erase_all
  EFL Driver, 213
efl_lld_start_erase_sector
  EFL Driver, 214
efl_lld_stop
  EFL Driver, 211
efl_lld_verify_erase
  EFL Driver, 215
EFlashConf
  EFL Driver, 208
EFlashDriver
  EFL Driver, 208
EFlashDriverVMT, 929
EFLD1
  EFL Driver, 215
eflInit
  EFL Driver, 208
eflObjectInit
  EFL Driver, 208
eflStart
  EFL Driver, 209
eflStop
  EFL Driver, 209
EID
  CANRxFrame, 923
  CANTxFrame, 926
enabled
  PWM Driver, 1019
end_cb
  hal_adc_configuration_group, 943
  hal_dac_conversion_group, 948
  hal_i2s_config, 955
  hal_spi_config, 961
  hal_wspi_config, 966
ep0_state
  USB Driver, 800
ep0config
  USB Driver, 801
ep0endcb
  USB Driver, 1062
ep0n
  USB Driver, 1062
ep0next
  USB Driver, 1061
ep0state
  USB Driver, 1061
ep__mode
  USBEndpointConfig, 1065
EP_STATUS_ACTIVE
  USB Driver, 776
EP_STATUS_DISABLED
  USB Driver, 776
EP_STATUS_STALLED
  USB Driver, 776
epc
  USB Driver, 1061
erased
  MFS Config, 996
error_cb
  hal_adc_configuration_group, 943
  hal_dac_conversion_group, 948
  hal_wspi_config, 966
error_event
  CAN Driver, 920
errors
  I2CDriver, 971
  SDCDriver, 1028
ETHD1
  MAC Driver, 368
event_cb
  USB Config, 1057
event_source, 930
cb, 931
<table>
<thead>
<tr>
<th>Documentation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_EOF</td>
<td>318</td>
</tr>
<tr>
<td>FILE_ERROR</td>
<td>318</td>
</tr>
<tr>
<td>FILE_OK</td>
<td>318</td>
</tr>
<tr>
<td>fileoffset_t</td>
<td>324</td>
</tr>
<tr>
<td>FileInputStream</td>
<td>321</td>
</tr>
<tr>
<td>vmt</td>
<td>933</td>
</tr>
<tr>
<td>fileStreamClose</td>
<td>321</td>
</tr>
<tr>
<td>fileStreamGet</td>
<td>321</td>
</tr>
<tr>
<td>fileStreamGetError</td>
<td>322</td>
</tr>
<tr>
<td>fileStreamGetPosition</td>
<td>323</td>
</tr>
<tr>
<td>fileStreamGetSize</td>
<td>323</td>
</tr>
<tr>
<td>fileStreamPut</td>
<td>323</td>
</tr>
<tr>
<td>fileStreamRead</td>
<td>320</td>
</tr>
<tr>
<td>fileStreamGetPosition</td>
<td>324</td>
</tr>
<tr>
<td>FileStreamVMT</td>
<td>934</td>
</tr>
<tr>
<td>fileStreamWrite</td>
<td>319</td>
</tr>
<tr>
<td>flags</td>
<td>930</td>
</tr>
<tr>
<td>event_source</td>
<td>930</td>
</tr>
<tr>
<td>param</td>
<td>931</td>
</tr>
</tbody>
</table>

### Generic Flash Interface, 219
- `FLASH_ATTR_ECC_CAPABLE`
- `FLASH_ATTR_ECC_ZERO_LINE_CAPABLE`
- `FLASH_ATTR_ERASED_IS_ONE`
- `FLASH_ATTR_MEMORY_MAPPED`

- `FLASH_ATTR_REWRITABLE`
- `FLASH_ATTR_SUSPEND_ERASE_CAPABLE`

### MII/RMII Header, 404
- `EXPANSION_ENABLENPAGE`
- `EXPANSION_LCWP`
- `EXPANSION_NPCAPABLE`
- `EXPANSION_NWAY`
- `EXPANSION_RESV`

### Abstract Files Interface, 318
- `FILE_EOF`
- `FILE_ERROR`
- `FILE_OK`

### MFSConfig, 996
- `flash_descriptor_t`
- `flash_error_t`
- `flash_offset_t`
- `flash_sector_descriptor_t`
- `flashp`

### CANRxFrame, 922
- `FMI`

### GPTConfig, 939
- `ICUConfig, 972`
- `PWMConfig, 1017`

### GPTConfig, 939
- `GPTConfig, 939`
- `ICUConfig, 972`
- `PWMConfig, 1017`

- `base_flash_data, 221`
- `base_flash_methods, 220`
- `base_flash_methods_alone, 220`
- `FLASH_ATTR_ECC_CAPABLE, 219`
- `FLASH_ATTR_ECC_ZERO_LINE_CAPABLE, 220`
INDEX

FLASH_ATTR_ERASED_IS_ONE, 219
FLASH_ATTR_MEMORY_MAPPED, 219
FLASH_ATTR_REWRITABLE, 219
FLASH_ATTR_SUSPEND_ERASE_CAPABLE, 220
flash_error_t, 226
flash_offset_t, 225
flash_sector_t, 226
flash_state_t, 226
flashGetDescriptor, 221
flashGetSectorOffset, 227
flashGetSectorSize, 227
flashProgram, 222
flashQueryErase, 224
flashRead, 222
flashStartEraseAll, 223
flashStartEraseSector, 224
flashVerifyErase, 225
flashWaitErase, 226
getBaseFlash, 221
get_descriptor_cb
USBConfig, 1057
getBaseFlash
Generic Flash Interface, 221
getBasePersistentStorage
Abstract Persistent Storage Interface, 326
GPT Driver, 228
_gpt_isr_invoke_cb, 232
GPT_CONTINUOUS, 235
gpt_lld_change_interval, 233
gpt_lld_init, 244
gpt_lld_poled_delay, 245
gpt_lld_start, 244
gpt_lld_start_timer, 245
gpt_lld_stop, 244
gpt_lld_stop_timer, 245
GPT_ONESHOT, 235
GPT_READY, 235
GPT_STOP, 235
GPT_UNINIT, 235
gptcallback_t, 234
gptChangeInterval, 238
gptChangeIntervalI, 231
gptcnt_t, 235
GPTD1, 246
GPTDriver, 234
gptfreq_t, 234
gptGetCounterX, 232
gptGetIntervalX, 231
gptInit, 235
gptObjectInit, 236
gptPolledDelay, 243
gptStart, 236
GPTDriver, 234
GPTDriverContinuous, 239
GPTDriverContinuousI, 239
gptStartOneShot, 240
GPTDriverOneShot, 241
gptstate_t, 235
gptStop, 237
gptStopTimer, 241
GPTDriverStopTimer, 242
GPTDriverStopTimerI, 242
PLATFORM_GPT_USE_GPT1, 233
GPT_CONTINUOUS
GPT Driver, 235
gpt_lld_change_interval
GPT Driver, 233
gpt_lld_init
GPT Driver, 244
gpt_lld_poled_delay
GPT Driver, 245
gpt_lld_start
GPT Driver, 244
gpt_lld_start_timer
GPT Driver, 245
gpt_lld_stop
GPT Driver, 244
gpt_lld_stop_timer
GPT Driver, 245
GPT_ONESHOT
GPT Driver, 235
GPT_READY
GPT Driver, 235
GPT_STOP
GPT Driver, 235
GPT_UNINIT
GPT Driver, 235
gptcallback_t
GPT Driver, 234
gptChangeInterval
GPT Driver, 238
gptChangeIntervalI
GPT Driver, 231
gptcnt_t
GPT Driver, 235
GPTConfig, 938
callback, 939
frequency, 939
GPTD1
GPT Driver, 246
GPTDriver, 940
config, 940
GPT Driver, 234
state, 940
gptfreq_t
GPT Driver, 234
gptGetCounterX
GPT Driver, 232
gptGetIntervalX
GPT Driver, 231
gptInit
GPT Driver, 235
gptObjectInit
GPT Driver, 236
gptPolledDelay
GPT Driver, 243
gptStart
  ChibiOS/HAL
GPT Driver, 236

gptStartContinuous
GPT Driver, 239

gptStartContinuousI
GPT Driver, 239

gptStartOneShot
GPT Driver, 240

gptStartOneShotI
GPT Driver, 241

gptstate_t
GPT Driver, 235

gptStop
GPT Driver, 237

gptStopTimer
GPT Driver, 241

gptStopTimerI
GPT Driver, 242

grpp
  hal_adc_driver, 945
  hal_dac_driver, 950

HAL, 838
HAL Driver, 23
  _CHIBIOS_HAL_, 24
  CH_HAL_MAJOR, 24
  CH_HAL_MINOR, 25
  CH_HAL_PATCH, 25
  CH_HAL_STABLE, 25
  HAL_FAILED, 25
  hal_lld_init, 27
  HAL_SUCCESS, 25
  HAL_VERSION, 24
  halInit, 25

hal.c, 1074
hal.h, 1075
hal_adc.c, 1076
hal_adc.h, 1077
hal_adc_config, 941
hal_adc_configuration_group, 941
  circular, 943
  end_cb, 943
  error_cb, 943
  num_channels, 943
hal_adc_driver, 944
  config, 945
  depth, 945
  grpp, 945
  mutex, 946
  samples, 945
  state, 945
  thread, 946
hal_adc_lld.c, 1079
hal_adc_lld.h, 1080
hal_buffers.c, 1081
hal_buffers.h, 1082
hal_can.c, 1084
hal_can.h, 1085
hal_can_lld.c, 1086
hal_can_lld.h, 1087
hal_channels.h, 1089
HAL_CRY_ENFORCE_FALLBACK
  Configuration, 849
  Crypto Driver, 110
HAL_CRY_USE_FALLBACK
  Configuration, 848
  Crypto Driver, 109
hal_crypto.c, 1090
hal_crypto.h, 1092
hal_crypto_lld.c, 1096
hal_crypto_lld.h, 1098
hal_dac.c, 1101
hal_dac.h, 1102
hal_dac_config, 946
hal_dac_conversion_group, 947
  end_cb, 948
  error_cb, 948
  num_channels, 947
hal_dac_driver, 948
  config, 950
  depth, 950
  grpp, 950
  mutex, 951
  samples, 950
  state, 950
  thread, 950
hal_dac_lld.c, 1104
hal_dac_lld.h, 1105
hal_efi.c, 1106
hal_efi.h, 1106
hal_efi_config, 951
hal_efi_driver, 952
  vmt, 953
hal_efi_lld.c, 1107
hal_efi_lld.h, 1108
HAL_FAILED
  HAL Driver, 25
hal_files.h, 1109
hal_flash.c, 1110
hal_flash.h, 1111
hal_gpt.c, 1113
hal_gpt.h, 1113
hal_gpt_lld.c, 1115
hal_gpt_lld.h, 1115
hal_i2c.c, 1116
hal_i2c.h, 1117
hal_i2c_lld.c, 1119
hal_i2c_lld.h, 1119
hal_i2c.c, 1120
hal_i2c.h, 1121
hal_i2s_config, 954
  end_cb, 955
  rx_buffer, 955
  size, 955
  tx_buffer, 954
hal_i2s_driver, 956
  config, 956
  state, 956
hal_i2s_lld.c, 1122
hal_i2s_lld.h, 1123
hal_icu.c, 1124
hal_icu.h, 1124
hal_icu_lld.c, 1126
hal_icu_lld.h, 1127
hal_ioblock.h, 1128
hal_lld.c, 1129
hal_lld.h, 1130
hal_lld_init
HAL Driver, 27
hal_mac.c, 1130
hal_mac.h, 1131
hal_mac_lld.c, 1132
hal_mac_lld.h, 1133
hal_mfs.c, 1134
hal_mfs.h, 1136
hal_mii.h, 1137
hal_mmc_spi.c, 1140
hal_mmc_spi.h, 1141
hal_mmcsd.c, 1142
hal_mmcsd.h, 1143
hal_objects.h, 1147
hal_pal.c, 1148
hal_pal.h, 1149
hal_pal_lld.c, 1152
hal_pal_lld.h, 1153
hal_persistent.h, 1155
hal_pwm.c, 1156
hal_pwm.h, 1156
hal_pwm_lld.c, 1158
hal_pwm_lld.h, 1159
hal_queues.c, 1161
hal_queues.h, 1162
hal_rtc.c, 1164
hal_rtc.h, 1164
hal_rtc_lld.c, 1166
hal_rtc_lld.h, 1166
hal_sdc.c, 1168
hal_sdc.h, 1169
hal_sdc_lld.c, 1171
hal_sdc_lld.h, 1172
hal_serial.c, 1174
hal_serial.h, 1175
hal_serial_lld.c, 1177
hal_serial_lld.h, 1177
hal_serial_nor.c, 1178
hal_serial_nor.h, 1179
hal_serial_usb.c, 1181
hal_serial_usb.h, 1182
hal_sio.c, 1183
hal_sio.h, 1184
hal_sio_config, 957
  rxevt_cb, 958
  rxne_cb, 958
  txend_cb, 958
  txnf_cb, 958
hal_sio_driver, 959
  config, 960
  state, 960
hal_sio_lld.c, 1185
hal_sio_lld.h, 1186
hal_spi.c, 1187
hal_spi.h, 1188
hal_spi_config, 960
  circular, 961
  end_cb, 961
  ssline, 961
  ssmask, 962
  sspad, 962
  ssport, 961
hal_spi_driver, 962
  config, 963
  mutex, 963
  state, 963
  thread, 963
hal_spi_lld.c, 1190
hal_spi_lld.h, 1191
hal_st.c, 1192
hal_st.h, 1193
hal_st_lld.c, 1194
hal_st_lld.h, 1194
hal_streams.h, 1195
HAL_SUCCESS
HAL Driver, 25
hal_trng.c, 1195
hal_trng.h, 1196
hal_trng_config, 964
hal_trng_driver, 964
  config, 965
  state, 965
hal_trng_lld.c, 1197
hal_trng_lld.h, 1198
hal_uart.c, 1198
hal_uart.h, 1199
hal_uart_lld.c, 1202
hal_uart_lld.h, 1202
hal_usb.c, 1203
hal_usb.h, 1205
hal_usb_cdc.h, 1208
hal_usb_lld.c, 1209
  in, 1211
  out, 1211
hal_usb_lld.h, 1211
HAL_USE_ADC
  Configuration, 843
HAL_USE_CAN
  Configuration, 843
HAL_USE_CRY
  Configuration, 844
HAL_USE_DAC
  Configuration, 844
HAL_USE_EFL
  Configuration, 844
HAL_USE_GPT
  Configuration, 844
hal_sio_driver, 959
INDEX 1249

HAL_USE_I2C  Configuration, 844
HAL_USE_I2S  Configuration, 844
HAL_USE_ICU  Configuration, 844
HAL_USE_MAC  Configuration, 845
HAL_USE_MMC_SPI  Configuration, 845
HAL_USE_PAL  Configuration, 843
HAL_USE_PWM  Configuration, 845
HAL_USE_RTC  Configuration, 845
HAL_USE_SDC  Configuration, 845
HAL_USE_SERIAL  Configuration, 846
HAL_USE_SERIAL_USB  Configuration, 846
HAL_USE_SIO  Configuration, 846
HAL_USE_SPI  Configuration, 846
HAL_USE_TRNG  Configuration, 846
HAL_USE_UART  Configuration, 846
HAL_USE_USB  Configuration, 847
HAL_USE_WDG  Configuration, 847
HAL_USE_WSPI  Configuration, 847
HAL_VERSION  HAL Driver, 24
hal_wdg.c, 1213
hal_wdg.h, 1213
hal_wdg_lld.c, 1214
hal_wdg_lld.h, 1215
hal_wspi.c, 1215
hal_wspi.h, 1216
hal_wspi_config, 965
  end_cb, 966
  error_cb, 966
hal_wspi_driver, 967
  config, 968
  mutex, 968
  state, 968
  thread, 968
hal_wspi_lld.c, 1219
hal_wspi_lld.h, 1220
halconf.h, 1221
halInit  HAL Driver, 25
HMACSvA256Context, 968
HMACSHA512Context, 969
hsconf
  MMConfig, 1002
I/O Buffers Queues, 53
  BQ_BUFFER_SIZE, 55
  bqGetLinkX, 56
  bqIsSuspendedX, 57
  bnotify_t, 61
  bqResumeX, 58
  bqSetLinkX, 57
  bqSizeX, 55
  bqSpacel, 56
  bqSuspendl, 58
  BUFFERS_CHUNKS_SIZE, 55
  ibqGetEmptyBufferl, 63
  ibqGetFullBufferTimeout, 65
  ibqGetFullBufferTimeoutS, 66
  ibqGetTimeout, 68
  ibqIsEmptyl, 59
  ibqIsFull, 59
  ibqObjectInit, 62
  ibqPostFullBuffer, 64
  ibqReadTimeout, 69
  ibqReleaseEmptyBuffer, 67
  ibqReleaseEmptyBufferS, 68
  ibqResetl, 63
  input_buffers_queue_t, 62
  io_buffers_queue_t, 61
  obqFlush, 79
  obqGetEmptyBuffer, 73
  obqGetEmptyBufferTimeout, 74
  obqGetFullBuffer, 71
  obqIsEmpty, 60
  obqIsFull, 61
  obqObjectInit, 70
  obqPostFullBuffer, 75
  obqPostFullBufferS, 76
  obqPutTimeout, 76
  obqReleaseEmptyBuffer, 72
  obqResetl, 71
  obqTryFlushl, 78
  obqWriteTimeout, 77
  output_buffers_queue_t, 62
I/O Bytes Queues, 540
  input_queue_t, 551
  io_queue_t, 550
  iq_read, 551
  iqGet, 547
  iqGetEmpty, 545
  iqGetFull, 545
  iqGetI, 555
  iqGetTimeout, 555
  iqIsEmpty, 546
  iqIsFull, 546
  iqObjectInit, 552
  iqPutI, 554
  iqReadI, 556
  iqReadTimeout, 557

ChibiOS/HAL
iqResetI, 553
oq_write, 552
oqGetEmptyI, 548
oqGetFullI, 547
oqGet, 561
oqIsEmptyI, 549
oqFullPath, 549
oqObjectInit, 558
oqPut, 550
oqPutI, 560
oqPutTimeout, 560
oqResetI, 559
oqWriteI, 562
oqWriteTimeout, 563
output_queue_t, 551
Q_EMPTY, 542
Q_FULL, 543
Q_RESET, 542
Q_TIMEOUT, 542
qGetLink, 544
qnotify_t, 551
qSetLink, 544
qSizeX, 543
qSpacel, 543
I2C Driver, 247
_i2c_wakeup_error_isr, 252
_i2c_wakeup_isr, 251
I2C_ACK_FAILURE, 250
I2C_ACTIVE_RX, 254
I2C_ACTIVE_TX, 254
I2C_ARBITRATION_LOST, 250
I2C_BUS_ERROR, 250
i2c_lld_get_errors, 253
i2c_lld_init, 261
i2c_lld_master_receive_timeout, 262
i2c_lld_master_transmit_timeout, 262
i2c_lld_start, 261
i2c_lld_stop, 261
I2C_LOCKED, 254
I2C_NO_ERROR, 250
I2C_OVERRUN, 250
I2C_PEC_ERROR, 251
I2C_READY, 254
I2C_SMB_ALERT, 251
I2C_STOP, 254
I2C_TIMEOUT, 251
I2C_UNINIT, 254
I2C_USE_MUTUAL EXCLUSION, 251
i2cAcquireBus, 259
i2cAcquireBus, 259
i2cdaddr_t, 253
I2CD1, 263
I2CDriver, 254
i2cflags_t, 254
i2cGetErrors, 257
i2cInit, 255
i2cMasterReceive, 252
i2cMasterReceiveTimeout, 258
i2cMasterTransmit, 252
i2cMasterTransmitTimeout, 258
i2cObjectInit, 255
i2cReleaseBus, 260
i2cStart, 256
i2cstate_t, 254
i2cStop, 256
PLATFORM_I2C_USE_I2C1, 253
I2C_ACK_FAILURE
I2C Driver, 250
I2C_ACTIVE_RX
I2C Driver, 254
I2C_ACTIVE_TX
I2C Driver, 254
I2C_ARBITRATION_LOST
I2C Driver, 250
I2C_BUS_ERROR
I2C Driver, 250
i2c_lld_get_errors
I2C Driver, 253
i2c_lld_init
I2C Driver, 261
i2c_lld_master_receive_timeout
I2C Driver, 262
i2c_lld_master_transmit_timeout
I2C Driver, 262
i2c_lld_start
I2C Driver, 261
i2c_lld_stop
I2C Driver, 261
I2C_LOCKED
I2C Driver, 254
I2C_NO_ERROR
I2C Driver, 250
I2C_OVERRUN
I2C Driver, 250
I2C_PEC_ERROR
I2C Driver, 251
I2C_READY
I2C Driver, 254
I2C_SMB_ALERT
I2C Driver, 251
I2C_STOP
I2C Driver, 254
I2C_TIMEOUT
I2C Driver, 251
I2C_UNINIT
I2C Driver, 254
I2C_USE_MUTUAL EXCLUSION
Configuration, 849
I2C Driver, 251
i2cAcquireBus
I2C Driver, 259
i2cdaddr_t
I2C Driver, 253
I2CConfig, 969
I2CD1
I2C Driver, 263
I2CDriver, 970
    config, 971
    errors, 971
    I2C Driver, 254
    state, 971
i2cflags_t
    I2C Driver, 254
i2cGetErrors
    I2C Driver, 257
i2cInit
    I2C Driver, 255
i2cMasterReceive
    I2C Driver, 252
i2cMasterReceiveTimeout
    I2C Driver, 258
i2cMasterTransmit
    I2C Driver, 252
i2cMasterTransmitTimeout
    I2C Driver, 258
i2cObjectInit
    I2C Driver, 255
i2cReleaseBus
    I2C Driver, 260
i2cStart
    I2C Driver, 256
i2cstate_t
    I2C Driver, 254
i2cStop
    I2C Driver, 256
I2S Driver, 264
    _i2s_isr_full_code, 268
    _i2s_isr_half_code, 267
I2S_ACTIVE, 270
I2S_COMPLETE, 270
i2s_lld_config_fields, 269
i2s_lld_driver_fields, 269
i2s_lld_init, 274
i2s_lld_start, 274
i2s_lld_start_exchange, 275
i2s_lld_stop, 275
i2s_lld_stop_exchange, 276
I2S_READY, 270
I2S_STOP, 270
I2S_UNINIT, 270
i2scallback_t, 269
I2SConfig, 269
I2SD1, 276
I2SDriver, 269
i2sInit, 270
i2sIsBufferComplete, 266
i2sObjectInit, 271
i2sStart, 271
i2sStartExchange, 273
i2sStartExchangel, 266
i2ystate_t, 270
i2sStop, 272
i2sStopExchange, 273
i2sStopExchangel, 267
PLATFORM_I2S_USE_I2S1, 268
I2S_ACTIVE
    I2S Driver, 270
I2S_COMPLETE
    I2S Driver, 270
i2s_lld_config_fields
    I2S Driver, 269
i2s_lld_driver_fields
    I2S Driver, 269
i2s_lld_init
    I2S Driver, 274
i2s_lld_start
    I2S Driver, 274
i2s_lld_start_exchange
    I2S Driver, 275
i2s_lld_stop
    I2S Driver, 275
i2s_lld_stop_exchange
    I2S Driver, 276
I2S_READY
    I2S Driver, 270
I2S_STOP
    I2S Driver, 270
I2S_UNINIT
    I2S Driver, 270
i2scallback_t
    I2S Driver, 269
I2SConfig
    I2S Driver, 269
I2SD1
    I2S Driver, 276
I2SDriver
    I2S Driver, 269
i2sInit
    I2S Driver, 270
i2sIsBufferComplete
    I2S Driver, 266
i2sObjectInit
    I2S Driver, 271
i2sStart
    I2S Driver, 271
i2sStartExchange
    I2S Driver, 273
i2sStartExchangel
    I2S Driver, 266
i2sstate_t
    I2S Driver, 270
i2sStop
    I2S Driver, 272
i2sStopExchange
    I2S Driver, 272
i2sStopExchangel
    I2S Driver, 267
ibnotify
    Serial over USB Driver, 635
ibqGetEmptyBuffer
    I/O Buffers Queues, 63
ibqGetFullBufferTimeout
ICU Buffers Queues, 65
ibqGetFullBufferTimeoutS
ICU Buffers Queues, 66
ibqGetTimeout
ICU Buffers Queues, 68
ibqIsEmpty
ICU Buffers Queues, 59
ibqIsFull
ICU Buffers Queues, 59
ibqObjectInit
ICU Buffers Queues, 62
ibqPostFullBuffer
ICU Buffers Queues, 64
ibqReadTimeout
ICU Buffers Queues, 69
ibqReleaseEmptyBuffer
ICU Buffers Queues, 67
ibqReleaseEmptyBufferS
ICU Buffers Queues, 68
ibqReset
ICU Buffers Queues, 63
ICU Driver, 277
   _icu_isr_invoke_overflow_cb, 285
   _icu_isr_invoke_period_cb, 284
   _icu_isr_invoke_width_cb, 284
   ICU_ACTIVE, 288
   ICU_INPUT_ACTIVE_HIGH, 288
   ICU_INPUT_ACTIVE_LOW, 288
   icu_lld_are_notifications_enabled, 286
   icu_lld_disable_notifications, 299
   icu_lld_enable_notifications, 298
   icu_lld_get_period, 286
   icu_lld_get_width, 285
   icu_lld_init, 296
   icu_lld_start, 296
   icu_lld_start_capture, 297
   icu_lld_stop, 297
   icu_lld_stop_capture, 298
   icu_lld_wait_capture, 297
   ICU_READY, 288
   ICU_STOP, 288
   ICU_UNINIT, 288
   ICU_WAITING, 288
   icuAreNotificationsEnabledX, 282
   icucallback_t, 287
   icucnt_t, 287
   ICUD1, 299
   icuDisableNotifications, 295
   icuDisableNotificationsS, 282
   ICUDriver, 287
   icuEnableNotifications, 294
   icuEnableNotificationsS, 281
   icufreq_t, 287
   icuGetPeriodX, 283
   icuGetWidthX, 283
   icuInit, 288
   icumode_t, 288
   icuObjectInit, 289
   icuStart, 289
   icuStartCapture, 292
   icuStartCaptureI, 290
   icustate_t, 288
   icuStop, 291
   icuStopCapture, 294
   icuStopCaptureI, 281
   icuWaitCapture, 292
   PLATFORM_ICU_USE_ICU1, 285
ICU_ACTIVE
   ICU Driver, 288
ICU_INPUT_ACTIVE_HIGH
   ICU Driver, 288
ICU_INPUT_ACTIVE_LOW
   ICU Driver, 288
   icu_lld_are_notifications_enabled
   ICU Driver, 288
   icu_lld_disable_notifications
   ICU Driver, 299
   icu_lld_enable_notifications
   ICU Driver, 298
   icu_lld_get_period
   ICU Driver, 286
   icu_lld_get_width
   ICU Driver, 285
   icu_lld_init
   ICU Driver, 296
   icu_lld_start
   ICU Driver, 296
   icu_lld_start_capture
   ICU Driver, 297
   icu_lld_stop
   ICU Driver, 297
   icu_lld_stop_capture
   ICU Driver, 299
   icu_lld_wait_capture
   ICU Driver, 297
ICU_READY
   ICU Driver, 288
ICU_STOP
   ICU Driver, 288
ICU_UNINIT
   ICU Driver, 288
ICU_WAITING
   ICU Driver, 288
   icuAreNotificationsEnabledX
   ICU Driver, 282
   icucallback_t
   ICU Driver, 287
   icucnt_t
   ICU Driver, 287
ICUD1

ChibiOS/HAL
ICU Driver, 299
icuDisableNotifications
ICU Driver, 295
icuDisableNotificationsI
ICU Driver, 282
ICUDriver, 973
cfg, 974
ICU Driver, 287
state, 974
icuEnableNotifications
ICU Driver, 294
icuEnableNotificationsI
ICU Driver, 281
icufreq_t
ICU Driver, 287
icuGetPeriodX
ICU Driver, 283
icuGetWidthX
ICU Driver, 283
icuInit
ICU Driver, 288
icumode_t
ICU Driver, 288
icuObjectInit
ICU Driver, 289
icuStart
ICU Driver, 289
icuStartCapture
ICU Driver, 292
icuStartCaptureI
ICU Driver, 280
icustate_t
ICU Driver, 288
icuStop
ICU Driver, 291
icuStopCapture
ICU Driver, 294
icuStopCaptureI
ICU Driver, 281
icuWaitCapture
ICU Driver, 292
id
mfs_data_header_t, 993
mfs_transaction_op_t, 994
IDE
CANRxFrame, 923
CANTxFrame, 925
in
hal_usb_lld.c, 1211
USB Driver, 800
in_cb
USBEndpointConfig, 1065
in_maxsize
USBEndpointConfig, 1066
in_params
USBDriver, 1061
in_state
USBEndpointConfig, 1066
Inner Code, 859
input_buffers_queue_t
I/O Buffers Queues, 62
input_queue_t
I/O Bytes Queues, 551
int_in
SerialUSBConfig, 1037
Interfaces and Classes, 858
io_buffers_queue, 974
bcounter, 976
bn, 977
brdptr, 976
bsize, 977
btop, 977
buffers, 977
bwrptr, 976
link, 978
notify, 978
ptr, 977
suspended, 976
top, 978
waiting, 976
io_buffers_queue_t
I/O Buffers Queues, 61
io_queue, 979
q_buffer, 980
q_counter, 980
q_link, 981
q_notify, 981
q_rdptr, 981
q_top, 980
q_waiting, 980
q_wrptr, 980
io_queue_t
I/O Buffes Queues, 550
IOBus, 981
mask, 982
offset, 982
portid, 982
IOBUS_DECL
PAL Driver, 468
ioline_t
PAL Driver, 501
iomode_t
PAL Driver, 501
iopadid_t
PAL Driver, 501
iopadid_t
PAL Driver, 501
iopadid_t
PAL Driver, 501
iopadid_t
PAL Driver, 501
iq_read
I/O Bytes Queues, 551
iqGet
I/O Bytes Queues, 547
iqGetEmpty
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>iO Bytes Queues, 545</td>
<td></td>
</tr>
<tr>
<td>iqGetFull</td>
<td>545</td>
</tr>
<tr>
<td>iqGet</td>
<td>545</td>
</tr>
<tr>
<td>iqGetTimeout</td>
<td>545</td>
</tr>
<tr>
<td>iqGetEmpty</td>
<td>546</td>
</tr>
<tr>
<td>iqGetFull</td>
<td>546</td>
</tr>
<tr>
<td>iqGetObjectInit</td>
<td>546</td>
</tr>
<tr>
<td>iqPut</td>
<td>552</td>
</tr>
<tr>
<td>iqRead</td>
<td>546</td>
</tr>
<tr>
<td>iqReadTimeout</td>
<td>552</td>
</tr>
<tr>
<td>iqReset</td>
<td>553</td>
</tr>
<tr>
<td>LPA_100BASE4</td>
<td>402</td>
</tr>
<tr>
<td>LPA_100FULL</td>
<td>402</td>
</tr>
<tr>
<td>LPA_100HALF</td>
<td>402</td>
</tr>
<tr>
<td>LPA_10FULL</td>
<td>402</td>
</tr>
<tr>
<td>LPA_10HALF</td>
<td>402</td>
</tr>
<tr>
<td>LPA_10PACK</td>
<td>403</td>
</tr>
<tr>
<td>LPA_LPAGE</td>
<td>403</td>
</tr>
<tr>
<td>LPA_PAUSE_ASYM</td>
<td>403</td>
</tr>
<tr>
<td>LPA_PAUSE_CAP</td>
<td>403</td>
</tr>
<tr>
<td>LPA_RESERVED</td>
<td>403</td>
</tr>
<tr>
<td>LPA_RGBAFTL</td>
<td>403</td>
</tr>
<tr>
<td>LPA_SLCFT</td>
<td>402</td>
</tr>
<tr>
<td>Lsconf</td>
<td>1002</td>
</tr>
<tr>
<td>MAC Driver, 349</td>
<td></td>
</tr>
<tr>
<td>ETDH1, 368</td>
<td></td>
</tr>
<tr>
<td>MAC_ACTIVE, 355</td>
<td></td>
</tr>
<tr>
<td>mac_lld_poll_link_status</td>
<td>365</td>
</tr>
<tr>
<td>mac_lld_read_receive_descriptor</td>
<td>366</td>
</tr>
<tr>
<td>mac_lld_release_receive_descriptor</td>
<td>365</td>
</tr>
<tr>
<td>mac_lld_release_transmit_descriptor</td>
<td>364</td>
</tr>
<tr>
<td>mac_lld_start</td>
<td>362</td>
</tr>
<tr>
<td>mac_lld_stop</td>
<td>363</td>
</tr>
<tr>
<td>mac_lld_write_transmit_descriptor</td>
<td>365</td>
</tr>
<tr>
<td>MAC_STOP, 355</td>
<td></td>
</tr>
<tr>
<td>MAC_SUPPORTS_ZERO_COPY, 354</td>
<td></td>
</tr>
<tr>
<td>MAC_UNINIT, 355</td>
<td></td>
</tr>
<tr>
<td>MAC_USE_EVENTS, 351</td>
<td></td>
</tr>
<tr>
<td>MAC_USE_ZERO_COPY, 351</td>
<td></td>
</tr>
<tr>
<td>MACDriver, 355</td>
<td></td>
</tr>
<tr>
<td>macGetNextReceiveBuffer</td>
<td>353</td>
</tr>
<tr>
<td>macGetNextTransmitBuffer</td>
<td>353</td>
</tr>
<tr>
<td>macGetReceiveEventSource</td>
<td>351</td>
</tr>
<tr>
<td>macInit, 355</td>
<td></td>
</tr>
<tr>
<td>macGetObjectInit</td>
<td>356</td>
</tr>
<tr>
<td>macPollLinkStatus</td>
<td>361</td>
</tr>
<tr>
<td>macReadReceiveDescriptor</td>
<td>352</td>
</tr>
<tr>
<td>macReleaseReceiveDescriptor</td>
<td>360</td>
</tr>
<tr>
<td>macReleaseTransmitDescriptor</td>
<td>359</td>
</tr>
<tr>
<td>macStart, 356</td>
<td></td>
</tr>
<tr>
<td>macstate_t, 355</td>
<td></td>
</tr>
<tr>
<td>macStop, 357</td>
<td></td>
</tr>
<tr>
<td>macWaitReceiveDescriptor</td>
<td>359</td>
</tr>
<tr>
<td>macWaitTransmitDescriptor</td>
<td>358</td>
</tr>
<tr>
<td>macWriteTransmitDescriptor</td>
<td>352</td>
</tr>
<tr>
<td>PLATFORM_MAC_USE_MAC1, 354</td>
<td></td>
</tr>
<tr>
<td>MAC_ACTIVE</td>
<td></td>
</tr>
<tr>
<td>MAC Driver, 355</td>
<td></td>
</tr>
<tr>
<td>mac_address</td>
<td>983</td>
</tr>
<tr>
<td>MACConfig, 1002</td>
<td></td>
</tr>
<tr>
<td>mac_lld_get_next_receive_buffer</td>
<td>367</td>
</tr>
<tr>
<td>MAC Driver, 367</td>
<td></td>
</tr>
<tr>
<td>mac_lld_get_next_transmit_buffer</td>
<td>366</td>
</tr>
<tr>
<td>MAC Driver, 366</td>
<td></td>
</tr>
<tr>
<td>mac_lld_get_receive_descriptor</td>
<td>364</td>
</tr>
<tr>
<td>MAC Driver, 364</td>
<td></td>
</tr>
<tr>
<td>mac_lld_get_transmit_descriptor</td>
<td>363</td>
</tr>
<tr>
<td>MAC Driver, 363</td>
<td></td>
</tr>
<tr>
<td>mac_lld_init</td>
<td>362</td>
</tr>
<tr>
<td>MAC Driver, 362</td>
<td></td>
</tr>
<tr>
<td>mac_lld_poll_link_status</td>
<td>365</td>
</tr>
<tr>
<td>MAC Driver, 365</td>
<td></td>
</tr>
<tr>
<td>mac_lld_read_receive_descriptor</td>
<td>365</td>
</tr>
<tr>
<td>MAC Driver, 366</td>
<td></td>
</tr>
<tr>
<td>mac_lld_release_receive_descriptor</td>
<td>365</td>
</tr>
<tr>
<td>MAC Driver, 365</td>
<td></td>
</tr>
<tr>
<td>mac_lld_release_transmit_descriptor</td>
<td>364</td>
</tr>
<tr>
<td>MAC Driver, 364</td>
<td></td>
</tr>
<tr>
<td>mac_lld_start</td>
<td>362</td>
</tr>
<tr>
<td>MAC Driver, 363</td>
<td></td>
</tr>
<tr>
<td>mac_lld_stop</td>
<td></td>
</tr>
<tr>
<td>MAC Driver, 365</td>
<td></td>
</tr>
<tr>
<td>mac_lld_write_transmit_descriptor</td>
<td></td>
</tr>
<tr>
<td>MAC Driver, 365</td>
<td></td>
</tr>
<tr>
<td>MAC_STOP</td>
<td></td>
</tr>
</tbody>
</table>
MAC Driver, 355
MAC_SUPPORTS_ZERO_COPY
MAC Driver, 354
MAC_UNINIT
MAC Driver, 355
MAC_USE_EVENTS
Configuration, 850
MAC Driver, 351
MAC_USE_ZERO_COPY
Configuration, 850
MAC Driver, 351
MACConfig, 983
mac_address, 983
MACDriver, 984
cfg, 985
MAC Driver, 355
rdevent, 985
rqueue, 985
state, 984
tdqueue, 985
macGetNextReceiveBuffer
MAC Driver, 353
macGetNextTransmitBuffer
MAC Driver, 353
macGetReceiveEventSource
MAC Driver, 351
macInit
MAC Driver, 355
macObjectInit
MAC Driver, 356
macPollLinkStatus
MAC Driver, 361
macReadReceiveDescriptor
MAC Driver, 352
MACReceiveDescriptor, 985
offset, 986
size, 986
macReleaseReceiveDescriptor
MAC Driver, 360
macReleaseTransmitDescriptor
MAC Driver, 359
macStart
MAC Driver, 356
macState_t
MAC Driver, 355
macStop
MAC Driver, 357
MACTransmitDescriptor, 986
offset, 987
size, 987
macWaitReceiveDescriptor
MAC Driver, 359
macWaitTransmitDescriptor
MAC Driver, 358
macWriteTransmitDescriptor
MAC Driver, 352
magic1
mfs_bank_header_t, 991
mfs_data_header_t, 992
magic2
mfs_bank_header_t, 991
mfs_data_header_t, 992
Managed Flash Storage Driver, 369
ALIGNED_DHDR_SIZE, 372
ALIGNED_REC_SIZE, 371
ALIGNED_SIZEOF, 372
mfs_bank_check_header, 378
mfs_bank_erase, 377
mfs_bank_get_state, 379
mfs_bank_scan_records, 379
mfs_bank_state_t, 375
mfs_bank_t, 374
mfs_bank_verify_erase, 377
mfs_bank_write_header, 378
MFS_CFG_BUFFER_SIZE, 373
MFS_CFG_MAX_RECORDS, 372
MFS_CFG_MAX_REPAIR_ATTEMPTS, 373
MFS_CFG_MEMORY_ALIGNMENT, 373
MFS_CFG_STRONG_CHECKING, 373
MFS_CFG_TRANSACTION_MAX, 374
MFS_CFG_WRITE_VERIFY, 373
mfs_error_t, 374
mfs_flash_copy, 376
mfs_flash_read, 375
mfs_flash_write, 375
mfs_gradient_collect, 380
mfs_id_t, 374
mfs_mount, 381
mfs_state_t, 374
mfs_try_mount, 381
mfsCommitTransaction, 389
mfsErase, 383
mfsEraseRecord, 385
mfsObjectInit, 382
mfsPerformGarbageCollection, 387
mfsReadRecord, 384
mfsRollbackTransaction, 389
mfsStart, 382
mfsStartTransaction, 388
mfsStop, 383
mfsWriteRecord, 385
PAIR, 372
RET_ON_ERROR, 372
mask
IOBus, 982
Memory Streams Class, 330
__memory_stream_data, 330
msObjectInit, 330
MemoryStream, 987
vmt, 989
memstreams.c, 1223
memstreams.h, 1224
MemStreamVMT, 990
mfs_bank_check_header
Managed Flash Storage Driver, 378
mfs_bank_erase
INDEX 1257

ADVERTISE_RFAULT, 401
ADVERTISE_SLCT, 400
BMCR_ANENABLE, 397
BMCR_ANRESTART, 397
BMCR_CTST, 397
BMCR_FULDDPLX, 397
BMCR_ISOLATE, 397
BMCR_LOOPBACK, 398
BMCR_PDOWN, 397
BMCR_RESET, 398
BMCR_RESV, 397
BMCR_SPEED100, 398
BMSR_100BASE4, 400
BMSR_100FULL, 400
BMSR_10HALF, 399
BMSR_10FULL, 399
BMSR_ANEGCOMPLETE, 399
BMSR_ANEGCAPABLE, 398
BMSR_ANEGCOMPLETE, 399
BMSR_ERCAP, 398
BMSR_JCD, 398
BMSR_LSTATUS, 398
BMSR_MFPRESUPPCAP, 399
BMSR_RESV, 399
BMSR_RFAULT, 399
EXPANSION_ENABLENPAGE, 404
EXPANSION_LCWP, 404
EXPANSION_NPCAPABLE, 404
EXPANSION_MFAULTS, 404
EXPANSION_NWAY, 403
EXPANSION_RESV, 404
LPA_100BASE4, 402
LPA_100FULL, 402
LPA_10HALF, 402
LPA_10FULL, 402
LPA_LPACK, 403
LPA_NPAGE, 403
LPA_PAUSE_ASYM, 403
LPA_PAUSE_CAP, 403
LPA_RESV, 403
LPA_RFAULT, 403
LPA_SLCT, 402
MII_ADVERTISE, 394
MII_ANNPTR, 394
MII_BMCRR, 393
MII_BMSR, 393
MII_CTRL1000, 394
MII_DCOUNTER, 395
MII_ESTATUS, 395
MII_EXPANSION, 394
MII_FCSCOUNTER, 395
MII_LBRError, 396
MII_LPA, 394
MII_MICR, 395
MII_NCONFIG, 396
MII_NWAYTEST, 395
MII_PHYADDR, 396
MII_PHYSID1, 393
MII_PHYSID2, 394
MII_PHYS1, 395
MII_RERRCOUNTER, 395
MII_RESV1, 396
MII_RESV2, 396
MII_SREVISION, 396
MII_STAT1000, 394
MII_TPSTATUS, 396
NWAYTEST_LOOPBACK, 404
NWAYTEST_RESV1, 404
NWAYTEST_RESV2, 405
MII_ADVERTISE
MII/RMII Header, 394
MII_ANNPTR
MII/RMII Header, 394
MII_BMCRR
MII/RMII Header, 393
MII_BMSR
MII/RMII Header, 393
MII_CTRL1000
MII/RMII Header, 394
MII_DCOUNTER
MII/RMII Header, 395
MII_ESTATUS
MII/RMII Header, 395
MII_EXPANSION
MII/RMII Header, 394
MII_FCSCOUNTER
MII/RMII Header, 395
MII_LBRError
MII/RMII Header, 396
MII_LPA
MII/RMII Header, 394
MII_MICR
MII/RMII Header, 395
MII_NCONFIG
MII/RMII Header, 396
MII_NWAYTEST
MII/RMII Header, 395
MII_PHYADDR
MII/RMII Header, 396
MII_PHYS1
MII/RMII Header, 393
MII_PHYS2
MII/RMII Header, 394
MII_PHYS1
MII/RMII Header, 395
MII_RERRCOUNTER
MII/RMII Header, 395
MII_RESV1
MII/RMII Header, 396
MII_RESV2
MII/RMII Header, 396
MII_SREVISION
MII/RMII Header, 396
MII_STAT1000
MII/RMII Header, 394
MII_TPISTATUS
MII/RMII Header, 396
millisecond
RTCDateTime, 1022
MMC over SPI Driver, 406
   _mmc_driver_methods, 408
crc7, 409
crc7_lookup_table, 429
MMC_NICE_WAITING, 408
mmc_vmt, 428
mmcConnect, 419
mmcDisconnect, 420
mmcErase, 427
mmcGetInfo, 427
mmcInit, 417
mmcIsCardInserted, 408
mmcIsWriteProtected, 409
mmcObjectInit, 417
mmcSequentialRead, 422
mmcSequentialWrite, 424
mmcStart, 418
mmcStartSequentialRead, 421
mmcStartSequentialWrite, 423
mmcStop, 418
mmcStopSequentialRead, 422
mmcStopSequentialWrite, 425
mmcSync, 426
read_CxD, 415
recv1, 411
recv3, 412
send_command_R1, 413
send_command_R3, 414
send_hdr, 411
sync, 416
wait, 410
MMC/SD Block Device, 430
   _mmcsd_block_device_data, 435
   _mmcsd_block_device_methods, 435
   _mmcsd_get_capacity, 437
   _mmcsd_get_capacity_ext, 438
   _mmcsd_get_slice, 437
   _mmcsd_unpack_csd_mmc, 440
   _mmcsd_unpack_csd_v10, 440
   _mmcsd_unpack_csd_v20, 441
   _mmcsd_unpack_mmc_cid, 439
   _mmcsd_unpack_sdc_cid, 438
MMCSD_BLOCK_SIZE, 434
MMCSD_CMD8_PATTERN, 435
MMCSD_R1_ERROR, 435
MMCSD_R1_ERROR_MASK, 434
MMCSD_R1_IS_CARD_LOCKED, 436
MMCSD_R1_STS, 436
mmcsdGetCardCapacity, 436
mm_cmbd6_construct
   SDC Driver, 588
mmc_detect_bus_clk
   SDC Driver, 592
mmc_init
 MMC_NICE_WAITING
   Configuration, 850
   MMC over SPI Driver, 408
mmc_set_bus_width
   SDC Driver, 594
mmc_switch_t
   SDC Driver, 586
mmc_vmt
   MMC over SPI Driver, 428
MMCDriver, 1001
   hscfg, 1002
   lscfg, 1002
   spip, 1002
mmcConnect
   MMC over SPI Driver, 419
mmcDisconnect
   MMC over SPI Driver, 420
MMCDriverVMT, 1005
mmcErase
   MMC over SPI Driver, 427
mmcGetInfo
   MMC over SPI Driver, 427
mmcInit
   MMC over SPI Driver, 417
mmcsCardInserted
   MMC over SPI Driver, 417
mmcsCardInserted
   MMC over SPI Driver, 408
mmcsWriteProtected
   MMC over SPI Driver, 409
mmcObjectInit
   MMC over SPI Driver, 417
MMCSDBlockDevice, 1006
   vmt, 1008
MMCSDBlockDeviceVMT, 1009
mmcsdGetCardCapacity
   MMC over SPI Driver, 422
mmcsSequentialWrite
   MMC over SPI Driver, 424
mmcStart
   MMC over SPI Driver, 418
mmcStartSequentialRead
MMC over SPI Driver, 421
mmcStartSequentialWrite
MMC over SPI Driver, 423
mmcStop
MMC over SPI Driver, 418
mmcStopSequentialRead
MMC over SPI Driver, 422
mmcStopSequentialWrite
MMC over SPI Driver, 425
mmcSync
MMC over SPI Driver, 426
mode
ICUConfig, 972
PWMChannelConfig, 1015
mode_detect
SDC Driver, 586
month
RTCDateTime, 1021
msg_t
OSAL, 875
msObjectInit
Memory Streams Class, 330
mutex
   hal_adc_driver, 946
   hal_dac_driver, 951
   hal_spi_driver, 963
   hal_wspi_driver, 968
   UARTDriver, 1052
mutex_t
   OSAL, 876
next_offset
MFSDriver, 999
Normal Drivers, 855
notify
   io_buffers_queue, 978
Null Streams Class, 332
   _null_stream_data, 332
   nullObjectInit, 332
nullObjectInit
   Null Streams Class, 332
NullStream, 1010
vmt, 1011
nullstreams.c, 1224
nullstreams.h, 1225
NullStreamVMT, 1012
num_channels
   hal_adc_configuration_group, 943
   hal_dac_conversion_group, 947
NWAYTEST_LOOPBACK
MII/RMII Header, 404
NWAYTEST_RESV1
MII/RMII Header, 404
NWAYTEST_RESV2
MII/RMII Header, 405
objGetInstance
   Base Object, 456
obnotify
   Serial over USB Driver, 636
obqFlush
   I/O Buffers Queues, 79
obqGetEmptyBufferTimeout
   I/O Buffers Queues, 73
obqGetEmptyBufferTimeoutS
   I/O Buffers Queues, 74
obqGetFullBuffer
   I/O Buffers Queues, 71
obqIsEmptyI
   I/O Buffers Queues, 60
obqIsFullI
   I/O Buffers Queues, 61
obqObjectInit
   I/O Buffers Queues, 70
obqPostFullBuffer
   I/O Buffers Queues, 75
obqPostFullBufferS
   I/O Buffers Queues, 76
obqPutTimeout
   I/O Buffers Queues, 76
obqReleaseEmptyBufferI
   I/O Buffers Queues, 72
obqResetI
   I/O Buffers Queues, 71
obqTryFlushI
   I/O Buffers Queues, 78
obqWriteTimeout
   I/O Buffers Queues, 77
offset
   flash_sector_descriptor_t, 938
   IObus, 982
   MACReceiveDescriptor, 986
   MACTransmitDescriptor, 987
   mfs_transaction_op_t, 994
oq_write
   I/O Bytes Queues, 552
oqGetEmptyI
   I/O Bytes Queues, 548
oqGetFullI
   I/O Bytes Queues, 547
oqGetI
   I/O Bytes Queues, 561
oqIsEmptyI
   I/O Bytes Queues, 549
oqIsFullI
   I/O Bytes Queues, 549
oqObjectInit
   I/O Bytes Queues, 558
oqPut
   I/O Bytes Queues, 550
oqPutI
   I/O Bytes Queues, 560
oqPutTimeout
   I/O Bytes Queues, 560
oqResetI
   I/O Bytes Queues, 559
oqWrite
I/O Bytes Queues, 562
oqWriteTimeout
I/O Bytes Queues, 563
OSAL, 861
   event_source_t, 876
   eventcallback_t, 876
   eventflags_t, 876
   msg_t, 875
   mutex_t, 876
OSAL_DBG_ENABLE_ASSERTS, 867
OSAL_DBG_ENABLE_CHECKS, 867
osal_halt_msg, 893
OSAL_IRQ_EPILOGUE, 869
OSAL_IRQ_HANDLER, 869
OSAL_IRQ_IS_VALID_PRIORITY, 869
OSAL_IRQ_MAXIMUM_PRIORITY, 867
OSAL_IRQ_PRIORITY_LEVELS, 866
OSAL_IRQ_PROLOGUE, 869
OSAL_MS2I, 870
OSAL_MS2RTC, 872
OSAL_S2I, 870
OSAL_S2RTC, 871
OSAL_ST_FREQUENCY, 866
OSAL_ST_MODE, 866
OSAL_ST_RESOLUTION, 866
OSAL_US2I, 871
OSAL_US2RTC, 872
osalDbgAssert, 867
osalDbgCheck, 868
osalDbgCheckClassI, 868
osalDbgCheckClassS, 868
osalEventBroadcastFlags, 885
osalEventBroadcastFlagsI, 885
osalEventObjectInit, 892
osalEventSetCallback, 886
osalInit, 877
osalMutexLock, 886
osalMutexObjectInit, 892
osalMutexUnlock, 887
osalOsGetSystemTimeX, 878
osalOsRescheduleS, 878
osalOsTimerHandlerI, 878
osalSysDisable, 887
osalSysEnable, 888
osalSysGetStatusAndLockX, 889
osalSysHalt, 877
osalSysLock, 888
osalSysLockFromISR, 888
osalSysPolledDelayX, 877
osalSysRestoreStatusX, 889
osalSysUnlock, 888
osalSysUnlockFromISR, 889
osalThreadDequeueAllI, 884
osalThreadDequeueNextI, 884
osalThreadEnqueueTimeoutS, 883
osalThreadQueueObjectInit, 891
osalThreadResumeI, 882
osalThreadResumeS, 882
osalThreadSleep, 879
osalThreadSleepMicroseconds, 874
osalThreadSleepMilliseconds, 874
osalThreadSleepS, 879
osalThreadSleepSeconds, 873
osalThreadSuspendS, 881
osalThreadSuspendTimeoutS, 881
osalTimeAddX, 890
osalTimeDiffX, 890
osalTimeIsInRangeX, 891
rtcnt_t, 875
sysinterval_t, 875
syssts_t, 875
sytme_t, 875
thread_reference_t, 875
osal.c, 1225
osal.h, 1227
OSAL_DBG_ENABLE_ASSERTS
OSAL, 867
OSAL_DBG_ENABLE_CHECKS
OSAL, 867
osal_halt_msg
OSAL, 893
OSAL_IRQ_EPILOGUE
OSAL, 869
OSAL_IRQ_HANDLER
OSAL, 869
OSAL_IRQ_IS_VALID_PRIORITY
OSAL, 869
OSAL_IRQ_MAXIMUM_PRIORITY
OSAL, 867
OSAL_IRQ_PRIORITY_LEVELS
OSAL, 866
OSAL_IRQ_PROLOGUE
OSAL, 869
OSAL_MS2I
OSAL, 870
OSAL_MS2RTC
OSAL, 872
OSAL_S2I
OSAL, 870
OSAL_S2RTC
OSAL, 871
OSAL_ST_FREQUENCY
OSAL, 866
OSAL_ST_MODE
OSAL, 866
OSAL_ST_RESOLUTION
OSAL, 866
OSAL_US2I
OSAL, 871
OSAL_US2RTC
OSAL, 872
osalDbgAssert
OSAL, 867
osalDbgCheck
OSAL, 868
osalDbgCheckClassI
OSAL, 868
osalDbgCheckClassS
OSAL, 868
osalEventBroadcastFlags
OSAL, 885
osalEventBroadcastFlagsI
OSAL, 885
osalEventObjectInit
OSAL, 892
osalEventSetCallback
OSAL, 886
osalInit
OSAL, 877
osalMutexLock
OSAL, 886
osalMutexObjectInit
OSAL, 892
osalMutexUnlock
OSAL, 887
osalOsGetSystemTimeX
OSAL, 878
osalOsRescheduleS
OSAL, 878
osalOsTimerHandlerI
OSAL, 878
osalSysDisable
OSAL, 887
osalSysEnable
OSAL, 888
osalSysGetStatusAndLockX
OSAL, 889
osalSysHalt
OSAL, 877
osalSysLock
OSAL, 888
osalSysLockFromISR
OSAL, 888
osalSysPolledDelayX
OSAL, 877
osalSysRestoreStatusX
OSAL, 889
osalSysUnlock
OSAL, 888
osalSysUnlockFromISR
OSAL, 889
osalThreadDequeueAllI
OSAL, 884
osalThreadDequeueNextI
OSAL, 884
osalThreadEnqueueTimeoutS
OSAL, 883
osalThreadQueueObjectInit
OSAL, 891
osalThreadResumeI
OSAL, 882
osalThreadResumeS
OSAL, 882
osalThreadSleep
OSAL, 879
osalThreadSleepMicroseconds
OSAL, 874
osalThreadSleepMilliseconds
OSAL, 874
osalThreadSleepS
OSAL, 879
osalThreadSleepSeconds
OSAL, 873
osalThreadSuspendS
OSAL, 881
osalThreadSuspendTimeoutS
OSAL, 881
osalTimeAddX
OSAL, 890
osalTimeDiffX
OSAL, 890
osalTimeIsInRangeX
OSAL, 891
out
hal_usb_lld.c, 1211
USB Driver, 800
out_cb
USBEndpointConfig, 1066
out_maxsize
USBEndpointConfig, 1066
out_params
USBDriver, 1061
out_state
USBEndpointConfig, 1066
Output Formatter Utility, 334
chprintf, 335
CHPRINTF_USE_FLOAT, 334
chsnprintf, 336
chvprintf, 334
chvsnprintf, 338
output_buffers_queue_t
I/O Buffers Queues, 62
output_queue_t
I/O Bytes Queues, 551
overflow_cb
ICUConfig, 973
page_size
flashDescriptor_t, 936
PAIR
Managed Flash Storage Driver, 372
PAL Driver, 457
_IOBUS_DATA, 467
_pal_clear_event, 469, 470
_pal_init_event, 468, 469
_pal_isr_code, 471, 472
_pal_lld_init, 508
_pal_lld_setgroupmode, 508
IOBUS_DECL, 468
ioline_t, 501
iomode_t, 501
iopadid_t, 501
pal_lld_init
  PAL Driver, 491
pal_lld_readgroup
  PAL Driver, 494
pal_lld_readlatch
  PAL Driver, 492
pal_lld_readpad
  PAL Driver, 496
pal_lld_readport
  PAL Driver, 491
pal_lld_setgroupmode
  PAL Driver, 495
pal_lld_setpad
  PAL Driver, 497
pal_lld_setpadmode
  PAL Driver, 499
pal_lld_setport
  PAL Driver, 493
pal_lld_togglepad
  PAL Driver, 499
pal_lld_toggleport
  PAL Driver, 494
pal_lld_writegroup
  PAL Driver, 495
pal_lld_writepad
  PAL Driver, 497
pal_lld_writeport
  PAL Driver, 492
PAL_LOW
  PAL Driver, 465
PAL_MODE_INPUT
  PAL Driver, 464
PAL_MODE_INPUT_ANALOG
  PAL Driver, 464
PAL_MODE_INPUT_PULLDOWN
  PAL Driver, 464
PAL_MODE_INPUT_PULLUP
  PAL Driver, 464
PAL_MODE_OUTPUT_OCPEDRAIN
  PAL Driver, 464
PAL_MODE_OUTPUT_PUSHPULL
  PAL Driver, 464
PAL_MODE_RESET
  PAL Driver, 463
PAL_MODE_UNCONNECTED
  PAL Driver, 463
PAL_NOLINE
  PAL Driver, 491
PAL_PAD
  PAL Driver, 490
PAL_PORT
  PAL Driver, 490
PAL_PORT_BIT
  PAL Driver, 466
PAL_USE_CALLBACKS
  Configuration, 847
  PAL Driver, 466
  PAL_WHOLE_PORT
  PAL Driver, 490
palcallback_t
  PAL Driver, 501
palClearLine
  PAL Driver, 482
palClearPad
  PAL Driver, 479
palClearPort
  PAL Driver, 474
PALConfig, 1012
palDisableLineEvent
  PAL Driver, 487
palDisableLineEventI
  PAL Driver, 486
palDisablePadEvent
  PAL Driver, 485
palDisablePadEventI
  PAL Driver, 484
palEnableLineEvent
  PAL Driver, 487
palEnableLineEventI
  PAL Driver, 486
palEnablePadEvent
  PAL Driver, 485
palEnablePadEventI
  PAL Driver, 484
palevent_t, 1013
  arg, 1014
  cb, 1013
  threads, 1013
palInit
  PAL Driver, 472
pallsLineEventEnabledX
  PAL Driver, 488
pallsPadEventEnabledX
  PAL Driver, 488
palReadBus
  PAL Driver, 502
palReadGroup
  PAL Driver, 475
palReadLatch
  PAL Driver, 473
palReadLine
  PAL Driver, 481
palReadPad
  PAL Driver, 481
palReadPort
  PAL Driver, 477
palSetBusMode
  PAL Driver, 472
palSetGroupMode
  PAL Driver, 503
palSetLine
  PAL Driver, 477
palSetLineCallback
  PAL Driver, 482
  PAL Driver, 466
PAL Driver, 489
palSetLineCallback
PAL Driver, 504
palSetLineMode
PAL Driver, 483
palSetPad
PAL Driver, 479
palSetPadCallback
PAL Driver, 489
palSetPadCallback
PAL Driver, 503
palSetPadMode
PAL Driver, 480
palSetPort
PAL Driver, 474
palToggleLine
PAL Driver, 483
palTogglePad
PAL Driver, 480
palTogglePort
PAL Driver, 475
palWaitLineTimeout
PAL Driver, 507
palWaitLineTimeoutS
PAL Driver, 506
palWaitPadTimeout
PAL Driver, 505
palWaitPadTimeoutS
PAL Driver, 504
palWriteBus
PAL Driver, 502
palWriteGroup
PAL Driver, 476
palWriteLine
PAL Driver, 481
palWritePad
PAL Driver, 478
palWritePort
PAL Driver, 473
param
event_source, 931
period
PWMConfig, 1017
PWMDriver, 1019
period_cb
ICUConfig, 973
Peripheral Interfaces, 510
PLATFORM_ADC_USE_ADC1
ADC Driver, 38
PLATFORM_CAN_USE_CAN1
CAN Driver, 87
PLATFORM_CRY_USE_CRY1
Crypto Driver, 110
PLATFORM_DAC_USE_DAC1
DAC Driver, 189
PLATFORM_EFL_USE_EFL1
EFL Driver, 207
PLATFORM_GPT_USE_GPT1
GPT Driver, 233
PLATFORM_I2C_USE_I2C1
I2C Driver, 253
PLATFORM_I2S_USE_I2S1
I2S Driver, 268
PLATFORM_ICU_USE_ICU1
ICU Driver, 285
PLATFORM_MAC_USE_MAC1
MAC Driver, 354
PLATFORM_PWM_USE_PWM1
PWM Driver, 522
PLATFORM_RTC_USE_RTC1
RTC Driver, 568
PLATFORM_SDC_USE_SDC1
SDC Driver, 584
PLATFORM_SERIAL_USE_USART1
Serial Driver, 621
PLATFORM_SIO_USE_SIO1
SIO Driver, 652
PLATFORM_SPI_USE_SPI1
SPI Driver, 673
PLATFORM_TRNG_USE_TRNG1
TRNG Driver, 709
PLATFORM_UART_USE_UART1
UART Driver, 730
PLATFORM_USB_USE_USB1
USB Driver, 772
PLATFORM_WDG_USE_WDG1
WDG Driver, 805
PLATFORM_WSPI_USE_WSPI1
WSPI Driver, 820
portid
IBus, 982
ps_error_t
Abstract Persistent Storage Interface, 328
ps_offset_t
Abstract Persistent Storage Interface, 328
psGetStorageSize
Abstract Persistent Storage Interface, 326
psRead
Abstract Persistent Storage Interface, 327
psWrite
Abstract Persistent Storage Interface, 327
ptr
io_buffers_queue, 977
PWM Driver, 511
PLATFORM_PWM_USE_PWM1, 522
PWM_CHANNELS, 522
PWM_DEGREES_TO_WIDTH, 516
PWM_FRACTION_TO_WIDTH, 515
pwm_lld_change_period, 522
pwm_lld_change_channel, 536
pwm_lld_disable_channel_notification, 538
pwm_lld_disable_periodic_notification, 537
pwm_lld_enable_channel, 535
pwm_lld_enable_channel_notification, 537
pwm_lld_enable_periodic_notification, 536
pwm_lld_init, 534

ChibiOS/HAL
INDEX 1265

pwm_lld_start, 534
pwm_lld_stop, 535
PWM_OUTPUT_ACTIVE_HIGH, 515
PWM_OUTPUT_ACTIVE_LOW, 515
PWM_OUTPUT_DISABLED, 514
PWM_OUTPUT_MASK, 514
PWM_PERCENTAGE_TO_WIDTH, 516
PWM_READY, 525
PWM_STOP, 525
PWM_UNINIT, 525
pwmcallback_t, 523
pwmChangePeriod, 527
pwmChangePeriodt, 517
pwmcnt_t, 524
pwmcntck_t, 524
pwmcntc_t, 524
PWM_CHANNELS
PWMDriver, 522
PWM_DEGREES_TO_WIDTH
PWM Driver, 516
PWM_FRACTION_TO_WIDTH
PWM Driver, 516
pwm_lld_change_period
PWM Driver, 515
pwm梠ld_disable_channel
PWM Driver, 536
pwm梠ld_disable_channel_notification
PWM Driver, 538
pwm梠ld_disable_periodic_notification
PWM Driver, 537
pwm梠ld_enable_channel
PWM Driver, 435
pwm梠ld_enable_channel_notification
PWM Driver, 537
pwm梠ld_enable_periodic_notification
PWM Driver, 536
pwm梠ld_init
PWM Driver, 536
PWM Driver, 534
PWM Driver, 534
PWM Driver, 535
PWM Driver, 515
PWM Driver, 515
PWM Driver, 514
PWM Driver, 514
PWM Driver, 516
PWM Driver, 525
PWM Driver, 525
PWM Driver, 525
PWM Driver, 523
PWM Driver, 528
PWM Driver, 518
PWM Driver, 533
PWM Driver, 521
PWM Driver, 531
PWM Driver, 520
PWMDriver, 523
PWM Driver, 522
PWM Driver, 516
PWM Driver, 526
PWM Driver, 526
PWM Driver, 526
PWM_CHANNELS
PWM Driver, 522
PWM DEGREES TO WIDTH
PWM Driver, 516
PWM FRACTION TO WIDTH
PWM Driver, 516
PWM Driver, 535
PWM Driver, 518
PWM Driver, 533
PWM Driver, 521
PWM Driver, 520
PWMDriver, 522
PWM Driver, 516
PWM Driver, 526
PWM Driver, 526
PWM_CHANNELS
PWM Driver, 522
PWM DEGREES TO WIDTH
PWM Driver, 516
PWM FRACTION TO WIDTH
PWM Driver, 516
PWM Driver, 535
PWM Driver, 518
PWM Driver, 533
PWM Driver, 521
PWM Driver, 520
PWMDriver, 522
PWM Driver, 516
PWM Driver, 526
PWM Driver, 526
PWM_CHANNELS
PWM Driver, 522
PWM DEGREES TO WIDTH
PWM Driver, 516
PWM FRACTION TO WIDTH
PWM Driver, 516
PWM Driver, 535
PWM Driver, 518
PWM Driver, 533
PWM Driver, 521
PWM Driver, 520
PWMDriver, 522
PWM Driver, 516
PWM Driver, 526
PWM Driver, 526
PWM_CHANNELS
PWM Driver, 522
PWM DEGREES TO WIDTH
PWM Driver, 516
PWM FRACTION TO WIDTH
PWM Driver, 516

ChibiOS/HAL
enabled, 1019
period, 1019
PWM Driver, 523
state, 1019
pwmEnableChannel
PWM Driver, 528
pwmEnableChannelI
PWM Driver, 518
pwmEnableChannelNotification
PWM Driver, 532
pwmEnableChannelNotificationI
PWM Driver, 521
pwmEnablePeriodicNotification
PWM Driver, 530
pwmEnablePeriodicNotificationI
PWM Driver, 520
pwmInit
PWM Driver, 525
pwmIsChannelEnabledI
PWM Driver, 519
pwmmode_t
PWM Driver, 524
pwmObjectInit
PWM Driver, 525
pwmStart
PWM Driver, 526
pwmstate_t
PWM Driver, 524
pwmStop
PWM Driver, 526
q_buffer
io_queue, 980
q_counter
io_queue, 980
Q_EMPTY
I/O Bytes Queues, 542
Q_FULL
I/O Bytes Queues, 543
q_link
io_queue, 981
q_notify
io_queue, 981
Q_OK
I/O Bytes Queues, 542
q_rdptr
io_queue, 981
Q_RESET
I/O Bytes Queues, 542
Q_TIMEOUT
I/O Bytes Queues, 542
q_top
io_queue, 980
q_waiting
io_queue, 980
q_wrptr
io_queue, 980
qGetLink
I/O Bytes Queues, 544
qnotify_t
I/O Bytes Queues, 551
qSetLink
I/O Bytes Queues, 544
qSizeX
I/O Bytes Queues, 543
qSpacesI
I/O Bytes Queues, 543
rca
SDCDriver, 1028
rdevent
MACDriver, 985
rdqueue
MACDriver, 985
read_CxD
MMC over SPI Driver, 415
receiving
USBDriver, 1060
recv1
MMC over SPI Driver, 411
recv3
MMC over SPI Driver, 412
requests_hook_cb
USBConfig, 1057
reserved1
mfs_bank_header_t, 991
RET_ON_ERROR
Managed Flash Storage Driver, 372
RTC Driver, 565
_RTC_driver_methods, 567
PLATFORM_RTC_USE_RTC1, 568
RTC_ALARMS, 567
RTC_BASE_YEAR, 567
RTC_HAS_STORAGE, 567
rtc_lld_driver_fields, 568
rtc_lld_get_alarm, 577
rtc_lld_get_time, 576
rtc_lld_init, 575
rtc_lld_set_alarm, 576
rtc_lld_set_time, 575
RTC_SUPPORTS_CALLBACKS, 567
rtcalarm_t, 568
rtccb_t, 568
rtcConvertDateTimeToFAT, 574
rtcConvertDateTimeToStructTm, 573
rtcConvertStructTmToDateTime, 574
RTCDriver, 568
rtcevent_t, 569
rtcGetAlarm, 572
rtcGetTime, 570
rtcInit, 569
rtcObjectInit, 569
rtcSetAlarm, 571
rtcSetCallback, 573
rtcSetTime, 570
RTC_ALARMS
RTC Driver, 567
RTC_BASE_YEAR
RTC Driver, 567
RTC_HAS_STORAGE
RTC Driver, 567
rtc_lld_driver_fields
RTC Driver, 568
rtc_lld_get_alarm
RTC Driver, 577
rtc_lld_get_time
RTC Driver, 576
rtc_lld_init
RTC Driver, 575
rtc_lld_set_alarm
RTC Driver, 576
rtc_lld_set_time
RTC Driver, 575
RTC_SUPPORTS_CALLBACKS
RTC Driver, 567
RTCAlarm, 1020
rtcalarm_t
RTC Driver, 568
rtcb_t
RTC Driver, 568
rtcConvertDateTimeToFAT
RTC Driver, 574
rtcConvertDateTimeToStructTm
RTC Driver, 573
rtcConvertStructTmToDateTm
RTC Driver, 574
RTCDateTime, 1020
day, 1022
dayofweek, 1022
dstflag, 1021
millisecond, 1022
month, 1021
year, 1021
RTCDriver, 1022
RTC Driver, 568
vmt, 1023
RTCDriverVMT, 1024
rtcevent_t
RTC Driver, 569
rtcGetAlarm
RTC Driver, 572
rtcGetTime
RTC Driver, 570
rtcInit
RTC Driver, 569
rtcnt_t
OSAL, 875
rtcObjectInit
RTC Driver, 569
rtcSetAlarm
RTC Driver, 571
rtcSetCallback
RTC Driver, 573
rtcSetTime
RTC Driver, 570
RTR
CANRxFrame, 922
CANTxFrame, 925
rx_buffer
hal_i2s_config, 955
rxbuf
USBOUTendpointState, 1069
rxchar_cb
UARTConfig, 1049
rxcnt
USBOUTendpointState, 1069
rxend_cb
UARTConfig, 1049
rxerr_cb
UARTConfig, 1049
rxevt_cb
hal_sio_config, 958
rfull_event
CANDriver, 919
rxne_cb
hal_sio_config, 958
rqueue
CANDriver, 919
rxsize
USBOUTendpointState, 1069
rxstate
UARTDriver, 1051
samples
hal_adc_driver, 945
hal_dac_driver, 950
saved_state
USBDriver, 1063
SD1
Serial Driver, 631
SD_BREAK_DETETECTED
Serial Driver, 615
SD_FRAMING_ERROR
Serial Driver, 615
sd_lld_init
Serial Driver, 629
sd_lld_start
Serial Driver, 630
sd_lld_stop
Serial Driver, 630
SD_NOISE_ERROR
Serial Driver, 615
SD_OVERRUN_ERROR
Serial Driver, 615
SD_PARITY_ERROR
Serial Driver, 615
SD_QUEUE_FULL_ERROR
Serial Driver, 616
SD_READY
Serial Driver, 622
SD_STOP
Serial Driver, 622
sd_switch_function_t
SDC Driver, 586
sd_switch_t
INDEX

SDC_NICE_WAITING
  Configuration, 851
  SDC Driver, 582
sdc_set_bus_width
  SDC Driver, 593
sd_vmt
  SDC Driver, 610
sdcbusclk_t
  SDC Driver, 586
sdcbusmode_t
  SDC Driver, 586
SDCConfig, 1025
  bus_width, 1026
sdConnect
  SDC Driver, 598
SDCD1
  SDC Driver, 611
sdDisconnect
  SDC Driver, 598
SDCDriver, 1026
  buf, 1028
  cardmode, 1028
  config, 1028
  errors, 1028
  rca, 1028
  SDC Driver, 585
  vmt, 1028
SDCDriverVMT, 1029
sdErase
  SDC Driver, 603
sdflags_t
  SDC Driver, 585
sdGetAndClearErrors
  SDC Driver, 601
sdGetInfo
  SDC Driver, 602
sdInit
  SDC Driver, 595
sdIsCardInserted
  SDC Driver, 583
sdIsWriteProtected
  SDC Driver, 584
sdmode_t
  SDC Driver, 585
sdObjectInit
  SDC Driver, 585
sdObjectInit
  SDC Driver, 596
sdControl
  Serial Driver, 629
sdRead
  SDC Driver, 599
sdStart
  SDC Driver, 596
sdStop
  SDC Driver, 597
sdSync
  SDC Driver, 601
sdWrite
  SDC Driver, 600

sdGet
  Serial Driver, 618
sdGetl
  Serial Driver, 617
sdGetTimeout
  Serial Driver, 618
sdGetWouldBlock
  Serial Driver, 628
sdIncomingData
  Serial Driver, 625
sdInit
  Serial Driver, 623
sdObjectInit
  Serial Driver, 623
sdPut
  Serial Driver, 617
sdPutl
  Serial Driver, 616
sdPutTimeout
  Serial Driver, 617
sdPutWouldBlock
  Serial Driver, 627
sdRead
  Serial Driver, 620
sdReadl
  Serial Driver, 620
sdReadTimeout
  Serial Driver, 620
sdRequestData
  Serial Driver, 626
sdStart
  Serial Driver, 624
sdstate_t
  Serial Driver, 622
sdStop
  Serial Driver, 622

SDU_READY
  Serial over USB Driver, 635
SDU_STOP
  Serial over USB Driver, 635
SDU_UNINIT
  Serial over USB Driver, 635
sdConfigureHook
  Serial over USB Driver, 639
sdControl
  Serial over USB Driver, 643
sdDataReceived
  Serial over USB Driver, 642
sdDataTransmitted
  Serial over USB Driver, 641
sdulInit
  Serial over USB Driver, 636
sdInterruptTransmitted
  Serial over USB Driver, 642
sdObjectInit
  Serial over USB Driver, 636
sduRequestsHook
  Serial over USB Driver, 640
SIO_UNINIT
   SIO Driver, 655
siocb_t
   SIO Driver, 655
SIOConfig
   SIO Driver, 654
sioControlX
   SIO Driver, 651
SIOD1
   SIO Driver, 660
SIODriver
   SIO Driver, 654
sioebc_t
   SIO Driver, 655
sioflags_t
   SIO Driver, 654
sioGetFlagsX
   SIO Driver, 648
sioInit
   SIO Driver, 656
sioObjectInit
   SIO Driver, 656
sioReadX
   SIO Driver, 650
sioRXGetX
   SIO Driver, 649
sioRXIsEmptyX
   SIO Driver, 648
sioStart
   SIO Driver, 656
siostate_t
   SIO Driver, 655
sioStop
   SIO Driver, 657
sioTXIsFullX
   SIO Driver, 648
sioTXPutX
   SIO Driver, 649
sioWriteX
   SIO Driver, 650
size
   flash_descriptor_t, 937
   flash_sector_descriptor_t, 938
   hal_i2s_config, 955
   MACReceiveDescriptor, 986
   MACTransmitDescriptor, 987
   mfs_data_header_t, 993
   mfs_transaction_op_t, 994
sleep_event
   CANDriver, 920
SNOR BUS DRIVER
   Serial NOR Flash Driver, 445
snor_get_descriptor
   Serial NOR Flash Driver, 446
SNOR_SHARED BUS
   Serial NOR Flash Driver, 445
snor_vmt
   Serial NOR Flash Driver, 454
SNORConfig, 1043
SNORDriver, 1043
   config, 1046
   device_id, 1046
   vmt, 1045
SNORDriverVMT, 1046
snorMemoryMap
   Serial NOR Flash Driver, 452
snorMemoryUnmap
   Serial NOR Flash Driver, 453
snorObjectInit
   Serial NOR Flash Driver, 451
snorStart
   Serial NOR Flash Driver, 451
snorStop
   Serial NOR Flash Driver, 452
sof_cb
   USBConfig, 1057
speed
   SerialConfig, 1031
SPI Driver, 661
   _spi_isr_code, 671
   _spi_isr_full_code, 672
   _spi_isr_half_code, 671
   _spi_wakeup_isr, 670
   PLATFORM_SPI_USE_SPI1, 673
   SPI_ACTIVE, 675
   SPI_COMPLETE, 675
   spi_lld_abort, 695
   spi_lld_config_fields, 673
   spi_lld_driver_fields, 673
   spi_lld_exchange, 693
   spi_lld_ignore, 692
   spi_lld_init, 690
   spi_lld_polled_exchange, 695
   spi_lld_receive, 694
   spi_lld_select, 692
   spi_lld_send, 694
   spi_lld_start, 691
   spi_lld_stop, 691
   spi_lld_unselect, 692
   SPI_READY, 675
   SPI_SELECT_MODE, 665
   SPI_STOP, 674
   SPI_SUPPORTS_CIRCULAR, 673
   SPI_UNINIT, 674
   SPI_USE_CIRCULAR, 665
   SPI_USE_MUTUAL_EXCLUSION, 665
   SPI_USE_WAIT, 664
   spiAbort, 684
   spiAbortI, 683
   spiAcquireBus, 689
   spicableback_t, 674
   SPIConfig, 674
   SPI_D1, 696
   SPIDriver, 674
   spiExchange, 686
   spigate, 685
ChibiOS/HAL
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>spiStartIgnore</td>
<td>679</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spiStartIgnoreI</td>
<td>667</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spiStartReceive</td>
<td>682</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spiStartReceiveI</td>
<td>669</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spiStartSend</td>
<td>681</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spiStartSendI</td>
<td>668</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spistate_t</td>
<td>674</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spiStop</td>
<td>677</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spiUnselect</td>
<td>678</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>spiUnselectI</td>
<td>666</td>
<td>SPI Driver</td>
</tr>
<tr>
<td>ssline</td>
<td>961</td>
<td>hal_spi_config</td>
</tr>
<tr>
<td>ssmsk</td>
<td>962</td>
<td>hal_spi_config</td>
</tr>
<tr>
<td>sspad</td>
<td>962</td>
<td>hal_spi_config</td>
</tr>
<tr>
<td>ssport</td>
<td>961</td>
<td>hal_spi_config</td>
</tr>
<tr>
<td>ST Driver, st_lld_get_alarm</td>
<td>705</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_get_counter</td>
<td>704</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_init</td>
<td>704</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_is_alarm_active</td>
<td>706</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_set_alarm</td>
<td>705</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_start_alarm</td>
<td>705</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_stop_alarm</td>
<td>705</td>
<td></td>
</tr>
<tr>
<td>ST Driver, stGetAlarm</td>
<td>700</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stGetAlarmN</td>
<td>704</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stGetCounter</td>
<td>700</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stInit</td>
<td>698</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stIsAlarmActive</td>
<td>701</td>
<td></td>
</tr>
<tr>
<td>ST Driver, stIsAlarmActiveN</td>
<td>702</td>
<td></td>
</tr>
<tr>
<td>ST Driver, stSetAlarm</td>
<td>699</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stSetAlarmN</td>
<td>703</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stStartAlarm</td>
<td>698</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stStartAlarmN</td>
<td>702</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stStopAlarm</td>
<td>699</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, stStopAlarmN</td>
<td>703</td>
<td>ST Driver</td>
</tr>
<tr>
<td>ST Driver, st_lld_get_alarm</td>
<td>706</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_get_counter</td>
<td>704</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_init</td>
<td>704</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_is_alarm_active</td>
<td>706</td>
<td></td>
</tr>
<tr>
<td>ST Driver, st_lld_set_alarm</td>
<td>705</td>
<td></td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>919</td>
<td>CANDriver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>928</td>
<td>CRYDriver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>940</td>
<td>GPTDriver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>945</td>
<td>hal_adc_driver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>950</td>
<td>hal_dac_driver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>956</td>
<td>hal_i2s_driver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>960</td>
<td>hal_sio_driver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>963</td>
<td>hal_spi_driver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>965</td>
<td>hal_trng_driver</td>
</tr>
<tr>
<td>ST Driver, state</td>
<td>968</td>
<td>hal_wspi_driver</td>
</tr>
<tr>
<td>ST Driver, stGetAlarm</td>
<td>971</td>
<td>I2CDriver</td>
</tr>
<tr>
<td>ST Driver, stGetAlarmN</td>
<td>974</td>
<td>ICUODriver</td>
</tr>
<tr>
<td>ST Driver, stGetCounter</td>
<td>984</td>
<td>MACDriver</td>
</tr>
<tr>
<td>ST Driver, stGetCounter</td>
<td>998</td>
<td>MFSDriver</td>
</tr>
<tr>
<td>ST Driver, stGetCounter</td>
<td>1019</td>
<td>PWMDriver</td>
</tr>
<tr>
<td>ST Driver, stGetCounter</td>
<td>1051</td>
<td>UARTDriver</td>
</tr>
<tr>
<td>ST Driver, stGetCounter</td>
<td>1060</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stGetCounter</td>
<td>1071</td>
<td>WDGDriver</td>
</tr>
<tr>
<td>ST Driver, stGetAlarm</td>
<td>1062</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stGetAlarmN</td>
<td>700</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stGetAlarmN</td>
<td>704</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stGetCounter</td>
<td>700</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stInit</td>
<td>698</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stIsAlarmActive</td>
<td>701</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stIsAlarmActiveN</td>
<td>702</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stSetAlarm</td>
<td>699</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stSetAlarmN</td>
<td>703</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stStartAlarm</td>
<td>698</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stStartAlarmN</td>
<td>702</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stStopAlarm</td>
<td>699</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ST Driver, stStopAlarmN</td>
<td>703</td>
<td>USBDriver</td>
</tr>
<tr>
<td>ChibiOS/HAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INDEX

Support Code, 860
suspended
   io_buffers_queue, 976
sync
   MMC over SPI Driver, 416
sysinterval_t
   OSAL, 875
syssts_t
   OSAL, 875
 systime_t
   OSAL, 875
tdqueue
   MACDriver, 985
thread
   hal_adc_driver, 946
   hal_dac_driver, 950
   hal_spi_driver, 963
   hal_wspi_driver, 968
   USBInEndpointState, 1068
   USBOutEndpointState, 1069
thread_reference_t
   OSAL, 875
threaddx
   UARTDriver, 1052
threads
   palevent_t, 1013
   threads_queue_t, 1047
threadx
   UARTDriver, 1052
TIME
   CANRxFrame, 922
top
   io_buffers_queue, 978
tr_limit_offset
   MFSDriver, 1000
tr_next_offset
   MFSDriver, 999
tr_nops
   MFSDriver, 1000
tr_ops
   MFSDriver, 1000
transmitting
   USBDriver, 1060
TRNG Driver, 708
   PLATFORM_TRNG_USE_TRNG1, 709
   trng_config_fields, 710
   trng_driver_fields, 709
   trng_generate, 715
   trng_init, 714
   trng_start, 714
   trng_stop, 715
   TRNG ready, 711
   TRNG_RUNNING, 711
   TRNG_STOP, 711
   TRNG_UNINIT, 711
   TRNGConfig, 710
   TRNGD1, 716
   TRNGDriver, 710
trngGenerate, 713
trngInit, 711
trngObjectInit, 711
trngStart, 712
trngstate_t, 710
trngStop, 712
trng_driver_fields
   TRNG Driver, 709
   TRNG_ready, 710
   TRNG_generate, 715
   TRNG_init, 714
   TRNG_start, 714
   TRNG_stop, 715
   TRNG_RUNNING, 711
   TRNG_STOP, 711
   TRNG_UNINIT, 711
   TRNGConfig, 710
   TRNGD1, 716
   TRNGDriver, 710
trng_init
   TRNG Driver, 714
trng_start
   TRNG Driver, 714
   TRNG_start, 715
   TRNG_UNINIT
   TRNG Driver, 711
   TRNGConfig, 710
   TRNGD1
   TRNG Driver, 716
   TRNGDriver
   TRNG Driver, 710
trngGenerate
   TRNG Driver, 710
   trngInit
   TRNG Driver, 711
   trngObjectInit
   TRNG Driver, 711
   trngStart
   TRNG Driver, 712
trngstate_t
   TRNG Driver, 710
trngStop
   TRNG Driver, 712
tx_buffer
   hal_i2s_config, 954
txbuf
   USBInEndpointState, 1068
txcnt
   USBInEndpointState, 1067
txempt_event
   CANDriver, 919
txend1_cb
   UARTConfig, 1049
txend2_cb
   UARTConfig, 1049
txend_cb
   hal_sio_config, 958

ChibiOS/HAL
txnf_cb
  hal_sio_config, 958
txqueue
  CANDriver, 919
txsize
  USBInEndpointState, 1067
txstate
  UARTDriver, 1051
UART Driver, 717
  _uart_rx_char_match_isr_code, 730
  _uart_rx_complete_isr_code, 727
  _uart_rx_error_isr_code, 728
  _uart_rx_idle_code, 728
  _uart_timeout_isr_code, 729
  _uart_tx1_isr_code, 726
  _uart_tx2_isr_code, 726
  _uart_wakeup_rx_cm_isr, 725
  _uart_wakeup_rx_complete_isr, 724
  _uart_wakeup_rx_error_isr, 724
  _uart_wakeup_rx_timeout_isr, 725
  _uart_wakeup_tx1_isr, 723
  _uart_wakeup_tx2_isr, 723
PLATFORM_UART_USE_UART1, 730
UART_BREAK_DETECTED, 722
UART_FRAMING_ERROR, 722
uart_lld_init, 748
uart_lld_start, 749
uart_lld_start_receive, 751
uart_lld_start_send, 750
uart_lld_stop, 749
uart_lld_stop_receive, 751
uart_lld_stop_send, 750
UART_NO_ERROR, 721
UART_NOISE_ERROR, 722
UART_OVERRUN_ERROR, 722
UART_PARITY_ERROR, 722
UART_READY, 732
UART_RX_ACTIVE, 733
UART_RX_COMPLETE, 733
UART_RX_IDLE, 733
UART_STOP, 732
UART_TX_ACTIVE, 732
UART_TX_COMPLETE, 732
UART_TX_IDLE, 732
UART_UNINIT, 732
UART_USE_MUTUAL_EXCLUSION, 723
UART_USE_WAIT, 722
uart_acquireBus, 747
uartcb_t, 731
uartccb_t, 731
UARTD1, 752
UARTDriver, 731
uartecb_t, 732
uartflags_t, 731
uartInit, 733
uartObjectInit, 733
uartReceiveTimeout, 746
uartReleaseBus, 748
uarttxstate_t, 733
uartSendFullTimeout, 745
uartSendTimeout, 744
uartStart, 734
uartStartReceive, 740
uartStartReceiveI, 741
uartStartSend, 736
uartStartSendl, 737
uartstate_t, 732
uartStop, 735
uartStopReceive, 742
uartStopReceiveI, 743
uartStopSend, 738
uartStopSendl, 739
uarttxstate_t, 732
UART_BREAK_DETECTED
UART Driver, 722
UART_FRAMING_ERROR
UART Driver, 722
uart_lld_init
  UART Driver, 748
uart_lld_start
  UART Driver, 749
uart_lld_start_receive
  UART Driver, 751
uart_lld_start_send
  UART Driver, 750
uart_lld_stop
  UART Driver, 749
uart_lld_stop_receive
  UART Driver, 751
uart_lld_stop_send
  UART Driver, 750
UART_NO_ERROR
UART Driver, 721
UART_NOISE_ERROR
UART Driver, 722
UART_OVERRUN_ERROR
UART Driver, 722
UART_PARITY_ERROR
UART Driver, 722
UART_READY
UART Driver, 732
UART_RX_ACTIVE
UART Driver, 733
UART_RX_COMPLETE
UART Driver, 733
UART_RX_IDLE
UART Driver, 733
UART_STOP
UART Driver, 732
UART_TX_ACTIVE
UART Driver, 732
UART_TX_COMPLETE
UART Driver, 732
UART_TX_IDLE
UART Driver, 732
UART_UNINIT
UART Driver, 732
UART_USE_MUTUAL_EXCLUSION
  Configuration, 853
UART Driver, 723
UART_USE_WAIT
  Configuration, 853
UART Driver, 722
uartAcquireBus
  UART Driver, 747
uartcb_t
  UART Driver, 731
uartccb_t
  UART Driver, 731
UARTConfig, 1048
rxchar_cb, 1049
rxchar_cb, 1049
rxend_cb, 1049
rxerr_cb, 1049
txend1_cb, 1049
txend2_cb, 1049
UARTDriver, 1050
UARTD1
  UART Driver, 752
UARTDriver, 1050
config, 1051
early, 1051
mutex, 1052
rxstate, 1051
state, 1051
threadrx, 1052
txstate, 1051
UART Driver, 731
uartecb_t
  UART Driver, 732
uartflags_t
  UART Driver, 731
uartinit
  UART Driver, 733
uartObjectInit
  UART Driver, 733
uartReceiveTimeout
  UART Driver, 746
uartReleaseBus
  UART Driver, 748
uartrxstate_t
  UART Driver, 733
uartSendFullTimeout
  UART Driver, 745
uartSendTimeout
  UART Driver, 744
uartStart
  UART Driver, 734
uartStartReceive
  UART Driver, 740
uartStartReceiveI
  UART Driver, 741
uartStartSend
  UART Driver, 736
uartStartSendI
  UART Driver, 737
uartstate_t
  UART Driver, 732
uartStop
  UART Driver, 735
uartStopReceive
  UART Driver, 742
uartStopReceiveI
  UART Driver, 743
uartStopSend
  UART Driver, 738
uartStopSendI
  UART Driver, 739
uarttxstate_t
  UART Driver, 732
ud_size
  USB Descriptor, 1058
ud_string
  USB Descriptor, 1058
unpacked_mmc_cid_t, 1052
unpacked_mmc_csd_t, 1053
unpacked_sdc_cid_t, 1053
unpacked_sdc_csd_10_t, 1054
unpacked_sdc_csd_20_t, 1055
USB CDC Header, 802
USB Driver, 753
_usb_ep0in, 791
_usb_ep0out, 791
_usb_ep0setup, 790
_usb_isr_invoke_event_cb, 769
_usb_isr_invoke_in_cb, 770
_usb_isr_invoke_out_cb, 771
_usb_isr_invoke_setup_cb, 770
_usb_isr_invoke_sof_cb, 769
_usb_reset, 788
_usb_suspend, 789
_usb_wakeup, 790
default_handler, 778
ep0_state, 800
ep0config, 801
EP_STATUS_ACTIVE, 776
EP_STATUS_DISABLED, 776
EP_STATUS_STALLED, 776
in, 800
out, 800
PLATFORM_USB_USE_USB1, 772
set_address, 778
USB_ACTIVE, 776
USB_DESC_BCD, 760
USB_DESC_BYTE, 760
USB_DESC_CONFIGURATION, 761
USB_DESC_CONFIGURATION_SIZE, 761
USB_DESC_DEVICE, 761
USB_DESC_ENDPOINT, 763
USB_DESC_ENDPOINT_SIZE, 763
USB_DESC_INDEX, 760
USB_DESC_INTERFACE, 762
USB_DESC_INTERFACE_ASSOCIATION, 763
USB_DESC_INTERFACE_ASSOCIATION_SIZE, 762
USB_DESC_INTERFACE_SIZE, 762
USB_DESC_WORD, 760
USB_EP0_ERROR, 777
USB_EP0_IN_SENDING_STS, 777
USB_EP0_IN_TX, 777
USB_EP0_IN_WAITING_TX0, 777
USB_EP0_OUT_RX, 777
USB_EP0_OUT_WAITING_STS, 777
USB_EP0_STATUS_STAGE, 771
USB_EP0_STP_WAITING, 777
USB_EP_MODE_TYPE, 763
USB_EP_MODE_TYPE_BULK, 764
USB_EP_MODE_TYPE_CTRL, 764
USB_EP_MODE_TYPE_INTR, 764
USB_EP_MODE_TYPE_ISO, 764
USB_EVENT_ADDRESS, 777
USB_EVENT_CONFIGURED, 777
USB_EVENT_RESET, 777
USB_EVENT_SUSPENDED, 777
USB_EVENT_UNCONFIGURED, 777
USB_EVENT_WAKEUP, 777
usb_lld_clear_in, 799
usb_lld_clear_out, 799
usb_lld_connect_bus, 773
usb_lld_disable_endpoints, 795
usb_lld_disconnect_bus, 773
usb_lld_get_frame_number, 772
usb_lld_get_status_in, 796
usb_lld_get_status_out, 795
usb_lld_get_transaction_size, 773
usb_lld_init, 792
usb_lld_init_endpoint, 794
usb_lld_prepare_receive, 797
usb_lld_prepare_transmit, 797
usb_lld_read_setup, 796
usb_lld_reset, 793
usb_lld_set_address, 794
usb_lld_set_out, 798
usb_lld_start, 792
usb_lld_start_in, 798
usb_lld_start_out, 797
usb_lld_stop, 793
usb_lld_wakeup_host, 774
USB_MAX_ENDPOINTS, 771
USB_READY, 776
USB_SELECTED, 776
USB_SET_ADDRESS_ACK_HANDLING, 772
USB_SET_ADDRESS_MODE, 772
USB_STOP, 776
USB_SUSPENDED, 776
USB_UNINIT, 776
USB_USE_WAIT, 764
usbcallback_t, 774
usbConnectBus, 765
USB1, 800
usbDisableEndpointsI, 782
usbDisconnectBus, 765
USBDriver, 774
usbep0state_t, 777
usbep_t, 774
usbepcallback_t, 775
usbepstatus_t, 776
usbevent_t, 777
usbeventcb_t, 775
usbGetDescriptor_t, 800
usbGetDriverStateI, 764
usbGetReceiveStatusI, 766
usbGetReceiveTransactioSizeX, 767
usbGetTransmitStatusI, 766
usblnit, 779
usblnitEndpointI, 781
usbObjectInit, 779
usbReadSetup, 768
usbReceive, 784
usbreqhandler_t, 775
usbSetupTransfer, 768
usbStartReceiveI, 786
usbStartTransmitI, 787
usbStart, 780
usbStartReceiveI, 783
usbStartTransmitI, 783
usbstate_t, 776
usbStop, 780
usbTransmit, 785
usbWakeupHost, 788
USB_ACTIVE
USB Driver, 776
USB_DESC_BCD
USB Driver, 760
USB_DESC_BYTE
USB Driver, 760
USB_DESC_CONFIGURATION
USB Driver, 761
USB_DESC_CONFIGURATION_SIZE
USB Driver, 761
USB_DESC_ENDPOINT
USB Driver, 761
USB_DESC_ENDPOINT_SIZE
USB Driver, 763
USB_DESC_INDEX
USB Driver, 760
USB_DESC_INTERFACE
USB Driver, 762
USB_DESC_INTERFACE_ASSOCIATION
USB Driver, 763
USB_DESC_INTERFACE_ASSOCIATION_SIZE
USB Driver, 762
USB_DESC_INTERFACE_SIZE
USB Driver, 762
USB Driver, 800
USB Descriptor, 1058
ud_size, 1058
ud_string, 1058
usbDisableEndpointI
USB Driver, 782
usbDisconnectBus
USB Driver, 765
USBDriver, 1059
address, 1062
cfg, 1060
configuration, 1062
ep0endcb, 1062
ep0n, 1062
ep0next, 1061
ep0state, 1061
epc, 1061
in_params, 1061
out_params, 1061
receiving, 1060
saved_state, 1063
setup, 1062
state, 1060
status, 1062
transmitting, 1060
USB Driver, 774
USBEndpointConfig, 1063
ep_mode, 1065
in_cb, 1065
in_maxsize, 1066
in_state, 1066
out_cb, 1066
out_maxsize, 1066
out_state, 1066
setup_cb, 1065
usbep0state_t
USB Driver, 777
usbep_t
USB Driver, 774
usbepcallback_t
USB Driver, 775
usbepstatus_t
USB Driver, 776
usbevent_t
USB Driver, 777
usbeventcb_t
USB Driver, 775
usbgetdescriptor_t
USB Driver, 800
usbGetDriverStateI
USB Driver, 764
usbGetFrameNumberX
USB Driver, 765
usbGetReceiveStatusI
USB Driver, 766
usbGetReceiveTransactionSizeX
USB Driver, 767
usbGetTransmitStatusI
USB Driver, 766
USBInEndpointState, 1067
thread, 1068
txbuf, 1068
txcnt, 1067
txsize, 1067
usblnit
USB Driver, 779
usbInitEndpointI
USB Driver, 781
usbObjectInit
USB Driver, 779
USBOutEndpointState, 1068
rxbuf, 1069
rxcnt, 1069
rxsize, 1069
thread, 1069
usbp
SerialUSBConfig, 1036
usbReadSetup
USB Driver, 768
usbReceive
USB Driver, 784
usbreqhandler_t
USB Driver, 775
usbSetupTransfer
USB Driver, 768
usbStallReceiveI
USB Driver, 786
usbStallTransmitI
USB Driver, 787
usbStart
USB Driver, 780
usbStartReceiveI
USB Driver, 783
usbStartTransmitI
USB Driver, 783
usbstate_t
USB Driver, 776
usbStop
USB Driver, 780
usbTransmit
USB Driver, 785
usbWakeupHost
USB Driver, 788
used_space
MFSDriver, 999
vmt
BaseAsynchronousChannel, 896
BaseBlockDevice, 900
BaseChannel, 903
BaseFlash, 907
BaseObject, 910
BasePersistentStorage, 912
BaseSequentialStream, 914
FileStream, 933
hal_efi_driver, 953
MemoryStream, 989
MMCDriver, 1004
MMCSDBlockDevice, 1008
NullStream, 1011
RTCDriver, 1023
SDCDriver, 1028
SerialDriver, 1033
SerialUSBDriver, 1039
SNORDriver, 1045

wait
  MMC over SPI Driver, 410
waiting
  io_buffers_queue, 976
wakeup_event
  CANDriver, 920
WDG Driver, 804
  PLATFORM_WDG_USE_WDG1, 805
    wdg_lld_init, 808
    wdg_lld_reset, 809
    wdg_lld_start, 809
    wdg_lld_stop, 809
  WDG READY, 806
  WDG STOP, 806
  WDG UNINIT, 806
  WDGDriver, 805
  wdgInit, 806
  wdgReset, 808
  wdgReSet, 805
  wdgStart, 806
  wdgstate_t, 806
  wdgStop, 807

wdg_lld_init
  WDG Driver, 808
wdg_lld_reset
  WDG Driver, 809
wdg_lld_start
  WDG Driver, 809
wdg_lld_stop
  WDG Driver, 809
WDG READY
  WDG Driver, 806
WDG STOP
  WDG Driver, 806
WDG UNINIT
  WDG Driver, 806
WDGConfig, 1070
  WDGDriver, 1070
    config, 1071
    state, 1071
  WDG Driver, 805
wdgInit
  WDG Driver, 806
wdgReset
  WDG Driver, 808
wdgResetI
  WDG Driver, 805
wdgStart
  WDG Driver, 806
wdgstate_t
  WDG Driver, 806
wdgStop
  WDG Driver, 807
width cb
  ICUConfig, 972
WSPI Driver, 811
  __wspi_error_code, 819
  __wspi_isr_code, 819
  __wspi_wakeup_isr, 818
  PLATFORM_WSPI_USE_WSPI1, 820
  WSPI COMPLETE, 822
  wspi_lld_command, 835
  wspi_lld_config_fields, 820
  wspi_lld_driver_fields, 820
  wspi_lld_init, 834
  wspi_lld_map_flash, 836
  wspi_lld_receive, 836
  wspi_lld_send, 835
  wspi_lld_start, 834
  wspi_lld_stop, 834
  wspi_lld_unmap_flash, 837
  WSPI_MEMMAP, 822
  WSPI READY, 822
  WSPI RECEIVE, 822
  WSPI SEND, 822
  WSPI STOP, 822
  WSPI UNINIT, 822
  WSPI_USE_MUTUAL_EXCLUSION, 815
  WSPI_USE_WAIT, 814
  wspiAcquireBus, 832
  wspiCallback_t, 821
  wspiCommand, 829
  WSPIConfig, 821
  WSPI_D1, 837
  WSPIDriver, 821
  wspiInit, 822
  wspiMapFlash, 831
  wspiMapFlashI, 817
  wspiObjectInit, 822
  wspiReceive, 830
  wspiReleaseBus, 833
  wspiSend, 830
  wspiStart, 824
  wspiStartCommand, 826
  wspiStartCommandI, 815
  wspiStartReceive, 828
  wspiStartReceiveI, 816
  wspiStartSend, 827
  wspiStartSendI, 816
  wspistate_t, 821
  wspiStop, 825
  wspiUnmapFlash, 832
  wspiUnmapFlashI, 818
  wspi_command_t, 1071
    addr, 1072
    alt, 1072
    cfg, 1072
cmd, 1072
dummy, 1072
WSPI_COMPLETE
  WSPI Driver, 822
wspi_lld_command
  WSPI Driver, 835
wspi_lld_config_fields
  WSPI Driver, 820
wspi_lld_driver_fields
  WSPI Driver, 820
wspi_lld_init
  WSPI Driver, 834
wspi_lld_map_flash
  WSPI Driver, 836
wspi_lld_receive
  WSPI Driver, 836
wspi_lld_send
  WSPI Driver, 835
wspi_lld_start
  WSPI Driver, 834
wspi_lld_stop
  WSPI Driver, 834
wspi_lld_unmap_flash
  WSPI Driver, 837
WSPI_MEMMAP
  WSPI Driver, 822
WSPI_READY
  WSPI Driver, 822
WSPI_RECEIVE
  WSPI Driver, 822
WSPI_SEND
  WSPI Driver, 822
WSPI_STOP
  WSPI Driver, 822
WSPI_UNINIT
  WSPI Driver, 822
WSPI_USE_MUTUAL_EXCLUSION
  Configuration, 854
  WSPI Driver, 815
WSPI_USE_WAIT
  Configuration, 854
  WSPI Driver, 814
wspiAcquireBus
  WSPI Driver, 832
wspicallback_t
  WSPI Driver, 821
wspiCommand
  WSPI Driver, 829
WSPIConfig
  WSPI Driver, 821
WSPID1
  WSPI Driver, 837
WSPIDriver
  WSPI Driver, 821
wspiInit
  WSPI Driver, 822
wspiMapFlashI
  WSPI Driver, 817
wspiObjectInit
  WSPI Driver, 822
wspiReceive
  WSPI Driver, 830
wspiReleaseBus
  WSPI Driver, 833
wspiSend
  WSPI Driver, 830
wspiStart
  WSPI Driver, 824
wspiStartCommand
  WSPI Driver, 826
wspiStartCommandI
  WSPI Driver, 815
wspiStartReceive
  WSPI Driver, 828
wspiStartReceiveI
  WSPI Driver, 816
wspiStartSend
  WSPI Driver, 827
wspiStartSendI
  WSPI Driver, 816
wspistate_t
  WSPI Driver, 821
wspiStop
  WSPI Driver, 825
wspiUnmapFlash
  WSPI Driver, 832
wspiUnmapFlashI
  WSPI Driver, 818
year
  RTCDateTime, 1021